Approches statiques et dynamiques pour l'optimisation de la consommation énergétique des applications de calcul à hautes performances

Mathieu Stoffel

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**
Arrêté ministériel : 25 mai 2016

Présentée par
Mathieu Stoffel

Thèse dirigée par **Frédéric Desprez**,
et coencadrée par **François Broquedis** et **Abdelhafid Mazouz**

préparée au sein du **Laboratoire d'Informatique de Grenoble (LIG)**,
membre de **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII)**

Approches statiques et dynamiques pour l'optimisation de la consommation énergétique des applications de calcul à hautes performances

Static and dynamic approaches for the optimisation of the energy consumption associated with High Performance Computing applications

Thèse soutenue publiquement le **01 Octobre 2021**,
devant le jury composé de :

Laurent Philippe
Professeur des Universités (Université de Bourgogne Franche-Comté - DISC [FEMTO-SC] (Besançon), Président du jury et examinateur

Laurent Lefèvre
Chargé de Recherche Habilité INRIA - Avalon [LIP] (Lyon), Rapporteur

Jean-Marc Pierson
Professeur des Universités (Université Toulouse 3 - Paul Sabatier) - SEPIA [IRIT] (Toulouse), Rapporteur

Denis Trystram
Professeur des Universités (Université Grenoble Alpes) - LIG [UGA] (Grenoble) , Examinateur

Pablo de Oliveira Castro
Maître de Conférence - LIPARAD [UVSQ] (Versailles), Examinateur

Andry Razafinjatovo
Président Directeur Général de Ryax Technologies (Lyon), Examinateur

Frédéric Desprez
Directeur de Recherche INRIA - CORSE [INRIA] (Grenoble), Directeur de thèse

Approches statiques et dynamiques pour l'optimisation de la consommation énergétique des applications de calcul à hautes performances

Le domaine du calcul à hautes performances est un enjeu industriel et académique crucial : de l'astrophysique à la météorologie en passant par la chimie des matériaux, d'Airbus à Total, en passant par Pfizer. Les sciences numériques leurs sont devenues indispensables, et cette dépendance se traduit par un besoin pour toujours plus de puissance de calcul.

Au moment de l'écriture de ce manuscrit, tous les acteurs du calcul à hautes performances redoublent d'efforts pour atteindre l'`ExaScale` : 10^{18} opérations sur nombres à virgule flottante par seconde. Néanmoins, contrairement au passage des précédents jalons (e.g. le `PetaScale`), la puissance de calcul atteinte par un supercalculateur n'est pas la seule grandeur d'intérêt. En effet, les premières estimations de la puissance électrique consommée par un système exaflopique étaient bien trop élevées pour être acceptables, aussi bien du point de vue économique qu'écologique. En conséquence, de nombreux efforts de recherche et de développement visant à réduire la consommation énergétique des supercalculateurs ont vu le jour.

C'est précisément le thème central des travaux présentés par ce manuscrit, qui incluent des contributions significatives à `Bull Dynamic Power Optimizer` (`BDPO`), ainsi que la conception, le développement, et la validation expérimentale de `Phase - Temporality Analyser` (`Phase-TA`).

`BDPO` est un outil de reconfiguration dynamique, c'est-à-dire un daemon exécuté en parallèle d'une application de calcul intensif, qui adapte la fréquence des cœurs de calcul à la charge de travail que ces derniers exécutent. Il a la particularité d'être complètement agnostique de ladite application, ainsi que du support d'exécution, tout en ne requérant aucune configuration de la part de l'utilisateur. L'utilisation de `BDPO` permet de réduire l'énergie consommée associée à l'exécution des applications `NEMO` et `HPCG` d'environ 15%, tout en maintenant la dégradation de performance associée sous les 4%.

`Phase-TA` est un outil d'analyse des profils des applications **itératives** de calcul intensif, notamment ceux produits par `BDPO`. Il détecte les comportements localement périodiques, et les caractérise en construisant des motifs représentatifs des périodicités associées. Ce qui a motivé le développement de `Phase-TA` est de pouvoir fournir à `BDPO` une prédiction pertinente et fiable du comportement à venir de l'application exécutée, de sorte à améliorer l'efficacité des reconfigurations qu'il opère. Il a été montré expérimentalement que les motifs construits par `Phase-TA` sont des représentations pertinentes des périodicités exhibées par les applications de calcul intensif, et qu'une part significative (i.e. plus de deux tiers) du temps d'exécution de ces dernières leur est imputable. Enfin, les performances de `Phase-TA` permettent d'envisager son utilisation pendant l'exécution d'une application de calcul intensif.

Mots clés : Calcul intensif, Application itératives, Consommation énergétique des supercalculateurs, Monitoring, Reconfiguration dynamique, Détection et caractérisation de périodicités

Static and dynamic approaches for the optimisation of the energy consumption associated with High Performance Computing applications

The High Performance Computing (HPC) field is a crucial issue, for both industry and academics: from astrophysics to meteorology, passing by materials science, from Airbus to Total, passing by Pfizer. Computational sciences have become essential, and this dependence implies a neverending urge for more computational power.

At the time of writing, all the actors of the HPC field redouble their efforts to reach the ExaScale: 10^{18} operations on floating point numbers per second. Nevertheless, contrary to the previous milestones (e.g. the PetaScale), the computational power achieved by a supercomputer is not the only key performance indicator. Indeed, the first assessment of the electrical power consumed by an exaflopic system were way too high to be acceptable, from both economic and ecological points of view. Consequently, numerous research and development efforts aiming at making supercomputer more energy-efficient were initiated during the last decade.

That is precisely the main topic of the work presented in this manuscript, which includes significant contributions to Bull Dynamic Power Optimizer (BDPO), and the conception, development, and experimental validation of Phase - Temporality Analyser (Phase-TA).

BDPO is a dynamic reconfiguration tool, that is to say a daemon executed in parallel of an HPC application, which changes the functioning frequency of the cores of the CPUs to the workload the latter are executing. It has the distinctive feature of being completely agnostic of both the aforementioned executed application and its execution environment, while requiring no specific configuration from the user. Using BDPO induces a 15% decrease of the energy consumption associated with the execution of the two applications NEMO and HPCG, while maintaining the associated performance degradation under 4%.

Phase-TA is designed to analyse the profile of an **iterative** HPC application, notably those produced by BDPO. It detects the locally periodic behaviours, and caracterises them by infering representative patterns for the associated periodicities. What motivated the development of Phase-TA was the possibility to build a relevant and reliable prediction of the future behaviour of the executed application, so as to make the reconfigurations performed by BDPO more efficient. It was experimentally shown that the patterns inferred by Phase-TA are relevant representations of the periodicities featured by HPC applications, and that those periodicities are accountable for a significant part (i.e. more than two thirds) of the execution time of the aformentioned applications. Finally, the performances of Phase-TA make it suitable for on-the-fly analysis of the profile of HPC applications.

Key words: High Performance Computing, Iterative applications, Energy consumption of supercomputers, Monitoring, Dynamic reconfiguration, Detection and characterization of periodicities

Sommaire

Glossaire

Ce glossaire regroupe l'intégralité des abréviations utilisées dans le présent manuscrit, ainsi que leurs significations :

[AMD] Advanced Micro Devices ;

[ARM] Acorn RISC Machine ;

[BDPO] Bull Dynamic Power Optimizer ;

[BLAS] Basic Linear Algebra Subprograms ;

[CISC] Complex Instruction Set Computer ;

[CPU] Central Processing Unit ;

[DBA] Dynamic time warping Barycentric Averaging ;

[DCT] Dynamic Concurrency Throttling ;

[DSL] Domain Specific Language ;

[DTW] Dynamic Time Warping ;

[DVFS] Dynamic Voltage Frequency Scaling ;

[EPI] European Processor Initiative ;

[FPGA] Field-Programmable Gate Array ;

[GPGPU] General Purpose Graphics Processing Unit ;

[HPC] High Performance Computing ;

[HPCG] High Performance Conjugate Gradients ;

[HPL] High Performance Linpack ;

[IA] Intelligence Artificielle ;

[IPC] Instructions retired Per reference Cycle ;

[MPI] Message Passing Interface ;

[MSR] Model Specific Register ;

[NAMD] NAnoscale Molecular Dynamics ;

[NEMO] Nucleus for European Modelling of the Ocean ;

[NPB] NAS Parallel Benchmarks ;

[NUMA] Non-Uniform Memory Access ;

[OSPM] Operating System-directed configuration and Power Management ;

[PMC] Performance Monitoring Counter ;

[PMem] Persistent Memory ;

[PMU] Performance Monitoring Unit ;

[Phase-TA] Phase - Temporality Analyser ;

[PUE] Power Usage Effectiveness ;

[RISC] Reduced Instruction Set Computer ;

[SSD] Solid State Drive ;

[TDP] Thermal Design Power ;

[USS] Système UltraScale ;

[WGSS] Within-Group Sum of Squares.

Introduction

1

Pour commencer, la section 1.1 contextualisera la thématique autour de laquelle les travaux de recherche présentés par ce manuscrit gravitent, à savoir la consommation énergétique des supercalculateurs. Ensuite, la section 1.2 esquissera les grandes lignes du déroulé de la thèse, de sorte à permettre au lecteur d'appréhender l'orientation et la démarche suivies au cours des recherches. Enfin, l'organisation du manuscrit sera brièvement présentée par la section 1.3.

1.1 Pourquoi s'intéresser à la consommation énergétique des supercalculateurs ?

La course effrénée à la puissance de calcul est intrinsèque à l'informatique, et ce depuis ses prémisses. En effet, la naissance de l'informatique moderne, généralement associée aux travaux de Turing dans le cadre de la cryptanalyse d'Enigma, avait pour but principal d'accélérer le décodage du chiffrement appliqué par cette dernière.

Pendant plusieurs décennies, les sujets techniques directement liés à l'augmentation de la puissance de calcul des machines à calculer (aussi bien ordinateurs que supercalculateurs) étaient la miniaturisation des transistors et circuits imprimés, et l'augmentation des performances brutes des composants (principalement processeurs, mémoires, bus et réseaux de communication et technologies de stockage). Néanmoins, vers la fin des années 1990, l'augmentation de la fréquence de fonctionnement des processeurs commença à rencontrer des premières limitations, notamment concernant la dissipation de la chaleur produite par les transistors. En conséquence, les fondeurs ne pouvaient plus simplement miser sur une augmentation de la cadence des processeurs et une miniaturisation des transistors pour augmenter la puissance de calcul. Ils ont alors exploré de nouvelles voies pour améliorer les performances de leurs processeurs, notamment la parallélisation.

En effet, au début des années 2000 [1], tous les principaux manufacturiers de processeurs ont présenté leurs premiers processeurs multicœurs. Cela a conduit à une modification profonde de la manière de développer les applications (des logiciels de bureautique aux

1. https://fr.wikipedia.org/wiki/Microprocesseur_multi-c%C5%93ur

simulations numériques à visée scientifique), qui sont conçues pour tirer profit du parallélisme offert par le matériel. Réciproquement, pour répondre à cette faim de parallélisme des applications de calcul intensif, les supercalculateurs offrent une puissance de calcul toujours plus importante via un degré de parallélisme toujours plus grand.

L'évolution du Top500, représentée par la Figure 1.1, illustre notamment le virage du parallélisme : les superordinateurs exhibent un nombre d'unités de calcul toujours plus important, ce qui implique notamment une consommation énergétique toujours plus élevée.

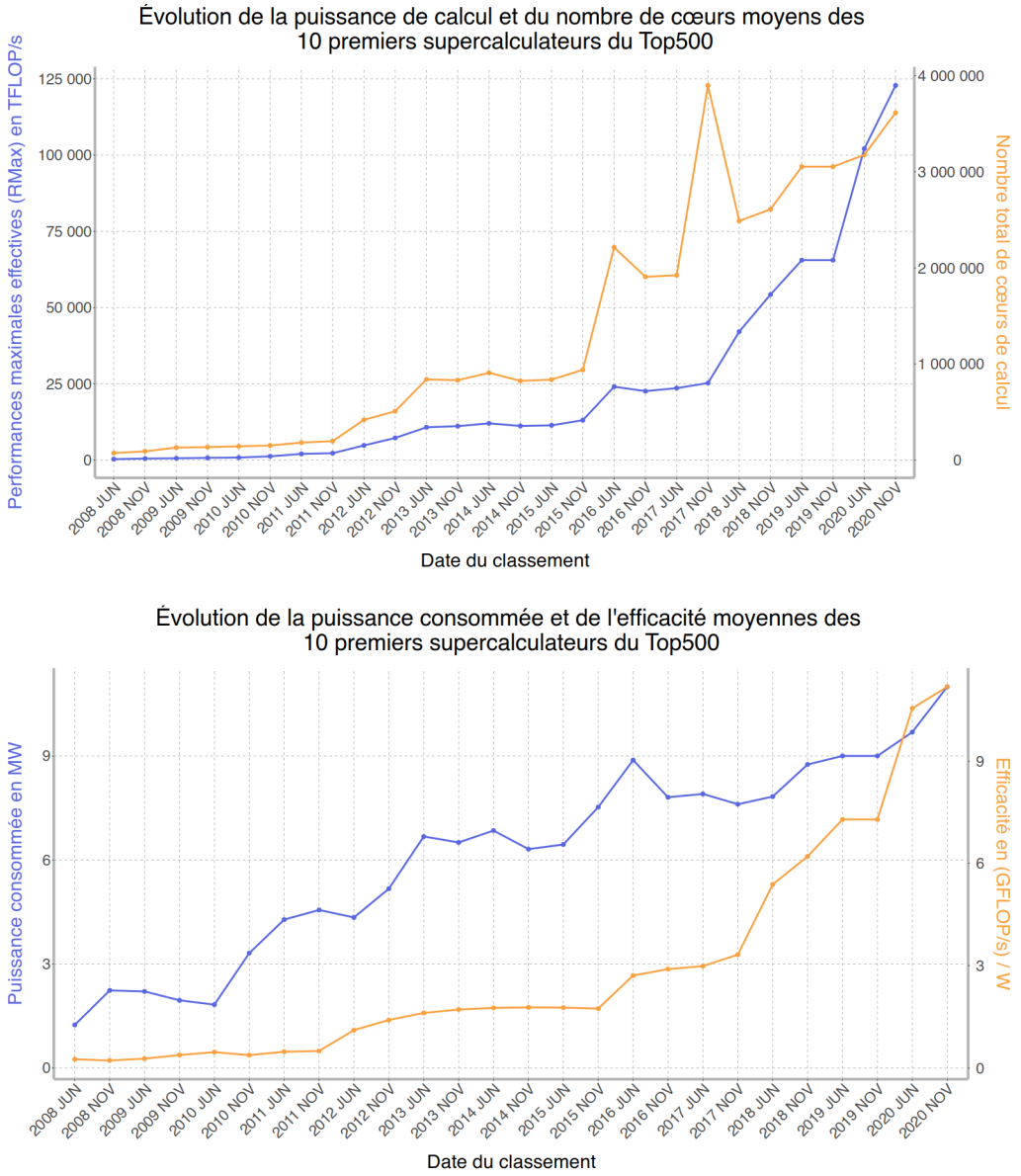


FIGURE 1.1 – Évolution, en moyenne, de la puissance de calcul, du nombre de cœurs, de la puissance consommée et de l'efficacité associée pour les 10 premiers supercalculateurs du Top500

Et cette augmentation de la consommation énergétique est devenue problématique à

l'orée de l'Exascale (atteindre les $10^{18} FLOPS$, c'est-à-dire 1 milliard de milliards d'opérations numériques sur des nombres à virgule flottante par seconde).

En effet, les premières estimations de la puissance consommée par les futurs premiers systèmes exhibant une telle puissance de calcul étaient gargantuesques : plus de $100MW$. À titre de comparaison, une ville d'une centaine de milliers de foyers, soit environ 250000 personnes, consomme une puissance similaire [2]. Aussi bien économiquement qu'écologiquement, une telle consommation énergétique s'est avérée inenvisageable. En conséquence, une borne sur la puissance que consommeront les futurs premiers systèmes exaflopiques a été définie [Shalf 2011], à savoir $20MW$.

C'est pourquoi les considérations énergétiques sont devenues de première importance dans le domaine du calcul à haute performance, au même titre, ou presque, que la puissance de calcul. La création, en 2013, du Green500 traduit cette évolution : le Top500 a un pendant « vert » qui met l'accent sur l'efficacité énergétique plutôt que sur la puissance de calcul. Il est donc logique que des efforts de recherche et de développement **à tous les niveaux** visent à améliorer l'efficacité énergétique des supercalculateurs.

Pour commencer, la conception des infrastructures d'accueil des supercalculateurs est passée au crible [Strevell 2019] dans le but de faire tendre leurs Power Usage Effectiveness (`PUE`) vers 1, et diminuer autant que possible la consommation d'énergie pour faire autre chose que des calculs.

Les technologies de refroidissement des nœuds de calcul, auxquelles une part non négligeable de la consommation énergétique de ces derniers est imputable, reçoivent une attention toute particulière. On peut notamment citer les efforts entourant les systèmes de refroidissement direct par liquide à haute température [Shoukourian 2017] équipant les supercalculateurs pétaflopiques SuperMUC-NG et JEWELS, conçus respectivement par Lenovo et Atos.

L'architecture des nœuds de calcul évolue également pour proposer une meilleure efficacité énergétique, notamment en intégrant des accélérateurs (principalement des `General Purpose Graphics Processing Unit` (`GPGPU`)). Les trois futurs premiers systèmes exaflopiques américains, Aurora, El Capitan, et Frontier, reposent d'ailleurs sur des architectures hétérogènes `CPU/GPGPU` [3] [4].

Bon nombre d'outils logiciels ont été, et/ou sont, développés dans le but d'améliorer l'efficacité énergétique des supercalculateurs. Certains s'appuient sur les spécificités des environnements de programmation typiques du `HPC`, à l'image de [Rountree 2009, Li 2010, Halimi 2014, Cesarini 2020] qui exploitent notamment les latences de synchronisation des communications collectives `Message Passing Interface` (`MPI`). D'autres s'appuient notamment sur des données de monitoring associées aux nœuds de calcul, à l'instar de [Isci 2006, Chetsa 2015, Schuchart 2017, Silvano 2019, Gholkar 2019, Corbalan 2020].

Les applications sont également parfois raffinées et réglées en vue d'être les plus efficaces possibles du point de vue de la consommation énergétique associée à leurs exécutions, à l'image des travaux réalisés dans [Lively 2014, Sourouri 2017, Vysocky 2018].

Malgré tous ces efforts, et même si l'ExaScale paraît à portée de main avec la livraison des premiers systèmes exaflopiques d'ici une à deux années, il y a encore fort à faire du

2. Estimation basée sur les données de consommation énergétique concernant 2020, mis à disposition par `https://ember-climate.org/data/`.

3. `https://www.alcf.anl.gov/aurora`

4. `https://www.amd.com/fr/products/exascale-era`

point de vue de l'efficacité énergétique des systèmes HPC. En effet, cela est notamment illustré par la Figure 1.2, qui reprend l'évolution de la puissance consommée et de l'efficacité énergétique des 10 premiers supercalculateurs du Top500, en y faisant figurer l'objectif d'efficacité énergétique pour l'ExaScale. En complément de ce graphe, on peut noter que le supercalculateur le plus efficace du top 10 du classement de Novembre 2020 est JUWELS, en septième position, avec à peine plus de 25.0 $GFLOP/s.W^{-1}$ pour 44 $PFLOP/s$, et que le supercalculateur le plus efficace du même classement figure en $170^{\text{ème}}$ position, il s'agit du NVIDIA DGX SuperPOD qui affiche une efficacité de 26.195 $GFLOP/s.W^{-1}$ pour 2.356 $PFLOP/s$. De manière encore plus pragmatique, le fait que les premiers systèmes exaflopiques ne respecteront vraisemblablement pas la borne des 20 MW mentionnée précédemment (les puissances nécessaires au fonctionnement de Frontier et El Capitan sont estimées aux alentours de 35 MW) traduit que l'efficacité énergétique demeurera un sujet central dans les années à venir dans le domaine du calcul à haute performance.

Tous ces éléments nous permettent de comprendre pourquoi il est **crucial** de s'intéresser à la consommation énergétique des supercalculateurs, notamment pour permettre à la science et l'industrie de repousser les frontières de la connaissance et du réalisable.

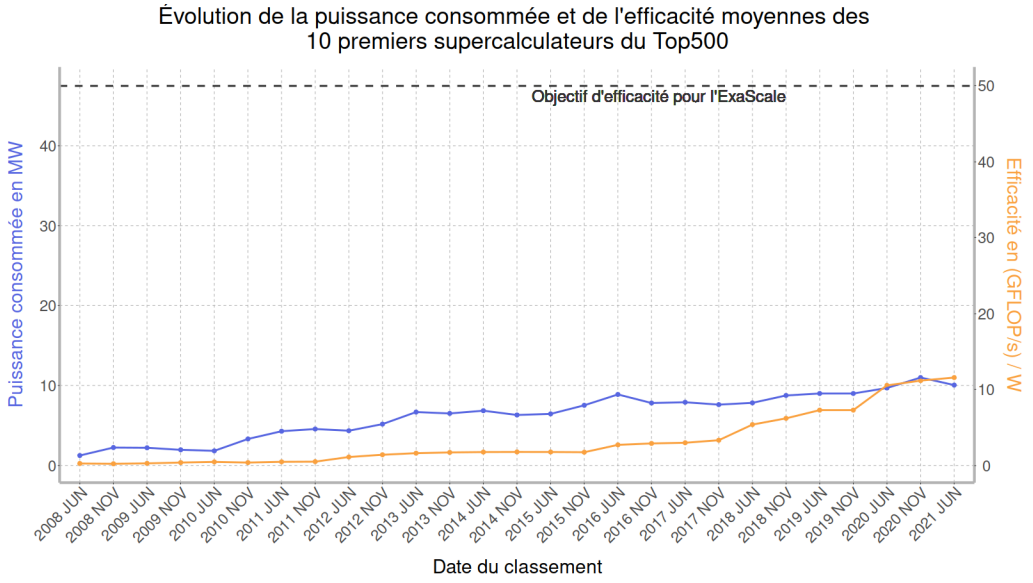


FIGURE 1.2 – Évolution, en moyenne, de la puissance consommée et de l'efficacité associée pour les 10 premiers supercalculateurs du Top500.
La ligne horizontale pointillée correspond à l'efficacité énergétique associée à la borne de 20 MW fixée pour l'ExaScale.

Les travaux menés dans le cadre cette thèse font partie de l'effort de recherche autour de l'efficacité énergétique des supercalculateurs. Plus précisément, ils sont centrés autour du développement d'un outil logiciel visant à améliorer l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance. Pour conclure cette section introductive, tâchons donc de mettre en lumière pourquoi la composante logicielle de l'effort d'amélioration de l'efficacité énergétique des supercalculateurs est primordiale.

Afin d'atteindre un haut niveau de performance, il faut que le système HPC offre une grande puissance de calcul **et** que l'application exécutée exploite au mieux cette puissance

de calcul. C'est notamment cette dualité qui engendre la notion de co-conception : pour atteindre les meilleures performances possibles, une approche pertinente est de concevoir et dimensionner le système matériel en fonction des applications qu'il exécutera, et de développer lesdites applications de sorte à ce qu'elles exploitent au mieux les caractéristiques du matériel (quitte à ce que leurs exécutions soient sous-efficaces sur d'autres machines). L'écrasante majorité des supercalculateurs occupant les premières places du Top500 avaient des applications, ou à minima des types d'applications, cibles lors de leurs conceptions (les futurs systèmes exaflopiques que sont Aurora, Frontier, et El Capitan ne font pas exception).

Une conséquence de cette notion de co-conception est qu'une application « externe » (i.e. qui n'a pas été développée pour tirer profit de l'architecture d'un supercalculeur spécifique, et n'était pas une des cibles lors de la conception du calculateur) a de très fortes probabilités (c'est quasiment une certitude) de ne pas exploiter le plein potentiel du système `HPC` sur lequel elle est exécutée. Aussi bien en termes de performances brutes que d'efficacité énergétique, les deux notions étant connexes. Observons par exemple le Top500, qui étalonne les supercalculateurs via l'exécution du benchmark « externe » `High Performance Linpack`[5] (`HPL`). On remarque alors que les performances mesurées (`RMax`) sont généralement, pour ne pas dire toujours, significativement inférieures aux performances théoriquement atteignables (`RPeak`). Comme mentionné précédemment, à cela s'ajoute le fait que les architectures hétérogènes `CPU/GPGPU` s'imposent à l'heure actuelle comme les alternatives les plus efficaces du point de vue énergétique, et sont donc de plus en plus dominantes dans le domaine du calcul à haute performance. Cela implique que les applications doivent supporter le paradigme `CPU/GPGPU`, ne serait-ce que pour être exécutées sur ces systèmes `HPC` hétérogènes, sans notion d'efficacité.

En conséquence, pour passer le cap de l'ExaScale du point de vue applicatif, deux approches complémentaires émergent : (1) faire évoluer les applications pour qu'elles profitent des ressources mises à disposition, et (2) développer un environnement logiciel de gestion des supercalculateurs permettant d'optimiser l'utilisation du matériel par les applications.

Du point de vue énergétique, le principal levier pour optimiser l'utilisation des ressources matérielles par une application est la reconfiguration des composants du supercalculateur. En effet, ces derniers exposent généralement plusieurs points de fonctionnements, chacun exhibant un compromis entre performances maximales atteignables et consommation énergétique. Si une application n'utilise pas l'intégralité de la puissance de calcul mise à disposition par le matériel, il est alors possible de reconfigurer les composants du supercalculateur vers des points de fonctionnement offrant des performances maximales atteignables inférieures mais adaptées aux besoins de l'application, induisant une réduction de la consommation énergétique associée.

Outre le levier de la reconfiguration des composants matériels, on peut citer l'adaptation du niveau de parallélisme mise en œuvre par les différentes parties de l'application, notamment lorsqu'une desdites parties ne tire pas profit du degré de parallélisme maximal mis à disposition par le matériel, ou bien encore la recompilation de l'application en adaptant certains paramètres tels que l'arité des nombres à virgule flottante ou les jeux d'instructions de vectorisation utilisés [Raïs 2018b].

5. https://www.netlib.org/benchmark/hpl

Néanmoins, la mise en place de tous ces leviers nécessite une expertise technique spécifique, que l'acteur du domaine du calcul à haute performance moyen ne possède généralement pas. Il apparaît donc clairement que le développement d'une pile logicielle dédiée à l'efficacité énergétique des supercalculateurs est fondamental, que ce soit pour fournir à l'utilisateur lambda d'un supercalculateur un actionneur pour les leviers mentionnés précédemment, ou pour accompagner un développeur lors de leurs intégrations à l'application qu'il développe.

1.2 Démarche suivie au cours des travaux de recherche

Cette section a trois buts principaux : (1) esquisser les démarches scientifique et logique suivies à l'échelle globale de la thèse ainsi que la chronologie associée, (2) présenter les lignes directrices de la méthodologie expérimentale, et (3) celles des travaux de développement. Une grande partie des éléments abordés par cette section seront repris et détaillés dans la suite du manuscrit, mais sont ici rassemblés et articulés pour proposer une vision globale et condensée de l'approche sous-jacente aux travaux de recherche.

Comme écrit dans la section 1.1, la thèse s'inscrit dans le cadre de la conception et du développement d'un outil logiciel pour améliorer l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance. L'outil en question se nomme `Bull Dynamic Power Optimizer` (`BDPO`), et le chapitre 5 décrit en détail son fonctionnement et son architecture. J'ai contribué, au sein de l'équipe R&D PowerEfficiency de la division HPC, Big Data et Sécurité (HPC BDS) d'Atos, au développement de la première version de `BDPO`, lors de mon Projet de Fin d'Études (PFE) d'ingénieur, entre Février et Août 2017.

En quelques mots, `BDPO` monitore à grain fin, i.e. à l'échelle de la dizaine de millisecondes, l'évolution temporelle de métriques caractéristiques de l'utilisation du matériel (e.g. le nombre d'instructions exécutées) au cours de l'exécution d'une application `HPC`. Lorsque lesdites métriques passent sous un certain seuil, `BDPO` considère que l'application n'utilise pas la puissance de calcul mise à disposition par le matériel. Il reconfigure alors le point de fonctionnement { tension ; fréquence } des cœurs de calcul du nœud de sorte à diminuer les performances maximales atteignables (puisque l'application ne les utilise pas complètement) et la consommation énergétique associée. À l'inverse, lorsque les métriques monitorées repassent au-dessus des seuils qui leurs sont associés, `BDPO` restaure le point de fonctionnement { tension ; fréquence } nominal. Ce faisant, la consommation énergétique associée à l'exécution de l'application diminue sans pour autant augmenter de manière significative le « temps jusqu'à la solution ». Néanmoins, cette première version de `BDPO` souffrait de deux défauts majeurs :

— la définition des seuils et des points de fonctionnement { tension ; fréquence } ciblés, dépendante de l'architecture sur laquelle l'application est exécutée, était faite par l'utilisateur, qui devait ainsi posséder une certaine expertise technique ;

— les reconfigurations étaient purement **réactives**, c'est-à-dire qu'elles intervenaient en réponse au franchissement des seuils, sans intuition ou considération pour les futurs évolutions des métriques monitorées. Or, un enchaînement de reconfigurations sur un court laps de temps peut être très pénalisant du point de vue du ratio { performances / énergie }.

Réaliser une thèse autour des thématiques liées à `BDPO` était l'occasion parfaite pour travailler sur ces deux problématiques. Cette dernière a été effectuée au sein d'Atos, dans la même équipe que celle au sein de laquelle j'ai effectué mon PFE, et l'équipe Compiler, Optimization, and Run-time SystEms (CORSE) de l'INRIA, sous la direction du Dr. Fréderic Desprez et du Dr. François Broquedis.

La première année de la thèse, qui commença en Février 2018, fut en partie dédiée à travailler, sous l'impulsion et en collaboration avec le Dr. Abdelhafid Mazouz (référent de la thèse au sein d'Atos) à la résolution de la première problématique. Cela a conduit à

la définition d'une méthodologie de calibrage associée à BDPO, exécutée à son installation sur un nœud de calcul, pour déterminer les seuils et points de fonctionnement { tension ; fréquence } les plus adaptés à l'architecture du nœud en question. Cette méthodologie est décrite en profondeur par la section 6.1 et a donné lieu à la publication [Stoffel 2018].

Par la suite, les deuxième et troisième années furent l'occasion de s'affairer à la seconde problématique. Lors des travaux de recherche et de développement associés à BDPO, j'ai eu l'occasion d'analyser bon nombre de profils de monitoring d'applications HPC. Tous exhibaient des motifs récurrents, qui se répétaient en longues séquences. Cela n'était pour autant pas surprenant, étant donné qu'une majorité des applications de calcul à haute performance sont **itératives**, c'est-à-dire qu'elles sont articulées autour d'un ensemble restreint de noyaux de calcul qui sont exécutés un grand nombre de fois pour un vaste jeu de données d'entrée. Être capable de détecter et d'isoler ces motifs récurrents est alors apparu comme un moyen de fournir à BDPO les connaissances nécessaires pour mettre en place des reconfigurations **prédictives**. En effet, avoir la connaissance d'un motif récurrent, lorsque ce dernier engendre de longues séries d'occurrences, revient à connaître (ou prédire) une large portion du profil d'une application. Deux années de travail en complète autonomie ont conduit à la conception et au développement de `Phase - Temporality Analyser` (Phase-TA), un outil de détection et de modélisation des comportements localement périodiques dans les séries temporelles, et donc dans les profils des applications HPC. La partie III aborde en détail le fonctionnement de Phase-TA, ainsi que les expérimentations associées. Ces travaux expérimentaux ont d'ailleurs été présentés dans une conférence scientifique internationale, et publiés dans les actes associés [Stoffel 2021].

L'exploitation par BDPO, du point de vue énergétique, des motifs représentatifs inférés par Phase-TA constitue le prochain item de travail auquel s'atteler. Les étapes de recherche bibliographique et de conception associées ont été traitées au cours des derniers mois de la thèse, période pendant laquelle le travail d'implémentation nécessaire à une première preuve de concept a été initié.

Maintenant que la chronologie globale de la thèse a été présentée, il convient d'esquisser les lignes directrices appliquées dans le cadre des travaux expérimentaux.

Pour commencer, toutes les expérimentations présentées dans ce manuscrit ont été menées « en conditions réelles », c'est-à-dire :

— sur des machines physiques, appartenant à des clusters ayant des configurations matérielles et logicielles typiques du domaine du calcul à haute performance ;
— avec des applications issues du HPC, pour des cas d'utilisation dimensionnés en fonction de la taille de la partition de nœuds de calcul utilisée.

De plus, une image expérimentale (i.e. un système d'exploitation spécifiquement configuré et agrémenté d'un ensemble de composants logiciels) a été construite, de sorte à permettre d'utiliser des clusters différents de manière transparente, ainsi que la reproducibilité des expérimentations. La mise en place, et le développement continu, de cette image expérimentale ont demandé des efforts conséquents mais nécessaires, par exemple pour le support du réseau d'interconnexion rapide Intel® Omni-Path, qui ne seront pas présentés par le manuscrit.

De la même manière, effectuer les expérimentations en conditions réelles a engendré une quantité de travail importante (notamment pour maîtriser les applications HPC considérées et dimensionner correctement les cas d'utilisation), dans la mesure où une

partie significative desdites expérimentations aurait pu être effectuées sur un ordinateur de travail, en utilisant des suites de benchmarks multi-threadés. Néanmoins, cela a permis d'inscrire les résultats dans un contexte « industriel », puisqu'ils sont issus d'expériences menées sur des machines typiques d'un supercalculateur, et des applications de calcul à haute performance. Et cela apporte aux résultats une forte valeur ajoutée, notamment du point de vue d'Atos mais pas seulement, puisque le domaine du calcul à haute performance est intrinsèquement ancré au monde industriel.

Enfin, intéressons-nous aux lignes directrices ayant encadré les travaux de développement. Le contrat de laboratoire liant Atos et l'INRIA dans le cadre de la thèse accorde à Atos la propriété exclusive de l'ensemble des développements effectués au cours de ces trois années de travail. Cela traduit la volonté d'Atos d'intégrer ces derniers à son catalogue d'outils logiciels propriétaires. En conséquence, `Phase-TA` n'intègre aucune bibliothèque libre et/ou ouverte : j'ai écrit l'intégralité de son code source, qui est donc propriété exclusive d'Atos.

De plus, dans l'optique de réduire au maximum le temps d' industrialisation de `Phase-TA`, ce dernier a été développé en `C`, avec une base de code richement commentée, et dont les parties les plus critiques ont été unitairement et fonctionnellement testées. À cela s'ajoute le fait que le code source de `Phase-TA` a été finement optimisé, notamment parce que cela présentait des intérêts pratiques et scientifiques, comme cela est détaillé par la partie III.

Il est clair que ces lignes directrices ont induit un travail de développement conséquent. Néanmoins, elles résultent d'un choix que j'ai sciemment fait, à savoir ancrer cette thèse CIFRE aussi équitablement que possible entre les mondes académique et industriel, entre la recherche et l'ingénierie.

1.3 Organisation du manuscrit

Ce manuscrit est organisé en trois parties, chacune étant sub-divisée en chapitres. La partie I aborde des thématiques et apporte des connaissances nécessaires à la bonne compréhension et à la mise en perspective des travaux présentés par le reste du manuscrit. Pour cela, le chapitre 2 présente et définit un ensemble d'éléments techniques afin de permettre au lecteur de : (1) saisir la structure globale d'un supercalculateur, notamment en se positionnant du point de vue de l'efficacité énergétique, et (2) se familiariser avec des notions et outils scientifiques et mathématiques sous-jacents au fonctionnement de `BDPO` et `Phase-TA`. Ensuite, le chapitre 3 synthétise les recherches bibliographiques et la veille technologique effectuées au cours de la thèse afin de décrire l'état de l'art concernant les domaines dans lesquels les travaux de recherche s'inscrivent. Le chapitre 4 clôt la première partie en présentant l'environnement expérimental, à savoir les plateformes matérielles et les applications `HPC` sur lesquelles les expériences présentées par ce manuscrit ont été réalisées.

`BDPO` est le centre de gravité de la partie II, qui s'articule selon deux chapitres : le chapitre 5 détaille le fonctionnement de `BDPO`, et le chapitre 6 présente la méthodologie expérimentale de calibrage qui lui est associée, ainsi que les expérimentations menées avec `BDPO`.

La partie III est quant à elle dédiée à `Phase-TA`, dont la méthodologie associée est décrite en grande largeur dans le chapitre 7. L'ensemble des expérimentations menées pour valider la pertinence de `Phase-TA`, et étudier les comportements localement périodiques des applications de calcul à haute performance sont décrites par le chapitre 8.

Pour finir, le chapitre 9 retrace les points clés et rassemble les conclusions découlant des travaux menés au cours de la thèse, puis esquisse les perspectives de poursuite desdits travaux de recherche.

Première partie

Travaux préparatoires

Éléments techniques

2

Ce chapitre a vocation à introduire et présenter succintement un ensemble de notions et de concepts nécessaires à l'appréhension du domaine du calcul à haute performance, et à la compréhension des travaux menés dans le cadre de la thèse.
Pour ce faire, la section 2.1 donne les principales clés permettant d'appréhender l'architecture d'un supercalculateur. La section 2.2 pourra alors introduire le concept d'intensité opérationnelle et la modélisation `Roofline`, qui permettent de quantifier le niveau de performance maximal atteignable par un noyau de calcul. Quant à la section 2.3, elle définit un ensemble d'outils mathématiques, autour du concept de distance, qui sont utilisés par `BDPO` et `Phase-TA`.

2.1 Architecture matérielle des supercalculateurs

Cette section a pour but de faire un tour d'horizon des principaux composants d'un supercalculateur, de sorte à permettre au lecteur d'appréhender l'architecture typique et le fonctionnement d'un système de calcul à haute performance, notamment du point de vue énergétique.

Un supercalculateur est une machine complexe, possédant plusieurs centaines de milliers voire millions de composants. Un tel système de calcul a généralement une taille imposante, et nécessite des infrastructures spécifiques de grandes envergures : alimentation électrique très haute tension, systèmes de refroidissement, systèmes d'asservissement

des conditions de pression, température, et hygrométrie de l'enceinte dans laquelle il se trouve, etc. Les centres de calcul s'apparentent donc à des entrepôts s'articulant autour de ces monstres de puissance de calcul que sont les supercalculateurs, comme la figure 2.1 le montre. On peut y voir `JUWELS`, l'actuel $8^{ème}$ supercalculateur le plus puissant du monde.

FIGURE 2.1 – Photo du supercalculateur `JUWELS`, actuellement en $8^{ème}$ position du Top500.
(Source : https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUWELS/JUWELS_node.html)

La figure 2.1 laisse aussi apparaître une autre caractéristique des supercalculateurs : leurs architectures matérielles sont modulaires. On voit en effet que `JUWELS` est constitué de plusieurs **armoires**, également appelées « racks ». Les détails d'une armoire `Bull Sequana` sont présentés par la figure 2.2, qui présente également une lame de calcul `Bull Sequana X1120`.

FIGURE 2.2 – À gauche, une armoire ou « rack », une brique de base pour un supercalculateur. Elle accueille notamment les lames ou « blades », qui accueillent les nœuds de calcul.
La lame présentée à droite est une `Bull Sequana X1120`, qui compte trois nœuds de calcul possédant chacun deux processeurs Intel® `Xeon Scalable Platform` d'architecture `Skylake`.

Le rôle d'une armoire est d'accueillir un ensemble de **lames de calcul** (également appelées « blades »), et d'encapsuler le matériel nécessaire à leur fonctionnement : alimentations électriques, système de refroidissement, etc.

Les lames de calcul se glissent dans les interstices d'une armoire, à la façon d'un tiroir. Elles accueillent un ou plusieurs nœuds de calcul, et mutualisent certaines ressources partagées telles que les systèmes de refroidissement et les unités d'alimentation électrique.

Les **nœuds de calcul** sont les machines qui effectuent les calculs à proprement parler. Un nœud est notamment constitué d'une ou plusieurs unités de calcul, telles que les processeurs (cf. sous-section 2.1.1) et les accélérateurs (cf. sous-section 2.1.6), de mémoire principale (cf. sous-section 2.1.2), d'interfaces vers les réseaux de management et d'interconnexion rapide, et optionnellement de stockage local. À titre d'exemple, la lame de calcul présentée par la figure 2.2 accueille trois nœuds [1] de calcul possédant chacun deux processeurs Intel® `Xeon Scalable Platform` d'architecture `Skylake`.

Les armoires hébergent également les commutateurs (« switches » pour les anglophones) des réseaux de management (cf. sous-section 2.1.4) et d'interconnexion rapide (cf. sous-section 2.1.3).

Pour finir cette succinte présentation des éléments essentiels au fonctionnement d'un supercalculateur, précisons que le stockage permanent est généralement centralisé et hébergé sur un ensemble de nœuds de calcul dédiés et connectés au réseau d'interconnexion rapide. La sous-section 2.1.5 offre plus de détails à ce sujet.

2.1.1 Processeurs

Cette sous-section s'articule autour du processeur, élement central d'un nœud de calcul. Après avoir présenté l'architecture typique d'un processeur moderne dans la sous-section 2.1.1.1, la sous-section 2.1.1.2 se concentrera sur les fonctionnalités associées à la gestion de l'énergie des processeurs. Enfin, la sous-section 2.1.1.3 abordera un type spécifique de processeur : les `manycores`.

2.1.1.1 Éléments d'architecture d'un processeur moderne

Un processeur est un composant complexe, contenant de nombreux sous-composants et proposant un très large éventail de fonctionnalités. La figure 2.3 donne idée de la complexité d'un processeur en proposant une vue de l'implantation matérielle d'un processeur Intel® `Xeon Ice Lake` à gauche, et sa représentation schématique à droite.

Parmi les différents composants d'un processeur, on trouve ainsi des contrôleurs mémoires qui permettent d'interfacer ce dernier avec la mémoire principale d'un nœud de calcul (cf. sous-section 2.1.2), des interfaces `PCIe` qui relient le processeur aux différents composants connectés au bus `PCIe` (e.g. les accélérateurs, présentés par la sous-section 2.1.6), ou bien encore des interfaces Intel® `UltraPath` qui permettent de relier les différents processeurs d'un nœud de calcul multi-processeurs. Parmi la liste des composants d'un processeur figure également le « quartz », un système oscillatoire qui permet de générer un signal de fréquence constante qui cadence le processeur. On parle alors de **fréquence nominale** du processeur, et la période associée à cette fréquence devient l'unité de mesure

1. La sous-section 4.1.1 détaille les caractéristiques techniques des nœuds de calcul en question.

du temps à l'échelle du processeur : le **cycle** processeur. Nous avons gardé les composants centraux des processeurs pour la fin : les **cœurs de calcul**. Ces derniers sont les éléments qui exécutent les instructions composant une application, et implémentent une vaste majorité des fonctionnalités proposées par un processeur. En « conditions nominales de fonctionnement », un cœur de calcul peut compléter plusieurs (généralement entre 1 et 4, et maximum 6 pour la vaste majorité des processeurs récents) instructions par cycle processeur. Le fonctionnement d'un cœur de calcul est complexe, et le reste de cette sous-section cherche à en décrire brièvement les points clés.

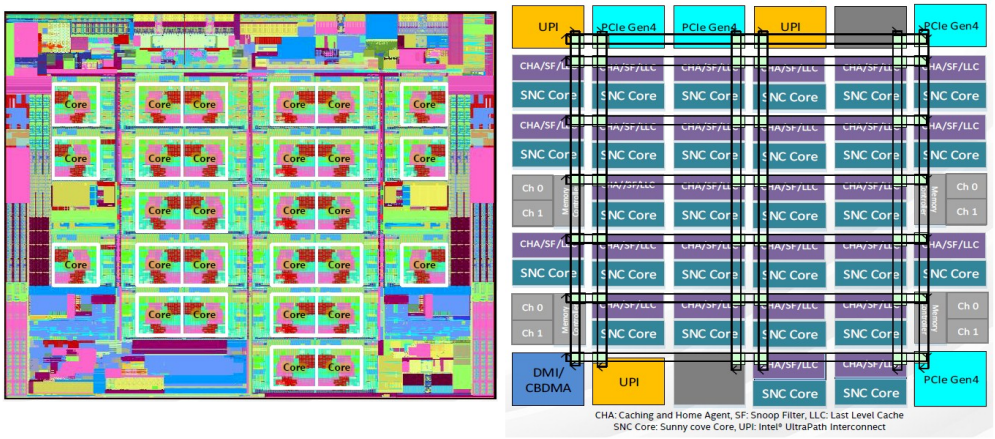


FIGURE 2.3 – Implantation matérielle (à gauche), et sa représentation schématique (à droite), d'un processeur Intel® `Xeon Ice Lake`.
(Source : https://www.nextplatform.com)

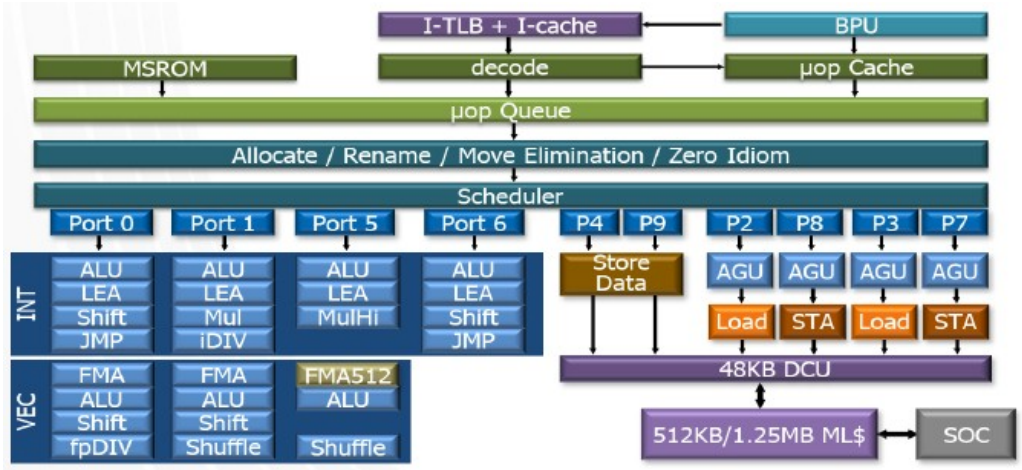


FIGURE 2.4 – Schéma illustrant la micro-architecture et le fonctionnement d'un cœur de calcul d'un processeur Intel® `Xeon Ice Lake`.
(Source : https://www.nextplatform.com)

Une des caractéristiques fondamentales des architecture de cœur de calcul est le jeu d'instructions (« instruction set » pour les anglophones) qu'il peut exécuter. À noter que

les jeux d'instructions sont des « entités » indépendantes des cœurs de calcul, au sens où des architectures et micro-architectures différentes peuvent les implanter. Un jeu d'instruction définit l'ensemble des opérations qu'un cœur de calcul l'implantant doit être en mesurer d'effectuer. Lesdites opérations ont des nombres d'opérandes variables, qui sont associés à des **registres** : des entités mémorisantes stockant les données sur lesquelles les opérations sont effectuées. On peut, en première approximation, considérer ce jeu d'instructions comme un cahier des charges guidant la conception du cœur. Deux paradigmes principaux co-existent à l'heure actuelle : les jeux d'instructions complexes, ou `CISC`, et les jeux d'instructions réduits, ou `RISC`. Les architectures de cœurs de calcul `CISC` sont généralement plus lourdes et plus coûteuses à produire, et consomment plus d'énergie que les architectures `RISC`. En contrepartie, les architectures `CISC` proposent des instructions, et donc des circuits matériels, qui permettent d'effectuer des opérations spécialisées, là où une architecture `RISC` nécessitera l'exécution d'une séquence d'instructions basiques pour mener à bien l'exécution d'une telle opération spécialisée, induisant un temps d'exécution généralement bien supérieur.

Continuons en nous intéressant aux détails architecturaux et micro-architecturaux de l'implantation d'un cœur de calcul. Afin de ne pas rendre le propos trop complexe, notre approche sera spécifique et non généraliste. En effet, nous allons nous concentrer sur un cœur de calcul en particulier pour introduire toutes les notions fondamentales à la compréhension de la suite du manuscrit. En l'occurrence, le cœur de calcul étudié sera celui d'un processeur Intel® `Xeon Ice Lake`. Les travaux de recherche menés dans le cadre de cette thèse ayant exclusivement eu pour support des processeurs Intel®, il m'a semblé fort à propos de sélectionner la dernière mouture estampillée `HPC` de la firme de Mountain View. Avant de commencer, précisons que bien que les cœurs de calcul des processeurs Intel® soient historiquement considérés d'architecture `CISC`, les cœurs des processeurs Intel® récents sont parfois considérés d'architecture « pseudo-`RISC` ». En effet, comme cela est détaillé par la suite, un cœur de calcul Intel® moderne est constitué d'une partie traduisant les instructions complexes du jeu d'instruction en des « sous-instructions » basiques (partie souvent appelée « frontend »), et d'une partie exécutant lesdites sous-instructions basiques. La partie en question est généralement appelée « backend », et s'apparente à un cœur de calcul « basique », que l'on peut facilement être tenté de qualifier de cœur `RISC`. D'où l'appellation « pseudo-`RISC` ».

Première étape de notre plongée dans les arcanes d'un cœur de calcul : la notion de **cache**. Il s'agit de composants mémorisants permettant l'exploitation des principes de localités (1) temporelle et (2) spatiale : (1) une donnée qui vient d'être accédée a de grandes chances d'être accédée à nouveau dans un futur proche, et (2) les données qui lui sont contigües ont de fortes chances d'être accédées dans un futur proche.

Or, les registres du cœur de calcul ne peuvent contenir qu'un ensemble très restreint d'octets, incommensurablement inférieur à la taille des jeux de données des applications `HPC`, qui résident en mémoire principale (se référer à la sous-section 2.1.2 pour plus de détails). Et la latence pour transférer une donnée depuis la mémoire principale vers un registre est très élevée, et se compte en centaines de cycles processeurs. Les caches sont accédés plus rapidement que la mémoire principale, mais sont plus coûteux à manufacturer, et donc disponibles en moins grandes quantités.

Lorsqu'une donnée est transférée de la mémoire principale vers les registres d'un cœur

de calcul, elle est mémorisée au sein de la hiérarchie de caches, pour tirer profit de la localité temporelle et être rapidement ré-accédée. De plus, la donnée n'est pas transférée « seule » : un groupe d'octets contigüs (64 octets sur la vaste majorité des processeurs récents [2]) auquel appartient la donnée ciblée est transférée, pour tirer profit de la localité spatiale en rendant rapidement accessibles les données « proches » de cette dernière. Un tel groupe d'octets contigüs est appelé une **ligne de cache**, et définit la granularité des caches et de la mémoire principale. Lorsque la donnée à transférer vers les registres du cœur de calcul est mémorisée en cache, on parle de **référence en cache**. Lorsque ce n'est pas le cas, on parle de **défaut de cache**, et la donnée en question doit être transférée depuis la mémoire principale, en étant mémorisée par la hiérarchie de caches au passage. Précisons également que lorsqu'une donnée mémorisée en cache est modifiée par une instruction exécutée par le cœur de calcul, il faut répercuter cette modification dans la mémoire principale, et dans la hiérarchie de cache au passage. On parle alors d'**écriture en retour** (ou de « write back » pour les anglophones).

Les processeurs actuels destinés au domaine du calcul à haute performance possèdent une hiérarchie de caches à trois niveaux, dans l'ordre de proximité aux registres des cœurs de calcul : le cache `L1`, le cache `L2`, et le cache `L3`. À noter que le cache `L1` est parfois appelé le cache `L1d`, en référence au fait qu'il est dédié aux données qui sont transférées depuis la mémoire principale. Une telle distinction est parfois nécessaire car les cœurs de calcul possèdent également un cache `L1i` dédié aux instructions constituant le programme à exécuter. Chaque cœur de calcul possède des caches `L1`, `L2` et `L3`, les deux premiers lui sont spécifiques (i.e. il est le seul cœur de calcul à pouvoir y accéder), alors que les caches `L3`, bien que rattachés à un cœur en particulier, sont partagés par l'ensemble des cœurs de calcul.

Il est alors possible que des cœurs différents possèdent des copies locales de la même donnée dans leurs caches privatifs, ou encore qu'un cœur de calcul doive appliquer une instruction à une donnée qui se trouve dans une portion du cache `L3` rattachée à un autre cœur de calcul. On comprend qu'il est nécessaire d'avoir un agent et un protocole dédié à la gestion des caches à l'échelle du processeur, et que les cœurs de calcul doivent être interconnectés. Ces deux fonctionnalités sont assurées respectivement par une circuiterie dédiée adjointe à chaque cache, et le **réseau maillé d'interconnexion** (« mesh interconnect » pour les anglophones) des cœurs de calcul, tous deux visibles sur la figure 2.3. L'agent de cache est également responsable de l'éviction des lignes de cache. En effet, étant donné que la taille des caches est faible devant celle de la mémoire principale, seule une infime partie de cette dernière peut y être mémorisée. Lorsque les caches sont remplis, toute nouvelle ligne de cache transférée depuis la mémoire principale doit être mémorisée à la place d'une ligne de cache mémorisée par le cache. Pour conclure ce paragraphe dédié aux caches, abordons la notion de `prefetcher`. Un `prefetcher` est un agent impliqué dans la gestion des caches `L1` et `L2`. Pour chaque ligne de cache transférée vers un des deux caches qui viennent d'être mentionnés, il transfère une ou plusieurs des lignes de cache qui lui sont contigües. Le but est d'exploiter au maximum la localité spatiale en pré-transférant les données proches de celles accédées par le cœur de calcul, ce qui permet de s'abstraire de la latence inhérente au transfert des lignes de cache depuis les niveaux

2. Une exception notable est l'`IBM Power 9`, pour lequel une ligne de cache fait 128 octets.

de cache supérieurs.

Maintenant que la hiérarchie de caches a été présentée, abordons les composants d'un cœur de calcul impliqués dans l'exécution d'une instruction. Pour commencer, les instructions à exécuter sont transférées depuis la mémoire principale vers le cache `L1i`. Ces instructions sont ensuite décodées pour établir la séquence de `micro-opérations` à exécuter, c'est-à-dire les opérations « unitaires / élémentaires » que le cœur de calcul peut exécuter, et injectées dans la file de `micro-opérations` à exécuter. L'opération de décodage de l'instruction pouvant être « longue » à l'échelle d'un cycle processeur, une mémoire tampon mémorise les « traductions » instructions → `micro-opérations` les plus récentes, dans un souci d'amélioration du temps de décodage. À noter également que certaines instructions induisent des longues séquences de `micro-opérations`, telles que `reps`, notamment utilisée pour implanter la fonction `memcpy`, qui peut induire plusieurs itérations d'une séquence de `micro-opérations`. L'injection des `micro-opérations` à injecter dans la file est alors déléguée au séquenceur de `micro-opérations`, couramment appelé `MSROM`.

Toujours dans une problématique d'optimisation des performances délivrées par le processeur, le file de `micro-opérations` peut être modifiée. Des `micro-opérations` peuvent être « fusionnées » (dans les faits, le terme le plus approprié est « regroupées ») pour occuper moins de place dans la file de `micro-opérations`. Mais la modification la plus commune est probablement le réordonnancement des `micro-opérations`, qui peuvent être effectuées « dans le désordre ». En effet, les différents types d'unités de calcul contenues dans un cœur sont en nombre limité, et exécuter les `micro-opérations` dans l'ordre d'arrivée des instructions n'est pas, dans la quasi-totalité des cas, l'ordonnancement permettant de maximiser le débit de `micro-opérations` exécutées[3]. Le rôle de l'ordonnanceur est précisément d'améliorer le débit de `micro-opérations` exécutées en sélectionnant l'ordre dans lequel elles seront envoyées vers les différents ports du cœur de calcul pour être exécutées. Un composant nommé **tampon de réorganisation** (« Re-Order Buffer », ou `ROB`, pour les anglophones) s'assure que : (1) toutes les `micro-opérations` composant une instruction aient été exécutées avant de déclarer cette dernière comme **retirée**, et que (2) les instructions soient retirées dans selon leur ordre d'apparition dans le programme exécuté, tout en respectant les dépendances de données que cela implique.

Mentionnés précédemment, intéressons-nous maintenant aux unités de calcul et aux ports d'un cœur de calcul. Chaque unité de calcul implante une famille d'opérations. Par exemple, les Unités Arithmétiques et Logiques (`ALU`) implantent les opérations arithmétiques et logiques basiques telles que l'addition et le « ou exclusif ». On peut également citer les unités de calcul dédiées à la manipulation des données (e.g. transferts depuis et vers la mémoire principale) qui sont assignées aux ports 2, 3, 4, 7, 8, et 9, et directement interfacées avec la hiérarchie de caches. La circuiterie implémentant l'ordonnanceur est complexe, et l'est d'autant plus que l'ordonnanceur possède un nombre de ports élevé auxquels il peut assigner les `micro-opérations`. En conséquence, plusieurs unités de calcul peuvent partager le même port. Parmi les unités de calcul qui partagent un port, une seule peut entamer l'exécution d'une `micro-opération` par cycle processeur.

Les unités vectorielles sont des unités de calcul à part. En effet, ces dernières per-

3. Par exemple, en raison de contention sur une unité de calcul car certains opérandes de la `micro-opération` exécutée doivent être transférés depuis la mémoire principale.

mettent de réaliser des opérations sur plusieurs données d'un même type en parallèle, par exemple 8 additions sur des nombres flottants peuvent être réalisées en une instruction, en un nombre de cycles processeurs grandement inférieur à celui requis pour exécuter 8 additions individuellement. Pour ce faire, des registres particuliers ont été implantés aux processeurs. Ces derniers font la taille de plusieurs données d'un même type, par exemple 8 nombres flottants pour reprendre l'exemple précédent. En opérant une « addition vectorielle » entre deux tels registres, les 8 nombres flottants contenus dans un des registres sont additionnés individuellement à leurs pendants dans l'autre registre (e.g. le quatrième nombre flottant du premier registre avec le quatrième nombre flottant du second registre). Plusieurs formats d'instructions et registres vectoriels se sont succédés au gré des innovations technologiques. À l'heure actuelle, les processeurs Intel® récents implantent le format `AVX512`, qui proposent des registres d'une arité de 512 bits, soit la possibilité de faire des opérations vectorielles sur 8 nombres flottants en précision double. Brève digression pour terminer ce paragraphe : soulignons le fait que le code source d'une application doit respecter plusieurs contraintes pour qu'un compilateur puisse détecter les opportunités d'utilisation d'instructions vectorielles. Dans de nombreux cas, c'est le développeur d'une application `HPC` qui incorpore explicitement l'utilisation d'instructions vectorielles, soit en intégrant des extraits de code en assembleur au code source de l'application en question, soit en utilisant les intrinsèques dédiés mis à disposition par le compilateur.

L'ensemble des étages traversés par une instruction pour être exécutée et retirée constituent le `pipeline` du cœur de calcul. Le fait de séparer les étapes de l'exécution d'une instruction selon plusieurs étages indépendants permet à la même instruction (potentiellement sur des opérandes différents) d'apparaître à différents étages du `pipeline`. Par exemple, une instruction d'addition sur des entiers peut entrer dans l'étage du décodage alors qu'une autre instruction d'addition est à l'étage des unités de calcul, en train d'être exécutée. C'est une des caractéristiques majeures des architectures « **pipelinées** » de cœurs de calcul.

Point de vocabulaire supplémentaire : un cœur de calcul pouvant retirer deux instances d'une même instruction durant un même cycle processeur est dit **superscalaire**. C'est par exemple le cas du processeur Intel® `Xeon Ice Lake`, qui possède plusieurs unités de calcul `ALU` assignées à des ports différents, et peut donc, par exemple, retirer deux instructions d'addition sur des entiers au même cycle processeur.

Les processeurs modernes implantent généralement des technologies de multi-threading simultané, à l'image de l'`HyperThreading` pour les processeurs Intel®. Cela consiste à implanter deux copies de tous les registres d'opérande et de contrôle d'un cœur de calcul, ainsi qu'un agent d'échange de contextes adapté. Deux processus peuvent alors « partager » un cœur de calcul : chacun utilise un jeu de registres, et accède à tour de rôle au `pipeline`. L'agent d'échange de contextes a pour but de minimiser l'impact sur les performances du changement de processus « actif », c'est-à-dire ayant accès au `pipeline`. Lorsque le processus actif est « bloqué »[4], un changement de contexte matériel permet de donner rapidement à l'autre processus

4. Par exemple, lorsqu'il n'y a plus de `micro-opérations` à répartir entre les ports du cœur de calcul et que celles en cours d'exécution sont en attente de données en cours de transfert depuis la mémoire principale.

l'accès au `pipeline`, afin de recouvrir le temps d'exécution non-productif du premier processus par un temps d'exécution productif pour le second. On dit alors que le cœur de calcul **physique** possède deux `hyper-threads`, et qu'il correspond à deux cœurs de calcul **logiques**.

Abordons maintenant un autre composant des processeurs modernes visant à améliorer le débit d'instructions retirables, à savoir l'Unité de Prédiction de Branchement (`BPU`). Lorsqu'une instruction conditionnelle de branchement doit être exécutée par un cœur de calcul, le flux d'instructions à exécuter par la suite dépend du résultat de l'évaluation de la condition en question. Néanmoins, si le décodage d'instructions est mis en attente jusqu'à ce que l'instruction conditionnelle de branchement soit retirée, un nombre significatifs de cycles processeurs peuvent être gâchés. La solution apportée à cette problématique consiste à exécuter une des deux branches possibles en attendant que l'instruction conditionnelle de branchement soit retirée. Si la branche exécutée s'avère être la bonne, le flux d'exécution d'instructions n'a pas été interrompu, et les performances du cœur de calcul n'ont pas été impactées. Si, au contraire, la branche exécutée n'est pas la bonne, les instructions exécutées pendant l'évaluation de la condition de branchement sont annulées. Le rôle de la `BPU` est d'améliorer, d'un point de vue statistique, la sélection de la bonne branche [5]. À titre d'information, c'est cette fonctionnalité d'exécution spéculative des branches qui a été le médium de l'attaque par canal auxiliaire `Spectre` [6], qui a fait grand bruit en 2019.

Pour finir, intéressons-nous aux `Model Specific Register` (`MSR`), et aux compteurs de performance, des composants que nous mentionnerons par la suite. Un `MSR` est un registre permettant d'interagir avec un processeur : ces derniers permettent soit d'obtenir une information et/ou une donnée concernant l'état du processeur, soit de modifier sa configuration. Chaque `MSR` a une adresse et un mode d'emploi (e.g. quel bit véhicule quelle information), qui peuvent varier avec le modèle du processeur, et qui sont décrits dans le manuel d'utilisation du processeur en question. Sur les processeurs Intel®, un `MSR` est accessible en lecture et en écriture via les instructions `rdmsr` et `wrmsr`, respectivement. À condition d'avoir les privilèges d'administration nécessaires, bien entendu.

Quant aux compteurs de performance, leur rôle est de permettre de compter les occurrences de certains évènements associés au fonctionnement des processeurs. Par exemple, un compteur de performance peut être incrémenté à chaque occurrence d'un défaut de cache `L2`. Certains évènements sont couramment utilisés, suffisamment pour se voir adjoindre un `MSR` spécifique, qui expose le compte de leurs occurrences. C'est par exemple le cas du nombre d'instructions retirées. Néanmoins, du fait de la complexité des processeurs modernes, il y a une très grande quantité de compteurs de performance. Trop pour que chacun puisse avoir un `MSR` dédié. Pour répondre à cette problématique, chaque cœur de calcul est doté d'une `Performance Monitoring Unit` (`PMU`). Il s'agit d'un ensemble de compteurs de performance génériques (8 pour les processeurs récents), appelés `Performance Monitoring Counter` (`PMC`), agrémenté d'une circuiterie de configuration. Chaque `PMC` peut compter les occurrences de n'importe quel évènement, en fonction de la configuration de la circuiterie qui l'accompagne (configuration spécifiée via un `MSR`).

5. En l'absence de `BPU`, pour l'exécution d'un grand nombre d'instructions conditionnelles de branchement, la bonne branche est sélectionnée la moitié du temps, soit 50% du temps.

6. Plus de détails dans [Kocher 2019].

Ainsi, si on exclut les compteurs de performance ayant un MSR dédié, on peut monitorer, pour chaque cœur de calcul, autant d'évènements différents que la PMU compte de PMC.

2.1.1.2 Gestion de l'énergie

Tout processeur moderne dispose de nombreuses fonctionnalités permettant d'agir sur son comportement d'un point de vue énergétique, fonctionnalités qui induisent généralement une modification du compromis entre débit d'instructions exécutables et puissance électrique consommée exhibé par le processeur en question. La plupart des fonctionnalités qui viennent d'être mentionnées sont standardisées par le modèle Operating System-directed configuration and Power Management (OSPM), décrit pas la figure 2.5. Le modèle OSPM définit un ensemble d'états associés aux différentes configurations que peut prendre un processeur, déclinés selon les fonctionnalités associées.

Pour commencer, mentionnons les G-States et les S-States qui définissent respectivement les états d'alimentation et les niveaux de veille qu'un système (i.e. pas uniquement le processeur, le nœud complet) peut prendre. On peut faire trois observations quant à la lecture du modèle OSPM :

— Il peut exister des dépendances entre les différentes fonctionnalités et états. Par exemple, les S-States n'agissent que lorsque le G-State associé à l'état de veille du processeur est actif ;

— Aux différents états sont assignés des identifiants dont la numérotation commence à 0, avec l'état « nominal », généralement le plus « performant » prenant la plus petite valeur d'identifiant, cette dernière s'incrémentant au fur et à mesure que les états « s'éloignent » de ladite configuration nominale. Par exemple, la veille la plus légère est associée au S-State S0, lorsque la veille la plus profonde est associée au S-State S1 ;

— Un point de vocabulaire pour finir : G0 est dit plus élevé que G1, et, à contrario, G1 est dit plus profond que G0.

Afin de réduire la consommation énergétique moyenne des processeurs, ces derniers possèdent également des niveaux de veille qui consistent à couper l'alimentation de certains composants du processeur pour qu'ils ne consomment (presque) plus d'énergie.

Les cœurs de calcul et le Package, c'est-à-dire tous les composants du processeur qui ne sont pas des cœurs de calcul, sont gérés séparément, ce qui implique des états différents, respectivement les C-States et les Package C-States. Les C-State C0 et Package C-State Package C0 correspondent à la pleine alimentation des ressources matérielles du processeur, et sont en vigueur dès lors que ce dernier exécute des instructions. Les C-States et Package C-States plus profonds sont utilisés, sur ordre du système d'exploitation, lorsque le processeur n'a aucune charge de travail à exécuter, et qu'il est alors possible de désactiver certains de ses composants. Plus les C-States et Package C-States sont profonds, plus il y a de composants désactivés. Ce qui implique une plus faible consommation d'énergie, mais également une latence de retour à l'état le plus élevé et donc à une période de productivité du processeur plus longue. Précisons que les Package C-States s'appliquent à l'échelle d'un processeur, lorsque les C-States s'appliquent à l'échelle d'un cœur de calcul. À noter également que l'utilisateur ne peut pas manuellement déclencher un changement de C-State ou de Package C-State. Il peut

néanmoins, via les paramètres d'initialisation du noyau du système d'exploitation, préciser quels sont les `C-State` et `Package C-State` les plus profonds autorisés. Pour finir concernant les `C-States` et `Package C-States`, précisons que dans le domaine du calcul à haute performance, les `C-State` et `Package C-State` les plus profonds autorisés sont généralement `C1` et `Package C1`, de sorte à minimiser la latence de retour à une période de productivité (lorsque la mise en veille de certains composants du processeur n'est tout simplement pas désactivée).

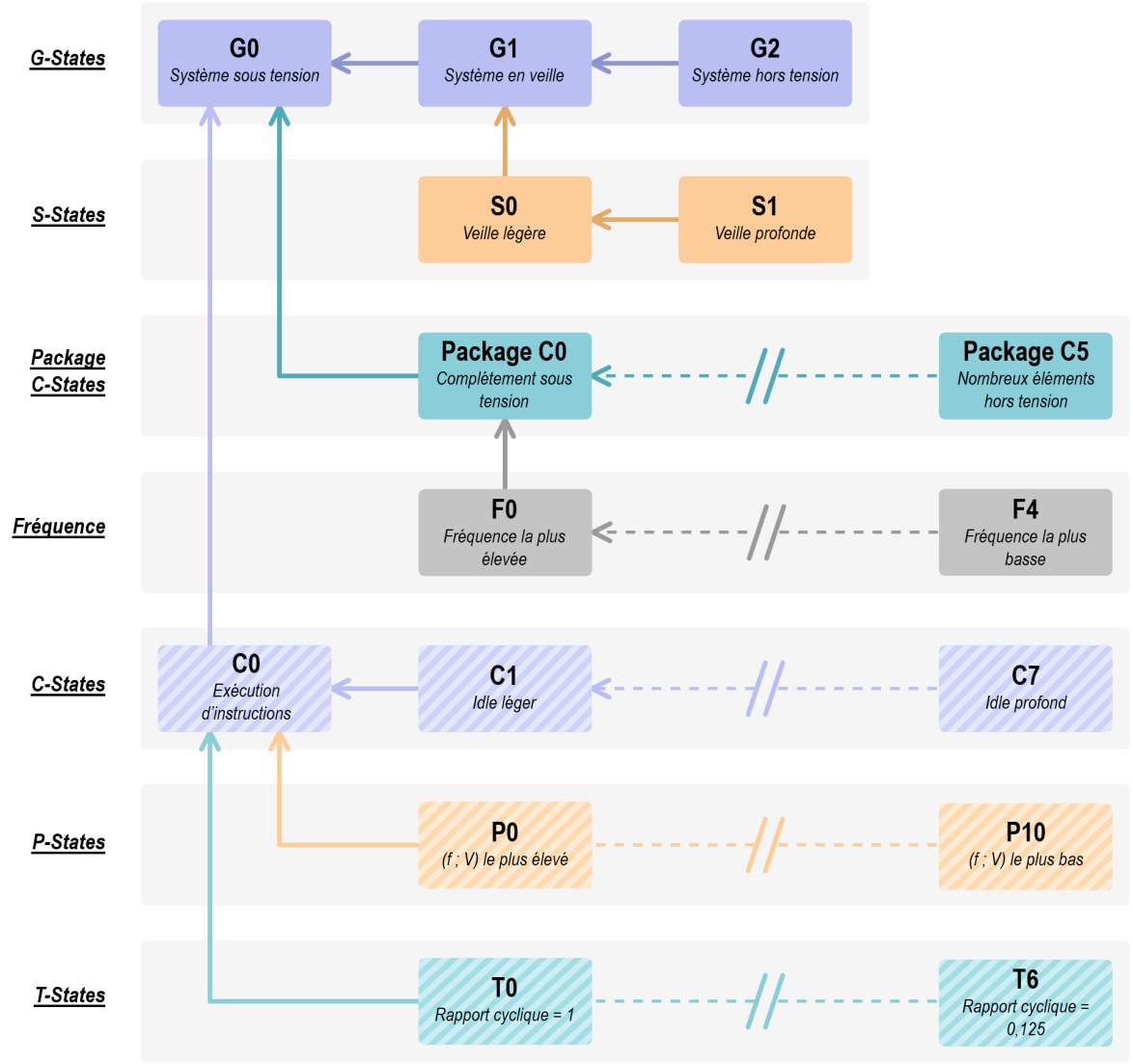


FIGURE 2.5 – Modèle `OSPM` encadrant les technologies de gestion énergétique des processeurs.

Lorsque le processeur est en train d'exécuter des instructions, c'est-à-dire lorsque les `C-State C0` et `Package C-State Package C0` sont en vigueur, il est possible de modifier le point de fonctionnement du `Package`, à l'échelle du processeur complet, et les points de fonctionnement des cœurs de calcul, appelés `P-States`, à la granularité du cœur de calcul. Ces points de fonctionnement sont modifiables dynamiquement par l'utilisateur,

qui peut également laisser au système d'exploitation le soin de les gérer. Pour le `Package`, le point de fonctionnement correspond à sa fréquence de fonctionnement : le point de fonctionnement le plus élevé est associé à la fréquence nominale, et le point de fonctionnement le plus profond est associé à la fréquence la plus basse. Pour les cœurs de calcul, le point de fonctionnement correspond à un couple { tension ; fréquence }. Similairement, le `P-State` le plus élevé est associé à la fréquence nominale, lorsque le `P-State` le plus profond est associé à la fréquence la plus basse. Globalement, un point de fonctionnement plus profond engendrera une consommation de puissance électrique plus basse, mais diminuera également le niveau maximale de performance atteignable, puisqu'il y aura moins de cycles processeurs par unité de temps.

Les cœurs de calcul exposent un autre levier permettant leur gestion du point de vue énergétique, la modification de leur rapport cyclique. Le modèle `OSPM` associe la notion de `T-State` à ce levier, avec le `T-State T0` correspondant à un rapport cyclique de 1 et le `T-State T6` correspondant généralement à un rapport cyclique de 0.125. Afin de mieux comprendre la distinction entre `P-States` et `T-States`, la figure 2.6 l'illustre par un exemple pratique.

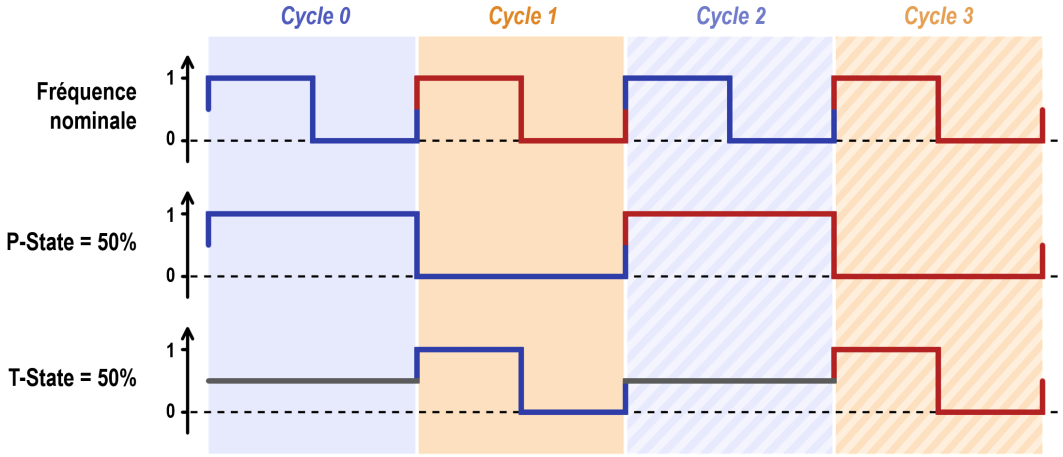


FIGURE 2.6 – Schéma illustrant la différence entre `T-States` et `P-States`.

Le premier signal correspond au signal d'horloge effectif d'un cœur de calcul lorsqu'il fonctionne à sa fréquence nominale, qui est égale à la fréquence de référence du processeur. Le schéma fait d'ailleurs apparaître 4 cycles processeurs, indicés de 0 à 3. Le second signal correspond alors au signal d'horloge effectif du cœur de calcul considéré lorsque le `P-State` en vigueur est associé à une fréquence égale à la moitié de la fréquence nominale. On voit que le cœur de calcul est constamment en période de productivité, mais que ses cycles sont deux fois plus longs que les cycles processeurs de référence. Au contraire, lorsqu'on applique plutôt le `T-State` correspondant à un rapport cyclique de 50% comme c'est le cas pour le troisième signal, le cœur de calcul fonctionne toujours à la fréquence nominale, mais une proportion égale à 50% des cycles processeurs de référence lui sont masqués. Pendant ces cycles processeurs masqués, le cœur de calcul est « inactif », au sens où l'état de ses transistors ne peut pas changer. Ainsi, bien que le processeur demeure pleinement alimenté, la consommation énergétique induite par l'exécution d'instructions est évitée. Si les cycles masqués et actifs sont entrelacés sur la figure 2.6, ce n'est pas

(forcément) le cas dans des conditions réelles d'utilisation, puisqu'un rapport cyclique de 50% signifie qu'**en moyenne** 50% des cycles processeurs sont masqués. Ainsi, bien que pour les deuxième et troisième signaux de la figure 2.6 le cœur de calcul considéré n'a une période de productivité effective équivalente qu'à deux cycles processeurs, les moyens mis en œuvre pour y parvenir sont fondamentalement différents. Pour finir concernant les `T-States`, précisons que [Schöne 2016] propose une étude en profondeur de leur fonctionnement et de leur implantation pour plusieurs architectures de processeurs Intel®, dont `Haswell`.

Pour finir concernant le modèle `OSPM`, précisons que les manufacturiers de processeurs ne sont pas obligés d'implanter l'intégralité des états prévus par le modèle. Il est par exemple très fréquent qu'un processeur n'exhibe qu'un sous-ensemble des `C-States` standardisés par le modèle `OSPM`. Dans la même veine, la gamme de points de fonctionnement exposés par un cœur de calcul varie d'une architecture matérielle à l'autre, ce qui induit des nombres de `P-States` différents d'un processeur à l'autre.

Bien que le modèle `OSPM` a été présenté, d'autres éléments impliqués dans la gestion énergétique d'un processeur méritent d'être abordés. Pour commencer, définissons la notion de `Thermal Design Power` (`TDP`), qui est une caractéristique cruciale de la conception d'un processeur. Elle correspond à la puissance thermique que le système de refroidissement adjoint au processeur doit être capable de dissiper pour garantir que son fonctionnement nominal[7] sous charge maximale n'entraîne pas de dégradation du processeur en question.

Lorsque l'enveloppe thermique du processeur le permet, c'est-à-dire lorsque la puissance thermique induite par le fonctionnement du processeur actuellement à dissiper est suffisamment inférieure à la `TDP` du processeur, il est possible de surcadencer les cœurs de calcul. Pour les processeurs Intel®, cette technologie s'appelle le `TurboBoost`. Cela consiste à élever la fréquence de fonctionnement du cœur de calcul au-delà de la fréquence nominale du processeur. Le nombre de cycles processeurs effectifs par unité de temps du cœur de calcul en question augmente, ce qui permet **potentiellement**[8] de retirer plus d'instructions par unité de temps que pour un fonctionnement à la fréquence nominale. Ce surcadençage induit une hausse significative de la puissance consommée, et donc une augmentation de la puissance thermique à dissiper. C'est ce qui explique que l'activation de technologies de surcadençage telles que Intel® `TurboBoost` ne peut généralement se faire que sur de courtes durées.

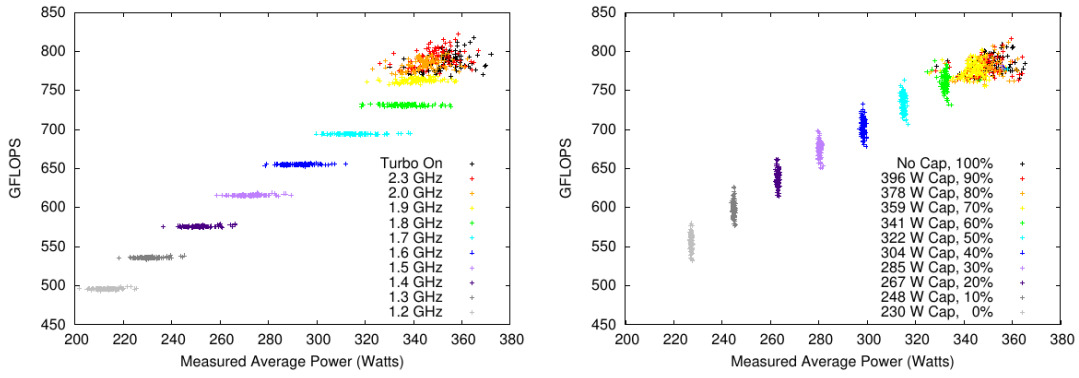
Bref retour sur les unités de calcul vectoriel, et les registres associés. Comme mentionné dans la sous-section 2.1.1.1, l'arité de ces derniers est plus élevée que celle des registres et unités de calcul classiques d'un cœur de calcul : entre 128 bits pour le standard d'instructions `SSE`, et 512 bits pour le standard `AVX512`. Du fait de leurs arités élevées, pour une même fréquence, les utiliser consomme beaucoup plus de puissance électrique, encore une fois comparativement aux unités de calcul et registres standards. Il n'est donc pas pos-

7. C'est-à-dire pour un `C-State C0`, un `P-State P0`, et un `T-State T0` sur l'ensemble des cœurs de calcul, ainsi qu'un `Package C-State Package C0` et une fréquence du `Package` maximale.

8. Le mot « potentiellement » est capital. En effet, si l'état du `pipeline` du cœur de calcul ne permet pas de retirer plus d'instructions, alors le cœur en question peut se retrouver à ne « rien » faire, mais à le faire très vite.
Un tel exemple est largement discuté dans le chapitre 5.

sible d'utiliser les registres et unités de calcul vectoriels pour la même gamme de points de fonctionnement { tension ; fréquence } que celle proposée par le cœur de calcul. Ainsi, lorsque des instructions vectorielles sont exécutées par un cœur de calcul, l'intervalle de points de fonctionnement { tension ; fréquence } adéquat est imposé silencieusement et de manière transparente au niveau matériel, ce qui peut induire un changement de `P-State` non initié par l'utilisateur, sans que celui-ci n'en soit prévenu. Par exemple, [Schöne 2019] précise que pour les processeurs Intel® `Xeon SP 6154 Gold`, la plage de `P-States` autorise des fréquences allant de 1.20 GHz à 3.70 GHz (`TurboBoost` pris en compte), et que l'utilisation des registres et unités de calcul vectorielles contraint cette plage à 2.10 GHz à 3.50 GHz (`TurboBoost` pris en compte).

Pour clore cette sous-section, attardons-nous sur une dernière des fonctionnalités liées à la gestion de l'énergie que les processeurs exposent : la limitation de la puissance consommée (« power capping » pour les anglophones). Cela permet à l'utilisateur de spécifier une borne à ne pas dépasser pour la puissance électrique consommée par le processeur, et le micrologiciel de ce dernier va adapter les `P-States` et `T-States` des cœurs de calcul de manière autonome pour respecter la borne supérieure en question.



(a) HPL: P-State Sweep

(b) HPL: Power Cap Sweep

FIGURE 2.7 – Comparaison des impacts d'un changement de `P-State` d'une part, et d'une limitation de puissance d'autre part, sur les performances observées et la puissance consommée, par les processeurs Intel® `Xeon E5-2698v3` d'une partition de 100 nœuds de calcul, lors de l'exécution de `HPL`. (Source : [Pedretti 2018])

Pour ce faire, tout processeur moderne se voit muni de sondes wattmétriques lui permettant d'évaluer sa consommation énergétique, et une boucle de contrôle est implémentée au sein de l'agent logiciel supervisant la limitation de puissance. Pedretti et al. offrent, dans [Pedretti 2018], une comparaison des impacts sur la puissance consommée et les performances mesurées de la mise en place d'une limitation de puissance d'une part, et d'un changement de `P-State` d'autre part. La figure 2.7 présente les résultats à l'échelle d'une partition de 100 nœuds de calcul embarquant deux processeurs Intel® `Xeon E5-2698v3`, pour l'exécution de `HPL`.

Deux observations méritent d'être faites. Premièrement, d'une part les changements de point de fonctionnement { tension ; fréquence } induisent des impacts sur les performances, et donc des fréquences effectives, similaires, mais une disparité élevée des puissances consommées associées. D'autre part, les limitations de puissance induisent des

puissances consommées similaires, mais des performances, et donc des fréquences effectives hétérogènes. Ainsi, on peut en conclure que la limitation de puissance cherche à garantir la puissance effectivement consommée par le processeur, lorsque les points de fonctionnement { tension ; fréquence } cherche à garantir la fréquence effective des cœurs de calcul. Cela souligne que même des processeurs de même modèle peuvent présenter de fortes disparités d'efficacité énergétique.

Deuxièmement, aussi bien pour une limitation de puissance qu'un changement de `P-State`, on peut observer que les points de données pour les fréquences supérieures ou égales à 2.00 GHz, y compris celle pour laquelle le `TurboBoost` est forcé à chaque fois que l'enveloppe thermique le permet, constituent un nuage de points relativement homogène. Cela met en exergue que le `TurboBoost`, même lorsque son activation n'est pas forcée, s'active lorsque la fréquence effective le permet et qu'une opportunité est identifiée. Ainsi, pour se prémunir d'une activation non souhaitée du `TurboBoost`, il faut le désactiver complètement.

2.1.1.3 Architecture spécifique : `manycore`

Pour terminer cette sous-section dédiée à la description de l'architecture d'un processeur, intéressons-nous à un type de processeur à l'architecture spécifique, que l'on rencontre typiquement dans le domaine du calcul à haute performance [9] : les processeurs `manycores`.

L'architecture d'un processeur `manycore` se caractérise par un nombre de cœurs de calcul important, généralement entre 256 et 2048 pour les modèles récents. Un processeur `manycore` exhibe ainsi un très haut degré de parallélisme pour compenser des performances individuelles des cœurs de calcul comparativement plus faibles que celles offertes par un cœur de calcul d'un processeur standard. Ces cœurs de calcul plus simples exhibent généralement des efficacités énergétiques plus élevées que les cœurs de calcul standards.

Un tel nombre de cœurs rend notamment la mise en place d'un anneau d'interconnexion des cœurs de calcul, d'agents de gestion des caches, et de bus d'interconnexion des composants d'un nœud de calcul particulièrement compliquée et potentiellement inefficace. Pour remplacer ces éléments, on trouve généralement des mémoires `scratchpads` se substituant aux caches, des réseaux sur puce qui interconnectent les cœurs de calcul, ou encore l'utilisation intensive d'Accès Direct à la Mémoire (`DMA`).

L'exécution d'une application `HPC` sur des processeurs `manycores` nécessite généralement d'adapter cette dernière pour qu'elle exploite pleinement les différentes ressources matérielles mises à disposition. Néanmoins, en tirant profit du degré de parallélisme élevé proposé par les processeurs `manycores`, l'application en question peut atteindre des niveaux de performance et d'efficacité énergétique généralement plus élevés que ceux qu'elle atteindrait sur une architecture matérielle classique.

9. On peut par exemple citer le `Sunway TaihuLight`, supercalculateur chinois actuellement en quatrième position du Top500, qui est équipé de processeurs `manycores Sunway SW26010`.
Une description des processeurs en question est notamment proposée dans [Ao 2018].

2.1.2 Mémoire principale

Les jeux de données traités par les applications `HPC` sont vastes, pouvant représenter jusqu'à plusieurs dizaines de téraoctets. Ils sont assurément bien trop imposants pour résider dans la hiérarchie de caches des processeurs. C'est là qu'intervient la mémoire principale des nœuds de calcul. Comme illustré par la figure 2.8, les technologies actuelles telles que la `DRAM DDR4` offrent des volumes mémorisants atteignant les 128 *Go*, ce qui permet d'équiper un nœud de calcul de plus d'un téraoctet de mémoire principale, dépendamment de son architecture matérielle.

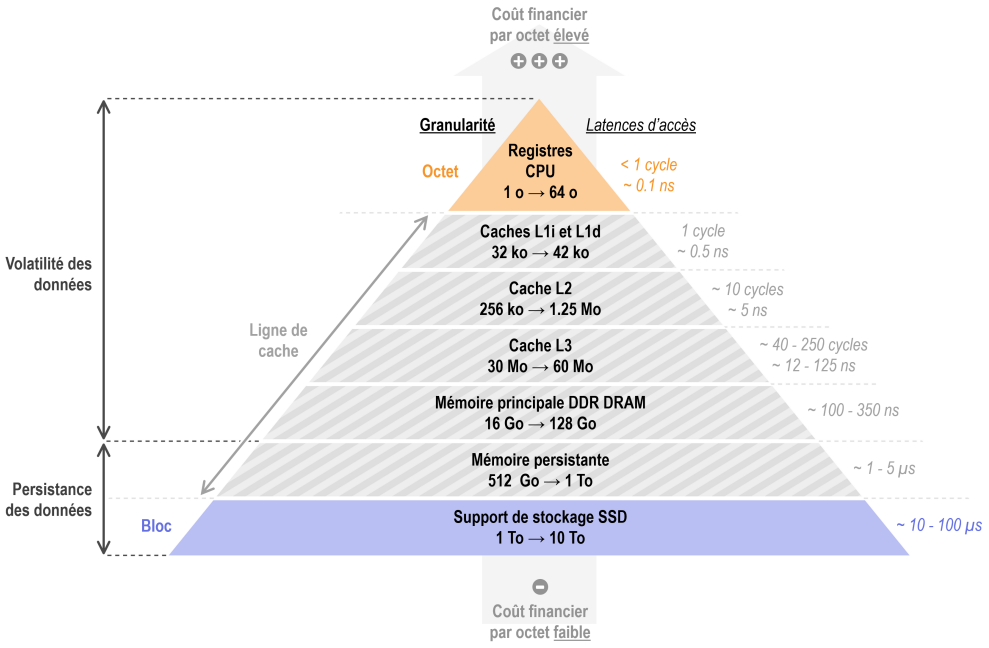


FIGURE 2.8 – Schéma illustrant la « hiérarchie » des composants mémorisant typiques de l'architecture d'un supercalculateur.
Plus on s'élève dans la hiérarchie, plus on s'approche des `pipeline` des cœurs de calcul et plus les latences d'accès aux données sont faibles, mais plus le volume de données mémorisables est faible en raison de coûts de fabrication plus élevés.
NB : les latences d'accès et les volumétries figurant sur ce schéma sont des ordres de grandeur pour une architecture matérielle typique du domaine du calcul à haute performance au moment de l'écriture de ce manuscrit.

Néanmoins, comme la figure 2.8 le laisse également paraître, la latence d'accès à la mémoire principale est également bien plus élevée que celle associée aux registres du processeur ou à la hiérarchie de caches. En conséquence, la mémoire principale peut se révéler être un goulot d'étranglement pour les performances des applications `HPC` engendrant beaucoup d'accès à cette dernière.

Pour être précis, la latence d'accès à la mémoire principale n'est pas le seul facteur qui peut expliquer qu'elle agisse parfois comme un goulot d'étranglement pour les performances. En effet, la bande passante entre la mémoire principale et le processeur joue également un rôle dans l'approvisionnement des données vers les registres des cœurs de

calcul. Afin de maximiser cette dernière, un processeur possède plusieurs contrôleurs mémoires, chacun en charge de plusieurs unités de mémoire. En répartissant les données entre les différentes unités en question, il est possible de transférer des données via les différents contrôleurs mémoires en parallèle. La bande passante effective est alors la somme de celles des contrôleurs mémoires.

Notons que la mémoire persistante (`PMem`) a fait son apparition dans le domaine du calcul à haute performance. Cette dernière s'intercale entre le stockage sur support et la mémoire principale, en termes de latences et de volumes mémorisant. Elle est structurée selon la granularité de la ligne de cache, à l'image de la mémoire principale, mais permet la persistance des données, à l'image des supports de stockage. Pour finir, précisons que la mémoire persistante exhibe plusieurs modes de fonctionnement, dont deux principaux : (1) le mode « mémoire », dans lequel elle prend le rôle de mémoire principale, et où cette dernière joue alors le rôle de cache pour la mémoire persistante, le tout étant fait de manière transparente pour l'utilisateur, et (2) le mode « accès direct » qui expose la mémoire persistante comme une ressource adressable indépendante de la mémoire principale, munie d'une interface de programmation pour permettre au développeur d'une application de choisir le support mémorisant à cibler pour les objets manipulés par ladite application.

2.1.3 Réseau d'interconnexion rapide

Le domaine du calcul à haute performance repousse les limites des paradigmes de la programmation parallèle et de la programmation distribuée : des millions de processus s'exécutent sur des milliers de nœuds de calcul en échangeant potentiellement des quantités gargantuesques de données pour mener à bien une quantité pantagruélique de calcul en un minimum de temps.

On comprend alors que le réseau de communication permettant les échanges de données en question doit être extrêmement performant pour ne pas agir comme un goulot d'étranglement sur les performances du supercalculateur. Le réseau de communication en question, c'est le réseau d'interconnexion rapide. Il offre une bande passante très élevée et une latence très faible [10] pour que les paquets transitent le plus rapidement possible entre les nœuds de calcul qu'il relie.

Pour ce faire, les réseaux d'interconnexion rapides combinent des technologies matérielles de pointe (e.g. en matière de fibre optique), et des procédés logiciels permettant d'accélérer les traitements associés à l'envoi et la réception de données (e.g. utilisation de l'accès distant direct à la mémoire principale (`RDMA`) pour éviter de monopoliser les processeurs et s'affranchir de plusieurs recopies des données). De plus, pour garantir le passage à l'échelle du réseau d'interconnexion rapide, des topologies hiérarchiques avec plusieurs niveaux de commutateurs sont mises en place.

Pour finir, le fait que tous les principaux constructeurs de supercalculateurs développent des réseaux d'interconnexion rapides propriétaires souligne une dernière fois à quel point c'est une ressource cruciale.

10. Respectivement jusqu'à 200 Gb/s et moins de 200 ns pour la technologie `InfiniBand HDR` de `Mellanox`, comme le précise [Mellanox 2019].

2.1.4 Réseau de management

Le principal rôle du réseau de management est de permettre à l'administrateur du supercalculateur d'accéder à distance aux interfaces d'administration des nœuds de calcul. En effet, il est peu pratique, voire impossible et/ou interdit sans autorisation préablable, selon les centres de calcul, de se rendre physiquement auprès du supercalculateur. Et il serait contre-productif d'utiliser le réseau d'interconnexion rapide pour effectuer les tâches d'administration puisque cela consommerait de la bande passante dont les applications `HPC` exécutées pourraient profiter. D'où la nécessité d'un réseau dédié.

Les usages classiques du réseau de management incluent notamment la gestion du cycle de vie (i.e. démarrage, arrêt, redémarrage, etc.) des nœuds de calcul, l'affichage de leurs consoles, ou bien encore leurs mises à jour. Pour finir, précisons que le réseau de management est généralement un réseau `Ethernet` (e.g. `Ethernet 10G`) car il n'y a pas de contrainte forte sur les performances du réseau de management.

2.1.5 Stockage

Comme cela est présenté par la figure 2.8 les supports de stockage offrent de larges espaces de stockage permanents, mais affichent des temps d'accès aux données largement supérieurs à ceux affichés par la mémoire principale et la hiérarchie de caches.

En conséquence, les applications de calcul à haute performance sont conçues de sorte à ne solliciter les systèmes de stockage que ponctuellement, typiquement pour effectuer des sauvegardes (intermédiaires) de leurs résultats. À l'approche de l'ère de ExaScale, les résultats en question représentent d'énormes volumes de données qui peuvent par exemple atteindre plusieurs dizaines, voire centaines, de téraoctets. Si le système de stockage ne parvient pas à absorber suffisamment rapidement de telles salves d'E/S, il agit alors comme un goulot d'étranglement pour les performances des applications `HPC`.

Afin de satisfaire ces contraintes de performance, les systèmes de stockage des supercalculateurs sont constitués d'un ensemble de nœuds de calcul spécifiquement conçus, et entièrement dédiés, à la gestion des E/S. Les nœuds en question sont équipés de supports de stockage à la pointe de la technologie, offrant des débits de lecture et d'écriture élevés [11]. De plus, ils sont connectés au réseau d'interconnexion rapide pour minimiser autant que possible le temps de transfert des données depuis les nœuds de calcul vers le système de stockage.

À cela s'ajoute la mise en place de systèmes de fichiers spécifiquement conçus pour le domaine du calcul à haute performance, à l'image de `Lustre` [12]. Ces systèmes de fichiers permettent notamment de segmenter les E/S en plusieurs « blocs » afin de les accélérer en les parallélisant, de se passer de recopies intermédiaires inutiles, ou bien encore de répartir la charge d'E/S efficacement entre plusieurs nœuds du système de stockage pour absorber les salves d'E/S. Précisons également que ces systèmes de fichiers sont « montables » et utilisables classiquement par les nœuds de calcul, à l'image de ce que permet un système de fichiers standard tel que `ext4`.

Pour conclure cette sous-section, deux remarques supplémentaires. Premièrement, le

11. Les supports de stockage Intel® `SSD Optane` offrent par exemple un débit maximale de 2600 Mo/s en lecture, et de 2200 Mo/s en écriture. Source : https://www.intel.fr/.
12. `https://www.lustre.org/`

système de stockage d'un supercalculateur est partagé par l'ensemble de ses utilisateurs. Ainsi, l'espace de stockage qu'il met à disposition d'un utilisateur en particulier ne correspond généralement qu'à une fraction de son espace de stockage complet, et il faut s'assurer logiciellement que l'utilisateur respecte le quota qui lui a été attribué.

Deuxièmement, les nœuds de calcul peuvent optionnellement être équipés de supports de stockage locaux. Cela offre notamment aux utilisateurs une alternative au système de stockage mutualisé du supercalculateur pour les E/S liées à des données temporaires locales aux nœuds de calcul.

2.1.6 Accélérateurs

Dernier composant matériel typique de l'architecture d'un supercalculateur sur lequel nous allons nous concentrer : l'accélérateur. Deux types d'accélérateurs seront présentés par cette sous-section : les `General Purpose Graphics Processing Unit` (`GPGPU`) seront l'objet de la sous-section 2.1.6.1, puis la sous-section 2.1.6.2 introduira les `Field-Programmable Gate Array` (`FPGA`).

Afin de conclure ce préambule, précisons que l'écrasante majorité des accélérateurs ne se substituent pas à un processeur. Ils ont besoin d'être pilotés, généralement par un processeur standard, auquel ils offrent la possibilité de déléguer des calculs, qu'ils exécuteront plus rapidement et plus efficacement que le processeur en question. D'où leurs noms d'accélérateurs. Cela implique notamment la gestion des transferts de données entre processeur, mémoire principale, et accélérateur, qui doivent être aussi efficaces que possible pour profiter pleinement de l'accélération induite par la délégation des calculs sur l'accélérateur.

2.1.6.1 `General Purpose Graphics Processing Unit` (`GPGPU`)

Les `GPGPU` sont devenus incontournables dans le domaine du calcul à haute performance, notamment dans le cadre de l'ExaScale. En effet, les futurs premiers supercalculateurs exaflopiques, El Capitan, Frontier, et Aurora, s'appuyeront tous sur des architectures hybrides `CPU`+`GPGPU`.

Bien que l'approche sous-jacente à l'architecture d'un `GPGPU` soit similaire à celle sous-jacente aux processeurs `manycores` présentée par la sous-section 2.1.1.3, elles diffèrent par bien des points. Pour commencer, un `GPGPU` est un accélérateur, et a donc besoin d'être piloté par un processeur. Ensuite, un `GPGPU` affiche un nombre de cœurs grandement supérieur, et ses cœurs sont simples, cadencés à des fréquences peu élevées, et spécialisés. Par exemple, l'architecture d'un `GPGPU NVIDIA AMPERE A100` compte un total de 8192 unités de calcul pour des nombres en virgule flottante simple précision, 8192 unités de calcul pour des nombres entiers, 4096 unités de calcul pour des nombres en virgule flottante double précision, et 512 unités de calcul tensorielles, spécifiques aux charges de travail typiques du domaine de l'apprentissage profond.

Le but est ici d'offrir un haut degré de parallélisme, profitable aux applications qui passent à l'échelle, afin de compenser très largement les plus faibles performances individuelles des cœurs d'un `GPGPU` comparativement à un processeur. S'articuler autour de cœurs de calcul plus simples et faiblement cadencés permet également aux `GPGPU` d'être particulièrement efficaces du point de vue énergétique. Un atout notable dans le domaine

du calcul à haute performance, quand on sait l'importance que l'efficacité énergétique revêt et revétira dans les décennies à venir.

Pour finir, précisons que les GPGPU embarquent de la mémoire (e.g. $6 \times 8\ Go = 48\ Go$ de mémoire HBM2, pour un GPGPU NVIDIA AMPERE A100), et des interfaces d'interconnexion rapides (e.g. PCIe 4.0, toujours pour un GPGPU NVIDIA AMPERE A100). Si on ajoute à cela la gestion de DMA (pour rappel, Accès Direct à la Mémoire), cela permet de transférer efficacement de larges (portions de) jeux de données depuis la mémoire principale vers la mémoire du GPGPU. La finalité est de pouvoir profiter d'une bande passante très élevée, et d'une moindre latence, entre la mémoire du GPGPU et les cœurs de calcul, afin que la mémoire principale n'agisse pas comme un goulot d'étranglement pour les performances de l'accélérateur.

2.1.6.2 Field-Programmable Gate Array (FPGA)

Le cœur d'un FPGA est un vaste réseau de cellules logiques reconfigurables, grâce à une circuiterie spécifique. Chaque cellule peut exécuter des opérations logiques et arithmétiques, ainsi que des opérateurs basiques, tels qu'un multiplexeur, en fonction de sa configuration (applicable via une interface logicielle). De plus, le réseau de cellules logiques est lui aussi re-routable, également logiciellement, ce qui permet de concevoir des circuits complexes, puis de les implanter sur le FPGA. Pour compléter cette description de l'architecture des FPGA, précisons que le réseau de cellules logiques est notamment agrémenté de composants mémorisants (e.g. mémoires scratchpads), et de dispositifs de génération de signaux d'horloge.

La principale utilité d'un FPGA est qu'il permet de prototyper des circuits imprimés, voire de les remplacer dans des cas précis. On peut par exemple citer HDEEM (présenté par [Hackenberg 2014]), un système permettant d'échantillonner à haute fréquence (jusqu'à 1 kHz) des sondes wattmétriques pour monitorer finement la consommation énergétique d'un nœud de calcul. La technologie FPGA a été retenue car elle permettait de réduire les coûts de fabrication (manufacturer un circuit imprimé peut s'avérer extrêmement onéreux) tout en conférant des possibilités d'évolutivité au système.

Et l'utilisation d'un FPGA en guise d'accélérateur dans le domaine du HPC est très largement envisagée, comme en atteste [Christodoulis 2019]. En effet, il est possible de programmer un FPGA pour qu'il implante matériellement un noyau de calcul (ou une partie de ce dernier). Or, « l'implantation matérielle » d'une fonction est en général significativement plus rapide et efficace que « l'implémentation logicielle » correspondante. Un FPGA étant reconfigurable, cela permet donc d'envisager d'accélérer les parties clés d'une application HPC en déléguant leurs exécutions à des circuits matériels correspondant, implantés sur un FPGA.

2.2 Intensité opérationnelle et modélisation `Roofline`

Avant de définir la modélisation `Roofline` à proprement parler, intéressons-nous à une notion fondamentale sur laquelle elle s'appuie : l'**intensité opérationnelle**. Pour terminer ce préambule, précisons que l'intensité opérationnelle et la modélisation `Roofline` ont été définies par S. Williams, A. Waterman, et D. Patterson dans [Williams 2009].

L'intensité opérationnelle d'un noyau de calcul est le nombre d'opérations en virgule flottante qui peuvent être exécutées par octet de données à rapatrier depuis la mémoire principale vers les registres du cœur de calcul exécutant le noyau en question.

Précision importante : seuls les octets de données absents de la hiérarchie de cache au moment où ils sont accédés sont considérés. Ainsi, l'intensité opérationnelle met l'accent sur le trafic entre la mémoire principale et la hiérarchie de cache. Là où d'autres métriques, telles que l'intensité arithmétique, considère les octets de données transférés depuis la hiérarchie de caches vers les registres des cœurs de calcul, mettant ainsi l'accent sur le trafic entre ces deux sous-systèmes.

Plus l'intensité opérationnelle d'un noyau de calcul est élevée, plus le temps d'exécution de ce dernier est dominé par l'exécution de calculs. À l'opposé, plus l'intensité opérationnelle d'un noyau de calcul est proche de 0, plus le temps d'exécution de ce dernier est dominé par les latences inhérentes aux transferts de données depuis la mémoire principale vers les registres des cœurs de calcul. Une intensité opérationnelle d'une valeur de 1 correspond alors à un « équilibre » : en moyenne sur l'ensemble du noyau de calcul, il faut rapatrier un octet de données depuis la mémoire principale pour une instruction exécutée.

Terminons ce paragraphe concernant l'intensité opérationnelle en remarquant que cette dernière n'est pas forcément triviale à déterminer par une simple lecture du code source du noyau de calcul considéré. De nombreux travaux de recherche ont pour objectif d'automatiser la détermination de l'intensité opérationnelle d'un noyau de calcul dans des cas particuliers. On peut par exemple citer les travaux présentés par [Olivry 2020], qui permettent le calcul d'une borne inférieure de l'intensité opérationnelle des noyaux de calcul affines.

Concentrons-nous maintenant sur la modélisation `Roofline`. Cette dernière met en relation la puissance de calcul offerte par un processeur, la bande passante entre la mémoire principale et le processeur en question, et l'intensité opérationnelle du noyau de calcul considéré. Ce faisant, la modélisation `Roofline` a trois principaux cas d'utilisation, qui s'avèrent connexes : (1) fournir une aide graphique à la co-conception entre l'architecture d'un nœud de calcul et les noyaux de calcul qu'il sera spécifiquement amené à exécuter, (2) proposer une estimation du niveau maximum de performance atteignable pour un noyau de calcul donné simplement à partir de l'évaluation de son intensité arithmétique, et (3) accompagner le développeur dans le processus d'optimisation du noyau de calcul en question.

Afin de mieux comprendre le fonctionnement de la modélisation `Roofline`, détaillons la manière dont elle est construite. La figure 2.9, extraite de [Williams 2009], illustre le processus de construction d'une modélisation `Roofline`. Pour commencer, les deux **plafonds globaux** de performance, représentés par des traits noirs épais, sont tracés. Le trait horizontal représente la puissance de calcul maximale que le processeur considéré

peut fournir, lorsque la bande passante de la mémoire principale n'est pas saturée et tous les ports arithmétiques des `pipelines` de l'ensemble de ses cœurs de calcul sont saturés. Le trait oblique représente la puissance de calcul maximale atteignable lorsqu'il y a saturation de la bande passante de la mémoire principale. Plus on s'approche de l'origine de l'axe des abscisses, plus l'intensité opérationnelle diminue, et donc plus il faut transférer de données depuis la mémoire principale vers les registres des cœurs de calcul pour que ces derniers puissent exécuter une instruction. En conséquence, lorsqu'il y a saturation de la bande passante de la mémoire principale, le temps moyen nécessaire à l'exécution d'une instruction augmente lorsque l'intensité opérationnelle diminue. D'où la décroissance de la puissance de calcul maximale atteignable au fur et à mesure que l'intensité opérationnelle diminue.

Ces deux plafonds globaux s'intersectent au niveau du **point crête**, qui est associé à l'intensité opérationnelle « crête », pour laquelle la puissance de calcul mise à disposition par le processeur et la bande passante de la mémoire principale sont entièrement consommées. Pour une intensité opérationnelle inférieure à l'intensité opérationnelle crête, la bande passante de la mémoire principale limite les performances atteignables par le noyau de calcul associé. À l'opposé, pour une intensité opérationnelle supérieure ou égale à l'intensité opérationnelle crête, les performances atteignables par le noyau de calcul considéré sont limitées par la bande passante des cœurs de calcul en termes d'instructions exécutables par cycle du processeur.

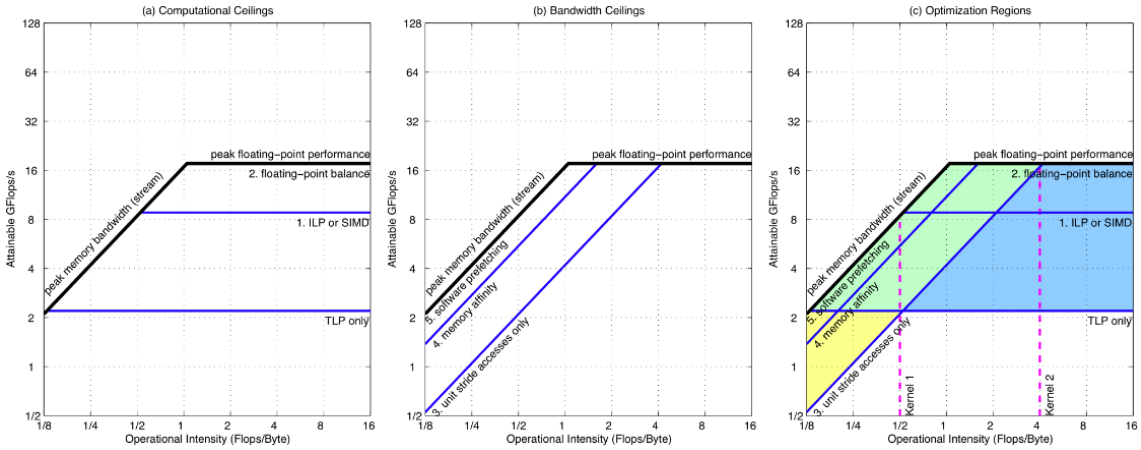


FIGURE 2.9 – Construction et utilisation d'une modélisation `Roofline`. De gauche à droite : (a) construction des plafonds liés aux unités de calcul, (b) construction des plafonds liés à la mémoire principale, et (c) rassemblement de l'ensemble des plafonds, et détermination du facteur limitant pour deux noyaux de calcul.
(Source : [Williams 2009])

La seconde étape de la construction d'une modélisation `Roofline` consiste à tracer les **plafonds intermédiaires**, représentés par des traits bleus sur la figure 2.9. Ces plafonds intermédiaires représentent des paliers pour atteindre les plafonds globaux. En effet, pour exploiter au maximum les ressources matérielles d'un nœud de calcul, il faut utiliser parfaitement l'intégralité des fonctionnalités proposées par ses composants. Et donc mettre en œuvre un ensemble de techniques pour y parvenir. La mise en œuvre d'une desdites techniques permet d'améliorer les performances du noyau de calcul considéré, et constitue

donc un palier dans son optimisation. Par exemple, la figure 2.9 présente les différents paliers pour atteindre le plafond global des cœurs de calcul pour un processeur AMD Opteron X2, à savoir : (1) exploiter le parallélisme de thread complètement (TLP), (2) exploiter pleinement le parallélisme d'instruction, c'est-à-dire tirer profit du caractère superscalaire du processeur en faisant en sorte que tous les ports des cœurs de calcul puissent être sollicités équitablement et continuellement, et (3) équilibrer les opérations d'addition et de multiplication au sein du noyau de calcul pour permettre la fusion desdites opérations (FMA).

En calculant l'intensité opérationnelle d'un noyau de calcul, il est alors possible d'estimer les performances maximales qu'il pourra atteindre sur l'architecture matérielle associée à la modélisation Roofline. Et les différents paliers guideront le développeur du noyau de calcul au cours de son optimisation dans le but d'atteindre lesdites performances maximales.

Pour finir, précisons que la modélisation Roofline a été la pierre angulaire de bon nombre de travaux de recherche, dont certains ont eu pour but de l'étendre et de la perfectionner. On peut notamment citer [Koskela 2018], qui étend la modélisation Roofline pour qu'elle prenne en compte la hiérarchie mémoire complète (i.e. mémoire principale **et** caches), avec un plafond global et des plafonds intermédiaires pour chaque niveau de ladite hiérarchie mémoire. Pour être complets, précisons que cette extension de la modélisation Roofline s'appuie sur l'utilisation d'un simulateur de cache.

2.3 Distances et mesures de similarité

Cette section définit un ensemble d'outils mathématiques autour de la notion de distance, et notamment le `Dynamic Time Warping` (`DTW`) et le `Within-Group Sum of Squares` (`WGSS`) (qui sont respectivement les objets des sous-sections 2.3.3 et 2.3.4), qui sont au cœur du fonctionnement de `Phase-TA`.

2.3.1 $d_1(\cdot)$ ou distance de Manhattan

Conceptuellement, la distance de Manhattan, également notée d_1, entre deux objets potentiellement multidimensionnels, est calculée en cumulant les différences entre les composantes des deux objets pour chaque dimension.

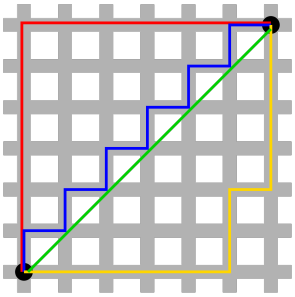


FIGURE 2.10 – Schéma illustrant les distances euclidienne, en vert, et de Manhattan, en rouge, bleu et jaune, à l'aide de la traditionnelle métaphore du taxi new-yorkais.
(source : https://fr.wikipedia.org/wiki/Distance_de_Manhattan)

Le parcours d'un taxi dans les rues quadrillées de Mahnattan est très souvent choisi pour illustrer son principe, et c'est d'ailleurs de là que lui vient son nom. Comme le montre la figure 2.10, afin de relier le point B depuis le point A, un taxi new-yorkais doit circuler dans les rues à angles droits de la ville, et peut indifféremment emprunter un des chemins bleu, rouge, ou jaune. La longueur de ces trois chemins est identique, et égale à la distance de Manhattan entre A et B, c'est-à-dire à la somme des distances entre A et B respectivement sur les axes horizontal et vertical.

Définissons mathématiquement la distance de Manhattan pour deux vecteurs de nombres réels. Soit $n \in \mathbb{N}^*$, soient $a, b \in \mathbb{R}^n$ deux vecteurs de nombres réels tels que $a = [a_0, a_1, \cdots, a_{n-1}]$ et $b = [b_0, b_1, \cdots, b_{n-1}]$. Alors, la distance de Manhattan entre a et b est définie par :

$$d_1 = \sum_{i=0}^{n-1} |a_i - b_i| \tag{2.1}$$

2.3.2 $d_2(\cdot)$ ou distance euclidienne

Quant à la distance euclidienne, généralement notée d_2, elle correspond conceptuellement à la distance « à vol d'oiseau », c'est-à-dire en ligne droite, entre deux objets. Cela est d'ailleurs illustré par le chemin vert de la figure 2.10.
Mathématiquement, si l'on reprend les définitions des vecteurs de nombres réels a et b

de la sous-section 2.3.2 définissant la distance de Manhattan, la distance euclidienne est alors définie par :

$$d_2 = \sqrt{\sum_{i=0}^{n-1} (a_i - b_i)^2} \tag{2.2}$$

2.3.3 Dynamic Time Warping (DTW)

DTW est une mesure de similarité notamment applicable aux séries temporelles à valeurs réelles, présentée et définie par [Sakoe 1971]. Le calcul de la DTW entre deux séries temporelles implique de créer des associations entre les points des deux séries temporelles. Deux points associés sont dits alignés. Tout point d'une des deux séries temporelles est forcément aligné avec au moins un point de l'autre série temporelle.

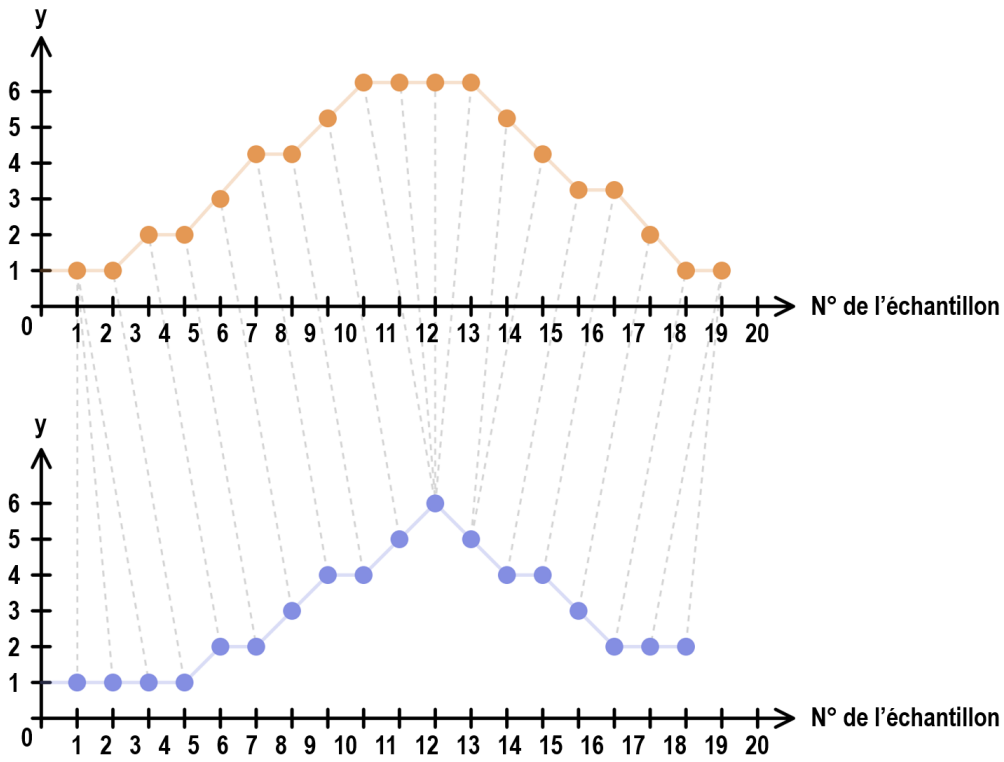


FIGURE 2.11 – Exemple d'alignements entre les points de deux séries temporelles à valeurs réelles, lors du calcul de la DTW entre lesdites séries temporelles

Les alignements sont définis de sorte à minimiser la somme des distances entre deux points alignés. Cette somme est d'ailleurs la valeur de la DTW entre les deux séries temporelles considérées. Conceptuellement, le fait qu'un point d'une série temporelle puisse être aligné avec plusieurs points de l'autre série temporelle rend la DTW particulièrement résistante aux perturbations temporelles. La figure 2.11 présente un exemple d'alignement entre les points de deux séries temporelles, lors du calcul de la DTW entre lesdites séries temporelles, les lignes pointillées représentant les alignements.

Soient $m, n \in \mathbb{N}^*$, soient $\mathcal{A} \in \mathbb{R}^m$ et $\mathcal{B} \in \mathbb{R}^n$ deux séries temporelles à valeurs réelles telles que $\mathcal{A} = (a_1, \cdots, a_m)$ et $\mathcal{B} = (b_1, \cdots, b_n)$. Alors, mathématiquement, `DTW` est définie par :

$$\begin{aligned}
\forall (i,j) \in [\![2;m]\!] \times [\![2;n]\!], \delta(a_i, b_j) &= d_2(a_i, b_j) + \\
&\quad min\left(\delta(a_{i-1}, b_j), \delta(a_i, b_{j-1}), \delta(a_{i-1}, b_{j-1})\right) && (2.3)\\
\delta(a_1, b_1) &= 0 && (2.4)\\
\forall i \in [\![2;m]\!], \delta(a_i, b_1) &= d_2(a_i, b_1) + \delta(a_{i-1}, b_1) && (2.5)\\
\forall j \in [\![2;n]\!], \delta(a_1, b_j) &= d_2(a_1, b_j) + \delta(a_1, b_{j-1}) && (2.6)\\
DTW(\mathcal{A}, \mathcal{B}) &= \delta(a_m, b_n) && (2.7)
\end{aligned}$$

Définissons alors $\texttt{DTW}_2$ de la même manière que `DTW`, à l'exception du fait que la distance euclidienne est élevée au carré dans les équations 2.3, 2.5, et 2.6.

2.3.4 Within-Group Sum of Squares (WGSS)

Définissons maitenant `WGSS` en partant de sa définition mathématique. Soit $\mathcal{I}$ un ensemble de séries temporelles à valeurs réelles, soit p une série temporelle à valeur réelles, alors le `WGSS` de p par rapport à $\mathcal{I}$ est défini par :

$$WGSS(p, \mathcal{I}) = \sum_{i \in \mathcal{I}} DTW_2(p, i) \tag{2.8}$$

En d'autres termes, le `WGSS` est un indicateur de la distance cumulée de la série temporelle p à l'ensemble de séries temporelles $\mathcal{I}$.

Étude de l'état de l'art

3

Ce chapitre a pour but de présenter l'état de l'art des thématiques d'intérêt du domaine du calcul à haute performance, du point de vue des travaux présentés par ce manuscrit.

Pour ce faire, la section 3.1 commencera par souligner le fait que le domaine du calcul à haute performance est témoin de profondes transformations dans le cadre du passage de l'ExaScale. Par la suite, la section 3.2 se concentrera sur la thématique de l'optimisation énergétique de l'exécution des applications `HPC` par des outils logiciels. Cela apportera des éléments de contextualisation pour la partie II, qui s'articule autour de `BDPO`, un outil de reconfiguration dynamique ayant pour but d'améliorer l'efficacité énergétique associée à l'exécution d'une application `HPC`. Enfin, la section 3.3 abordera la caractérisation des comportements périodiques exhibés par les séries temporelles. Cela permettra de contextualiser les travaux présentés par la partie III, qui est centrée sur `Phase-TA`, un outil de détection et de caractérisation des comportements localement périodiques des applications `HPC`.

Pour conclure ce préambule, précisons que la comparaison des deux outils sous-jacents aux travaux menés dans le cadre de la thèse aux solutions concurrentes pertinentes présentées par ce chapitre sera proposée au sein des parties dédiées à `BDPO` et `Phase-TA`. Pour être précis, dans la sous-section 6.4.3 pour `BDPO`, dans la sous-section 8.5.3 pour `Phase-TA`. Cela permettra au lecteur d'avoir connaissance des fonctionnements des deux outils lorsqu'ils seront comparés à l'état de l'art.

3.1 Des transformations profondes pour atteindre l'ExaScale

Chaque cap qui a été passé, du téraflops au pétaflops, a représenté un défi technique. Et leurs réalisations ont nécessité des progrès scientifiques et technologiques majeurs, toujours plus conséquents à mesure que les limites de la puissance de calcul étaient repoussées. L'ExaScale ne fait pas exception, et le domaine du calcul à haute performance

connait de grandes transformations pour parvenir à atteindre 1 $EFLOP/s$.

De ces transformations, celle qui vient en premier à l'esprit est aussi la plus profonde : la prédominance, pour ne pas dire l'inévitabilité, des architectures hétérogènes accélérées par des `GPGPU`. En Juin 2008, `Roadrunner` devenait le supercalculateur le plus puissant du monde, en étant le premier à dépasser le $PFLOP/s$. À l'époque, les 10 premières places du TOP500 étaient occupées par des supercalculateurs dont la puissance de calcul venait uniquement de processeurs. 13 ans plus tard, 6 des 10 supercalculateurs les plus puissants sont accélérés par des `GPGPU`, et les trois futurs premiers systèmes exaflopiques le seront également. La principale raison sous-jacente à cette transformation est l'efficacité énergétique proposée par les accélérateurs, globalement largement supérieure à celles des processeurs standards [Lobet 2018].

Et cette transformation sert de fondation à la convergence « `HPC+IA` ». En effet, les techniques d'Intelligence Artificielle (`IA`), apprentissage profond en tête, ont le vent en poupe depuis environ une décennie, notamment parce que les `GPGPU` ont permis d'accélérer les phases d'apprentissage. Si la convergence en question trouve tout d'abord naissance dans la possibilité de partager les supports matériels d'exécution, elle devient de plus en plus profonde, jusqu'au sein des applications. En effet, un exemple de cas d'utilisation particulièrement populaire en ce moment est l'utilisation de réseaux de neurones pour construire des modèles permettant d'approximer la solution de systèmes d'équations complexes. Une fois l'entraînement du réseau de neuronne complet, l'inférence d'une approximation de la solution via le modèle construit est potentiellement plusieurs ordres de grandeurs plus rapide que la résolution du système d'équations par des méthodes classiques. [Archibald 2020, Huang 2020, Di Matteo 2020] présentent précisément des cas pratiques d'utilisation de techniques d'`IA` dans des applications `HPC`. Afin de conclure ce paragraphe, mentionnons le fait que le benchmark `HPL` a maintenant une version « `IA` », nommée `HPL-IA`, qui utilise notamment des nombres à virgule flottante de moindre précision (sur 32 voire 16 bits). `Fugaku`, actuel supercalculateur le plus puissant du monde, a d'ailleurs passé le cap de l'exaflops sur ce benchmark [Kudo 2020].

Et ce phénomène de convergence s'étend à d'autres thématiques de l'informatique distribuée, telles que les « Masses de Données » (« Big Data » pour les anglophones). L'aboutissement de ce rapprochement serait l'avènement des systèmes `UltraScale` : des mastodontes numériques rassemblant des partitions de machines aux architectures hétérogènes et variées pour couvrir une large gamme de champs d'application, et fédérer les thématiques de l'informatique parallèle, distribuée, et de l'apprentissage en mutualisant les ressources matérielles les exécutant. D'ores et déjà, les centres de données atteignent des échelles démesurées et pèsent significativement dans les chiffres de consommation énergétique mondiaux, et cela va en s'accélérant [Nicola 2018]. Le domaine du calcul à haute performance ne fait pas exception et les tailles des supercalculateurs et des infrastructures les accueillant croîent également. Tout comme leurs consommations énergétiques, au point que cela devienne une des principales problématiques du passage de l'ExaScale, comme cela a déjà été longuement discuté dans ce manuscrit. Une des solutions envisagées pour rendre les futurs grands centres de calcul exaflopiques viables énergétiquement et économiquement parlant est le sur-provisionnage (ou « over-provisioning » pour les anglophones). En quelques mots, cela consiste à : (1) équiper le supercalculateur de plus de ressources matérielles que l'alimentation de l'infrastructure ne peut en faire fonctionner dans des conditions nominales, puis à (2) utiliser des moyens logiciels tels que la

limitation de puissance des nœuds de calcul pour faire fonctionner les équipements en étant sous-alimentés. Les articles [Sakamoto 2017] et [Patel 2020] abordent notamment la question du sur-provisionnage.

Dernière transformation notable sur laquelle nous allons nous attarder dans le cadre de cette section : la généralisation du monitoring des jobs HPC et de la collection des données que cela engendre. Cette tendance qui s'inscrit dans une volonté d'appliquer des techniques et outils issus de l'étude des masses de données (« Big Data »), de l'analyse de données (« Data Analytics »), et de l'apprentissage (« Machine Learning ») au domaine du calcul à haute performance. Parmi les nombreux cas d'utilisation, on peut par exemple citer :

— L'amélioration des ordonnanceurs et des politiques d'ordonnancement via l'utilisation d'algorithmes d'apprentissage, et notamment de classification [Legrand 2019, Zrigui 2020] ;

— La prédiction de problèmes matériels tels que la surchauffe d'un processeur, en appliquant des algorithmes d'apprentissage sur les journaux d'évènements des nœuds de calcul [Platini 2021] ;

— Accompagner le développeur d'application HPC lors du processus d'optimisation de cette dernière, en lui offrant des informations sur son comportement et lui permettant de le visualiser [Servat 2018] ;

— Permettre l'auto-tunage des applications de calcul à haute performance à partir des données de monitoring associées à son exécution.

Ce dernier exemple de cas d'utilisation mérite d'être détaillé pour conclure cette section. En effet, l'intégration d'une boucle de rétroaction entre monitoring et auto-tunage des applications HPC dans la pile logicielle de gestion d'un supercalculateur semble inévitable pour tirer profit de la puissance de calcul à disposition de manière efficace, et ainsi atteindre l'ExaScale. L'importance capitale de cette boucle de rétroaction est notamment soulignée par les nombreux projets de recherche gravitant autour de l'ExaScale qui la mettent en leurs cœurs : ANTAREX [1], READEX [2], REGALE [3], ou bien encore les projets DEEP [4], pour ne citer que les projets européens récents.

1. Cf. [Silvano 2019]
2. https://www.readex.eu/
3. https://regale-project.eu/
4. https://www.deep-projects.eu/

3.2 Outils logiciels pour l'optimisation énergétique de l'exécution d'une application de calcul à haute performance

Dans un premier temps, cette section sera l'occasion de présenter l'état de l'art concernant des approches logicielles ayant pour but d'améliorer l'efficacité énergétique associée à l'exécution d'une application HPC différentes de celle sous-jacente à BDPO (qui, pour rappel, est l'objet de la partie II).

Ensuite, dans un second temps, les outils implémentant des approches similaires à celle de BDPO seront abordés individuellement. Cela permettra notamment, dans la sous-section 6.4.3, une fois que BDPO aura été présenté, de le comparer à l'état de l'art.

Mesurer la consommation énergétique associée à l'exécution d'une application HPC Comme cela a déjà été mentionné à plusieurs reprises dans ce manuscrit, la maîtrise de la consommation énergétique des futurs systèmes exaflopiques est cruciale. Et ce constat est tout aussi vrai lorsque l'on se projete une à deux décennies dans le futur, et que l'on considère les systèmes UltraScale. Or, comme le souligne [Barbosa 2018], avant de pouvoir maîtriser la consommation énergétique de tels systèmes, il faut être capable de la monitorer. Du fait du nombre imposant et de l'hétérogénéité des composants constituant ces futurs géants numériques, de nombreuses problématiques se posent. Par exemple, comment gérer l'incommensurable quantité de données que ce monitoring énergétique générera à moindres coûts (notamment financier, énergétique, et écologique) ?

Et ces problématiques se posent déjà lorsque l'on considère le passage de l'ExaScale. Des travaux de recherche centrés sur le monitoring de la consommation énergétique des nœuds de calcul lors de l'exécution d'une application ont donc vu le jour, tels que [Hackenberg 2014], [Rashti 2015], ou bien encore [Libri 2018]. Ces travaux cherchent à remplir trois buts principaux : (1) fournir aux développeurs le moyen d'évaluer l'efficacité énergétique de leurs applications pour qu'ils prennent ce facteur en compte lors de la phase d'optimisation desdites applications, (2) permettre aux administrateurs de suivre finement la consommation énergétique des différents nœuds et jobs, et (3) permettre la mise en place de rétroactions d'autotunage motivées par des considérations énergétiques, notamment au niveau de l'ordonnanceur.

Présenté par [Hackenberg 2014], le fonctionnement de HDEEM a déjà été esquissé par la sous-section 2.1.6.2. En quelques mots, HDEEM est un système sur puce construit autour d'un FPGA, destiné à équiper un nœud de calcul et ayant pour but d'agréger les mesures à haute fréquence (1 kHz) effectuées par des sondes wattmétriques réparties entre les différents composants du nœud de calcul. Les échantillons associés aux 7 dernières heures de fonctionnement peuvent être mémorisés dans une mémoire locale à HDEEM. L'exploitation, via une interface de programmation en langage C, et/ou la journalisation de ces données est laissée à la charge de l'utilisateur.

À l'instar de HDEEM, [Rashti 2015] présente WattProf, un système sur puce connectable via une interface PCIe à un nœud de calcul, et articulé autour d'un FPGA. De plus, WattProf est accompagné de sondes wattmétriques dédiées interposables au niveau des interfaces PCIe des différents composants du nœud de calcul considéré, et des connectiques des bancs de mémoire principale DRAM. De manière similaire à HDEEM, WattProf embarque de la mémoire afin de mémoriser les échantillons issus des mesures de consommation

énergétique qu'il effectue, et propose une interface de programmation pour permettre à l'utilisateur d'accéder aux dits échantillons.

Quant à `Dwarf in a Giant` (`DiG`), présenté par [Libri 2018], il s'agit également d'un système sur puce, mais s'appuyant cette fois sur une carte `Arduino`[5]. De plus, `DiG` se connecte au bloc d'alimentation du nœud de calcul, et permet donc de monitorer la consommation énergétique à l'échelle de ce dernier, et non pas à la granularité de ses composants, contrairement à `HDEEM` et `WattProf`. La carte `Arduino` est notamment utilisée pour traiter les données de consommation énergétique, par exemple à l'aide d'une `FFT`, ainsi que pour publier les données en question via le réseau de management du supercalculateur (cf. section 2.1.4) grâce au protocole `MQTT`[6]. Il est alors nécessaire d'exécuter un relais `MQTT` sur une machine n'étant pas un nœud de calcul (e.g. un nœud d'administration) pour agréger les données de consommation énergétique des différents nœuds, et les mettre à disposition de l'utilisateur qui peut alors y accéder depuis n'importe quel client `MQTT`. Enfin, ajoutons que `DiG` permet également un échantillonnage précis et à haute fréquence, tout en demeurant un système sur puce à bas coût pour le domaine du calcul à haute performance.

Ce qui ressort de l'étude de ces différents travaux de recherche est que le monitoring énergétique des nœuds de calcul des futurs systèmes exaflopiques comportera vraisemblablement un échantillonnage à une fréquence largement supérieure au hertz, en plus de l'échantillonnage basse fréquence traditionnel, utilisé pour suivre la consommation énergétique d'un supercalculateur à gros grain[7]. Cet échantillonnage à haute fréquence se fera vraisemblablement via des systèmes sur puce et des sondes wattmétriques dédiés. Pour finir, les problématiques liées à la mémorisation, au stockage, et à la mise à disposition des données de monitoring seront fondamentales.

Caractériser le comportement des applications de calcul à haute performance Un des socles communs à toutes les approches d'optimisation est l'idée suivante : être capable de caractériser le comportement de l'objet à optimiser (e.g. par des équations mathématiques) est essentiel pour comprendre son fonctionnement, pouvoir raisonner sur ce dernier, et ainsi l'optimiser. Le domaine du calcul à haute performance ne fait pas exception, et c'est pour cela que l'ensemble des outils qui seront présentés dans la suite de ce manuscrit, aussi bien ceux développés dans le cadre de cette thèse que ceux appartenant à l'existant, s'appuient sur une caractérisation de l'application à optimiser. Cette caractérisation peut par exemple être construite à partir d'une analyse de leurs codes sources, ou encore via des données de monitoring associées à leurs exécutions.

Un large éventail de méthodologies de caractérisation est esquissé dans la suite de ce manuscrit, au gré des outils présentés. Il ne s'agit ici donc pas de les cataloguer de manière exhaustive, mais plus précisément de s'attarder sur les travaux dirigés par B. Calder au début des années 2000. En effet, ces derniers se sont révélés fondateurs, et ont inspiré, de près ou de loin, de nombreux travaux d'optimisations à visée énergétique. [Sherwood 2001], [Sherwood 2002], and [Eeckhout 2005] présentent la notion de `Basic Block Vector`, et comment l'exploiter pour caractériser le comportement d'une application. En quelques mots, un `Basic Block` est une portion du code source d'une

5. https://www.arduino.cc/
6. https://fr.wikipedia.org/wiki/MQTT
7. Par exemple via `Bull Energy Optimizer` (`BEO`), développé par l'équipe PowerEfficiency d'Atos.

application qui est exécutée séquentiellement, avec un point d'entrée et un point de sortie uniques (i.e. une portion de code qui ne contient pas d'instruction de branchement). Un programme est ainsi constitué d'un ensemble fini de `Basic Blocks`, déterminable à la compilation. Un `Basic Block Vector` est alors un vecteur d'entiers unidimensionnel, associant à chaque `Basic Block` du programme le nombre de fois qu'il a été exécuté au cours d'un intervalle fixe (e.g. les 30 dernières secondes, ou bien encore les 100 derniers millions d'instructions exécutées). L'analyse des `Basic Block Vectors` d'une application au cours de son exécution sert alors d'outil pour la construction d'une approche d'optimisation de cette dernière. Par exemple, si un sous-ensemble de `Basic Blocks` sont exécutés un grand nombre de fois par rapport à l'ensemble des `Basic Blocks`, ils sont probablement des cibles de choix pour le développeur qui souhaiterait optimiser le code source de son application. Ou bien encore, la clustérisation des `Basic Block Vectors` peut révéler des comportements récurrents au sein d'une application, voire même entre plusieurs applications, *et cætera*. Et cette notion de comportement récurrent donne (quasiment) directement naissance à la notion de **phase**, qui est sous-jacente à la vaste majorité des outils d'optimisation énergétique des applications de calcul à haute performance. `BDPO` ne fait pas exception, et le chapitre lui étant dédié (pour rappel, il s'agit du chapitre 5) propose une formalisation de la définition du concept de **phase**.

Identification des leviers permettant d'optimiser l'efficacité énergétique associée à l'exécution d'une application de calcul intensif Avant de penser à améliorer l'efficacité énergétique associée à l'exécution d'une application `HPC`, il est fondamental d'identifier les moyens mis à disposition pour agir sur le ratio { performances / énergie } d'un nœud de calcul, également appelés **leviers**, et de caractériser leurs impacts sur le ratio en question. Dans ce contexte, la lecture de [Hackenberg 2015] et [Schöne 2019] est plus que conseillée. En effet, dans ces deux articles, les auteurs présentent en détail les principales fonctionnalités liées à la gestion de l'énergie pour les processeurs Intel® d'architecture `Haswell` et `Skylake`. Ces dernières ne seront pas détaillées par cette section, mais plutôt abordées au fil du manuscrit, lorsque cela sera pertinent et enrichissant pour le propos tenu.

Si les deux références bibliographiques qui viennent d'être mentionnées s'intéressent aux fonctionnalités de gestion énergétique exposées par certaines architectures matérielles spécifiques, d'autres travaux se sont intéressés à l'identification et la caractérisation des techniques pouvant servir de leviers pour l'optimisation de l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance.

On peut par exemple citer [Raïs 2018a], qui définit la « version du code source » comme un levier. L'idée principale est de générer plusieurs implémentations du même algorithme, et de caractériser les ratios { performances / énergie } associés à ces différentes implémentations pour déterminer laquelle utiliser préférentiellement en fonction des contraintes sur les performances et la consommation énergétique associées à l'exécution de l'application en question. Dans cet article, les auteurs utilisent un `Domain Specific Language` (`DSL`) pour générer quatre mises en œuvre différentes du parallélisme au sein d'une application `HPC` (s'appuyant sur un modèle de programmation hybride `MPI+OpenMP`). L'article en question décrit également le processus d'automatisation de la mise en place de ce levier [8] et montre qu'il est possible d'adapter la version du code source

8. Pour être précis, la version du code source est couplé à un autre levier, plus couramment employé :

aux contraintes pesant sur l'exécution de l'application, en termes d'énergie consommée et de temps d'exécution. Cela achève la preuve de concept de l'utilité du levier « version du code source ».

Toujours dans l'optique de définir les leviers exploitables du point de vue énergétique, [Raïs 2018b] propose la formalisation d'une méthodologie permettant de construire la « table des leviers pour l'énergie et la puissance électrique ». Dans cet article, Raïs et al. appliquent la méthodologie à l'exécution sur un nœud de calcul d'un noyau de calcul réalisant la multiplication de deux matrices denses ligne par ligne, et se concentrent sur 3 leviers : (1) la parallélisation du code (monothreadé d'une part, et multi-threadé avec un thread `OpenMP` par cœur logique d'autre part), (2) le type et la précision des nombres (entiers, flottants sur 32 bits, et flottants sur 64 bits, et (3) le standard d'instructions vectorielles utilisé (aucun, `SSE3`, et `AVX2`). La construction de la table des leviers leur a notamment permis de déterminer quels étaient les impacts des trois leviers considérés sur 3 métriques : (1) la puissance moyenne consommée par le nœud lors de l'exécution, (2) l'énergie consommée par le nœud à l'échelle de l'exécution complète, et (3) le temps d'exécution. De cet ensemble de connaissances compilées par la table des leviers, les auteurs ont par exemple pu extraire l'observation suivante concernant les conditions d'exécution du benchmark : afin de minimiser l'énergie consommée, il vaut mieux utiliser des instructions vectorielles `AVX2` lorsque l'on travaille avec des nombres à virgule flottante sur 64 bits, et des instructions vectorielles `SSE3` lorsque l'on travaille avec des nombres à virgule flottante sur 32 bits.

Pour finir, l'article [Ferreira Lima 2019] mérite aussi d'être cité. Dans cet article, Ferreira Lima et al. étudient d'autres leviers : (1) l'implémentation du modèle de programmation `OpenMP` utilisé, (2) l'activation du surcadençage (ici Intel® `TurboBoost`), (3) la disponibilité des `C-States` (soit aucun, soit tous disponibles), et (4) le **gouverneur** `acpi-cpufreq`. En quelques mots, un gouverneur est un agent intégré au noyau du système d'exploitation qui implémente une politique de gestion des `P-States` des cœurs de calcul. Généralement, `performance` est le gouverneur utilisé par défaut dans le domaine du `HPC`. Ce dernier implique peu ou prou l'utilisation constante du point de fonctionnement associé à la fréquence nominale du processeur, avec possiblité d'activation du surcadençage. Les benchmarks pour lesquels les leviers mentionnés à l'instant sont trois noyaux de calcul dédiés à la factorisation de matrices denses, respectivement selon les méthodes `LU`, `QR`, et de `Cholesky`.

De la même manière que précédemment, l'exécution des noyaux de calcul a été monitorée, de sorte à pouvoir évaluer l'impact des leviers à l'étude sur les performances et la consommation énergétique associées aux exécutions en question. Cela a permis de tirer plusieurs conclusions, dont deux sont particulièrement intéressantes : (1) le choix de l'implémentation du modèle de programmation `OpenMP` constitue un levier pour l'efficacité énergétique associée à l'exécution d'une application `HPC`, et (2) les impacts du surcadençage et de la disponibilité des `C-States` sur ces deux métriques sont significatifs.

Il apparaît donc clairement que de multiples leviers permettent d'influer significativement sur le ratio { performances / énergie } exhibé par l'exécution d'une application `HPC`. Dans la suite de cet état de l'art, seuls les leviers que je juge les plus intéressants dans le cadre de la contextualisation des travaux sur `BDPO` (présentés par la partie II) seront

l'adaptation des paramètres de parallélisme, i.e. du nombre de rangs `MPI` et de threads `OpenMP`.

abordés.

Reconfiguration dynamique de la fréquence du `Package` Comme présenté dans la sous-section 2.1.1.2, le `Package` expose plusieurs fréquences de fonctionnement, chacune associée avec un ratio { performances / énergie } différent. Ici, les performances en question sont notamment les bandes passantes et les latences d'accès associées à la mémoire principale et au bus d'interconnexion du nœud de calcul.

`DUF`, présenté par [André 2020], et `UPSCavenger`, présenté par [Gholkar 2019], utilisent ce levier pour améliorer l'efficacité énergétique associée à l'exécution d'une application `HPC`. Pour ce faire, tous deux implémentent une boucle de régulation en circuit fermé que l'on peut approximativement résumer de la sorte : (1) ils monitorent des métriques construites à partir de compteurs de performance pour déterminer si la charge de travail induite par l'application `HPC` exécutée est dominée par des calculs ou par des accès mémoires, (2) lorsqu'une opportunité de reconfiguration se présente, ils changent la fréquence du `Package`, (3) via le monitoring mis en place, ils déterminent si la reconfiguration a eu un impact sur les performances supérieur à la limite autorisée, et (4) si ce n'est pas le cas, ils continuent de baisser la fréquence du `Package`, sinon ils rétablissent la fréquence du `Package` la plus basse permettant de respecter la limite de dégradation des performances. Si un changement de type de charge de travail est détecté lors des étapes (1) ou (4), le processus d'optimisation empirique de la fréquence du `Package` est réinitialisé. [André 2020] et [Gholkar 2019] tirent globalement les mêmes conclusions, dont trois particulièrement intéressantes :

— Les portions de l'exécution de l'application dominées intensivement par des calculs n'utilisent généralement pas la totalité de la bande passante vers la mémoire principale, ni celle entre les caches `L3` et `L2`. Ce sont donc de bonnes candidates à une baisse de la fréquence du `Package`, qui affectera principalement les bandes passantes en question ;

— La reconfiguration dynamique de la fréquence du `Package` permet de réaliser des économies d'énergie pouvant dépasser les 10% (jusqu'à plus de 16% pour `DUF` pour l'application `EP` des `NPB`), tout en gardant la dégradation de performance associée aux alentours des 5% ;

— La reconfiguration dynamique de la fréquence du `Package` permet même, dans certaines conditions, d'améliorer les performances de l'application `HPC` exécutée. En effet, en diminuant la fréquence du `Package`, on réduit la puissance thermique à dissiper, ce qui offre plus d'opportunités d'activation du `TurboBoost` pour les cœurs de calcul, ce qui se révèle généralement bénéfique pour les performances des applications dominées intensivement par des calculs.

À noter que [Sundriyal 2018] confirme également les observations ci-dessus, dans le cadre d'une étude des possibilités offertes par la reconfiguration des points de fonctionnement des cœurs de calcul et du `Package` pour l'exécution mono-nœud d'une application `HPC`.

Optimisation de l'efficacité énergétique sous limitation de puissance consommée Comme précisé dans la section 3.1, une des pistes envisagées pour permettre l'opération des futurs systèmes exaflopiques avec une enveloppe énergétique contrôlée est le

sur-provisionnement. Étant donné que dans des conditions normales d'utilisation, il est impossible d'alimenter à pleine puissance l'entièreté des nœuds de calcul d'un supercalculateur sur-provisionné, un compromis entre nombre de nœuds disponibles et puissance d'alimentation des nœuds doit être mis en place. La capacité de limitation de la puissance consommée par les processeurs, présentée par la sous-section 2.1.1.2, devient alors cruciale pour agir sur le compromis qui vient d'être mentionné.

Les auteurs de `GEOPM` (présenté par [Eastep 2016]), et de `PShifter` (présenté par [Gholkar 2018]) partent du constat suivant : lorsque l'enveloppe énergétique attribuée à un job implique la mise en place d'une limitation de puissance, répartir cette dernière de manière homogène entre les nœuds de calcul impliqués dans l'exécution du job en question n'est généralement pas la répartition qui optimise les performances associées à l'exécution de l'application. La raison sous-jacente à ce constat est parfaitement illustrée par la figure 2.7, extraite de [Pedretti 2018] : les performances affichées par des processeurs de même modèle sous une même contrainte de limitation de puissance consommée ont tendance à exhiber une très forte variabilité. En conséquence, les processeurs les plus rapides sous la contrainte de limitation de puissance termineront les premiers les charges de travail qui leurs auront été attribuées, et attendront les processeurs les plus lents.

L'idée sous-jacente à `GEOPM` et `PShifter` est alors d'imposer des limitations de puissance plus fortes aux nœuds les plus rapides afin de pouvoir relâcher celles des nœuds les plus lents, tout en respectant l'enveloppe de puissance assignée au job en question. En homogénéisant les puissances de calcul effectives des nœuds de calcul exécutant une application `HPC`, on accélère le **chemin critique**, à savoir la portion de l'exécution attribuée au nœud le plus lent. Ce faisant, on améliore les performances associées à l'exécution de l'application considérée.

Pour ce faire, `GEOPM` et `PShifter` mettent en place des structures hiérarchiques, avec un agent par nœud de calcul et une hiérarchie d'instances de supervision qui coordonnent les agents locaux aux nœuds de calcul. De plus, ils mettent en place des boucles de rétroaction : (1) les agents évaluent les performances des nœuds de calcul, puis (2) font parvenir ces données aux instances de supervision, qui (3) ajustent alors les limitations de puissance consommée des nœuds, afin que (4) les agents les mettent en place sur les nœuds de calcul, avant de retourner à l'étape (1).

Pour finir, précisons que l'amélioration de performances, et donc d'efficacité énergétique (étant donné que l'enveloppe énergétique attribuée reste la même), permise par l'utilisation de `GEOPM` et `PShifter` est substantielle : au moins 10%, en moyenne plus de 20%, et jusqu'à plus de 30%, par rapport à une répartition équitable de l'enveloppe énergétique entre tous les nœuds de calcul impliqués dans l'exécution de l'application `HPC` considérée.

Exploiter les synchronisations des communications collectives `MPI` [Li 2010], [Halimi 2014], et [Cesarini 2020] présentent des outils exploitant les synchronisations des communications collectives `MPI` pour améliorer l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance. Pour ce faire, `FoREST-mn` (présenté par [Halimi 2014]) utilise un levier appelé `Dynamic Voltage Frequency Scaling` (`DVFS`), qui consiste à adapter le point de fonctionnement { tension ; fréquence } des cœurs de calcul (i.e. leurs `P-States`) pendant l'exécution de

l'application pour adapter ces derniers à la charge de travail qu'elle induit. Comme chaque P-State expose un ratio { performances / énergie } différent, il est possible de rendre l'exécution d'une application plus efficace énergétiquement parlant en optant pour un P-State plus profond lorsque l'application en question n'utilise pas le niveau de performance maximal atteignable exhibé par le point de fonctionnement associé à la fréquence nominale. Cette présentation du DVFS est succinte mais suffisante pour le propos tenu dans cet état de l'art. Le chapitre 5 en offre une présentation bien plus détaillée, car le DVFS est un élément central de l'approche sous-jacente à BDPO. En ce qui concerne l'outil présenté par [Li 2010], il a également recours au DVFS comme levier énergétique, auquel il adjoint le Dynamic Concurrency Throttling (DCT), qui consiste à faire varier le nombre de threads OpenMP exécutant une région parallèle. Enfin, Countdown (présenté par [Cesarini 2020]) reconfigure quant à lui soit les P-States soit les T-States des cœurs de calcul lorsque l'opportunité se présente, toujours dans le but d'améliorer l'efficacité énergétique associée à l'exécution de l'application en question.

Les méthodologies implémentées par ces trois outils s'appuient sur le constat suivant : lorsqu'une communication collective MPI (i.e. globale à tous les rangs appartenant au communicateur considéré) est bloquante, tous les rangs ne vont pas nécessairement entrer dans la barrière de synchronisation associée au même moment. Il apparaît même que la durée séparant les entrées du premier et du dernier rang peut être significative, et atteindre plusieurs secondes dans certaines conditions. En conséquence, les rangs les plus rapides vont vraisemblablement consommer de l'énergie à ne rien faire, si ce n'est attendre que les rangs les plus lents entrent dans la barrière de synchronisation [9]. L'idée est alors de ralentir les rangs les plus rapides via les leviers sélectionnés, pour diminuer autant que possible leur temps d'attente, sans pour autant altérer le **chemin critique** qui est associé au rang le plus rang. De la sorte, les performances sont littéralement préservées à l'identique, puisqu'imposées par le chemin critique, mais l'énergie consommée est réduite car les rangs ralentis consomment moins de puissance électrique.

Pour mettre en œuvre ce principe de fonctionnement, les trois outils implémentent deux méthodologies distinctes. D'une part, Countdown s'appuie sur un constat supplémentaire, résultant d'une analyse détaillée de l'exécution mono-nœud de l'application QuantumExpresso : les communications collectives MPI qui présentent une opportunité de reconfiguration ont une durée supérieure à 500 μs. Dès lors, l'approche implémentée par Countdown consiste à surcharger le module PMPI de MPI pour mettre en place un « timer » de 500 μs au début de chaque routine de communication collective. Si le timer expire, alors une adaptation des P-States (ou des T-States, en fonction du mode de fonctionnement de l'outil) est mise en œuvre. D'autre part, les approches de FoREST-mn et de l'outil présenté par [Li 2010] s'appuient sur une boucle de rétroaction. Via une annotation du code source de l'application et une surcharge du module PMPI de l'implémentation de MPI utilisée, il est possible de mesurer le temps nécessaire à chaque rang pour accomplir la « charge de travail » qui lui est assignée, une charge de travail étant définie comme une portion de l'exécution délimitée par deux communications collectives. Il est alors possible de déterminer le chemin critique de l'application, c'est-à-dire le rang le plus lent, et d'appliquer du DVFS pour ralentir tous les rangs plus rapides. Néanmoins,

9. Cette observation tient au fait que l'utilisation de C-States profonds (i.e. autre que C0) est généralement empêchée, via une attente active, par les implémentations de MPI, de sorte à ce qu'il n'y ait pas une pénalité de réveil des cœurs de calcul à la sortie de la barrière de synchronisation (cf. [Cesarini 2020]).

il n'existe pas nécessairement de P-State permettant de faire en sorte qu'un rang précis se synchronise avec le chemin critique. Et ralentir un rang plus que nécessaire s'avèrerait contre-productif, puisque cela changerait le chemin critique de l'application et dégraderait ses performances. D'où l'importance de la boucle de rétroaction, qui permet de mesurer l'impact du changement de P-State pour s'assurer qu'il ne ralentit pas trop le rang considéré, ou pour déterminer si un P-State plus profond serait encore plus favorable.

Pour finir, notons que les trois outils permettent d'améliorer l'efficacité énergétique associée à l'exécution d'une application HPC de manière significative lorsque cette dernière présente de nombreuses communications MPI collectives, avec une réduction de l'énergie consommée d'au moins 15% pour une augmentation du temps d'exécution sous la barre des 5%.

DVFS Governor - présenté par [Da Costa 2015] DVFS Governor est un gouverneur pour le module acpi-cpufreq du noyau Linux, spécifique au domaine du HPC. Pour rappel, un gouverneur implémente une politique de gestion des P-States des cœurs de calcul. Il met en place du DVFS dans le but d'améliorer l'efficacité énergétique associée à l'exécution d'une application.

Pour ce faire, DVFS Governor s'appuie sur le monitoring du taux d'utilisation de la bande-passante du lien Ethernet des nœuds de calcul qui exécutent l'application pour identifier les opportunités de reconfiguration. En quelques mots, Pierson et Da Costa modélisent l'exécution d'une application HPC comme une alternance de portions dominées intensivement par des calculs, et de portions dédiées aux communications inter-nœuds (typiquement à l'aide du protocole MPI). De plus, seules les parties dédiées au calcul sont considérées sensibles à la fréquence de fonctionnement des cœurs de calcul. Il faut ajouter à cela une caractérisation préalable des points de fonctionnement { tension ; fréquence } les plus élevé et profond pour déterminer les puissances moyennes leurs étant associées lorsque la bande passante du lien Ethernet est saturée. Les auteurs définissent alors, à partir de toutes ces valeurs, deux bornes sur le taux d'occupation du dit lien Ethernet. Du point de vue de l'efficacité énergétique associée à l'exécution d'une application HPC, il est alors indiqué de diminuer ou d'augmenter la fréquence des cœurs de calcul, lorsque le taux d'occupation du lien traverse les bornes en question.

Pour conclure, précisons que DVFS Governor a été évalué pour l'exécution sur quatre nœuds de calcul des NAS Parallel Benchmarks. Globalement, il améliore le ratio { performances / énergie } associé à l'exécution de l'application (environ 3% de diminution de l'énergie consommée pour un impact acceptable, voire négligeable sur les performances). De plus, seules les performances de EP sont déteriorées, et le gain énergétique le plus significatif, plus de 20%, est réalisé pour FT, qui s'avère intensément dominé par des communications inter-nœuds.

E-AMOM - présenté par [Lively 2014] E-AMOM, qui est présenté par [Lively 2014], est une méthodologie permettant de construire des modèles de performance et de consommation énergétique d'une application de calcul à haute performance, dans le but d'améliorer l'efficacité énergétique de cette dernière via la mise en œuvre de DVFS et de DCT.

Pour commencer, l'application considérée est exécutée plusieurs fois en faisant varier le nombre de threads utilisé, ainsi que le point de fonctionnement { tension ; fréquence } des cœurs de calcul, pendant qu'un ensemble d'évènements sont monitorés grâce aux

compteurs de performance des processeurs. À noter que les consommations électriques du nœud complet, et de ses processeurs et de sa mémoire principale pris séparemment, sont également monitorées pendant ces exécutions. Il faut également préciser que l'intégralité des mesures sont faites à l'échelle du noyau de calcul, grâce à l'annotation des codes source des applications considérées.

`E-AMOM` applique ensuite un ensemble de techniques et outils mathématiques (notamment une analyse en composants principaux) pour sélectionner les évènements les plus pertinents pour construire des modèles de la consommation électrique d'un nœud de calcul complet, ainsi que de ses processeurs et de sa mémoire principale pris séparemment, et un modèle du temps d'exécution de l'application.

Une fois les modèles construits, Lively et al. se servent de ces derniers pour identifier les points de fonctionnement et nombres de threads optimaux du point de vue de l'efficacité énergétique pour chaque noyau de calcul des applications considérées. En annotant une fois de plus le code source des applications en question, ils mettent alors en place `DVFS` et `DCT` à l'échelle du noyau de calcul. Pour être complet, il faut préciser que Lively et al. optimise également manuellement les boucles principales des applications considérées en mettant notamment en œuvre du `déroulage de boucle`[10].

Et les résultats obtenus pour un ensemble de benchmarks et d'applications de calcul à haute performance sont très bons : les modèles sont précis, avec une erreur moyenne inférieure à 7%, et à la fois le temps d'exécution et l'énergie consommée associée sont significativement réduits. Pour être précis, la réduction du temps d'exécution peut atteindre les 21%, et celle de l'énergie consommée les 12%.

`EAR` - présenté par [Corbalan 2020] `EAR`, qui est présenté par [Corbalan 2020], comprend plusieurs composants (5 majeurs), et se veut être une solution de gestion de l'énergie complète pour une infrastructure de calcul à haute performance. `EAR` propose ainsi aussi bien un suivi et une journalisation de la consommation énergétique des nœuds de calcul à l'échelle d'un supercalculateur, qu'un actionneur, nommé `EARL`, reconfigurant dynamiquement les points de fonctionnement { tension ; fréquence } des cœurs de calcul. Dans le cadre de cette section, c'est principalement sur l'actionneur qui vient d'être mentionné que nous allons nous concentrer.

`EARL` s'appuie sur le module `DynAIS`[11] de `EAR` qui permet de détecter le début d'une itération de n'importe (en adéquation avec la configuration de `DynAIS`) quelle boucle de l'application exécutée incluant au moins un appel `MPI`. `EARL` est également muni de modèles de performance permettant d'estimer l'impact d'un changement des points de fonctionnement des cœurs de calcul sur les performances de l'application exécutée, et la puissance consommée par ces derniers. Ces modèles prennent en entrée des métriques construites à partir d'évènements monitorés via les compteurs de performance des cœurs de calcul. De plus, ils sont spécifiques à une architecture matérielle de nœud de calcul, et construits à l'installation d'`EAR`, via l'exécution d'un ensemble de benchmarks sélectionnés pour être représentatifs d'une large gamme d'applications `HPC`. L'action d'`EARL` consiste alors à optimiser une fonction de coût, spécifiée via la politique qu'il doit mettre en œuvre,

10. Pour une définition sommaire, consulter : https://fr.wikipedia.org/wiki/D%C3%A9roulage_de_boucle.

11. Connexe à la thématique de détection d'un comportement périodique, le fonctionnement de `DynAIS` est présenté par la section 3.3 car cela est plus pertinent.

en appliquant du DVFS. Les reconfigurations en question s'opèrent à l'échelle de plusieurs itérations de la boucle considérée, et sont ajustées via une boucle de rétro-action.

À l'heure actuelle, EARL implémente deux politiques : (1) minimisation du temps d'exécution pour atteindre la solution, et (2) minimisation de l'énergie consommée pour atteindre la solution. Les résultats rapportés par [Corbalan 2020] montre que la mise en œuvre de la seconde politique par EARL permet de réduire l'énergie consommée par l'exécution de 9 benchmarks et applications HPC par entre 3% et 8%, pour une dégradation associée d'un peu plus de 4% en moyenne.

MREEF - présenté par [Chetsa 2015] MREEF est l'aboutissement des travaux de Chetsa et al., présenté par l'ensemble des articles suivants [Chetsa 2012, Chetsa 2013, Chetsa 2014a, Chetsa 2014b, Chetsa 2015]. Cet outil de reconfiguration dynamique s'articule autour de trois étapes. Pour commencer, il échantillonne à une fréquence de 1 Hz un ensemble de métriques caractéristiques de l'état d'utilisation du nœud de calcul (e.g. nombre d'instructions retirées, nombre d'octets envoyés via le réseau d'interconnexion, etc.). Les valeurs résultant d'un échantillonnage de la collection de métrique en question constituent un **Vecteur d'Exécution** (EV). La distance de Manhattan [12] entre deux EV consécutifs est calculée pour déterminer si le type de charge de travail [13] induite par l'exécution de l'application a changé. Un ensemble d'EV contigus suffisamment similaires pour être assimilables au même type de charge de travail est appelé une **phase**. La méthodologie associée à la détection des changements de phase est décrite en grande largeur par [Chetsa 2012]. Chaque phase, y compris celle en cours, est caractérisée par un $\texttt{EV}_{phase}$, dont les composantes sont les moyennes arithmétiques des composantes de l'ensemble des EV appartenant à ladite phase.

Avant de continuer, précisons que les auteurs ont élaboré une taxonomie des différents types de charge de travail qu'une application HPC peut typiquement exhiber. De plus, ils ont conçu des micro-benchmarks exhibant spécifiquement chacun un des types de charge de travail qui viennent d'être mentionnés. Cela leur a permis de déterminer expérimentalement quels points de fonctionnement sont les plus adaptés à chaque type de charge de travail du point de vue de l'efficacité énergétique, pour l'architecture matérielle des nœuds de calcul considérée. Chetsa et al. ont également construit un **Vecteur d'Exécution de Référence** ($\texttt{EV}_{ref}$) par type de charge de travail à partir des EV associés aux exécutions des micro-benchmarks en question.

La seconde étape consiste alors à déterminer à quel type de charge de travail l'$\texttt{EV}_{phase}$ actuel est assimilable, en le comparant aux $\texttt{EV}_{ref}$ construits préalablement. Pour ce faire, MREEF utilise un ensemble d'outils mathématiques (notamment une analyse en composants principaux, ou PCA), et implémente un ensemble de règles permettant de décider à quel type de charge de travail se rapporte l'$\texttt{EV}_{phase}$.

Enfin, la troisième et dernière étape consiste à reconfigurer, si nécessaire, les composants matériels du nœud de calcul pour que leurs points de fonctionnement correspondent à ceux identifiés préalablement comme optimaux pour la charge de travail considérée. Pour être complets, précisons que MREEF met en place deux types de reconfiguration : du DVFS, et le basculement vers le mode Low Power Idle (LPI) de la carte réseau. Notons

12. Voir la sous-section 2.3.1.

13. Par exemple, une charge travail peut être intensivement dominée par des calculs, ou bien constituée majoritairement d'accès à la mémoire principale.

également que cette étape s'appuie sur un principe de localité temporelle : MREEF fait l'hypothèse que le prochain EV appartiendra à la phase en cours.

Pour finir, il convient de préciser que [Chetsa 2015] présente un ensemble d'expérience montrant que MREEF a permis de rendre les exécutions des NAS Parallel Benchmarks plus efficaces du point de vue énergétique : jusqu'à 15% d'énergie consommée en moins, pour une dégradation des performances sous les 7%.

READEX - présenté par [Schuchart 2017] Le projet européen READEX a eu pour objet la création d'une collection d'outils pour accompagner les développeurs d'applications de calcul intensif dans l'amélioration de l'efficacité énergétique de ces dernières. L'article [Schuchart 2017] est le point d'entrée officiel des travaux menés dans le cadre de READEX, mais de multiples articles décrivent ses différents composants, à l'image de [Oleynik 2015, Kjeldsberg 2017, Sourouri 2017, Kumaraswamy 2018].

Le cœur de la méthodologie implémentée par READEX consiste en la détermination des régions d'intérêt au niveau du code source, et la construction de modèles d'optimisation pour ces dernières. Les régions d'intérêt sont les noyaux de calcul constituant une application HPC, dont READEX cherche à exploiter le **dynamisme** : c'est-à-dire le fait que la configuration optimale d'un ensemble de leviers pour une fonction de coût donné change au cours de l'exécution. La fonction du modèle d'optimisation associé à une région d'intérêt est de permettre d'estimer l'impact d'une configuration donnée sur la fonction de coût associée sans avoir à évaluer empiriquement chaque configuration. En effet, l'ensemble des configurations possibles croît exponentiellement avec le nombre de leviers et le nombre de points de fonctionnement qu'ils exposent. En conséquence, l'exploration exhaustive des configurations possibles demande généralement un temps prohibitif.

L'identification des régions d'intérêt peut soit être faite automatiquement, grâce à un composant de READEX implémentant une analyse statique du code source, soit par l'utilisateur de l'application. À noter que READEX propose également une interface d'annotation poussée permettant à l'utilisateur de fournir des informations supplémentaires concernant le code source, telles que le type de charge de travail associé à la région d'intérêt considérée.

Une fois les régions d'intérêt identifiées, elles sont instrumentées automatiquement pour ajouter le code nécessaire au monitoring de métriques d'intérêt via Score-P, un autre composant de READEX. L'étape d'instrumentation comporte également l'intégration des extraits de code permettant d'actionner les leviers utilisés par READEX au code source de l'application. Parmi ces leviers, on trouve notamment le DVFS et le DCT.

Cela rend possible d'auto-tuner l'application lors de son exécution, en essayant différentes configurations et en monitorant l'impact de ces dernières sur les métriques monitorées. Ces données de monitoring sont injectées dans Periscope, le composant en charge de la construction des modèles d'optimisation. À noter que si l'utilisateur a fourni des informations concernant les régions d'intérêt via l'interface d'annotation, le processus de construction en question peut être accéléré et conduire à un modèle plus pertinent.

Une fois les modèles d'optimisation mis au point, READEX détermine les configurations optimales du point de vue de la fonction de coût considérée, et les met en œuvre pendant l'exécution de l'application. Les modèles finaux sont par ailleurs stockés dans une « base de données » afin d'être réutilisés par la suite, et ainsi éviter le processus de construction. Pour finir, les auteurs ont démontré expérimentalement que READEX permet de diminuer

la consommation énergétique associée à l'exécution d'une application de manière significative (plus de 8% en moyenne, et jusqu'à plus de 20%), pour des dégradations de performance qu'ils jugent acceptables, mais qui peuvent tout même atteindre les 10% et sont rarement négligeables (i.e. inférieures à 1%).

3.3 Caractérisation des comportements localement périodiques dans les séries temporelles

Cette section passe en revue un ensemble de travaux de recherche traitant la thématique de la caractérisation des comportements périodiques exhibés par les séries temporelles. Cela permettra au lecteur d'avoir une connaissance de l'état de l'art lorsqu'il commencera la lecture de la partie III qui présente `Phase-TA`, un outil de détection et de caractérisation des comportements localement périodiques exhibés par les profils des applications `HPC`, qui constituent une famille de séries temporelles.

`Sketches` **- présenté par** [Indyk 2000] Indyk et al. proposent une méthodologie pour détecter une « tendance » (ici, un couple constitué d'une période et du motif qui se répète régulièrement) dans une série temporelle. Ils proposent également un outil théorique, à savoir les `sketches`, pour rendre ladite méthodologie efficace du point de vue de la complexité algorithmique.

L'idée principale est de découper la série temporelle en sous-séries temporelles de taille L. La valeur de L qui minimise la somme des distances entre la première sous-série temporelle et l'ensemble des sous-séries temporelles est la période associée au comportement périodique. La sous-série temporelle dont la somme des distances aux autres sous-séries temporelles est minimale est alors le motif représentatif associé au comportement périodique.

Cette approche est très coûteuse du point de vue de la complexité algorithmique, puisqu'il faut calculer l'ensemble des distances pour chaque paire de sous-séries temporelles, et ce, pour chaque valeur possible de L. Les auteurs ont donc développé la notion de `sketch` pour compenser en partie cette importante complexité algorithmique. En quelques mots, à chaque sous-série est associé un `sketch`, qui est un vecteur de taille $k \ll L$. Les composantes du `sketch` sont les résultats des produits scalaires entre la sous-série considérée et un ensemble de k vecteurs aléatoires générés selon une distribution normale. Une telle construction permet alors de construire un encadrement **probabiliste** de la distance entre deux `sketches`, qui se calcule en un temps fixe. En substituant les sketches aux sous-séries temporelles, et en considérant les encadrements probabilistes en lieu et place des distances, il est possible de réduire la complexité algorithmique de l'approche.

`Periodicity detection in time series databases` **- présenté par** [Elfeky 2005] Elfeky et al. présentent une méthodologie permettant de détecter les comportements localement périodiques d'une série temporelle, et d'extraire les périodes associées. Leur approche nécessite de discrétiser la série temporelle vers un ensemble de symboles. Une fois discrétisée, la série temporelle est décalée de p symboles vers la droite, puis la distance de Hamming[14] entre les séries temporelles initiale et décalée est calculée. Si cette distance est inférieure à un seuil fixé par l'utilisateur, la série temporelle est considérée p-périodique.

Afin d'accélérer le calcul des distances de Hamming pour un balayage complet des valeurs possibles pour p, les auteurs utilisent un produit de convolution calculé via une *Transformée de Fourier Rapide* (fréquemment désignée par l'acronyme `FFT`), et une table

14. Cf. https://fr.wikipedia.org/wiki/Distance_de_Hamming.

de valeurs particulières pour les symboles (ces derniers doivent être des puissances de 2). En effet, la somme des distances de Hamming pour l'ensemble des valeurs possibles de p est égale à un produit de convolution, dont la complexité algorithmique dans le domaine fréquentiel est moindre. Néanmoins, pour pouvoir extraire du produit de convolution la valeur de la distance pour un décalage particulier, il est nécessaire que les symboles soient égaux à des puissances de 2. Imposer la table de valeurs particulières mentionnée précédemment permet donc de garantir que l'accélération de la méthodologie par l'application d'une `FFT` est possible.

`STNR` - présenté par [Rasheed 2010] Rasheed et al. présentent `STNR`, une méthodologie de détection des périodicités exhibées par une série temporelle discrète, résistant, dans une certaine mesure, à l'insertion et à la suppression de symboles.

La première étape de la méthodologie, potentiellement facultative, consiste à discrétiser les séries temporelles à valeurs réelles. Ensuite, il faut construire l'arbre à suffixes associé à la série temporelle. On dénombre les occurrences de chaque sous-chaîne de symboles incluse dans la série temporelle, en mémorisant les positions des premiers symboles des dites occurrences dans un « vecteur d'occurrence » (il y a donc un vecteur par sous-chaîne). Soit $n \in \mathbb{N}^*$. On note le vecteur d'occurrence $\mathcal{O}(o_1, o_2, \cdots, o_n)$, où $o_{i,i\in[\![1;n]\!]}$ est la position du premier symbole de la i-ème occurrence de la sous-chaîne considérée. Une fois ce vecteur construit, on construit le « vecteur des différences » en soustrayant chaque paire d'éléments consécutifs du vecteur d'occurrence. On note ce vecteur $\mathcal{D}(d_1, d_2, \cdots, d_{n-1})$, où $d_{i,i\in[\![1;n-1]\!]} = o_{i+1} - o_i$. Alors, tout couple $p_{c,i} = \{o_i; d_i\}_{i\in[\![1;n-1]\!]}$ est une périodicité candidate. En d'autres termes, il faut déterminer si la sous-chaîne considérée survient avec une période d_i à partir de la position o_i.

Pour déterminer si une périodicité candidate s'avère être une périodicité existante, les auteurs définissent $nov(p_{c,i})$, à savoir le « nombre d'occurrences vérifiées » de la périodicité candidate $p_{c,i}$ de longueur $l \in \mathbb{N}^*$. On initialise alors $nov(p_{c,i})$ à 0. On définit également $t \in \mathbb{N}$, le « seuil de dilatation temporelle », en d'autres termes le nombre de suppressions et d'insertions de symboles que l'on souhaite pouvoir compenser au maximum. Pour chaque $k \in \mathbb{N}$ tel qu'une sous-chaîne de longueur l commençant en position $o_i + k \times d_i \pm t$ existe, on compare ladite sous-chaîne à celle considérée. Si elles sont identiques, $nov(p_{c,i})$ est incrémenté de 1. À noter que les valeurs de k sont balayées par ordre croissant, et qu'à chaque fois qu'une correspondance avec la sous-chaîne considérée est trouvée, la périodicité candidate $p_{c,i}$ est mise à jour : (1) la position à partir de laquelle les potentielles occurrences suivantes sont recherchées est la position du premier symbole de la dernière occurrence vérifiée, et (2) la période candidate est la distance moyenne entre deux occurrences vérifiées.

Enfin, les auteurs définissent la « force périodique » d'une périodicité candidate $p_{c,i}$, notée τ, comme la différence entre $nov(p_{c,i})$ et le nombre maximal possible d'occurrences vérifiées. Les périodicités candidates exhibant une force périodique supérieure à un seuil préalablement fixé sont considérées comme des périodicités avérées.

`AUTOPERIOD` - présenté par [Vlachos 2005] Vlachos et al. décrivent `AUTOPERIOD`, un outil utilisant la dualité entre les domaines temporel et fréquentiel pour identifier les périodes (et donc les fréquences) significatives pour une série temporelle. `AUTOPERIOD` calcule tout d'abord le périodogramme de la série temporelle considérée, c'est-à-dire l'évolution de

la densité spectrale de puissance en fonction de la fréquence associée à une potentielle périodicité, à l'aide d'une *Transformée de Fourier Discrète* (généralement abrégé par l'acronyme `DFT`). En identifiant les fréquences pour lesquelles la densité spectrale de puissance est élevée, les auteurs isolent les fréquences, et donc les périodicités, candidates. En effet, candidates seulement car une densité spectrale de puissance élevée peut résulter de la superposition de plusieurs densités spectrales d'intensités moindres dans un intervalle de fréquence restreint.

Par la suite, `AUTOPERIOD` calcule l'autocorrélation circulaire de la série temporelle considérée pour déterminer si les périodicités candidates sont significatives. En quelques mots, si une fréquence identifiée comme candidate grâce au périodogramme correspond à un maximum local de l'autocorrélation circulaire, ou se situe dans son voisinage, et que ce maximum local est supérieur à un certain seuil fixé préalablement par les auteurs, alors la périodicité candidate est jugée significative. Ce qui signifie qu'un comportement périodique a été détecté par `AUTOPERIOD`.

`EAR` - présenté par [Corbalan 2020] `EAR`, présenté par Corbalan et al., est une solution de gestion énergétique à l'échelle d'une infrastructure de calcul à haute performance qui comprend plusieurs composants, dont 5 principaux. Le composant qui nous intéresse se nomme `DynAIS`.

Brièvement, `DynAIS` implémente une version de `Dynamic Periodicity Detector` (présenté par [Freitag 2001]) adaptée aux séries temporelles discrètes pour détecter le fait qu'une application de calcul à haute performance exécute une de ses boucles principales. Pour ce faire, `EAR` contient une implémentation du module `PMPI` de l'environnement de programmation `MPI`, qui vient surcharger celle utilisée par défaut lors du lancement de l'application. Cette implémentation de `PMPI` permet de tracer les appels aux routines `MPI`, qui sont des symboles, afin de construire la série temporelle discrète des symboles `MPI` appelés.

La méthodologie dérivée de `Dynamic Periodicity Detector` permet ensuite de déterminer si cette série temporelle exhibe des périodicités. Le cas échéant, elles sont vraisemblablement associées aux boucles principales de l'application. À noter que `EAR` agrège ensuite des relevés de compteurs de performance à l'échelle d'une itération d'une boucle principale, afin de définir une signature de l'application.

Pour résumer, le module `DynAIS` de `EAR` permet de détecter des comportements localement périodiques dans les séries temporelles **discrètes**.

À titre de rappel, la sous-section 8.5.3 propose, en guise de conclusion à la partie dédiée à la présentation de `Phase-TA`, une comparaison de ce dernier aux outils qui viennent d'être présentés par cette section.

Mise en place d'un environnement expérimental

4

Ce chapitre aborde le contexte expérimental des travaux présentés par ce manuscrit. Pour ce faire, la section 4.1 regroupe et détaille les caractéristiques techniques des plateformes matérielles utilisées pour mener les expérimentations, et la section 4.2 présente brièvement le panel d'applications de calcul à haute performance qui ont été exécutées.

4.1 Plateformes expérimentales

Dans un premier temps, la sous-section 4.1.1 spécifie l'architecture des nœuds de calcul mis à disposition par Atos, utilisés dans le cadre des travaux au cœur du chapitre 6. Puis, dans un second et dernier temps, la sous-section 4.1.2 présente le cluster `dahu`, mis à disposition par `Grid'5000`, et qui a été utilisé pour réaliser les expérimentations présentées par le chapitre 8.

Comme mentionné par la section 1.2, une « image expérimentale » a été mise en place : il s'agit d'une image d'un système d'exploitation agrémentée de toutes les dépendances logicielles et configurations requises pour mettre en place un environnement expérimental performant, stable, reproductible, et identique pour les plateformes matérielles. L'image en question est construite autour d'une version « vanilla » du système d'exploitation `RedHat RHEL` car il fait office de standard industriel dans le domaine du calcul à haute performance, et qu'il équipe les supercalculateurs conçus par Atos. Concernant les logiciels encapsulés dans l'image, il s'agit des paquets nécessaires au support des composants matériels des nœuds de calcul utilisés (e.g. `drivers` Intel® `Omni-Path`), des chaînes de compilation (e.g. `GCC`), des environnements de programmation (e.g. `OpenMPI`), et des applications `HPC` utilisées (e.g. `NEMO`). Pour finir, précisons que bien que la création et les évolutions de cette image expérimentale aient représenté une charge de travail significative, elles ne seront pas abordées dans de plus amples détails par ce manuscrit.

4.1.1 Plateforme `Bull`

Les spécifications de la partition de nœuds de calcul mise à disposition par Atos pour la réalisation de travaux expérimentaux dans le cadre de la thèse sont détaillées par

le tableau 4.1. Le système de fichiers adjoint est un « scratch » NFS offrant 500 Go de stockage, exposés via un réseau de management Ethernet 10 Gb/s.

Le supercalculateur SuperMUC-NG a occupé les $8^{ème}$ puis $9^{ème}$ places du Top500 entre Juin 2018 et Novembre 2019. Et sa configuration est similaire à celle des nœuds dont la configuration est présentée par cette sous-section, traduisant le fait que cette dernière est typique du HPC.

TABLE 4.1 – Caractéristiques techniques des nœuds de calcul mis à disposition par Atos.

Modèle de processeur	Intel Xeon Platinum 8176
Nombre de processeurs	2
Nombre de cœurs	56 (2×28)
Type de mémoire principale	DDR4@2.40 GHz
Taille de la mémoire principale	128 GB(4×32 GB)
Réseau d'interconnexion rapide	Mellanox® Infiniband FDR 56 Gb/s
Système d'exploitation	RedHat RHEL 7.4

4.1.2 Plateforme Grid'5000

Le tableau 4.2 rassemble les principales caractéristiques techniques des nœuds de calcul du cluster dahu. Une partition de 16 nœuds de calcul homogènes a été utilisée, auxquels 3 nœuds supplémentaires exposant un système de fichiers Lustre dédié ont été adjoints. Ce système de fichiers offre 1 To de stockage sur support SSD(520 Mb/s en lecture, et 480 Mb/s en écriture), compte 1 MDS et 2 OSS, et est accessible via le réseau d'interconnexion rapide.

TABLE 4.2 – Caractéristiques techniques du cluster dahu mis à disposition par Grid'5000.

Modèle de processeur	Intel Xeon Gold 6130
Nombre de processeurs	2
Nombre de cœurs	32 (2×16)
Type de mémoire principale	DDR4@2.67 GHz
Taille de la mémoire principale	192 GB(6×32 GB)
Réseau d'interconnexion rapide	Intel® Omni-Path 100 Gb/s
Système d'exploitation	RedHat RHEL 7.4

L'architecture matérielle des nœuds de calcul de dahu est typique du domaine du calcul à haute performance, comme en témoigne le fait que des supercalculateurs actuellement classés aux alentours de la $50^{ème}$ position du Top500 aient des spécifications similaires.

Pour finir, précisons que Grid'5000[1] est une plateforme matérielle flexible et de

1. https://www.grid5000.fr

grande échelle. Elle est dédiée aux travaux de recherche expérimentaux des sciences informatiques, et présente un intérêt tout particulier pour les domaines du calcul parallèle et distribué qui incluent notamment les thématiques du calcul à haute performance, du « Cloud », du « Big Data », et de l'Intelligence Artificielle.

4.2 Panel d'applications de calcul intensif

L'énumération suivante regroupe de brèves présentations de l'intégralité des applications, benchmarks, et noyaux de calcul du domaine du calcul à haute performance exécutés dans le cadre des expérimentations menées au cours de la thèse :

[Basic Linear Algebra Subprograms (BLAS)] Bibliothèque de fonctions standardisée implémentant un ensemble d'opérations d'algèbre linéaire, telles que la multiplication d'un vecteur par un scalaire, ou la multiplication matricielle (cf. http://www.netlib.org/blas/). Ces fonctions constituent des briques de base, optimisées, qui sont ensuite utilisées par les développeurs d'applications au sein des noyaux de calcul de ces dernières.
Les BLAS sont organisés en familles :

[Dense BLAS-1] Regroupe les opérations dont les opérandes sont vecteurs pleins (e.g. produit scalaire) ;

[Dense BLAS-2] Regroupe les opérations dont les opérandes sont mixtes (e.g. multiplication d'une matrice par un vecteur), pour des matrices et vecteurs pleins ;

[Dense BLAS-3] Regroupe les opérations dont les opérandes sont des matrices pleines (e.g. multiplication matricielle).

Les familles Sparse BLAS-1, Sparse BLAS-2, et Sparse BLAS-3 sont les pendants des trois familles qui viennent d'être présentées, pour des matrices et vecteurs creux.

[High Performance Conjugate Gradients (HPCG)] Benchmark conçu comme un complément du High Performance Linpack (HPL) pour évaluer les performances des supercalculateurs pour une autre gamme de problèmes, à savoir les opérations vectorielles et matricielles pour des opérandes creux, qui ne permettent généralement pas d'atteindre les performances maximales offertes par les unités de calcul (cf. https://www.hpcg-benchmark.org/).
Pour ce faire, HPCG résout un système d'équations linéaires en couplant l'utilisation de la méthode du gradient conjugué à une approche multigrille, après un pré-conditionnement par application de la méthode de Gauss-Seidel. La charge de travail associée induit une grande quantité d'accès à la mémoire principale et de communications.

[NAS Parallel Benchmarks (NPB)] L'acronyme NPB désigne un ensemble de benchmarks développés par la NASA dans le but d'évaluer la puissance de calcul d'un système HPC pour différents types de charge de travail (cf. https://www.nas.nasa.gov/software/npb.html).
Chaque membre des NPB est accompagné d'un ensemble de jeux de données d'entrée, proposant des charges de travail d'ordres de grandeur variés (on parle de « classes »).
Dans le cadre de la thèse, les benchmarks suivants ont été utilisés :

[CG] Construit une estimation de la plus petite valeur propre d'une matrice creuse définie positive en s'appuyant notamment sur la méthode du gradient conjugué

pour résoudre un système d'équations linéaires.
La charge de travail induite comporte des communications MPI point-à-point et des accès mémoire irréguliers ;

[MG] Approxime la solution d'une équation de Poisson discrète tridimensionnelle en utilisant une approche multigrille exécutée selon un cycle en V.
La charge de travail induite génère une forte pression sur la bande passante entre les processeurs et la mémoire principale ;

[EP] Génère des réalisations aléatoires selon une loi normale centrée réduite par application de la méthode de Box-Muller.
Exerce une forte pression sur les unités de calcul, et passe très bien à l'échelle ;

[FT] Résolution d'un système d'équations différentielles partielles tridimensionnel en utilisant une transformée de Fourier rapide.
La charge de travail associée implique notamment de nombreuses communications MPI globales ;

[BT-MZ] Résout un système d'équations non-linéaires en utilisant un solveur tridiagonal par bloc.
Plus qu'un benchmark, c'est une micro-application qui n'exerce donc pas un type spécifique de charge de travail, mais exhibe plutôt un comportement approchant celui d'une application réelle.
La version de BT considérée est la version MultiZone, conçue pour être exécutée selon un modèle hybride MPI+OpenMP ;

[LU-MZ] Similaire à BT-MZ en appliquant des relaxations successives selon la méthode de Gauss-Seidel ;

[SP-MZ] Similaire à BT-MZ en appliquant un solveur scalaire pentadiagonal.

[NEKbone] Proxy de l'application NEK5000 qui extrait uniquement son noyau de calcul principal afin de pouvoir l'étudier en isolation (cf. https://github.com/masmithanl/nekbone).
NEKbone résout une équation de Poisson tridimensionnelle par la méthode des éléments spectraux, en s'appuyant sur la recette numérique du gradient conjugué.
La charge de travail induite comprend notamment des phases de calcul intensives associées à des multiplications matricielles, ainsi que des communication MPI locales et globales ;

[NAnoscale Molecular Dynamics (NAMD)] Simule numériquement la dynamique moléculaire de larges systèmes biomoléculaires, tels que la solvatation du virtovirus du tabac 1 dans le cas d'utilisation STMV (cf. http://www.ks.uiuc.edu/Research/namd/).
NAMD est une application complexe, implémentée via l'environnement de programmation parallèle Charm++, qui intègre notamment un équilibreur de charges de travail entre les différentes unités de calcul.
La charge de travail induite par l'application est constituée de plusieurs phases, alternant calculs intensifs, et communications locales et globales, le tout ponctué de nombreux accès mémoires ;

[Nucleus for European Modelling of the Ocean (NEMO)] Cadriciel dédié à la simulation et l'analyse des phénomènes climato-océaniques, tels que l'évolution de la salinité des océans (cf. https://www.nemo-ocean.eu/).
NEMO compte plusieurs composants, dont TOP/PISCES qui modélise le transport de sédiments par la dynamique océanique, ainsi que les processus géologiques et biochimiques associés. Le cas d'utilisation considéré est GYRE, qui étude l'évolution d'un double vortex dans un pavé délimité, au fil du cycle saisonnier.
À l'instar de ce qui a été écrit pour NAMD, la charge de travail induite comprend des phases de calcul intensives, ponctuées de nombreux accès mémoire et de communications locales ;

[OpenFOAM] Boîte à outils pour la simulation numérique dans le domaine de la mécanique des fluides, regroupant notamment un ensemble de solveurs s'appuyant sur la méthode des volumes finis, et d'outils de pré- et post-traitement (cf. https://www.openfoam.com/).
Dans le cadre de la thèse, deux solveurs ont été utilisés :

[simpleFOAM] Simulateur adapté aux écoulements turbulents en régime permanent de fluides incompressibles. Il a été appliqué au cas d'utilisation motorBike qui modélise les flux d'air s'appliquant à une moto et de son pilote ;

[icoFOAM] Simulateur adapté aux écoulements laminaires en régime transitoire de fluides newtoniens. Il a été appliqué au cas d'utilisation cavity qui modélise les flux de liquides dans une cavité engendrés par le déplacement de la partie supérieure de l'enveloppe de cette dernière.

Les implémentations de HPCG et des BLAS utilisées dans le cadre des travaux autour de BDPO présentés par le chapitre 6 sont celles optimisées pour les processeurs Intel disposant d'unités vectorielles AVX512. Ces dernières sont livrées par Intel Compiler and Libraries v2018.1.163, et ont été exécutées avec les implémentations associées de IntelMPI et libiomp. Dans la même veine, l'implémentation de HPCG exécutée dans le cadre des travaux expérimentaux associés à Phase-TA (cf. chapitre 8) est celle optimisée pour AVX512 livrée par Intel Compiler and Libraries v2019.3.199, également avec les implémentations associées de IntelMPI et libiomp.

Toutes les autres applications, exécutées aussi bien dans le cadre du chapitre 6 que du chapitre 8, ont été compilées avec GCC 10.1.0, la version de libgomp livrée avec ce dernier, et OpenMPI 4.0.4.

Deuxième partie

BDPO : optimiser la puissance consommée par l'exécution des applications de calcul intensif

Description de BDPO

5

Ce chapitre présente BDPO, en commençant par préciser son cahier des charges dans la section 5.1. Puis, la section 5.2 détaillera le principe du fonctionnement de BDPO, en s'attardant sur les différents concepts sous-jacents. Enfin, la section 5.3 concluera en présentant l'architecture de BDPO, ainsi que les points clés de son implémentation.

Ce chapitre fait appel a plusieurs notions associées à l'architecture d'un nœud de calcul, et plus particulièrement à la micro-architecture des processeurs. Les notions en question ont été introduites par la section 2.1.

5.1 Quel cahier des charges pour BDPO ?

Comme la section 1.1 l'a longuement détaillé, rendre les supercalculateurs plus efficaces du point de vue énergétique s'avèrera fondamental pour entrer dans l'ère de l'ExaScale, et demeurera un élément, si ce n'est l'élément, clé de l'avenir du domaine du calcul intensif. Dans ce contexte, la principale motivation au développement de `Bull Dynamic Power Optimizer` (BDPO) est de participer à l'amélioration de l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance.

Pour ce faire, BDPO entend améliorer l'efficacité avec laquelle l'application exécutée utilise les ressources mises à disposition par les composants matériels du supercalculateur. En effet, ces derniers proposent plusieurs points de fonctionnement, exhibant différents ratios { performances / énergie }. BDPO se définit alors comme un **outil de reconfiguration dynamique** dont le but est d'offrir aux utilisateurs finaux des supercalculateurs la possibilité de déléguer l'adaptation de la configuration des nœuds de calcul à la charge de travail induite par l'exécution d'une application. L'adjectif « dynamique » renvoie ici au fait que les reconfigurations ont lieu pendant l'exécution de l'application. L'effet recherché est d'améliorer, du point de vue énergétique, l'efficacité avec laquelle les composants matériels des nœuds de calcul sont utilisés par l'application.

Adapter les points de fonctionnement des composants matériels d'un nœud de calcul à la charge de travail que l'application qu'il exécute induit implique tacitement la notion de compromis entre performances et énergie consommée. La manière d'approcher ce compromis a un impact fort sur la conception de l'outil de reconfiguration dynamique associé, comme l'étude de l'état de l'art proposée par la section 3.2 le démontre.

Par exemple, les auteurs de MREEF (présenté par [Chetsa 2015]) jugent qu'il est acceptable de dégrader les performances de l'application tant que : (1) l'action de MREEF permet d'améliorer l'efficacité énergétique associée à l'exécution de l'application, et (2) l'augmentation du temps d'exécution induite par l'action de MREEF est inférieure ou égale à 10%, comparativement à une exécution de référence[1]. Cela permet d'envisager des actions de reconfiguration relativement « agressives », ainsi qu'une algorithmique interne complexe pour l'outil de reconfiguration puisque le seuil de tolérance sur la dégradation des performances de l'application exécutée lui permet d'avoir un impact sur ces dernières. D'autres outils, à l'image de FoREST-mn (présenté par [Halimi 2014]), rendent le seuil de dégradation de performances à ne pas dépasser configurable par l'utilisateur.

Le positionnement de BDPO en ce qui concerne ce compromis entre performances et énergie consommée est de **minimiser autant que possible** la dégradation des performances induite par son action, quitte à ne pas drastiquement réduire la consommation énergétique. Le raisonnement sous-jacent compte deux arguments principaux :

— L'essence du calcul à haute performance est d'offrir le plus haut niveau de performance possible, en témoigne notamment le Top500. L'utilisateur final d'un supercalculateur souhaite que l'exécution de son application prenne le moins de temps possible. Ainsi, dégrader les performances d'un système HPC, en plus d'être « contre-nature », serait un frein notable à l'adoption de BDPO par la communauté du calcul intensif ;

— Toute diminution de l'énergie consommée pour exécuter une application HPC est **significative**, qu'elle soit d'un ou de dix pourcents. Et, en référence au premier point, elle est d'autant plus significative si elle n'induit aucune dégradation de performance.

Cela implique notamment que BDPO doit être **léger** (i.e. que la complexité algorithmique associée à son fonctionnement doit être limitée), et qu'il doit mettre en place des actions de reconfiguration avec **parcimonie**.

De plus, toujours dans l'optique de favoriser son adoption par la communauté HPC, BDPO doit être **agnostique** de l'application exécutée : son fonctionnement ne doit nécessiter aucune connaissance préalable de l'application exécutée, ni une quelconque modification de son code source.

1. C'est-à-dire pour les mêmes conditions expérimentales (i.e. configuration, jeu de données, etc.), mais sans que MREEF ne soit exécuté en parallèle de l'application.

5.2 Principe de fonctionnement de BDPO

Cette section détaille les trois principales composantes de l'approche sous-jacente à BDPO, avant de les rassembler pour définir le principe de fonctionnement de BDPO, tout en l'illustrant via un exemple concret.

Cibler les cœurs de calcul Comme cela a été précisé par la section 5.1, BDPO est un outil de reconfiguration dynamique. Son but est d'adapter le point de fonctionnement des composants matériels d'un nœud de calcul à la charge de travail induite par l'application exécutée, dans l'optique d'améliorer l'efficacité énergétique de l'exécution en question. Se pose alors la question suivante : quels composants reconfigurer ?

Aux prémices de la conception de BDPO, les cœurs de calcul des processeurs se sont avérés être des cibles de choix pour deux raisons principales :

— Les processeurs « étaient »[2] responsables de la part la plus importante (la proportion communément admise se situe entre 35 et 50%, cf. [Liu 2010]) de la consommation énergétique des nœuds de calcul ;

— Comme présenté par la sous-section 2.1.1.2, les processeurs exposent de nombreuses fonctionnalités liées à la gestion de l'énergie, et notamment les P-States qui permettent de faire varier dynamiquement le point de fonctionnement { tension ; fréquence }, afin de modifier le ratio { performances / énergie } du cœur de calcul associé.

Focus sur le Dynamic Voltage Frequency Scaling (DVFS), qui est le nom de la technique qui consiste à modifier le P-State des cœurs de calcul au cours de l'exécution d'une application. Comme mentionné précédemment, un P-State est un point de fonctionnement { tension ; fréquence } pour un cœur de calcul. Chaque P-State exhibe un compromis différent entre puissance consommée et performances maximales atteignables. Dans la configuration standard d'un supercalculateur, les cœurs de calcul fonctionnent constamment avec le P-State associé à la fréquence nominale du processeur, le surcadençage étant autorisé/activé (par exemple via TurboBoost pour les processeurs Intel®). Les performances maximales atteignables sont alors les performances maximales théoriques du processeur, et la puissance électrique consommée est la plus élevée possible, puissance pour laquelle la TDP du processeur a été calculée.

Or, une application de calcul à haute performance n'exploite qu'occasionnellement la pleine puissance de calcul mise à disposition par les cœurs de calcul. Cette observation est notamment illustrée par [Ao 2018] : malgré le fait qu'une équipe de chercheurs se soit attelée à l'optimisation de HPCG pour qu'elle exploite au mieux la puissance de calcul mise à disposition par le supercalculateur Sunway TaihuLight, l'application n'a atteint qu'un débit d'instructions d'environ 480 $TFlop/s$, lorsque le supercalculateur chinois possède un débit d'instructions supérieur à 100 $PFlop/s$ (démontré par la pratique).

L'idée sous-jacente au DVFS est alors d'appliquer un P-State plus profond lors des

2. À l'heure actuelle, avec le clair virage vers les architectures hybrides CPU+GPGPU, ce sont ces derniers qui dominent la consommation énergétique des nœuds de calcul. C'est pourquoi l'adaptation de l'approche de BDPO aux GPGPU est actuellement à l'étude au sein de l'équipe PowerEfficiency d'Atos.
Pour les nœuds n'étant pas équipés d'accélérateur, les processeurs demeurent toujours responsables de la part la plus importante de la consommation énergétique.

portions de l'application qui ne nécessitent pas la puissance de calcul maximale offerte par les cœurs de calcul, puis de revenir au `P-State` nominal pour les portions de l'application intensivement dominées par des calculs. Si le `P-State` plus profond ciblé permet de répondre aux besoins en puissance de calcul de l'application pendant les portions concernées, l'impact sur les performances des changements de points de fonctionnement { tension ; fréquence } est limité voire négligeable. Alors que la puissance électrique consommée pendant lesdites portions diminue significativement, ce qui se traduit globalement par une diminution de l'énergie consommée par l'exécution de l'application. Le ratio { performances / énergie } est alors vraisemblablement impacté dans le sens d'une amélioration de l'efficacité énergétique associée à l'exécution de l'application.

Dernier point à aborder concernant le `DVFS` : le changement de `P-State` n'est pas « gratuit ». En effet, comme démontré dans [Mazouz 2013] pour les architectures Intel® `Westmere`, `Sandy Bridge`, et `Ivy Bridge`, puis par [Hackenberg 2015, Schöne 2019] pour les architectures Intel® `Haswell` et `Skylake`, un changement de point de fonctionnement { tension ; fréquence } n'est pas immédiat et a un impact sur les performances. Dans le cas de l'architecture `Skylake` par exemple, un changement de `P-State` : (1) ne peut intervenir qu'une fois toutes les 500 μs environ, (2) nécessite un délai de l'ordre de la quinzaine de microsecondes pour être effectif car le changement de tension d'alimentation n'est pas immédiat, et (3) implique que le `pipeline` du cœur de calcul concerné soit purgé, induisant un surcoût en termes de performance. Le `DVFS` est donc une technique à utiliser avec parcimonie, sous peine de perturber significativement l'exécution de l'application considérée.

On peut donc retenir que `BDPO` est un outil de reconfiguration dynamique local au nœud de calcul, qui met en place du `DVFS` pour améliorer l'efficacité énergétique associée à l'exécution d'une application `HPC`.

Exploiter les mouvements de données Maintenant que nous avons établi que `BDPO` mettra en place du `DVFS` pour tirer profit des portions de l'application exécutée qui n'utilisent pas la totalité de la puissance de calcul mise à disposition par les cœurs de calcul, il reste à identifier de telles portions.

Pour ce faire, partons d'une observation faite par Spiliopoulos et al. dans [Spiliopoulos 2011], à savoir : lorsque l'accès à une donnée provoque un défaut de cache en `L3`, le temps nécessaire à ce que la ligne de cache associée soit rapatriée depuis la mémoire principale vers l'un des registres du cœur de calcul considéré correspond à plusieurs dizaines, voire centaines, de cycles du processeur. Lorsque les dépendances entre les instructions à exécuter et la donnée chargée via ce défaut de cache empêchent l'ordonnanceur d'envoyer de nouvelles `micro-opérations` vers les ports du cœur de calcul concerné, cela conduit à l'injection de `cycles d'attente` dans le `pipeline`[3] du cœur de calcul. Pendant ces `cycles d'attente`, la partie concernée du `pipeline` ne fait « rien », au sens où elle n'exécute pas de `micro-opération`, mais elle reste active et consomme donc la même quantité d'énergie que si elle exécutait des `micro-opérations` « effectives ». Et la durée d'attente est complètement indépendante du point de

3. Précisons que les latences d'accès à la mémoire ne sont pas les seules sources d'injection de `cycles d'attente`. Le manque temporaire de `micro-opérations` à exécuter en raison de latences des unités de décodage des instructions peut également engendrer des `cycles d'attente`, par exemple.

fonctionnement { tension ; fréquence } du cœur de calcul[4]. Les moments où le `pipeline` d'un cœur de calcul est abondamment rempli de `cycles d'attente`, et donc les phases d'accès intensifs à la mémoire principale, sont ainsi particulièrement propices à l'application de `DVFS`.

La conclusion précédente fait intervenir la notion de **phase**, tâchons donc de la définir. Toute portion contigüe de l'exécution d'une application pendant laquelle elle exhibe un comportement homogène est appelée une phase. Par exemple, lorsque la portion contigüe en question exhibe un nombre d'instructions retirées par cycle élevé et stable, elle sera qualifiée de phase de calcul intensives. Il est alors légitime de se demander si les phases d'accès intensifs à la mémoire principale, que nous abrègerons en « phases mémoires », sont suffisamment fréquentes pour être ciblées par `BDPO`.

La lecture de [Ao 2018] apporte un premier élément de réponse à cette question. Comme mentionné précédemment, le travail d'une équipe de chercheurs autour de `HPCG` n'a permis d'atteindre qu'un débit d'instructions retirées aux alentours des 480 $TFlop/s$ sur le `Sunway TaihuLight`, qui propose pourtant une puissance de calcul maximale dépassant les 100 $PFlop/s$. Ce qui explique cette observation est le fait que `HPCG` est largement dominée par des phases mémoires intensives, induisant un nombre important de `cycles d'attente`, ce qui limite le niveau de performance atteignable par l'exécution de l'application sur le supercalculateur chinois bien en deçà de son débit d'instructions retirées maximal atteignable.

Le fait que le sous-système associé à la mémoire principale[5] agisse comme un goulot d'étranglement pour les performances est **intrinsèque** au noyau de calcul de `HPCG`, dont l'intensité opérationnelle[6] est très faible : chaque opération en virgule flottante dépend d'un nombre élevé d'octets de données qu'il faut déplacer de la mémoire principale vers les registres du processeur.

Intéressons-nous aux simulations présentées par [Ibeid 2018], dont la modélisation `Roofline` présentée par la figure 5.1 est extraite. Ces dernières indiquent que plusieurs noyaux de calcul parmi les plus courants dans le monde du calcul intensif induisent, et induiront au moins pour les premières années de l'ère ExaScale, un étranglement par le système mémoire des performances des applications associées. Ce qui se traduira vraisemblablement par la présence de nombreuses phases mémoires lors de l'exécution d'une large gamme d'applications, et donc par la présence d'opportunités de mise en place de `DVFS` à des fins d'optimisation de l'efficacité énergétique associée aux dites exécutions.

Le noyau de calcul `SpMV`, exhibant la plus faible intensité opérationnelle sur la figure 5.1, consiste en la multiplication d'une matrice creuse par un vecteur, et le noyau de calcul de `HPCG` fait intensivement appel à ce dernier. La faible intensité opérationnelle associée à `SpMV` explique que les performances atteignables lors de l'exécution de `HPCG` sur le `Sunway TaihuLight` soient entre 2 et 3 ordres de grandeurs inférieures aux performances maximales offertes par le supercalculateur chinois, et que le travail d'optimisation de Ao et al. dans [Ao 2018] ait été centré autour du système mémoire, de sorte à maximiser

4. Elle dépend notamment de la mémoire principale, du bus, et de la fréquence du `Package`.
5. Par la suite désigné comme le **système mémoire**.
6. Se reporter à la section 2.2 pour des définitions de l'intensité opérationnelle et de la modélisation `Roofline`.

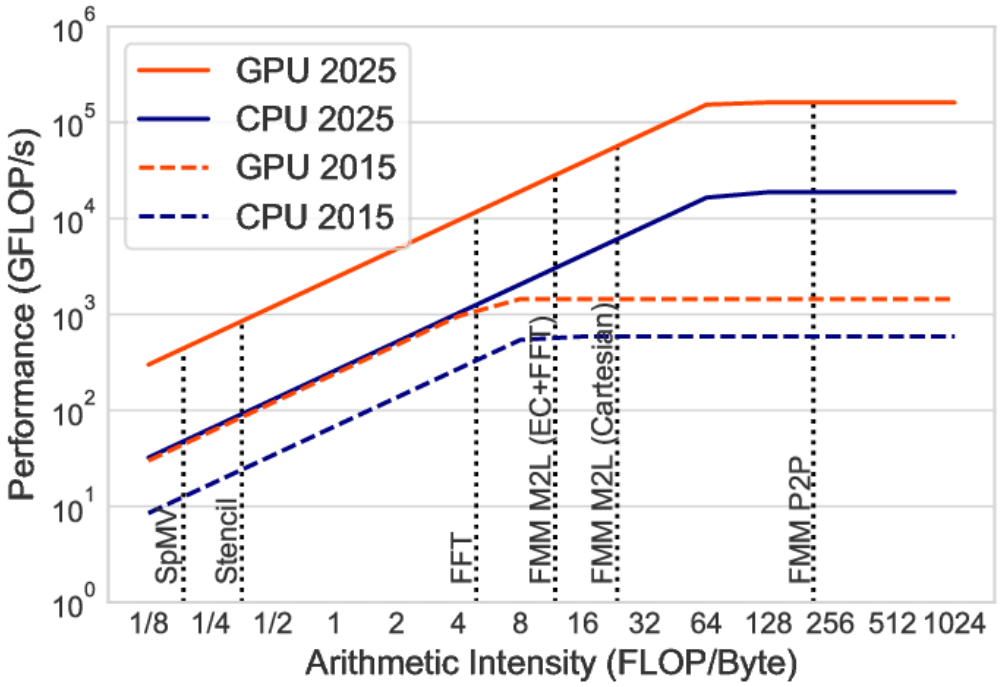


FIGURE 5.1 – Modélisation `Roofline` présentant une estimation des performances maximales atteignables d'un ensemble de noyaux de calcul typiques du domaine du calcul intensif, pour des architectures matérielles caractéristiques des années 2015 et 2025. (Source : [Ibeid 2018])

l'utilisation de sa bande passante.

Ainsi, `BDPO` ciblera les phases d'accès intensifs à la mémoire principale car ces dernières sont de très bonnes candidates à la mise en place de `DVFS`, et qu'une large gamme d'applications `HPC` en exhibent.

Détecter les phases d'accès intensifs à la mémoire principale via un monitoring à grain fin Les phases mémoires ayant été identifiées comme la cible de `BDPO`, reste à déterminer comment les détecter lors de l'exécution d'une application, étant donné que `BDPO` se veut être un outil de reconfiguration dynamique. D'autant plus que le cahier des charges présenté par la section 5.1 contient deux contraites fortes : `BDPO` doit (1) être agnostique de l'application exécutée, et (2) être aussi léger que possible du point de vue de la complexité algorithmique.

En ce qui concerne la première contrainte, les compteurs de performance (`PMC`, introduits par la sous-section 2.1.1.1) s'avérèrent être une solution toute indiquée. En effet, ils permettent de monitorer à l'unité près une large gamme d'évènements, tels que le nombre de références en cache `L2`, permettant de construire des métriques caractéristiques du niveau d'utilisation des différents composants du processeur. Les `PMC` rendent donc possible le suivi, de manière indirecte puisqu'en monitorant le matériel, du comportement dynamique d'une application `HPC` lors de son exécution. De plus, en sélectionnant méticuleusement les évènements monitorés, il est possible de détecter un type particulier

de phase, à l'image des phases mémoires. Cette approche permet ainsi de demeurer agnostique de l'application, tout en offrant à BDPO la possibilité de déterminer quel type de phase traverse ladite application à un moment donné de son exécution.

En ce qui concerne la deuxième contrainte, on peut observer que l'action de compter les différents évènements monitorés est effectuée au niveau matériel, par une circuiterie dédiée. Ainsi, seules la programmation des PMC à l'initialisation de leur utilisation, et leur lecture nécessitent une intervention logicielle qui transite par le noyau du système d'exploitation. En conséquence, il est possible de mettre en place une procédure légère de monitoring à grain fin par échantillonnage des compteurs de performance.

Dernière problématique à laquelle il faut apporter une réponse, quels évènements monitorer via les compteurs de performances, et quelles métriques construire à partir de ces derniers, pour détecter les phases mémoires pendant lesquelles appliquer du DVFS ?

Comme précisé précédemment, les phases mémoires sont caractérisées par de nombreux mouvements de lignes de cache depuis la mémoire principale vers la hiérarchie de caches, ce qui conduit à l'injection d'un nombre de cycles d'attente important, exécutés en lieu et place de micro-opérations. En considérant ces observations, les deux métriques suivantes ont été construites dans le but de détecter les phases mémoires :

$$IPC = \frac{INSTRUCTION\_RETIRED}{RDTSC}$$

$$L3RWC = \frac{L2\_RQSTS.MISS + L2\_TRANS.L2\_WB}{RDTSC}$$

L'IPC définit le nombre d'instructions retirées du pipeline d'un cœur de calcul par cycle de référence du processeur, c'est-à-dire par « coup d'horloge », pour l'horloge cadencée à la fréquence nominale du processeur. L'évènement RDTSC compte précisément ce nombre de cycles écoulés. L'IPC est une métrique représentative du niveau de performance associé à l'exécution d'une application : plus cette métrique est élevée, plus le nombre d'instructions exécutées par seconde est élevé. On peut rajouter qu'une phase de calcul intensive induira vraisemblablement un IPC très élévé.

Le L3RWC est quant à lui un indicateur du nombre de lignes de cache transférées entre les caches L2 et L3 par cycle de référence du processeur : $L2\_RQSTS.MISS$ compte tous les défauts de cache en L2, qui induisent le transfert de lignes de cache depuis le cache L3 vers le cache L2, et $L2\_TRANS.L2\_WB$ compte toutes les écritures en L3 d'une ligne de cache située en L2. Lorsque la valeur de L3RWC est élevée, cela traduit une localité spatiale des données accédées par l'application entre les caches L2 et L3. Au contraire, lorsqu'elle est faible, cela signifie que la localité spatiale des données accédées par l'application se situe soit en cache L1, soit en mémoire principale. Or, dans le domaine du HPC, une localité spatiale en cache L1 permet généralement au noyau de calcul exécuté d'exploiter la puissance de calcul mise à disposition par le matériel, ce qui se traduit par un IPC élevé. Ainsi, lorsqu'une faible valeur de L3RWC est couplée à une faible valeur de l'IPC pendant l'exécution d'une application de calcul à haute performance, cela indique vraisemblablement une localité spatiale des données en mémoire principale, et la présence nourrie de cycles d'attente, en d'autres termes une phase mémoire.

De plus, la phénoménologie associée aux deux métriques considérées par BDPO, à savoir l'IPC et le L3RWC, permet d'envisager l'utilisation de seuils pour détecter les phases

mémoires, puisque ce sont des valeurs basses conjointes des deux métriques qui sont caractéristiques de ces dernières. Du point de vue des performances, cette méthode de détection est particulièrement légère, ce qui est un avantage certain lorsque l'on considère la deuxième contrainte du cahier des charges à remplir.

Pour résumer, BDPO s'appuiera sur le monitoring de deux métriques construites à partir du relevé des compteurs de performances, ainsi qu'un système de seuils associés aux dites métriques pour détecter les phases mémoires, tout en restant léger du point de vue de la complexité algorithmique et agnostique de l'application exécutée.

Rassembler les pièces du puzzle Rassemblons les différents éléments constituant l'approche de BDPO pour en donner un résumé synthétique.

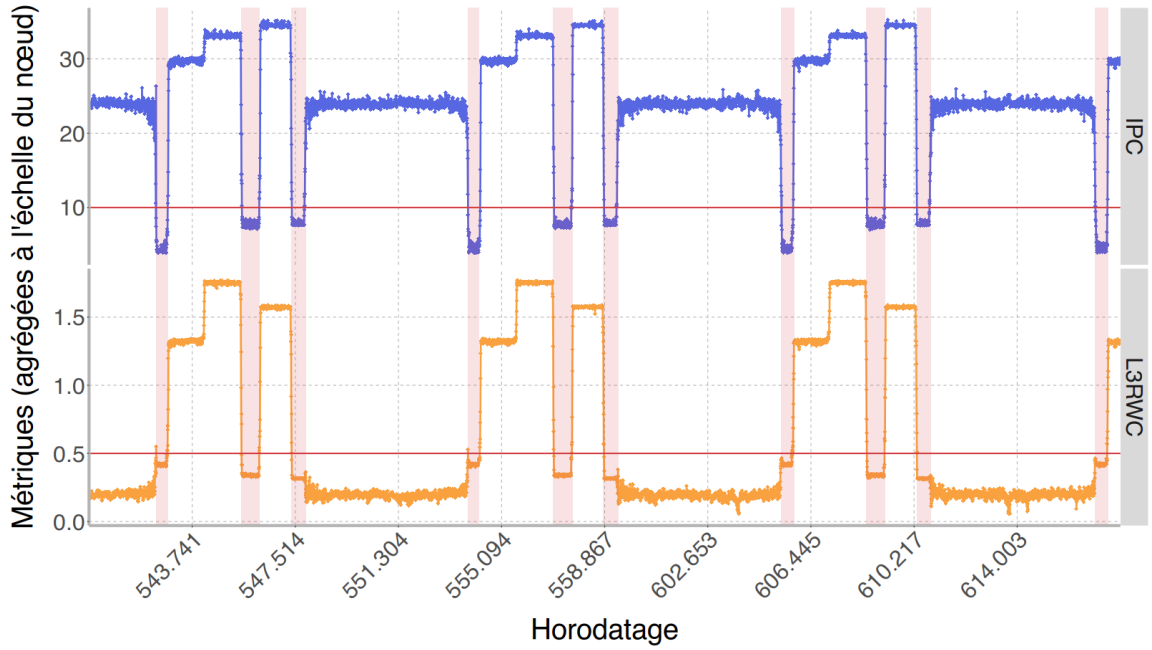


FIGURE 5.2 – Extrait des évolutions de l'IPC et du L3RWC pour une exécution de FT. Les lignes horizontales rouges représentent les seuils sur les métriques qui permettent d'identifier les phases mémoires, représentées par les zones rouges.

BDPO monitore l'évolution de deux métriques caractéristiques de l'utilisation du matériel engendrée par l'exécution d'une application HPC, à savoir l'IPC et le L3RWC. Ces deux métriques sont construites à partir de comptes d'évènements exposés par les PMU des cœurs de calcul, dont les PMC sont échantillonnés à intervalles réguliers par BDPO. Lorsqu'elles passent sous certains seuils, ces métriques permettent de détecter une phase mémoire, c'est-à-dire une portion contigüe de l'exécution de l'application dominée par des transferts de lignes de cache depuis la mémoire principale vers les registres du processeurs. Conséquence directe de ces transferts de lignes de cache, des `cycles d'attente` sont injectés dans les `pipelines` des cœurs de calcul, ce qui se révèle être une occasion parfaite d'appliquer du DVFS. En abaissant le point de fonctionnement { tension ; fréquence } des cœurs de calcul pendant les phases mémoire, BDPO diminue la puissance consommée par ces derniers pendant lesdites phases, sans induire de dégradation notable

des performances. Cela permet d'améliorer l'efficacité énergétique associée à l'exécution de l'application considérée.

À titre d'exemple, la figure 5.2 présente des extraits de l'évolution de l'IPC et du L3RWC pour une exécution de FT[7]. Les lignes horizontales rouges représentent les seuils sur les deux métriques. Les zones rouges correspondent aux portions pour lesquelles les valeurs de l'IPC et du L3RWC sont inférieures aux seuils qui leurs sont associés. Autrement dit, les zones rouges correspondent à des phases mémoires pendant lesquelles BDPO peut appliquer du DVFS : il abaisse le point de fonctionnement { tension ; fréquence } des cœurs de calcul à l'entrée d'une zone rouge, et le restaure à sa sortie.

Pour conclure, précisons que les détails techniques liés à l'implémentation de BDPO, tels que la fréquence à laquelle les PMC sont échantillonnés, sont donnés par la section 5.3, et que la section 6.1 présente la méthodologie expérimentale permettant de définir les seuils sur l'IPC et le L3RWC, ainsi que les points de fonctionnement { tension ; fréquence } ciblés par le DVFS.

7. Ces extraits ont été obtenus en monitorant les PMC des cœurs de calcul grâce à BDPO.

5.3 Architecture de BDPO

La première chose à préciser concernant l'architecture de BDPO, c'est qu'il a été conçu comme un outil de reconfiguration **local** à un nœud de calcul. En d'autres termes, une **instance isolée** de BDPO est exécutée sur **chaque** nœud de calcul impliqué dans l'exécution de l'application pour laquelle on cherche à améliorer l'efficacité énergétique associée. Il n'y a pas de synchronisation à une échelle globale entre les différentes instances de BDPO. Chaque instance agit donc en toute indépendance vis-à-vis des autres instances.

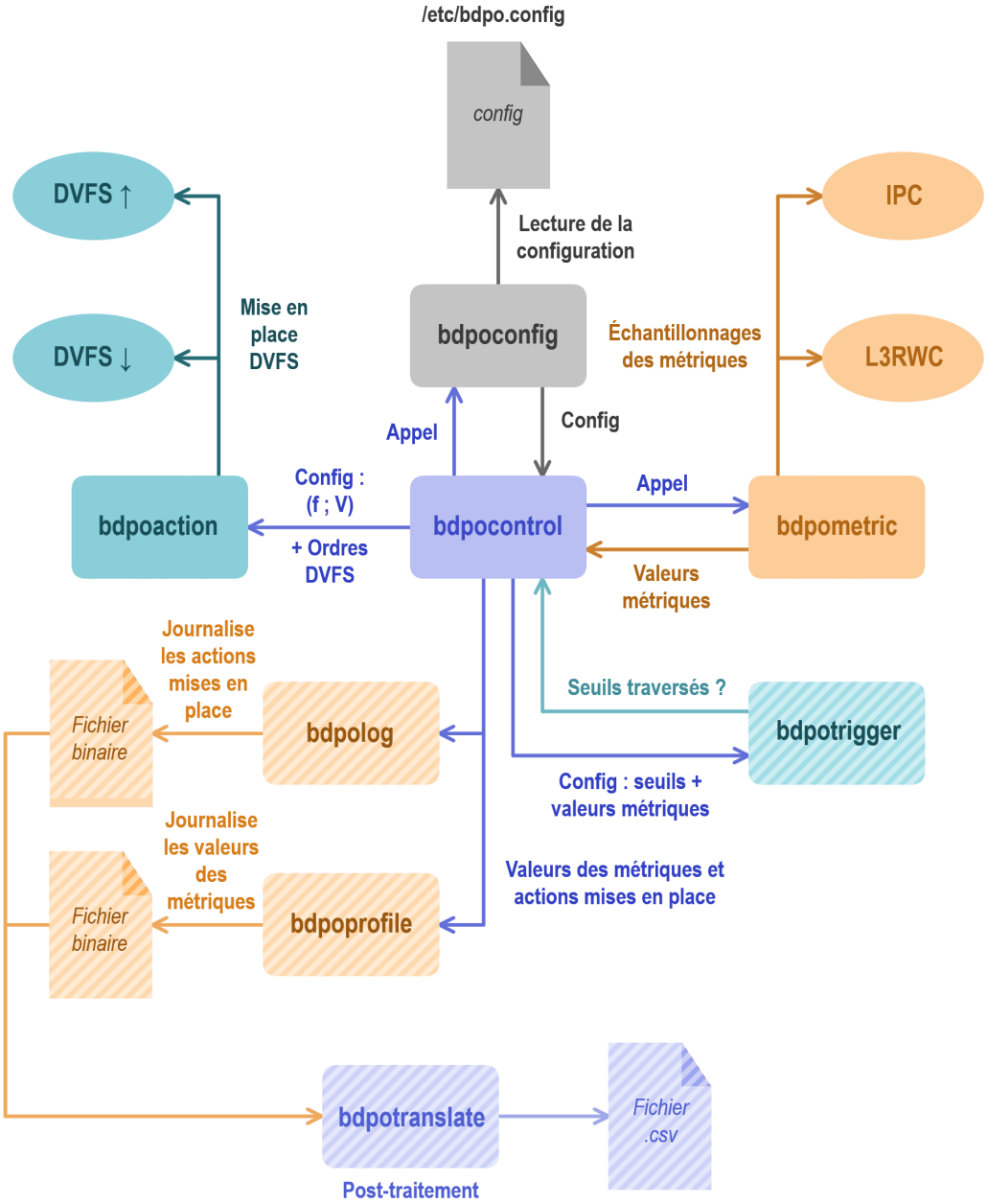


FIGURE 5.3 – Schéma décrivant l'architecture modulaire de BDPO.

L'implémentation de BDPO, réalisée en langage C, est modulaire, comme le présente la figure 5.3. L'exécution de BDPO commence par un appel à bdpoconfig pour lire sa configuration, spécifiée via le fichier de configuration /etc/bdpo.conf et la ligne de commande. Ladite configuration contient notamment les valeurs des seuils associés aux métriques, les points de fonctionnement { tension ; fréquence } ciblés, et la fréquence d'échantillonnage des PMC.

BDPO s'articule autour d'une boucle principale qui, une fois la configuration lue et l'initialisation des structures de données effectuée, est itérée jusqu'à ce que BDPO reçoive un signal de terminaison. Cette boucle principale, implémentée au sein du module bdpo-control, est constituée d'un ensemble d'étapes exécutées séquentiellement :

1. Appel au module bdpometric qui relève les compteurs de performance, et calcule les valeurs des métriques (i.e. IPC et L3RWC) ;
2. Les valeurs des métriques pour l'itération de la boucle principale considérée sont transmises au module bdpotrigger, qui détermine si la mise en place de DVFS est nécessaire ;
3. Si c'est le cas, le module bdpoaction reçoit l'ordre de reconfigurer le point de fonctionnement { tension ; fréquence } des cœurs de calcul ;
4. Si la journalisation des valeurs prises par les métriques, et des reconfigurations mises en place par BDPO, ont été activées via sa configuration, alors ces dernières sont respectivement effectuées par les modules bdpoprofile et bdpolog ;
5. BDPO s'endort pour une durée variable, définie de sorte à respecter la période d'échantillonnage des compteurs de performance.

De multiples précisions méritent d'être apportées. Pour commencer, le relevé des PMC est effectué à l'aide de la bibliothèque libpfm, pour l'ensemble des processus (i.e. pas seulement l'application HPC exécutée). En effet, les reconfigurations des cœurs de calcul mises en place par BDPO sont conditionnelles de l'état d'utilisation de ces derniers. Il faut donc considérer tous les « utilisateurs » des cœurs de calcul, c'est-à-dire l'ensemble des processus exécutés. De sorte à pouvoir monitorer les évènements générés par l'ensemble des processus, il faut que l'utilisateur démarrant l'exécution de BDPO ait les privilèges d'administration, ou bien que le paramètre perf_event_paranoid exposé par le noyau Linux soit fixé à -1.

Concernant les reconfigurations mises en place par BDPO, plusieurs éléments doivent être précisés. Premièrement, seulement deux points de fonctionnement { tension ; fréquence } sont ciblés par BDPO parmi ceux exhibés par les cœurs de calcul. L'un est désigné comme le point de fonctionnement « bas », et doit être utilisé lors des phases mémoires, et l'autre est désigné comme le point de fonctionnement « haut », et doit être utilisé le reste du temps. Le module bdpoaction implémente donc deux actions : (1) le passage du point de fonctionnement bas vers le point de fonctionnement haut, noté $DVFS_{bas \rightarrow haut}$, et (2) le passage du point de fonctionnement haut vers le point de fonctionnement bas, noté $DVFS_{haut \rightarrow bas}$.

Lorsque le point de fonctionnement bas est actif, cela signifie que les valeurs des deux métriques sont inférieures aux seuils qui sont associés à ces dernières. Le fait qu'au moins

une des deux métriques prenne alors une valeur supérieure au seuil associé déclenche l'action $DVFS_{bas \to haut}$. Lorsque le point de fonctionnement haut est actif, cela signifie qu'au moins une des valeurs des deux métriques est supérieure au seuil associé. Le fait que les deux métriques prennent des valeurs inférieures aux seuils associés déclenche alors l'action $DVFS_{haut \to bas}$.

Pour finir concernant les reconfigurations, précisons que ces dernières sont effectuées par le biais du module `acpi-cpufreq` du noyau `Linux`, via l'interface qu'il expose au sein du système de fichiers `sysfs`. Pour ce faire, des privilèges d'administration sont requis. Il est donc nécessaire que l'administrateur du supercalculateur les accorde à `BDPO`, ou bien qu'il fournisse une manière alternative d'appliquer du `DVFS` depuis l'espace utilisateur, à l'image de `msr-safe`, présenté par [Shoga 2014].

Concernant la journalisation des métriques et des actions prises par `BDPO`, on peut remarquer que les modules `bdpolog` et `bdpoprofile` produisent des fichiers binaires. En effet, cela permet d'écrire directement le contenu des structures données sans avoir à le traduire en caractères alphanumériques, ce qui présente deux avantages : (1) le volume des données à journaliser est réduit, ce qui permet notamment de mettre ces dernières en tampon (on parle de « buffering » dans la langue de Shakespeare), et (2) le coût en termes de performances de la traduction en caractères alphanumériques est évité.

Le module `bdpotranslate` permet alors, à postériori, de traduire ces fichiers binaires vers des fichiers au format `CSV`, compréhensibles et exploitables.

Point très important, l'implémentation de la boucle principale de `BDPO` est **parallèle** : un thread s'exécute sur chaque cœur de calcul, et prend en charge le relevé des métriques et la mise en place de `DVFS` pour le cœur auquel il est associé. La prise de décisions par le module `bdpotrigger`, ainsi que la journalisation des métriques et des actions mises en place, peuvent alors aussi bien être effectuées à l'échelle du cœur de calcul, qu'à l'échelle du nœud de calcul, en agrégeant les valeurs des métriques pour l'ensemble des cœurs.

Pour finir, un plugin `SPANK` permet la gestion du cycle de vie de `BDPO` à l'échelle d'un job via l'ordonnanceur `slurm`. Lorsque l'option `--bdpo=yes` est spécifiée dans une commande `slurm` de lancement de job (e.g. `srun`), ce dernier s'occupe d'exécuter `BDPO`, en parallèle de l'application `HPC` considérée, sur l'ensemble des nœuds de calcul impliqués dans le job en question. Cela permet également d'attribuer, via `slurm`, les droits d'administration nécessaires à l'ensemble des instances de `BDPO`.

Expérimentations

6

Pour commencer, il est important de noter que l'**ensemble** des expérimentations présentées par ce chapitre ont été réalisées sur la plateforme matérielle mise à disposition par Atos, dont les caractéristiques sont spécifiées par la section 4.1.1.

Dans un premier temps, la section 6.1 présente la méthodologie de calibrage associée à BDPO. Cette dernière permet de construire une configuration spécifique à une architecture matérielle donnée. Avec une configuration adaptée aux nœuds de calcul exécutant une application de calcul à haute performance, il est possible d'optimiser l'efficacité énergétique associée à l'exécution en question, comme le montre la section 6.2. La section 6.3 présente quant à elle une évaluation des performances de BDPO et de son impact sur l'application HPC dont l'exécution est optimisée d'un point de vue énergétique. Pour finir, la section 6.4 dresse un bilan des expériences présentées par ce chapitre, et fait une ouverture concernant l'un des principaux axes d'amélioration de BDPO : faire en sorte que les reconfigurations qu'il met en place soient prédictives et non plus réactives.

6.1 Méthodologie expérimentale de calibrage associée à BDPO

La section 5.2 a détaillé en profondeur le fonctionnement de BDPO, qui implique de définir des seuils sur les métriques monitorées, à savoir l'IPC et le L3RWC, afin d'identifier les phases mémoires. Ces dernières sont exploitables via la mise en place DVFS, ce qui nécessite d'identifier des points de fonctionnement { tension ; fréquence } bas et haut intéressants du point de vue de l'efficacité énergétique. De manière assez évidente, ces quatre paramètres de configuration sont fondamentaux, et ont un impact critique sur la capacité de BDPO à améliorer l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance. On comprend alors que définir ces paramètres de manière adéquate nécessite une expertise technique que l'utilisateur final d'un supercal-

culateur ne possède à priori pas. Ce qui va à l'encontre de la volonté centrale au cahier des charges de BDPO de favoriser son adoption par la communauté HPC.

C'est pourquoi BDPO est accompagné d'une méthodologie empirique de calibration, encadrant et guidant la définition des seuils et points de fonctionnement haut et bas. Ladite méthodologie s'appuie sur l'observation suivante : les quatre paramètres dépendent intrinsèquement et uniquement de l'architecture matérielle des nœuds de calcul sur lesquels l'application considérée est exécutée. En conséquence, il est possible de construire une configuration de BDPO spécifique à une architecture de nœuds de calcul, et c'est précisément le but de la méthodologie décrite par cette section.

Définition du protocole expérimental Il existe plusieurs problématiques sous-jacentes à la construction d'une configuration de BDPO spécifique à une architecture de nœuds de calcul :

1. Les seuils sur l'IPC et le L3RWC doivent permettre d'identifier les phases mémoires, et il faut donc que la méthodologie construite permette de séparer les cas où les données résident en mémoire principale des cas où elles résident dans la hiérarchie de caches ;
2. Pour identifier les points de fonctionnement { tension ; fréquence } d'intérêt, il faut être capable d'étudier l'impact des points de fonctionnement sur les ratios { performances / énergie } associés aux applications exécutées ;
3. Il faut que les observations réalisées concernant les deux premières problématiques soient vérifiées pour une large gamme d'applications HPC.

Intéressons-nous dans un premier temps à la troisième et dernière problématique. « Intégrer » un large panel d'applications HPC à la méthodologie nécessiterait un effort colossal, serait complexe à complètement automatiser, et donc vraisemblablement lourd à mettre en œuvre pour les utilisateurs de BDPO. En conséquence, l'approche retenue consiste à utiliser les noyaux de calcul BLAS [1] comme **proxy** pour les applications de calcul à haute performance. En effet, ces noyaux de calcul optimisés implémentent des recettes numériques d'algèbre linéaire variées et typiques du domaine du calcul intensif, et sont utilisés par un grand nombre d'applications HPC. Des micro-benchmarks faisant office de « lanceurs » pour les noyaux de calcul BLAS ont été implémentés, et leurs exécutions selon le protocole élaboré dans ce paragraphe ont été automatisées de sorte à être intégrées au processus d'installation de BDPO sur un nœud de calcul. Les micro-benchmarks appellent les noyaux de calcul BLAS sur un jeu de données de taille spécifiable, et généré pseudo-aléatoirement. Ces appels sont encapsulés dans une structure itérative, ce qui permet de spécifier au moment du lancement d'un micro-benchmark le nombre de répétitions à effectuer. En conséquence, en s'appuyant sur les BLAS, la méthodologie de calibrage de BDPO permet de couvrir un vaste spectre d'applications de calcul à haute performance, tout en demeurant simple et compacte, ce qui a permis d'automatiser son exécution.

Un autre avantage à l'utilisation de micro-benchmarks est qu'il est possible d'aisément faire varier les paramètres liés à leurs exécutions, tels que la résidence de leurs jeux de

1. Présentés par la section 4.2.

données d'entrée. Ainsi, les noyaux de calcul BLAS pourront être exécutés d'une part pour des jeux de données résidant en mémoire principale, et d'autre part pour des jeux de données résidant en cache L3. En couplant cela au monitoring des métriques IPC et L3RWC pendant les exécutions des noyaux de calcul, on répond à la première des problématiques listées ci-dessus.

Reste la deuxième problématique. Le fait que les processeurs exposent un nombre de points de fonctionnement relativement restreint (i.e. inférieur à 15), les évaluer de manière exhaustive est envisageable. Ainsi, pour déterminer quels points de fonctionnement sont intéressants du point de vue de l'application de DVFS par BDPO, les noyaux de calcul BLAS seront exécutés pour l'ensemble des points de fonctionnement { tension ; fréquence }, tout en monitorant les consommations énergétiques et temps d'exécution [2] associés.

Si l'on rassemble les éléments ci-dessus, le protocole expérimental suivant permet de répondre aux trois problématiques sous-jacentes à la conception d'une méthodologie de calibrage pour BDPO : exécuter l'ensemble des noyaux de calcul BLAS pour l'ensemble des points de fonctionnement { tension ; fréquence }, tout en monitorant la consommation énergétique, le temps d'exécution et les métriques IPC et L3RWC via les compteurs de performance, pour deux jeux de données distincts, l'un résidant en cache L3, l'autre en mémoire principale.

Pour finir, quelques détails techniques. Premièrement, les micro-benchmarks ont été exécutés sur un nœud de calcul, de manière concurrente, via OpenMP, sur l'ensemble de ses 56 cœurs de calcul physiques. Encore une fois, la sous-section 4.1.1 présente les caractéristiques techniques des nœuds de calcul utilisés, et la section 4.2 donne un ensemble de précisions supplémentaires concernant les noyaux de calcul BLAS, notamment concernant l'implémentation et la version utilisées).

De plus, pour chaque **point expérimental** (i.e. chaque couple { point de fonctionnement ; résidence des données }), les noyaux de calcul ont été exécutés 15 fois, et les valeurs médianes des grandeurs monitorées pour ces 15 exécutions ont été retenues.

Ensuite, pour chaque noyau de calcul, le nombre de répétitions du micro-benchmark associé est adapté, de sorte à ce que son exécution pour le point de fonctionnement nominal dure au moins 15 secondes. Les mêmes nombres de répétitions sont ensuite effectuées pour tous les points expérimentaux associés au micro-benchmark considéré partageant la même résidence des données.

Ajoutons que les temps d'exécution ont été mesurés en comptant, via RDTSC, le nombre de cycles de référence du processeur nécessaires au traitement de la charge de travail à accomplir. Dans la même veine, l'énergie consommée par l'exécution des noyaux de calcul a été monitorée à l'échelle du nœud de calcul complet, via un sonde matérielle qui relève la puissance consommée moyenne à une fréquence de 1 Hz et dont la résolution est $125mW$.

Enfin, ce protocole expérimental a été répété sur les deux nœuds de calcul de la partition utilisée, et a conduit à des conclusions similaires.

2. Le temps d'exécution pour effectuer une charge de travail donnée, est une métrique, certes simpliste et indirecte mais ici pertinente et suffisante, du niveau de performance associé à l'exécution d'une application.

Choix des points de fonctionnement { tension ; fréquence } « bas » et « haut » Les résultats expérimentaux détaillant l'impact du balayage des points de fonctionnement { tension ; fréquence } sur les performances et l'énergie consommée par l'exécution de l'ensemble des noyaux de calcul `BLAS` sont présentés par les figures 6.1, 6.2, 6.3, et 6.4. Commençons par un guide pour la lecture de ces figures.

Les histogrammes colorés représentent les réductions de consommation énergétique pour l'exécution d'un noyau de calcul, pour chacun des points de fonctionnement par rapport au point de fonction de référence. L'axe des ordonnées associé est celui situé à gauche du graphe. Chaque point de fonctionnement est identifié par la fréquence associée, exprimée en kHz, et s'est vu attribuer une couleur spécifique. La fréquence de référence est 2.10 GHz = 2100000 kHz, et « Turbo » désigne le point de fonctionnement pour lequel la fréquence est celle de référence et le `TurboBoost` est activé dès que l'enveloppe thermique du processeur et l'état des `pipelines` des cœurs de calcul le permettent. Les lignes horizontales dorées correspondent respectivement à des réductions de la consommation énergétique par 3% et 10%.

Les courbes noires qui viennent se superposer aux histogrammes représentent l'augmentation du temps d'exécution par rapport à celui associé au point de fonctionnement nominal. L'axe des ordonnées associé est celui situé à droite du graphe, et la ligne horizontale noire correspond à une augmentation du temps d'exécution de 3%.

Pour finir, précisons que les figures 6.1 et 6.2 sont associées aux noyaux `Dense BLAS`, pour des résidences des données respectivement en mémoire principale et en cache `L3`, alors que les figures 6.3 et 6.4 sont associées aux noyaux `Sparse BLAS`, également pour des résidences des données respectivement en mémoire principale et en cache `L3`.

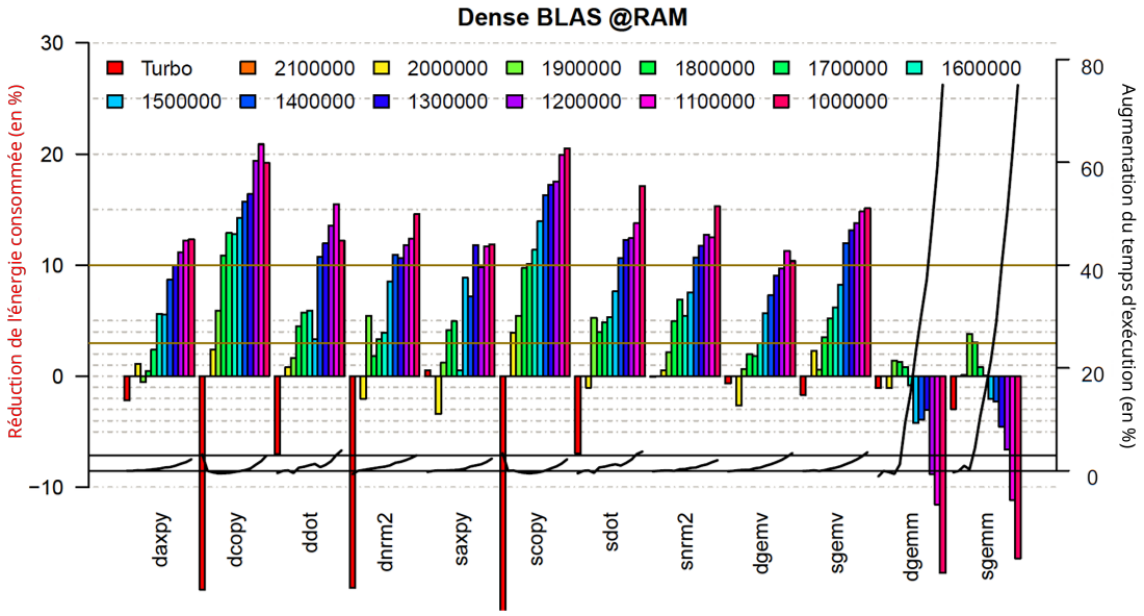


FIGURE 6.1 – Impact sur les performances et l'énergie consommée associées à l'exécution des noyaux de calcul `Dense BLAS` d'un balayage de l'ensemble des points de fonctionnement { tension ; fréquence } disponibles, pour une résidence des données en mémoire principale.

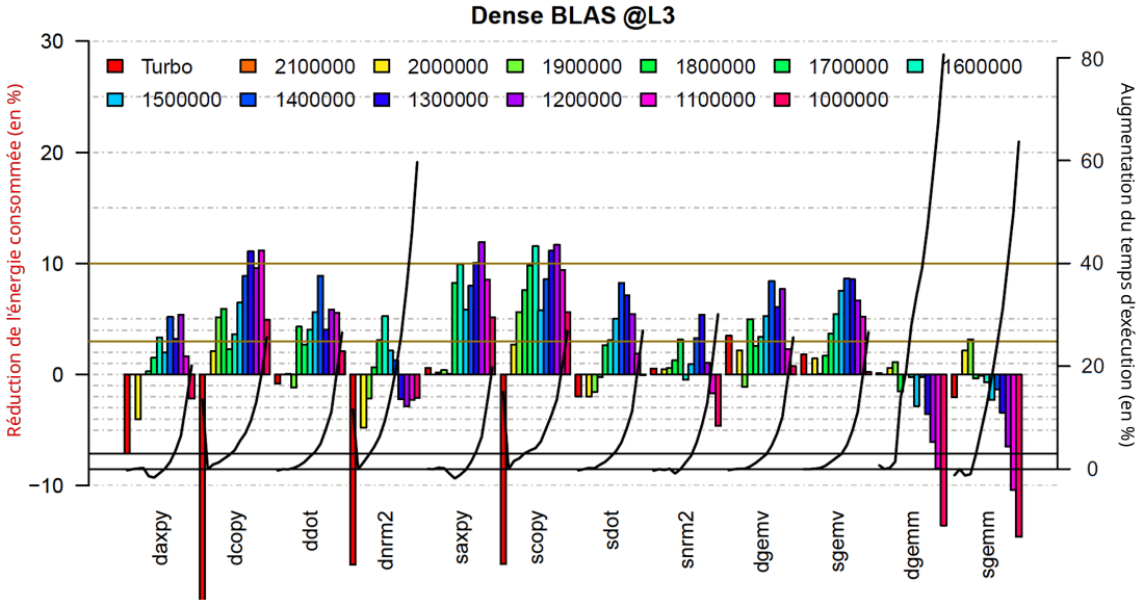


FIGURE 6.2 – Impact sur les performances et l'énergie consommée associées à l'exécution des noyaux de calcul `Dense BLAS` d'un balayage de l'ensemble des points de fonctionnement { tension ; fréquence } disponibles, pour une résidence des données en cache `L3`.

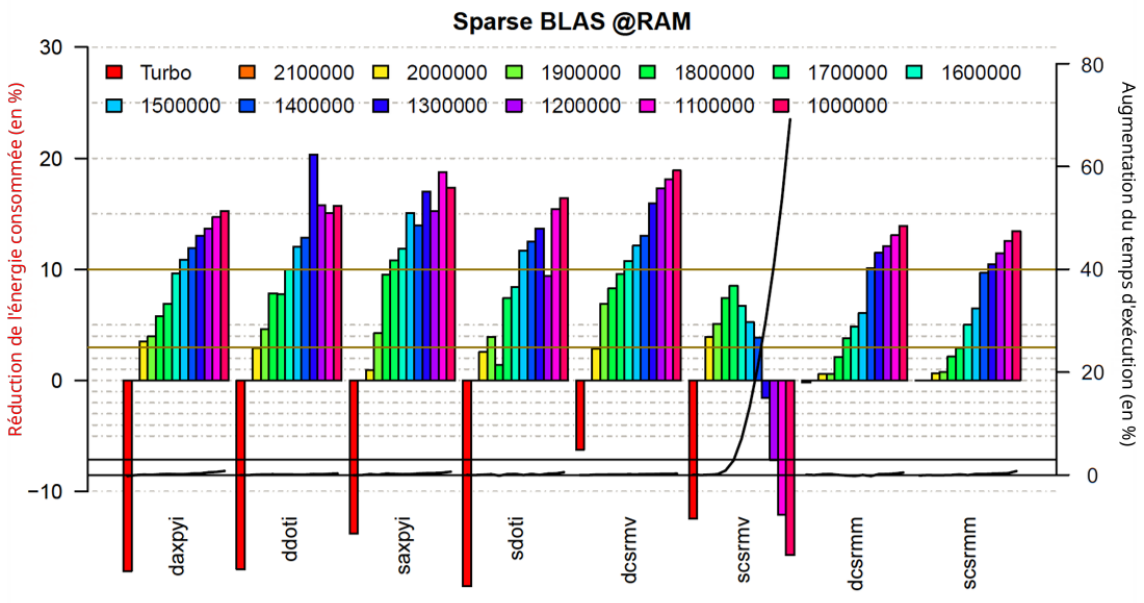


FIGURE 6.3 – Impact sur les performances et l'énergie consommée associées à l'exécution des noyaux de calcul `Sparse BLAS` d'un balayage de l'ensemble des points de fonctionnement { tension ; fréquence } disponibles, pour une résidence des données en mémoire principale.

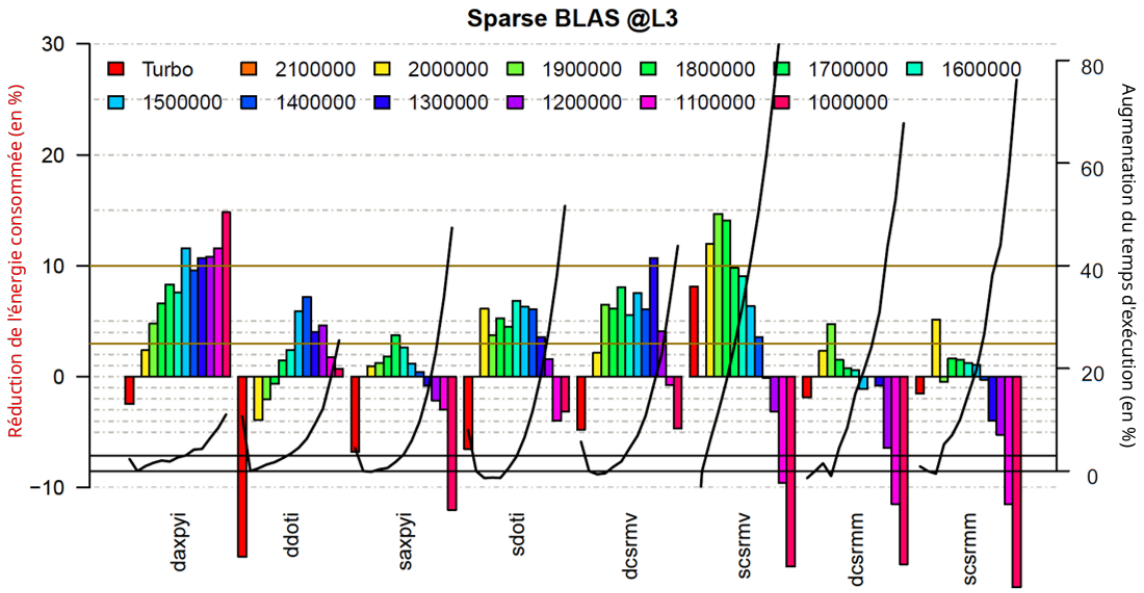


FIGURE 6.4 – Impact sur les performances et l'énergie consommée associées à l'exécution des noyaux de calcul `Sparse BLAS` d'un balayage de l'ensemble des points de fonctionnement { tension ; fréquence } disponibles, pour une résidence des données en cache `L3`.

Plusieurs observations peuvent être faites. Pour commencer, concentrons-nous sur les points expérimentaux associés à une résidence des jeux de données en mémoire principale. On remarque que pour tous les noyaux de calcul appartenant aux familles `Dense BLAS-1`, `Dense BLAS-2`, `Sparse BLAS-1`, et `Sparse BLAS-3`, des gains énergétiques significatifs (i.e. au moins 10% et jusqu'à plus de 20%) peuvent être accomplis en optant pour un point de fonctionnement { tension ; fréquence } associé à une fréquence basse, tout en maintenant l'augmentation du temps d'exécution sous la barre des 3%.

Il en va de même pour la famille `Sparse BLAS-2`, à l'exception du noyau de calcul `scsrmv`, qui implémente la multiplication d'une matrice creuse au format `CSR` par un vecteur dense. Du fait de la représentation `CSR`, toutes les valeurs non-nulles de la matrice creuse sont stockées dans un même tableau unidimensionnel, dans l'ordre des lignes de la matrice. Étant donné qu'il y a en moyenne peu d'éléments non-nuls par ligne de la matrice creuse, il y a en moyenne moins d'évictions en cache `L3` lors du parcours de la matrice creuse et du vecteur dense, comparativement aux noyaux de calcul appartenant aux familles `BLAS` déjà passées en revue. Cette « réutilisabilité partielle » se traduit par un équilibre entre phases mémoires « marquées » et phases mémoires « légères ». Ce qui explique qu'une diminution de la fréquence des cœurs de calcul ne soit profitable du point de vue du ratio { performances / énergie } que jusqu'à un certain point, ici 1.70 GHz.

En ce qui concerne les noyaux de calcul `Dense BLAS-3`, on remarque que la mise en place de `DVFS` est particulièrement défavorable dès que l'on s'aventure en deçà d'une fréquence de 1.90 Ghz. Les deux noyaux en question, `dgemm` et `sgemm`, implémentent la multiplication de deux matrices denses, opération qui se prête particulièrement bien à la mise en place du parcours par bloc des matrices en question. En adaptant la taille du bloc

à la taille du cache L1, on atteint des taux de réutilisabilité des données très élevés, et une résidence des données « effective » en cache L1, même pour un jeu de données complet résidant en mémoire principale. En conséquence, le nombre de phases mémoires est très limité, et ces dernières interviennent principalement lors du transfert des lignes de cache associées au bloc considéré depuis la mémoire principale vers la hiérarchie de caches. La proportion de phases mémoires étant ainsi relativement faible, l'utilisation d'un point de fonctionnement { tension ; fréquence } bas n'est pas profitable. En comparaison, les noyaux de calcul Sparse BLAS-3 ne permettent pas le parcours des matrices creuses par bloc, du fait de la représentation CSR. C'est pourquoi une diminution de la fréquence des cœurs de calcul est pleinement profitable du point de vue du ratio { performances / énergie } pour ces noyaux de calcul.

Si l'on s'intéresse maintenant aux résultats associés à une résidence des données en cache L3, on s'aperçoit que pour l'intégralité des noyaux de calcul BLAS, la réduction de l'énergie consommée induite par l'utilisation d'une fréquence inférieure à la fréquence nominale s'accompagne de dégradations significatives des performances. En effet, lorsque les données d'entrée résident en cache L3, les transferts de lignes de cache entre la mémoire principale et la hiérarchie de caches est limitée, ce qui implique que les phases mémoires sont sporadiques et peu fréquentes (comparativement aux expériences associées avec une résidence en mémoire principale). Par conséquent, peu de cycles d'attente sont injectés dans les pipelines des cœurs de calcul, et les opportunités d'utilisation de DVFS sont peu fréquentes. Couplé à l'impact fortement favorable du point de vue de l'efficacité énergétique que la mise en place de DVFS peut avoir lorsque le jeu de données réside en mémoire principale, cela tend à valider l'approche de BDPO.

Considérons maintenant les résultats associés au point de fonctionnement « Turbo ». Pour la vaste majorité des noyaux de calcul lorsque les données résident en mémoire principale, le sur-cadençage des cœurs de calcul conduit simplement à une augmentation significative de la puissance consommée sans pour autant augmenter la quantité de travail utile effectuée : les cycles d'attente sont ingérés plus rapidement et donc injectés en plus grandes quantités. Les résultats associés aux noyaux de calcul Sparse BLAS-3 servent de rappel au fait que l'activation du TurboBoost est conditionnelle à un ensemble complexe de paramètres matériels, ce qui semble avoir empêché son utilisation pour lesdits noyaux lorsque les données résident en mémoire principale. Lorsque les données résident en cache L3, le TurboBoost se montre également globalement défavorable du point du vue du ratio { performances / énergie }.

Trois observations supplémentaires méritent néanmoins d'être faites. Premièrement, lorsque son utilisation permet d'accélérer de manière effective et significative des phases de calcul intensives, comme c'est le cas pour scsrmv, l'activation du TurboBoost peut s'avérer très efficace sur le plan énergétique.

Deuxièmement, il est possible que l'utilisation du TurboBoost dégrade les performances, lorsqu'il est sensé les accélérer. Ce phénomène est notamment visible pour les noyaux de calcul doti, sdoti, et dnrm2 pour une résidence des données en cache L3, et dcopy et scopy quelque soit la résidence des données. Cette augmentation du temps d'exécution est vraisemblablement induite par la création d'une contention sur la hiérarchie de caches. En effet, le sur-cadençage des cœurs de calcul provoque une augmentation du nombre de requêtes par unité de temps à traiter pour la hiérarchie de caches, vrai-

semblablement suffisamment importante pour saturer les ports des tuiles constituant le cache L3, ces dernières étant partagées par plusieurs cœurs de calcul. Pour être complets, notons que les phases mémoires dominent le temps d'exécution de dcopy et scopy de manière écrasante, pour une résidence en mémoire principale. Tellement écrasante que la pression exercée sur le cache L3 est suffisante pour induire des effets que l'on observe pour une résidence en cache L3.

Troisièmement, il convient de s'apesantir sur le fait que les mécaniques de gestion énergétique et thermique des processeurs sont complexes. Des phénomènes pour le moins contre-intuitifs peuvent alors subvenir, comme c'est le cas pour les noyaux dgemv et sgemv lorsque le jeu de données d'entrée réside en cache L3. En effet, on s'aperçoit que sans engendrer de diminution du temps d'exécution, l'utilisation forcée du TurboBoost permet de réduire l'énergie consommée significativement (i.e. de plus d'un pourcent). Une explication plausible est que le sur-cadençage des cœurs de calcul pendant les phases entrecoupant les longues phases de calculs vectoriels empêche l'utilisation de la fréquence AVX512 maximale pendant ces dernières. Il y aurait alors un « jeu de compromis » favorable du point de vue de l'efficacité énergétique entre les diminutions et augmentations de puissances consommées et temps d'exécution entre les phases vectorielles ralenties et les phases accélérées par le TurboBoost.

De l'examen de l'ensemble de ces résultats expérimentaux, il ressort que, pour une résidence des données en mémoire principale : (1) la plus basse fréquence permettant de garder la dégradation de performance à un niveau négligeable (i.e. moins de 0.5%) tout en réduisant la puissance consommée est 1.90 GHz, et (2) la fréquence engendrant la diminution de l'énergie consommée la plus élevée tout en maintenant l'augmentation du temps d'exécution sous le seuil acceptable des 3% est 1.10 GHz. En conséquence, pour les nœuds de calcul considérés (cf. sous-section 4.1.1), les points de fonctionnement { tension ; fréquence } bas et haut sélectionnés pour BDPO sont respectivement ceux associés aux fréquences 1.10 GHz et 1.90 GHz.

Détermination des seuils associés à l'IPC et au L3RWC permettant la détection des phases mémoires La figure 6.5 présente les résultats expérimentaux associés à l'étude de l'impact d'un balayage des points de fonctionnement { tension ; fréquence } sur l'IPC et le L3RWC exhibés par les noyaux de calcul BLAS lors de leurs exécutions. Sur cette figure, les coordonnées d'un point expérimental sont l'IPC et le L3RWC moyens pour l'entièreté de l'exécution du noyau de calcul associé. Une couleur est associée à chaque famille de noyaux de calcul (e.g. les Dense BLAS-3 sont représentés en vert). Les triangles représentent les exécutions pour lesquelles les données d'entrée résidaient en cache L3, et les ronds celles pour lesquelles elles résidaient en mémoire principale. De plus, les points expérimentaux (i.e. triangles et ronds) associés aux exécutions d'un même noyau de calcul pour différents points de fonctionnement sont liés, le point expérimental exhibant le plus faible IPC étant associé à la fréquence la plus basse, et celui exhibant le plus haut IPC étant associé à la fréquence la plus haute.

La première chose que l'on peut observer est qu'il existe, pour chaque noyau de calcul, exception faite de la famille Dense BLAS-3 et de scsrmv, une claire séparation entre les points expérimentaux associés à une résidence des données en mémoire principale et ceux associés à une résidence en cache L3. Le comportement exhibé par dgemm et

`sgemm` s'explique par le fait que ces noyaux de calcul implémentent un parcours par bloc des matrices, comme cela a été détaillé précédemment. En conséquence, peu importe la résidence du jeu de données d'entrée, la résidence « effective » du bloc considéré est en cache `L1`, rendant ces deux noyaux très sensibles à la fréquence des cœurs de calcul. Concernant `scsrmv`, la réutilisabilité partielle des données induit une certaine sensibilité à la fréquence et permet d'atteindre un niveau de performance supérieur aux autres noyaux de calcul, pour une résidence des données en mémoire principale.

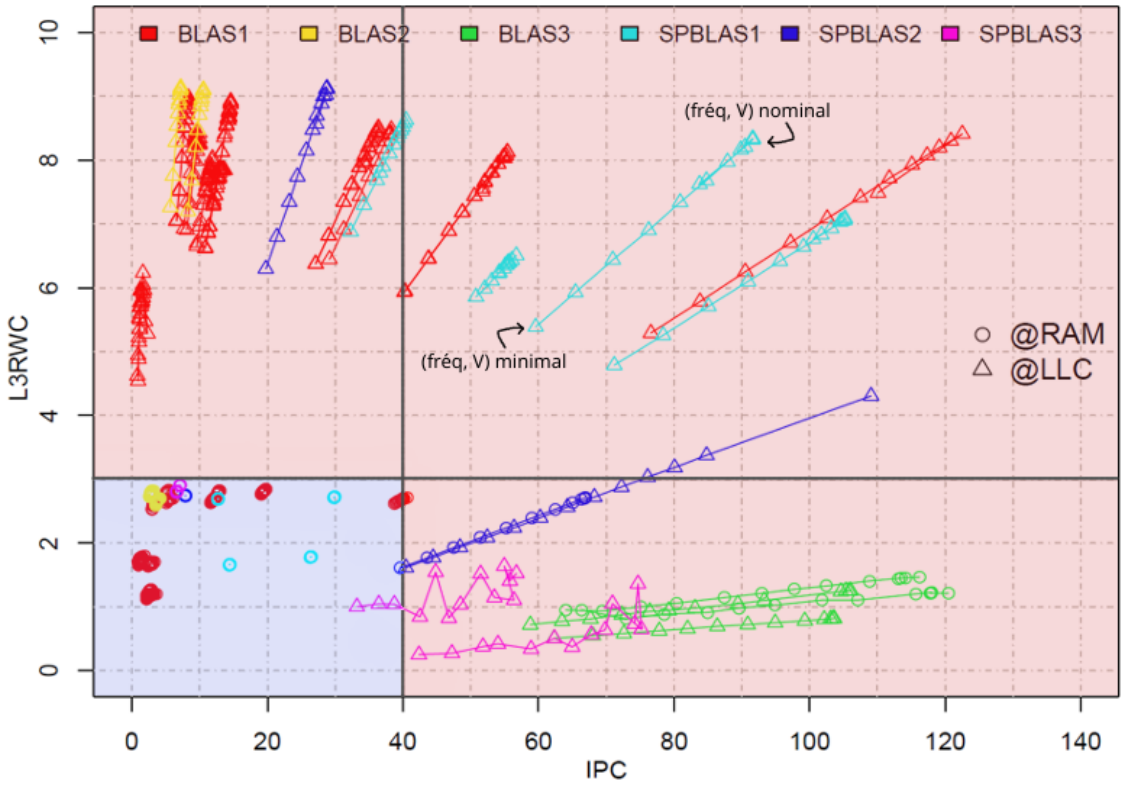


FIGURE 6.5 – Impact sur l'`IPC` et le `L3RWC` associés à l'exécution des noyaux de calcul `BLAS` d'un balayage de l'ensemble des points de fonctionnement { tension ; fréquence } disponibles, pour des résidences des données en cache `L3` et en mémoire principale.

De plus, on peut observer que le cadrant délimité par une zone d'un bleu violacé englobe la vaste majorité des points de données associés à une résidence en mémoire principale. À cela s'ajoute le fait que chacune de ces séries de points expérimentaux voit ces derniers être concentrés dans des zones restreintes, ce qui traduit une insensibilité à la fréquence des cœurs de calcul du point de vue des performances. À contrario, lorsque le jeu de données réside en cache `L3`, l'`IPC`, et donc le niveau de performance, sont largement impactés par un changement de point de fonctionnement { tension ; fréquence } des cœurs de calcul.

Il est ainsi possible de définir simplement et efficacement des seuils sur l'`IPC` et le `L3RWC` permettant de détecter les phases mémoires en considérant les bornes de la zone bleue. Pour l'architecture des nœuds de calcul utilisés dans le cadre de ces expériences, on obtient ainsi $seuil_{IPC} = 40.0$ et $seuil_{L3RWC} = 3.00$.

Pour finir, notons que les observations qui viennent d'être faites sont cohérentes avec les conclusions tirées plus tôt dans cette section. Cela est notamment mis en exergue par les corrélations entre les impacts des changements de point de fonctionnement { tension ; fréquence } sur les temps d'exécution d'une part, et sur les valeurs des métriques `IPC` et `L3RWC` d'autre part.

Ce qu'il faut retenir Cette section présente la méthodologie expérimentale de calibrage associée à `BDPO`, qui a pour but d'élaborer la configuration de ce dernier. La configuration en question comprend la définition des seuils sur les métriques `IPC` et `L3RWC`, ainsi que la spécification des points de fonctionnement { tension ; fréquence } haut et bas pour l'application de `DVFS`. De plus, la méthodologie conduit à la construction d'une configuration spécifique à l'architecture matérielle du nœud sur lequel elle est exécutée. Les noyaux de calcul `BLAS`, couramment utilisés dans le domaine du calcul à haute performance, sont utilisés comme proxy des applications `HPC` de sorte à ce que la configuration construite soit adaptée à une vaste majorité d'entre elles.

En plus de permettre la définition des paramètres de `BDPO` mentionnés précédemment, l'exécution de la méthodologie sur la plateforme expérimentale mise à disposition par Atos (présentée par 4.1.1) a permis de tirer les conclusions suivantes : (1) la conjonction de l'`IPC` et du `L3RWC` permettent de séparer les résidences en mémoire et en cache `L3`, (2) une résidence des données en mémoire principale induit une insensibilité significative à la fréquence des cœurs de calcul, vraisemblablement due à l'abondance des phases mémoires, ce qui constitue une opportunité de mise en place de `DVFS`, et (3) lesdites opportunités de mise en place de `DVFS` permettent de réduire significativement la consommation énergétique associée à l'exécution des noyaux de calcul `BLAS`, en maintenant la dégradation des performances à un niveau acceptable du point du vue du cahier des charges de `BDPO`. Il reste à confirmer ces résultats pour des applications `HPC` réelles, et à vérifier que la configuration construite par la méthodologie est bien adaptée aux applications en question.

Notons également que la méthodologie expérimentale de calibrage de `BDPO` permet de garantir que ce dernier soit agnostique des applications `HPC` considérées. Pour finir, précisons qu'elle est entièrement automatisée et exécutable à l'installation de `BDPO` sur le nœud de calcul, de sorte à accompagner les administrateurs des supercalculateurs et à favoriser son adoption par la communauté du calcul à haute performance.

6.2 Optimisation de l'efficacité énergétique d'applications de calcul intensif

La méthodologie de calibrage de BDPO, présentée par la section 6.1, a permis de construire une configuration spécifique à la plateforme expérimentale mise à disposition par Atos, présentée par la sous-section 4.1.1. Il s'agit maintenant de mettre cette configuration à l'épreuve de deux applications de calcul à haute performance pour valider non seulement la méthodologie de calibration, mais surtout le fait que BDPO permette d'optimiser l'efficacité énergétique des applications HPC.

Pour ce faire, HPCG et NEMO [3] ont été exécutées sur la partition de deux nœuds de calcul mise à disposition par Atos, avec un rang MPI attribué à chaque cœur de calcul physique pour un total de 112 rangs MPI.

En parallèle de ces applications HPC, une instance de BDPO a été exécutée par nœud de calcul. La configuration construite via la méthodologie de calibrage a été utilisée : les fréquences associées aux points de fonctionnement { tension ; fréquence } bas et haut sont respectivement 1.10 GHz et 1.90 GHz, le seuil sur le L3RWC est fixé à 3.00, et celui sur l'IPC est fixé à 40.0.

HPCG et NEMO ont été exécutées pour trois « configurations expérimentales » différentes, 15 fois par configuration, pour un jeu de données d'entrée résidant en mémoire principale ($\approx$ 60 Go). Les trois configurations expérimentales en question sont :

[Nominale] Utilisation du point de fonctionnement { tension ; fréquence } associé à la fréquence nominale des processeurs durant toute la durée de l'exécution. C'est la configuration par défaut de la majorité des supercalculateurs ;

[BDPO] Exécution de BDPO en parallèle de l'application HPC, avec une période d'échantillonnage des compteurs de performance égale à 5 ms, mise en place de DVFS entre les points de fonctionnement { tension ; fréquence } haut et bas ;

[TurboBoost] Utilisation du point de fonctionnement { tension ; fréquence } associé à la fréquence nominale des processeurs avec activation forcée du TurboBoost dès que cela est possible.

Toutes ces exécutions ont été chronométrées, et l'énergie qu'elles ont consommée a été monitorée via une sonde matérielle wattmétrique (fréquence d'échantillonnage de 1 Hz et précision de 125 mW). Les tableaux 6.1 et 6.2 présentent les résultats médians des 15 exécutions, normalisés par rapport à la configuration « Nominale », respectivement pour HPCG et NEMO.

Ainsi, BDPO permet de réduire significativement la consommation énergétique associée aux exécutions de HPCG et NEMO, respectivement de 16.5% et 15.0% par rapport à la configuration « Nominale », tout en maintenant l'augmentation du temps d'exécution associé sous les 4.0% (respectivement 2.8% et 3.8%). Cela valide l'approche de BDPO et la conception de sa méthodologie expérimentale de calibrage, puisqu'en utilisant cette dernière il est possible d'améliorer significativement l'efficacité énergétique associée à

3. Présentées par la section 4.2.

l'exécution d'une application HPC tout en gardant la dégradation de performance sous un seuil acceptable.

TABLE 6.1 – Temps d'exécution (T_{exec}) et consommations énergétiques ($\mathbb{E}$) médianes pour 15 exécutions de HPCG, pour les trois configurations expérimentales envisagées.
Les résultats sont normalisés par rapport à la configuration expérimentale « Nominale ».

HPCG	T_{exec} (s)	$\mathbb{E}$ (J)	T_{exec} normalisé	$\mathbb{E}$ normalisée
Nominale	325.26	241 803	1.000	1.000
BDPO	334.33	201 831	1.028	0.8346
TurboBoost	324.26	271 872	0.9969	1.124

Dans un soucis de complétude, on remarque que l'utilisation forcée du TurboBoost n'entraîne pas d'amélioration significative des performances, qu'il peut même légèrement dégrader dans le cas de NEMO. Pour autant, son activation engendre une augmentation significative de la consommation énergétique associée à l'exécution d'une application HPC, ici de l'ordre de 12%. Ces observations confirment celles qui ont été faites par la section 6.1 : l'impact du TurboBoost est globalement largement défavorable du point de vue du ratio { performances / énergie }, et son utilisation semble à proscrire.

TABLE 6.2 – Temps d'exécution (T_{exec}) et consommations énergétiques ($\mathbb{E}$) médianes pour 15 exécutions de NEMO, pour les trois configurations expérimentales envisagées.
Les résultats sont normalisés par rapport à la configuration expérimentale « Nominale ».

NEMO	T_{exec} (s)	$\mathbb{E}$ (J)	T_{exec} normalisé	$\mathbb{E}$ normalisée
Nominale	162.32	133 386	1.000	1.000
BDPO	168.49	113 415	1.038	0.8503
TurboBoost	165.59	149 727	1.020	1.123

6.3 Performances de BDPO et impact sur l'application HPC exécutée

La section 6.2 a montré que BDPO permet de réduire la consommation énergétique associée à une application de calcul à haute performance, tout en maintenant la dégradation de performance engendrée par son action inférieure à 4.0%, relativement à une exécution dans des conditions de référence. Néanmoins, le cahier des charges de BDPO met l'accent sur l'importance de minimiser son impact sur les performances. Il est donc crucial de déterminer les causes des dégradations des performances des applications HPC engendrées par BDPO, afin d'être en mesure de les minimiser.

Les dégradations de performance en question peuvent avoir deux sources : (1) l'exécution de BDPO en parallèle de l'application HPC induit une utilisation significative des cœurs de calcul, et/ou (2) certaines opportunités de DVFS détectées par BDPO sont des « faux-positifs » et se révèlent contre-productives.

Afin de se prémunir contre la première des sources qui viennent d'être mentionnées, le cahier des charges de BDPO stipule que son implémentation doit être légère. Le but des expériences présentées par cette section est de déterminer si BDPO en lui-même a un impact sur les performances de l'application dont il cherche à optimiser l'efficacité énergétique.

Pour ce faire, la capacité de reconfiguration de BDPO a été désactivée, et sa boucle principale a été annotée de sorte à pouvoir mesurer le temps d'exécution d'une de ses itérations, via `clock_gettime` pour l'horloge `CLOCK_MONOTONIC_RAW`. BDPO a ensuite été exécuté [4], pour différentes valeurs de la période d'échantillonnage des PMC, sur un des nœuds de calcul de la partition mise à disposition par Atos (présenté par la sous-section 4.1.1). Pour chacune de ces périodes d'échantillonnage, les temps d'exécution de 10000 itérations de la boucle principale ont été mesurés, et les résultats sont présentés synthétiquement par la figure 6.6 grâce à un diagramme « boîte à moustaches ».

On s'aperçoit que la durée de la boucle principale de BDPO semble totalement décorrélée de la période d'échantillonnage des compteurs de performance. De plus, les durées médianes des boucles principales se situent entre 240 μs et 290 μs, ce qui représente environ 4% d'une période d'échantillonnage de 5 ms. Cela semble indiquer que l'implémentation de BDPO répond au critère de légèreté de son cahier des charges.

Pour valider cette observation, il a donc fallu quantifier l'impact des mécaniques internes de BDPO sur les performances de l'application de calcul à haute performance dont on cherche à améliorer l'efficacité énergétique. Pour cela, BDPO a été exécuté en parallèle de HPCG, ses capacités de reconfiguration ayant été désactivées. De plus, plusieurs valeurs de la période d'échantillonnage de sa boucle principale ont été évaluées, et HPCG a été exécutée 15 fois pour chacune d'entre elles. Le tableau 6.3 présente les résultats de ces expérimentations. Les valeurs présentées sont les médianes pour les 15 exécutions, et les variations des performances de HPCG induites par les mécaniques internes de BDPO sont évaluées par rapport à la configuration « Défaut », qui correspond à une exécution de HPCG pour laquelle BDPO n'est pas exécuté en parallèle.

On peut observer que les variations des performances de HPCG induites par l'exécution de BDPO sont faibles, puisque globalement de l'ordre de 0.5%. De plus, il y a même,

4. Par lui-même, et non pas en parallèle d'une application HPC.

pour une période d'échantillonnage de 50 ms, une subtile amélioration de ces dernières. Il apparaît donc que les perturbations générées par l'exécution de `BDPO` en parallèle d'une application `HPC` sur les performances de cette dernière sont indiscernables de celles induites par l'action du système d'exploitation. En conclusion, les dégradations de performance observées dans le cadre de la section 6.2 semblent être imputables à la mise en place de `DVFS` dans des situations où cela s'avère contre-productif. Cette problématique sera au cœur de la sous-section 6.4.2.

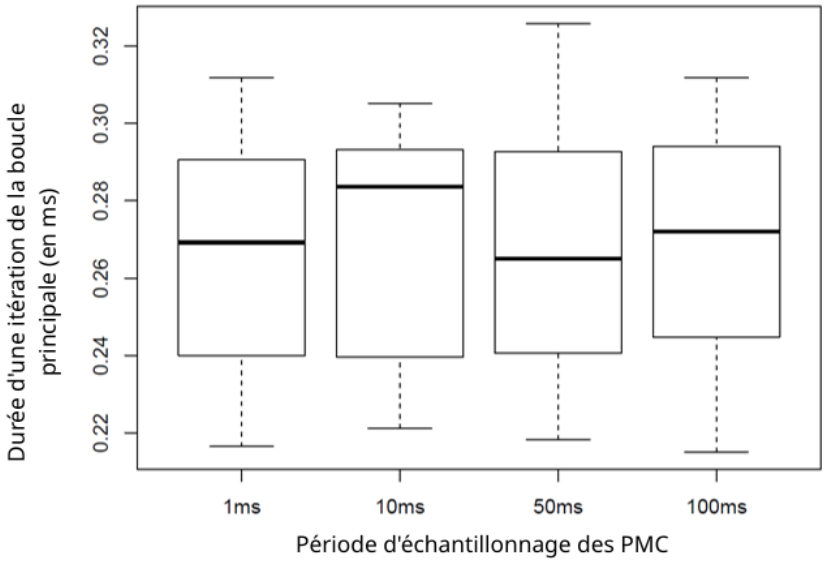


FIGURE 6.6 – Évaluation de la durée d'une itération de la boucle principale de `BDPO` pour différentes périodes d'échantillonnage des compteurs de performance, sur un nœud de calcul « à vide » et sans reconfiguration. Pour chaque période d'échantillonnage, les durées de 10000 itérations de la boucle principale ont été mesurées.

TABLE 6.3 – Impact de la boucle principale de monitoring des compteurs de performances de `BDPO` sur les performances de `HPCG`. Plusieurs périodes d'échantillonnage ont été évaluées. Les valeurs présentées sont les médianes pour 15 exécutions, et les variations de performance sont évaluées par rapport à la configuration « Défaut », qui correspond à une exécution de `HPCG` pour laquelle `BDPO` n'est pas exécuté en parallèle.

Période d'échantillonnage	Performances ($GFlop/s$)	Variation (%)
Défaut	49.71	+0.00%
100 ms	49.57	−0.27%
50 ms	49.74	+0.06%
10 ms	49.44	−0.53%
1 ms	49.47	−0.48%

6.4 Bilan des expériences

Après avoir récapitulé les résultats clés des expérimentations présentées par ce chapitre dans la sous-section 6.4.1, le concept de reconfiguration prédictive, moyen de perfectionnement envisagé pour BDPO, sera introduit par la sous-section 6.4.2. Enfin, un retour sur l'étude de l'état de l'art concernant les outils de reconfiguration dynamique à visée énergétique sera au cœur de la sous-section 6.4.3, afin de comparer BDPO aux solutions existantes.

6.4.1 Points clés

L'heure est venue de conclure ce chapitre dédié aux travaux empiriques articulés autour de BDPO. Pour commencer, il faut souligner que la méthodologie expérimentale de calibrage de BDPO permet de lui construire une configuration spécifique à une architecture matérielle de nœud de calcul. Grâce au protocole mis en place, exécutable automatiquement à l'installation de BDPO sur un nœud de calcul, il est possible d'identifier les points de fonctionnement { tension ; fréquence } intéressants du point de vue de l'efficacité énergétique, et de sélectionner des valeurs pour les seuils sur l'IPC et le L3RWC qui permettent de détecter les phases mémoires, exploitables via DVFS. De plus, du fait que la méthodologie de calibrage s'appuie sur les noyaux de calcul BLAS, briques de construction communes à de nombreuses applications de calcul à haute performance, la configuration en question est adaptée à une large gamme d'applications HPC.

Et cela a été confirmé empiriquement, puisque l'utilisation de BDPO avec la configuration construite via sa méthodologie de calibrage a permis de réduire la consommation énergétique de deux applications HPC réelles, HPCG et NEMO, d'au moins 15% tout en gardant les dégradations de performance associées sous la barre des 4%.

L'impact de BDPO sur les performances des applications dont il cherche à améliorer l'efficacité énergétique a alors été évalué. Il s'est avéré que l'implémentation de la boucle principale de BDPO donne à cette dernière une empreinte suffisamment faible pour que les perturbations de performance induites par les mécaniques internes de BDPO soient indiscernables de celles induites par l'action du système d'exploitation. En conséquence, les dégradations de performance mentionnées en sus semblent imputables à la mise en place de DVFS dans des cas où cela est contre-productif.

Ainsi, ce chapitre a été l'occasion de prouver expérimentalement que le cahier des charges associé à BDPO est rempli : c'est un outil de reconfiguration dynamique léger qui permet d'améliorer l'efficacité énergétique des applications de calcul à haute performance tout en restant agnostique de ces dernières, et en gardant les dégradations de performance associées à son action sous un seuil acceptable.

6.4.2 Axe d'amélioration : des reconfigurations prédictives

Comme cela a été mentionné précédemment, les dégradations de performance engendrées par l'action de BDPO sont vraisemblablement dues à la mise en place de DVFS lorsque la situation ne s'y prête pas. En effet, si BDPO diminue la fréquence des cœurs de calcul pendant une phase sensible à la fréquence, l'exécution de cette dernière prendra plus de

temps que si elle avait été exécutée par des cœurs de calcul fonctionnant à leur fréquence nominale.

Une telle situation peut survenir dans deux cas de figure : (1) `BDPO` détecte une phase mémoire là où il n'en est rien (faux-positif de détection), et (2) le fait que les reconfigurations auxquelles `BDPO` procède sont **réactives** induit que la mise en place de `DVFS` lors de phases mémoires puisse être contre-productif. Deux pistes à explorer donc. Néanmoins, la seconde constitue une problématique qui peut être commune à l'ensemble des outils de reconfiguration lorsque la première est spécifique à `BDPO`. C'est donc cette seconde problématique qui a été étudiée dans la suite des travaux de recherche. Attardons-nous sur cette dernière.

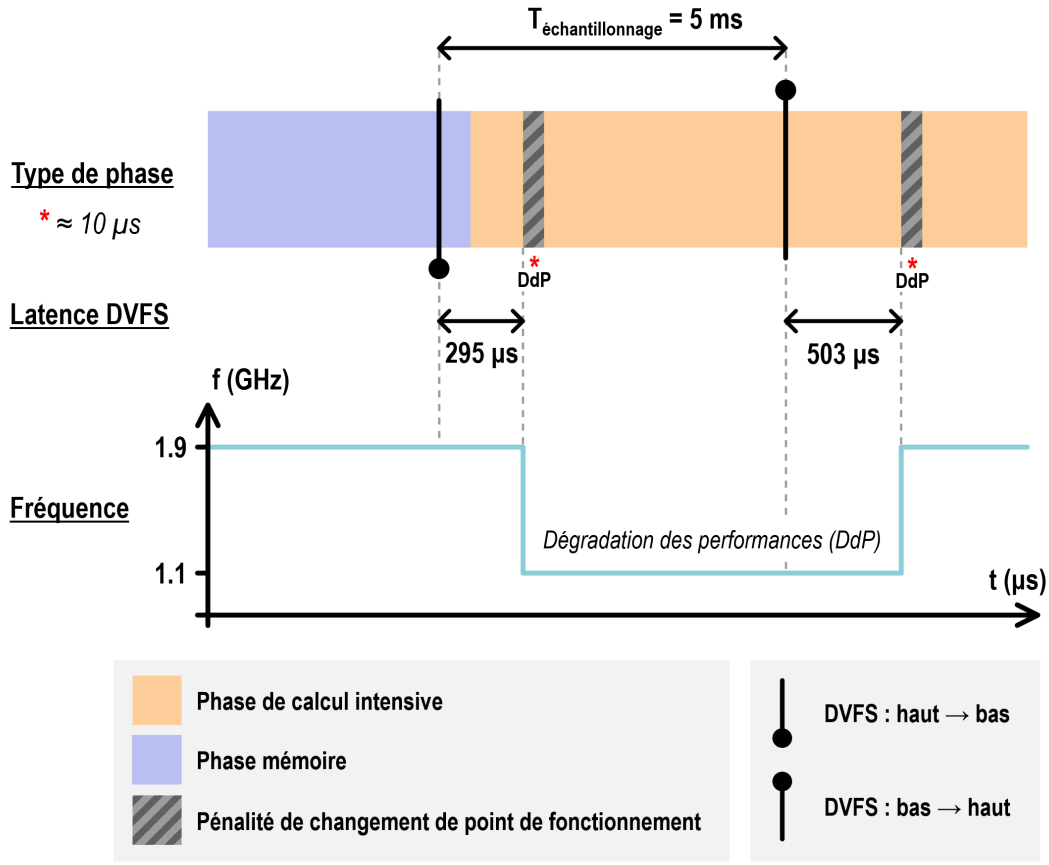


FIGURE 6.7 – Schéma illustrant une des problématiques liées au fait de mettre en place du `DVFS` **réactif** : le changement de point de fonctionnement { tension ; fréquence } n'étant pas immédiat, il est possible qu'il intervienne après que l'opportunité de reconfiguration soit terminée.

La figure 6.7 illustre la problématique en question. Pour le formuler simplement, `BDPO` met en place des reconfigurations **réactives** : elles s'appuient sur les valeurs « instantanées » des métriques `IPC` et `L3RWC`, sans avoir aucune garantie sur le fait que ces valeurs vont être représentatives de l'intégralité de la période d'échantillonnage qui va suivre. En d'autres termes, il est tout à fait possible que `BDPO` échantillonne les compteurs de performance quelques micro-secondes avant qu'une phase mémoire ne se termine, et qu'une

phase de calcul intensive lui fasse suite. Dans ce cas, jusqu'au prochain échantillonnage, soit pendant la durée d'une période d'échantillonnage, une phase de calcul intensive sera exécutée avec un point de fonctionnement { tension ; fréquence } bas, alors qu'elle est fortement sensible à la fréquence des cœurs de calcul. Cela se traduira donc par une dégradation des performances.

Dans la même veine, un entrelacement de phases mémoires et de phases de calcul intensives de courtes durées peut conduire à la mise en place contre-productive de `DVFS`. En effet, une telle configuration conduit à des oscillations des valeurs des métriques autour des seuils qui leurs sont associés. En conséquence, des changements de point de fonctionnement { tension ; fréquence } s'enchaînent à un rythme soutenu. Et même si, à tout instant, le point de fonctionnement { tension ; fréquence } en vigueur est adapté au type de phase en cours d'exécution, l'accumulation des pénalités [5] de performances associées aux changements de point de fonctionnement engendre une dégradation de performance notable. Par exemple, [Sourouri 2017] relate que lesdites dégradations peuvent atteindre 50% du temps d'exécution de référence (i.e. pour une exécution sans mise en place de `DVFS`) lorsque les injonctions de reconfigurations sont intégrées aux noyaux de calcul de l'application en question.

Pour apporter une solution à cette problématique, il faut être capable de prédire le comportement à venir de l'application pour déterminer à priori si la reconfiguration que `BDPO` s'apprête à mettre en place s'avèrera réellement favorable. On parlera alors de reconfiguration **prédictive**. Munir `BDPO` de telles capacités de prédiction est précisément le but de `Phase - Temporality Analyser`, qui est l'objet de la partie III.

6.4.3 Comparaison à l'état de l'art

Le fonctionnement de `BDPO` ayant été présenté, et la capacité de ce dernier à améliorer l'efficacité énergétique associée à l'exécution d'une application `HPC` ayant été évaluée, il est maintenant possible de le comparer aux outils de reconfiguration introduits lors de l'état de l'art (cf. section 3.2).

`DVFS Governor` - présenté par [Da Costa 2015] L'approche sous-jacente à `DVFS Governor` est similaire à celle de `BDPO`, dans la mesure où toutes deux : (1) s'appuient sur le monitoring de métriques représentatives de l'état d'utilisation des ressources matérielles mises à disposition par le nœud de calcul, (2) utilisent le `DVFS` comme levier pour améliorer l'efficacité énergétique associée à une application `HPC`, et (3) sont accompagnées d'une caractérisation préalable du nœud de calcul.

Néanmoins, l'approche de `DVFS Governor` repose sur la modélisation d'une application `HPC` comme une succession en alternance de phases de calcul, et de phases de communications inter-nœuds. De plus, ces phases de communications sont considérées insensibles à la fréquence des cœurs de calcul, et complètement disjointes des phases de calcul. Or, les applications `HPC` et les implémentations de `MPI` récentes mettent notamment l'accent sur le recouvrement des communications inter-nœuds par des phases de calcul. À cela s'ajoute le fait que les travaux présentés par [Da Costa 2015] ont été menés sur une

5. Pour rappel, un changement de point de fonctionnement { tension ; fréquence } entraîne une purge du `pipeline` du cœur de calcul, ce qui induit une dégradation brute des performances.
Pour plus de détails, se référer à [Mazouz 2013].

architecture utilisant des liens Ethernet pour interconnecter les nœuds de calcul. Les durées des phases de communication ont donc tendance à être beaucoup plus longues que lorsqu'elles sont effectuées via un réseau d'interconnexion rapide tel que ceux présentés par la sous-section 2.1.3 [6]. En conséquence, les portions de l'exécution d'une application qui seront exploitables seront moindres, l'approche de DVFS Governor paraît donc peu compatible avec le domaine du calcul à haute performance à l'ère de l'ExaScale.

E-AMOM - présenté par [Lively 2014] La méthodologie associée à E-AMOM s'appuie sur le monitoring d'évènements via les compteurs de performance des processeurs pour construire des modèles de performance et de consommation électrique associées à l'exécution d'une application HPC. Ces modèles servent ensuite à identifier les opportunités de mise en œuvre de DVFS et de DCT. Les reconfigurations associées aux dites opportunités sont intégrées directement au code source de l'application. Ce dernier est également optimisé localement par les auteurs de [Lively 2014].

Ainsi, de fait, l'approche sous-jacente à E-AMOM est incompatible avec le cahier des charges de BDPO, qui accorde la plus grande importance au fait de demeurer agnostique de l'application. Néanmoins, les expérimentations présentées par [Lively 2014] montre qu'il est possible de réduire de manière consistante la consommation énergétique associée à l'exécution d'une application en appliquant du DVFS, tout en préservant les performances de ladite application. Et ce résultat est significatif, et extrêment positif, lorsque l'on considère l'approche de BDPO.

EAR - présenté par [Corbalan 2020] Le fonctionnement du composant EARL d'EAR se rapproche véritablement de la synergie souhaitée entre BDPO et Phase-TA (présenté par la partie III). La principale différence, et elle est de taille, concerne l'échelle à laquelle les approches de BDPO et EARL s'appliquent. En effet, par défaut, pour EARL, le monitoring des métriques et les reconfigurations des points de fonctionnement { tension ; fréquence } des cœurs de calcul sont effectués à l'échelle de plusieurs itérations d'une boucle de l'application exécutée. Ce qui peut se traduire par une granularité à l'échelle de la dizaine de secondes dans certains cas. De surcroît, comme les reconfigurations mises en œuvre par EARL s'appliquent à l'échelle d'une boucle, elles sont adaptées au comportement majoritaire, mais pas nécessairement à l'ensemble des comportements. Par exemple, si, schématiquement, une boucle est constituée de 60% de phases de calcul intensives, et de 40% de phases d'accès mémoire, le comportement moyen observé par EARL sera mixte, avec une légère prédominance des phases de calcul. La reconfiguration associée à cette observation induira alors généralement, de manière inhérente, des dégradations de performance, qui seront maintenues sous un seuil défini comme acceptable grâce à la boucle de rétroaction.

Au contraire, BDPO demeure complètement agnostique de la structure de l'application, et ne considère que les valeurs « instantanées » des métriques pour mettre en œuvre du DVFS à l'échelle de la dizaine de millisecondes. De plus, les reconfigurations qu'il met en œuvre semblent s'avérer plus bénéfiques du point de vue de l'amélioration de l'efficacité énergétique associée à l'exécution d'une application HPC.

6. À titre d'exemple, une majorité des communications inter-nœuds de l'application QuantumExpresso semblent durer moins de 500 μs d'après [Cesarini 2020].

Ainsi, bien que les approches de BDPO et EAR soient très similaires, leurs mises en œuvre se traduisent par des principes de fonctionnement fondamentalement différents : EAR cherche à déterminer le point de fonctionnement { tension ; fréquence } le plus adapté au comportement moyen d'une application à l'échelle d'une de ses boucles, lorsque BDPO cherche à appliquer le point de fonctionnement { tension ; fréquence } le plus adapté à l'état d'utilisation du matériel à l'instant considéré.

MREEF - présenté par [Chetsa 2015] Tout comme BDPO, MREEF se base sur le monitoring de métriques et s'appuie sur une caractérisation préalable du matériel pour identifier les « phases »[7] propices à une reconfiguration dynamique des composants matériels du nœud de calcul. La principale différence entre les deux outils est que les mécaniques internes de MREEF sont plus complexes, algorithmiquement parlant, que celles de BDPO, notamment en raison de l'analyse en composantes principales calculée par MREEF. C'est fort probablement une des raisons qui forcent MREEF à se cantonner à une fréquence d'échantillonnage de 1 Hz, contrairement à BDPO qui peut échantillonner les métriques qu'il monitore à une fréquence allant jusqu'à 200 Hz.

De plus, MREEF implémente des reconfigurations réactives, tout comme BDPO actuellement. Couplé à la période d'échantillonnage de 1 seconde, cela signifie que toute reconfiguration contre-productive reste en vigueur pendant 1 seconde complète (contre 5 millisecondes pour BDPO). C'est probablement ce qui explique que les dégradations de performance engendrées par l'utilisation de MREEF sont plus importantes (presque le double, en moyenne) que celles induites par l'utilisation de BDPO, pour des gains énergétiques similaires. Ainsi, il est fort probable que MREEF gagnerait également à être doté de capacités de prédiction du comportement à venir de l'application, à l'instar de ce que Phase-TA apportera à BDPO.

READEX - présenté par [Schuchart 2017] L'approche de READEX se rapproche de celle d'E-AMOM, dans la mesure où toutes deux impliquent la construction de modèles de performance et de consommation électrique et/ou énergétique pour permettre une recherche de la configuration optimale pour une fonction de coût donnée. Néanmoins, READEX propose une collection d'outils automatisant le processus, notamment en ce qui concerne l'annotation du code source de l'application.

À l'instar de ce qui a été dit pour E-AMOM, READEX ne répond pas au cahier des charges de BDPO, notamment parce que la modification extensive du code source de l'application, bien qu'automatisée, va à l'encontre du principe d'agnosticisme. À cela s'ajoute le fait que l'approche de READEX implique une forte complexité algorithmique, et s'accompagne d'un impact sur les performances trop élevé pour être jugé négligeable, voire acceptable du point de vue du cahier des charges de BDPO.

7. Les définitions de la notion de « phase » utilisées par BDPO et MREEF diffèrent subtilement, mais véhiculent à peu de choses près la même phénoménologie.

Troisième partie

Phase-TA : caractériser les comportements localement périodiques des applications de calcul intensif

7 Méthodologie implémentée par Phase-TA

Ce chapitre décrit en profondeur le fonctionnement de Phase-TA, qui s'appuie notamment sur un ensemble de concepts et de notions qui sont définis par la section 7.1. À l'aide de ces définitions, il est alors possible de spécifier, dans la section 7.2, les motivations et objectifs sous-jacents à la conception de Phase-TA. Il sera alors temps de décrire en grande largeur la méthodologie en trois étapes implémentée par Phase-TA pour détecter et modéliser les comportements localement périodiques exhibés par les séries temporelles. La

section 7.3 donne un aperçu global de la méthodologie, alors que les sections 7.4, 7.5, 7.6 vont chacune dans le détail d'une de ses trois étapes. Avant de parachever ce chapitre en rappelant les points clés concernant `Phase-TA` dans la section 7.7 parachève ce chapitre en rappelant les points clés concernant `Phase-TA` au fil de l'analyse d'un profil `HPC` réel, illustrée étape par étape.

7.1 Définitions préliminaires

Cette section définit un ensemble de concepts et de notions qui servent de socle à l'approche de `Phase-TA` et sont essentiels à la compréhension de son fonctionnement.

7.1.1 Profil applicatif et échantillons

Un **profil applicatif** est l'évolution temporelle d'une métrique caractéristique de l'état d'utilisation du matériel lors de l'exécution d'une application `HPC`. Ainsi, un profil applicatif est une **série temporelle**. A titre d'exemple, la figure 7.2 présente trois profils applicatifs, chacun décrivant l'évolution temporelle d'une métrique différente représentant l'état d'utilisation de sous-systèmes du processeur.

Les points d'un profil applicatif sont également appelés des **échantillons**, ces derniers étant fréquemment, pour ne pas dire toujours, issus d'un procédé d'échantillonnage. Dans le cas présent, ils sont issus d'un échantillonnage des compteurs de performance des processeurs d'un nœud de calcul de `dahu`.

7.1.2 Périodicité

Notion centrale des travaux de recherche associés à `Phase-TA`, une **périodicité** est un motif minimal qui survient régulièrement dans une série temporelle, engendrant des comportements localement périodiques.
Or, les séries temporelles à valeurs réelles sont généralement associées à l'évolution temporelle de grandeurs tangibles, telles que la température monitorée par un capteur, ou les cours des marchés financiers. Elles sont donc le plus souvent intrinsèquement soumises à des variations aléatoires assimilables aux bruits d'insertion, de suppression, et de modification présentés par la sous-section 7.1.3. En conséquence, du fait de ces variations aléatoires, il est impossible de remonter au motif exact associé à une périodicité, également appelé **expression de référence**, puisque chacune de ses occurrences est bruitée de manière imprédictible.

7.1.3 Bruits d'insertion, de modification et de suppression

Les **bruits** d'**insertion**, de **suppression** et de **modification** constituent une représentation qualitative de la manière dont des variations aléatoires peuvent modifier une occurrence d'un motif au sein d'une série temporelle, par rapport à une occurrence de référence. Ces trois types de bruit, qui sont décrits ci-dessous, sont également illustrés par la figure 7.1 :

[Bruit d'insertion] un ou plusieurs échantillons ont été ajoutés à une occurrence d'un motif, par rapport à l'expression de référence du dit motif ;

[Bruit de suppression] un ou plusieurs échantillons ont été supprimés d'une occurrence d'un motif, par rapport à l'expression de l'occurrence de référence du dit motif ;

[Bruit de modification] un ou plusieurs échantillons d'une occurrence du motif ont des valeurs différentes des échantillons correspondant dans l'expression de l'occurrence de référence du dit motif.

Les bruits d'insertion et de suppression correspondent à des perturbations temporelles, par la suite respectivement appelées dilatations et compressions. En effet, l'insertion d'échantillons résulte généralement de l'allongement temporel d'un phénomène, par exemple une congestion sur les réseaux de communication induit une latence qui retarde un ordre d'achat sur un marché financier. Or, les échantillons sont issus d'un échantillonnage à fréquence fixe, il y aura donc plus d'échantillons associés au phénomène considéré. De manière similaire, la suppression d'échantillons résulte généralement du raccourcissement temporel d'un phénomène, qui implique que moins d'échantillonnages auront lieu pendant la durée du phénomène.

Le bruit de modification, quant à lui, résulte généralement des variations aléatoires inhérentes aux phénomènes engendrant les séries temporelles, e.g. les flux d'air d'une pièce engendrent de légères variations de la température monitorée par un capteur.

7.1.4 Motif représentatif

Un des enseignements de la définition de ce qu'est une périodicité offerte par la sous-section 7.1.2 est qu'il est généralement impossible de déterminer le motif exact associé à une périodicité car chacune de ses occurrences est bruitée aléatoirement.
Une approche classique est alors de construire un **motif représentatif** de la périodicité à partir de ses occurrences observables. En d'autres termes, un motif représentatif est un motif qui se veut similaire au motif associé à la périodicité. Similarité qui croît avec le nombre d'occurrences de ladite périodicité à partir desquelles le motif représentatif est inféré. La principale raison d'être d'un motif représentation est bien évidemment de servir de substitut au motif associé à une périodicité, ce dernier étant inaccessible.

7.1.5 Région périodique

Une **région périodique** est un ensemble d'au moins deux occurrences consécutives d'une périodicité. Toute région périodique est donc associée à une unique périodicité. De plus, du fait que les occurrences considérées doivent être consécutives, une région périodique constitue un « comportement localement périodique ». À titre d'exemple, le profil applicatif représentant l'évolution temporelle du nombre d'instructions retirées par cycle de référence, en violet, présenté par la figure 7.2, exhibe deux régions périodiques, mises en valeur en bleu.

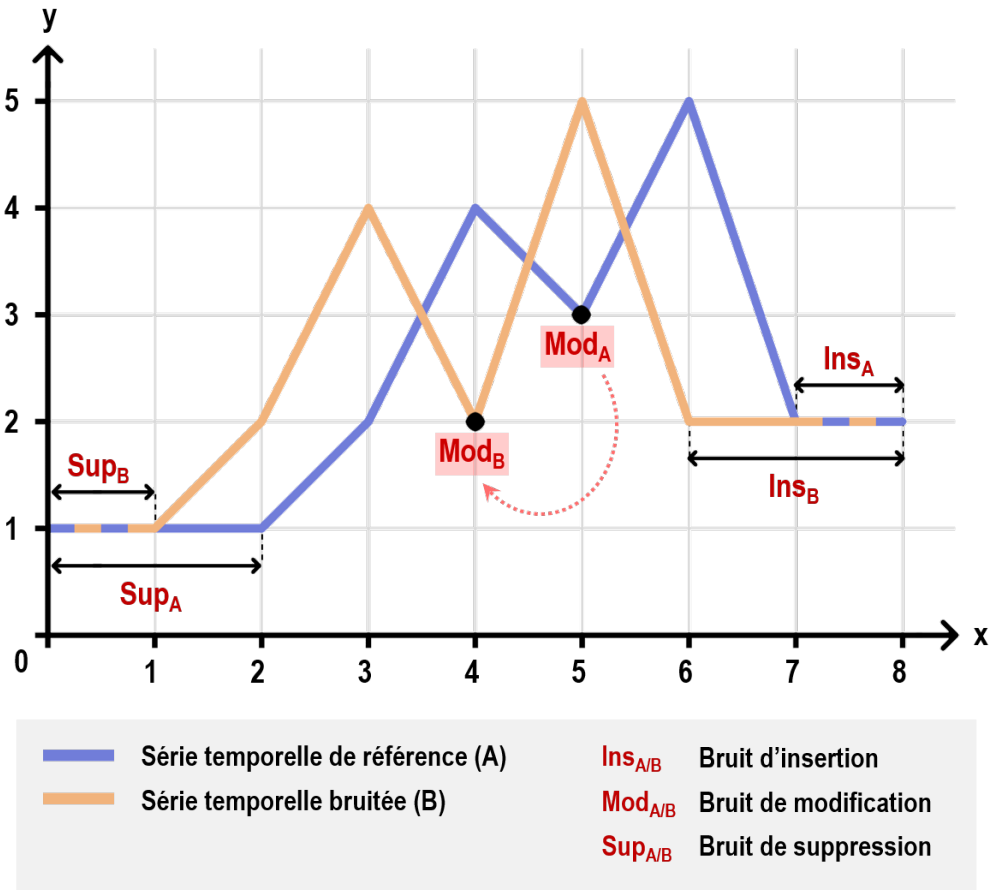


FIGURE 7.1 – Schéma illustrant les différents types de bruit : la courbe violette est une série temporelle de référence, à laquelle ont été appliqués, dans l'ordre, des bruits de suppression, de modification, et d'insertion, pour obtenir la courbe orange.

7.1.6 Instance périodique

Une **instance périodique** est une occurrence d'une périodicité appartenant à une région périodique, telle que définie par la sous-section 7.1.5. Afin d'illustrer cette définition, la figure 7.2 met en valeur 6 instances périodiques, délimitées par des barres verticales anthracites au sein des régions périodiques dessinées en bleu. Il est important de noter que l'occurrence de la périodicité mise en valeur en rouge **n'est pas** une instance périodique, car elle est isolée et n'appartient donc pas à une région périodique.

7.1.7 Retour sur le `WGSS`

Reprenons les notations introduites par la sous-section 2.3.4, à savoir p une série temporelle à valeurs réelles, et $\mathcal{I}$ un ensemble de séries temporelles à valeurs réelles. Alors, comme mentionné par ladite sous-section, le `WGSS` est un indicateur de la distance cumulée de la série temporelle p à l'ensemble de séries temporelles $\mathcal{I}$.

Or, une instance périodique est un extrait d'une série temporelle, et est donc une série temporelle en tant que telle. De la même manière, un motif représentatif est une série temporelle. Ainsi, si $\mathcal{I}$ représente un ensemble d'instances périodiques engendrées par une

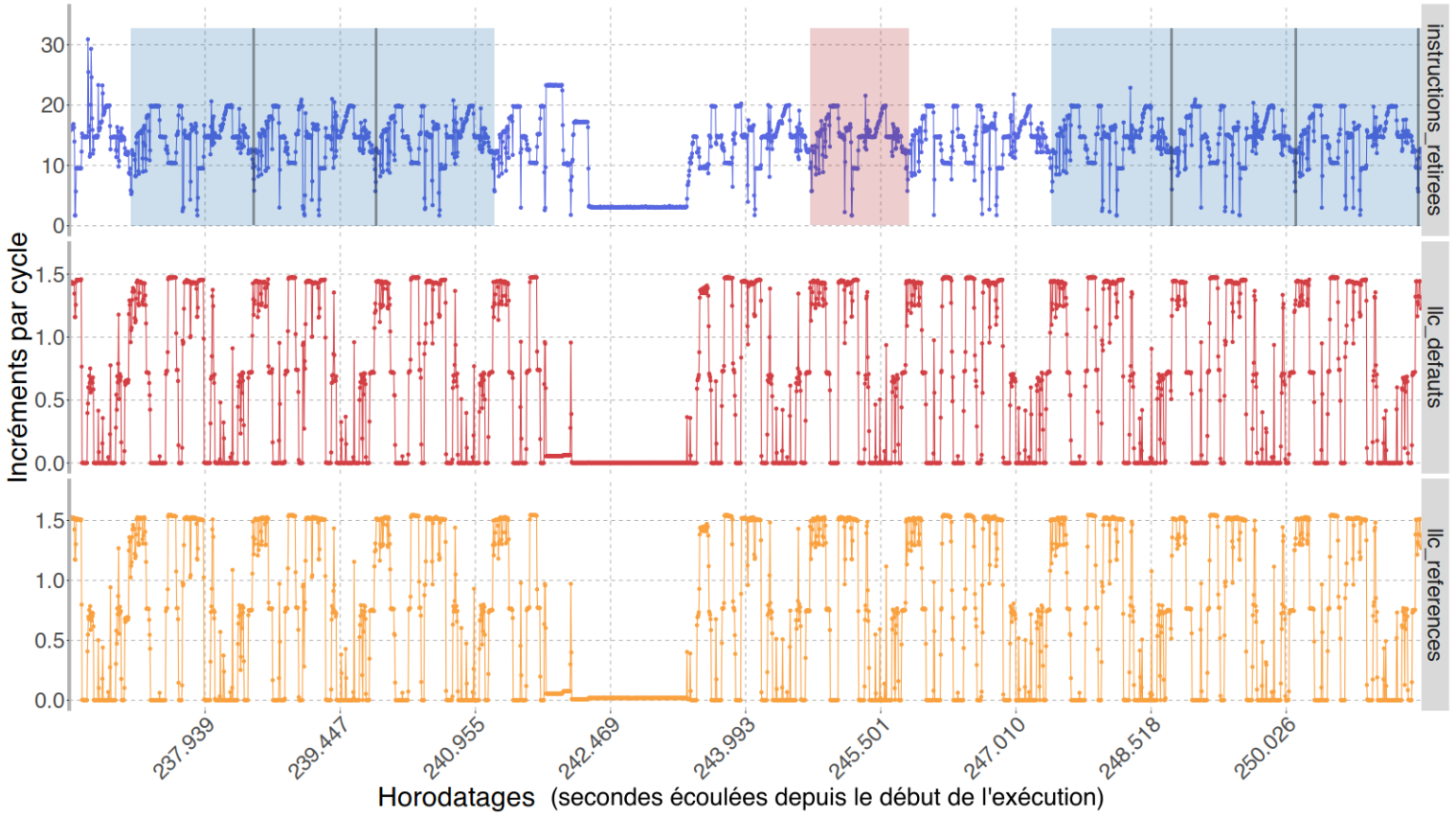


FIGURE 7.2 – Extraits de l'évolution temporelle du nombre d'incréments par cycle de référence de trois métriques (de haut en bas : le nombre d'instructions retirées, le nombre de défauts de cache L3, et le nombre de références en cache L3), agrégées à l'échelle d'un nœud de `dahu`, pour une exécution de `HPCG`.
Chaque graphe est un profil applicatif, les zones bleues sont des régions périodiques contenant des instances périodiques délimitées par les traits verticaux anthracites, et la zone rouge met en valeur une occurrence isolée de la périodicité, qui n'est donc pas une instance périodique.

périodicité, et p est un motif représentatif pour la périodicité en question, le calcul de $WGSS(p, \mathcal{I})$ permet **d'évaluer la pertinence** de p en tant que substitut pour les instances périodiques engendrées par la périodicité considérée, et donc la pertinence de p en tant que motif représentatif de cette dernière.

7.1.8 Analyse sur données froides versus analyse sur données chaudes

Il existe plusieurs taxonomies pour classer et caractériser les types de données, et les différentes analyses qu'on peut leur appliquer. Ce paragraphe se concentre sur une dichotomie opposant **l'analyse sur données froides**, ou analyse post-mortem, à **l'analyse sur données chaudes** ou analyse à l'exécution (ou « au runtime » pour ceux qui ne sont pas effrayés par un anglicisme). Un même jeu de données peut être candidat à une analyse à froid et à une analyse à chaud. La différence résidant dans le contexte, et plus précisément l'instant, où l'analyse est effectuée.

D'une part, une analyse à froid implique que le processus à l'origine de la création des données est terminé, et que le jeu de données associé est figé. En conséquence, il n'y a généralement pas de contrainte temporelle forte associée à une analyse à froid (bien qu'il y ait d'autres contraintes, telles que le stockage des jeux de données).

Une analyse à chaud, quant à elle, est réalisée pendant que le processus à l'origine de la création des données est toujours en cours d'exécution. Plusieurs types d'analyse à chaud existent : par paquets, en flux (en « streaming » pour les anglophones), avec ou sans agrégation des données, etc. Le fait que l'analyse soit effectuée en parallèle de l'exécution du processus qui produit les données rajoute de nombreuses contraintes par rapport à l'analyse à froid, notamment temporelles. En effet, pour que l'analyse à chaud soit possible, il faut que le processus d'analyse ait une durée compatible avec l'échelle temporelle associée au processus de création des données. Autrement dit, plus les données sont produites rapidement et à grande échelle, et plus l'analyse doit être efficace et rapide. Cette contrainte temporelle trouve généralement sa source dans le fait que les résultats de l'analyse des données servent de données d'entrée à un processus ayant une temporalité similaire et/ou liée à celle du processus de création de données à analyser.

7.2 Motivations et objectifs

Comme cela a été détaillé par les sections 1.2 et 6.4.2, le but original de `Phase-TA` est de fournir à `BDPO` la capacité de mettre en place des reconfigurations prédictives en exploitant le fait que les applications `HPC` itératives exhibent des comportements localement périodiques. Ainsi, en s'appuyant notamment sur les définitions formulées précédemment par la section 7.1, on peut condenser l'approche de `Phase-TA` en une phrase : détecter les régions périodiques dans les profils applicatifs pour construire, à partir de ces dernières, des motifs représentatifs des périodicités exhibées par les applications `HPC` associées.

Si l'intention initiale était uniquement de doter `BDPO` de capacités prédictives, il est vite apparu que l'étude des comportements périodiques dépassait le cadre de `BDPO`. En effet, à l'échelle du domaine du calcul à haute performance, la problématique de la prédiction des futurs échantillons d'un profil applicatif est commune à l'écrasante majorité des outils de reconfiguration dynamique, comme en témoignent notamment [Isci 2006] et [Chetsa 2015]. Et même lorsque l'on sort du `HPC`, l'étude des comportements périodiques est une thématique d'intérêt pour bon nombre de domaines impliquant la manipulation de séries temporelles, à l'instar de la prédiction des tendances d'évolution des marchés financiers. C'est pourquoi `Phase-TA` a été conçu de sorte à s'abstraire complètement de `BDPO`, et à pouvoir analyser n'importe quelle série temporelle. Il peut ainsi répondre à une double motivation : fournir à `BDPO` des capacités prédictives et permettre l'analyse des comportements périodiques exhibés par les séries temporelles.

Des objectifs supplémentaires sont venus se greffer au fil des travaux de recherche associés à `Phase-TA`. En effet, posséder un tel outil a permis d'ajouter l'exploration expérimentale de thématiques connexes à l'aspect périodique des applications de calcul à haute performance à la liste des tâches à accomplir. Cela a notamment conduit à l'étude de la question suivante : est-ce que les comportements localement périodiques des applications `HPC` dominent leurs temps d'exécution, ou sont-ils marginaux ?

Puisque, comme cela a été mentionné par la sous-section 7.1.2, il est impossible d'accéder à l'expression exacte des motifs associées aux périodicités exhibées par les profils applicatifs, évaluer la pertinence des motifs représentatifs construits par `Phase-TA` s'avèra tâche ardue. Ardue mais essentielle, étant donné que lesdits motifs représentatifs seront amenés à servir de base à d'autres outils et exploitations, tels que `BDPO`. C'est pourquoi l'étude, sous plusieurs angles, de la pertinence des motifs représentatifs inférés vis-à-vis des périodicités associées a été un objectif clé.

Si les performances de `Phase-TA` et la complexité algorithmique de la méthodologie qu'il implémente ne figurent pas en haut de la liste des priorités en ce qui concerne l'analyse à froid, c'est une toute autre histoire pour l'analyse à chaud d'une série temporelle. En effet, comme précisé par la sous-section 7.1.8, l'analyse à chaud nécessite que `Phase-TA` puisse analyser un profil applicatif d'intérêt en un temps restreint. Or, la capacité d'analyser des données à chaud est nécessaire pour que son fonctionnement soit compatible avec celui de `BDPO`. Optimiser le code source de `Phase-TA` s'est donc avéré être un objectif incontournable.

7.3 Principe de fonctionnement de `Phase-TA`

La méthodologie qu'implémente `Phase-TA` est constituée de trois étapes, comme la figure 7.3 l'illustre.

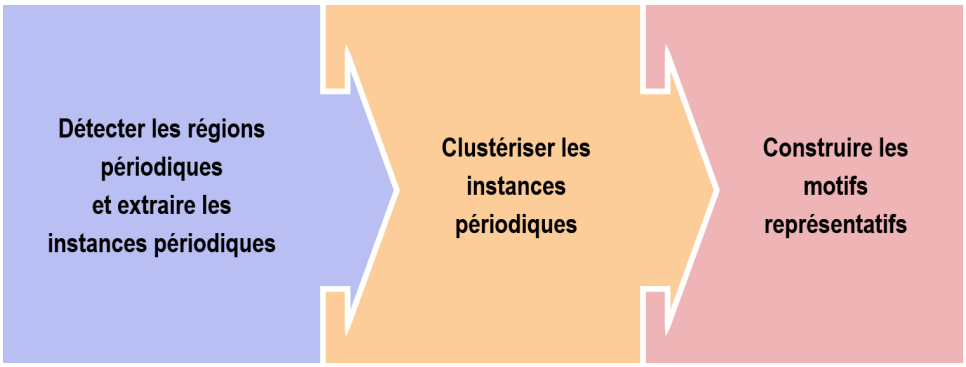


FIGURE 7.3 – Schéma illustrant les trois étapes de la méthodologie implémentée par `Phase-TA`, ainsi que leur enchaînement.

Pour commencer, le profil applicatif est analysé par morceaux pour détecter les régions périodiques et en extraire les instances périodiques. La section 7.4 explique pourquoi il est nécessaire, d'utiliser une approche « diviser pour régner » lors de cette étape. Une fois les instances périodiques extraites, il convient de les clustériser. En effet, chaque périodicité peut engendrer plusieurs régions périodiques disjointes. Et des régions périodiques issues de périodicités différentes peuvent être entrelacées : une région périodique associée à la première périodicité peut par exemple être intercalée entre deux régions périodiques associées à une seconde périodicité. Enfin, les instances périodiques sont moyennées par cluster, de sorte à construire un motif représentatif par périodicité.

Ces trois étapes s'enchaînent, ne sont pas parallélisables, et la méthodologie n'est pas applicable sur un flux de données. Par contre, il est possible d'appliquer cette approche à n'importe quelle série temporelle, même multidimensionnelle. En revanche, l'implémentation de `Phase-TA` ne permet de traiter que des séries temporelles unidimensionnelles, bien qu'elle soit extensible via une édition de liens dynamique pour permettre d'implémenter simplement le support des séries temporelles multidimensionnelles. Pour finir, précisons le format d'une série temporelle analysable par `Phase-TA` : il s'agit d'un ensemble de couples formés d'un horodatage (« timestamp » dans la langue de Shakespeare) et d'une valeur réelle entière ou à virgule flottante, organisés selon le format de fichier `CSV` [1].

1. Pour être précis, `Phase-TA` prend en entrée une représentation binaire intermédiaire d'un tel fichier au format `CSV`, à des fins d'optimisation. Un traducteur bi-directionnel entre ladite représentation binaire et le format `CSV` a été implémenté.

7.4 Détecter les comportements localement périodiques

La finalité de cette première étape est d'extraire les instances périodiques détectées par l'analyse de la série temporelle. En d'autres termes, cela revient à constituer une collection de portions (ou de « sous-séries » temporelles) de la série temporelle analysée, chaque portion correspondant à une occurrence d'une périodicité, comme cela est illustré par la figure 7.4.

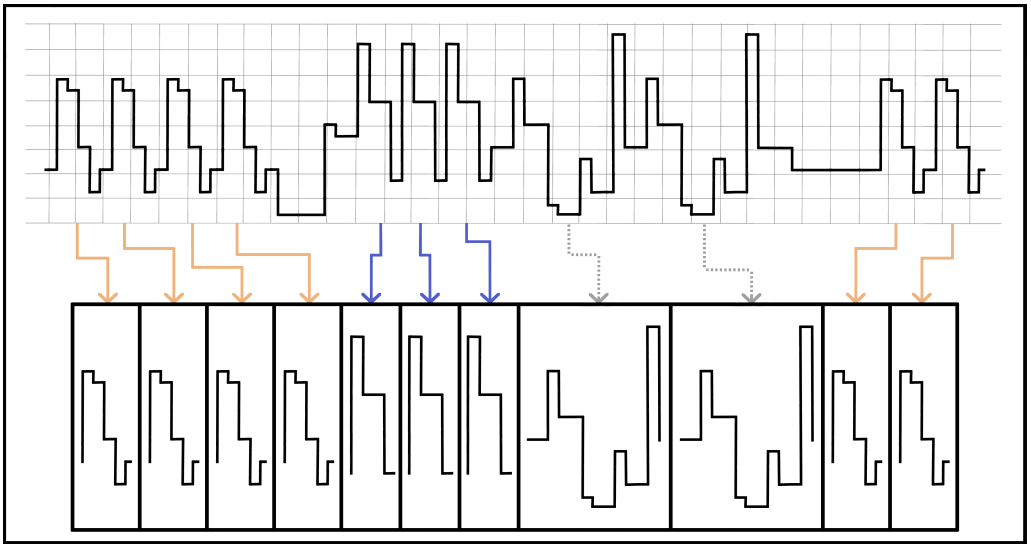


FIGURE 7.4 – Schéma de principe associé à l'étape de détection des comportements localement périodiques. On y voit un total de 11 instances périodiques, réparties entre 4 régions périodiques, elles-mêmes associées à 3 périodicités. Ces 11 instances périodiques sont extraites de la série temporelle analysée pour donner 11 « sous-séries » temporelles.

La **difficulté majeure** à laquelle doit répondre cette première étape de la méthodologie implémentée par `Phase-TA` est que la détection de comportements localement périodiques doit pouvoir **être appliquée sur des séries temporelles bruitées**. Ainsi, il est nécessaire d'autoriser que des variations aléatoires perturbent les occurrences des motifs périodiques, sans pour autant être trop permissif car cela risquerait d'induire la détection de faux-positifs. L'enjeu principal des travaux autour de cette partie de `Phase-TA` a donc été de concevoir une approche algorithmique permettant de moduler finement la tolérance au bruit, **à l'échelle d'une famille de séries temporelles**. En effet, à l'image des travaux menés autour de `BDPO` (cf. section 6.1), le but est qu'il soit possible de fournir une configuration par défaut « par domaine », qui permette de traiter une vaste majorité de ladite famille de séries temporelles sans qu'une expertise du domaine soit exigée de l'utilisateur. Et il va sans dire que la famille de séries temporelles qui nous intéresse dans le cadre de ce manuscrit, et pour laquelle une configuration par défaut a été construite, est celle des profils applicatifs `HPC`.

Par ailleurs, comme mentionné par la section 7.3, l'étape de détection des comportements localement périodiques nécessite d'utiliser une approche « diviser pour régner ». En effet, dans le cas général, une série temporelle n'est pas périodique lorsque considérée dans son intégralité. Par contre, elle peut l'être de manière localisée, sans que ces localités n'exhibent une récurrence spécifique (i.e. il n'y a pas de périodicité des comportements

localement périodiques). Or, les outils mathématiques à notre disposition pour déterminer si une périodicité existe considèrent l'intégralité de la série temporelle analysée. On comprend alors que pour détecter un comportement localement périodique, il est nécessaire de considérer seulement l'extrait de la série temporelle qui y est associé. D'où l'approche « diviser pour régner », dont l'algorithme est détaillé par la sous-section 7.4.1, qui consiste à analyser la série temporelle par morceaux. À la lecture de ce bref paragraphe introductif, il apparaît que la sélection de la longueur des extraits considérés a un impact fondamental sur la capacité de détection des régions périodiques. C'est pourquoi la problématique du choix de ce paramètre est abordée en grande largeur par la sous-section 7.4.2.

7.4.1 Algorithme de détection des régions périodiques et d'extraction des instances périodiques

Une série temporelle à valeurs réelles peut être représentée comme un vecteur unidimensionnel horizontal, dont les éléments correspondent aux échantillons de ladite série temporelle. Par convention, le premier élément du vecteur, d'indice 0, se situe à l'extrémité gauche du dit vecteur. Plus on se déplace vers l'extrémité droite du vecteur, plus les indices des éléments croissent, et plus les échantillons sont chronologiquement éloignés du début de la série temporelle.

L'algorithme implémentant la détection des régions périodiques modélise la série temporelle considérée comme un vecteur unidimensionnel horizontal, et l'analyse par morceaux, à l'aide d'une fenêtre glissante notée `SW` (pour « Sliding Window ») de longueur $2 \cdot L$, avec $L \in \mathbb{N}^*$. La fenêtre glissante est constituée de deux parties de longueur L, la partie gauche notée `LSW` (pour « Left Sliding Window »), et la partie droite notée `RSW` (pour « Right Sliding Window »). Au cours de l'analyse, la fenêtre glissante est translatée vers la droite du vecteur, et prend un total de $T \in \mathbb{N}^*$ positions. Pour tout $t \in [\![0; T]\!]$, on note $\texttt{SW}^t$ la fenêtre glissante lorsqu'elle est dans sa t-ième position, et $\texttt{LSW}^t$ et $\texttt{RSW}^t$ respectivement ses parties gauche et droite. Dans la suite, nous considérons une valeur de $t \in [\![0; T-1]\!]$ fixée.

Pour détecter si $\texttt{SW}^t$ exhibe un comportement localement périodique, un algorithme dérivé de `Dynamic Periodicity Detector`, présenté par [Freitag 2001], a été conçu et implémenté. Comme illustré par la figure 7.5, $\forall k \in \mathcal{A} = [\![1; L-1]\!], SW_k^t = SW^t \gg k$ est calculé, où $\gg$ est l'opérateur de décalage vers la droite. Il faut bien noter que $\texttt{SW}^t$ n'est pas glissée vers la droite, mais que $\texttt{SW}_k^t$ n'est autre qu'une « copie » de $\texttt{SW}^t$ dont les éléments ont été décalés de k éléments vers la droite. Ci-après, k est également appelé facteur de décalage (« sliding factor » en anglais). Ensuite, la distance de Manhattan entre $\texttt{RSW}^t$ et $\texttt{RSW}_k^t$, notée $d_1\left(RSW^t, RSW_k^t\right)$, est calculée pour toutes les valeurs de $k \in \mathcal{A}$. Le raisonnement sous-jacent est qu'une valeur de $d_1\left(RSW^t, RSW_k^t\right)$ proche de 0 signifiera que $\texttt{RSW}^t$ peut être considérée comme k-périodique. Ainsi, en analysant les minima locaux de $d_1\left(RSW^t, RSW_k^t\right)$, il est possible de déterminer si $\texttt{RSW}^t$ exhibe un comportement localement périodique, d'une période dont k est un multiple (potentiellement $1 \times k$). En effet, comme la figure 7.5 le fait apparaître, il peut exister une famille de périodes candidates, et il est alors possible d'en extraire la **période de base**, c'est-à-dire la plus petite des périodes candidates. Si une période de base existe, elle est notée k_{bp}, et est déterminée en appliquant successivement les algorithmes 1 et 2 pour, respectivement, déterminer les périodes candidates et isoler, parmi ces dernières, la période de base. Ces deux algorithmes sont

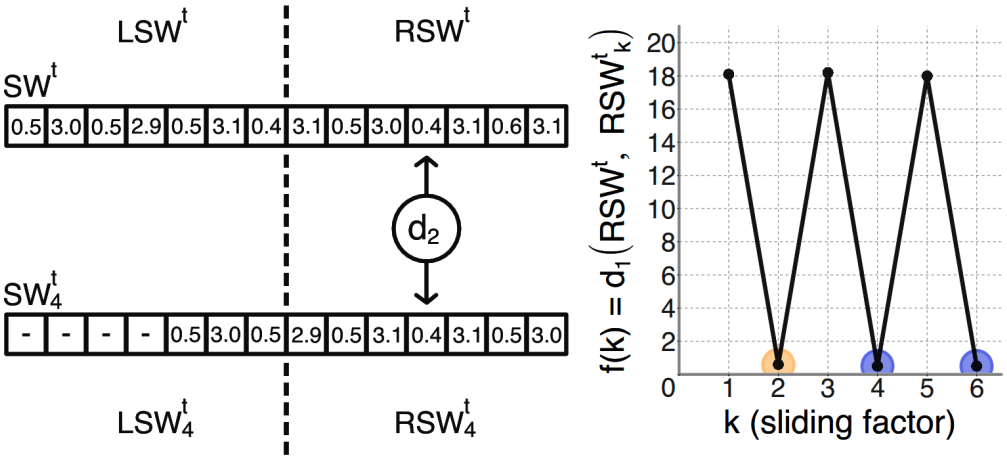


FIGURE 7.5 – Schéma illustrant l'algorithme de détection des comportements localement périodiques implémenté par `Phase-TA`. À gauche, un exemple de fenêtre glissante ainsi que sa version décalée vers la droite par un facteur 4. À droite, l'évolution de $f(k) = d_1(RSW^t, RSW_k^t)$ en fonction du facteur de décalage k. Le graphe fait apparaître la famille de périodes détectées (encerclées), et la période de base (encerclée en orange).

détaillés dans la suite de cette sous-section. La dernière étape consiste alors à extraire $\lfloor L/k_{bp} \rfloor$ instances périodiques de $\texttt{RSW}^t$ de longueur k_{bp}, puis $\texttt{SW}^{t+1}$ est obtenue en faisant glisser $\texttt{SW}^t$ de $k_{bp} \cdot \lfloor L/k_{bp} \rfloor$ échantillons vers la droite.

Tout d'abord, concernant les paramètres d'entrée de l'algorithme 1, `dist_vect`, `sample_max`, `dist_mean`, et `dist_min` sont internes à `Phase-TA` et calculés précédemment par ce dernier, alors que `t_sample_max` et `t_dist_mean` sont des paramètres de configuration possédant des valeurs par défaut (plus de détails à ce sujet par la suite), et exposés à l'utilisateur. `dist_vect` contient les valeurs de $f(k) = d_1(RSW^t, RSW_k^t)$ pour toutes les valeurs du facteur de décalage k autorisées par la largeur $2 \cdot L$ de la fenêtre glissante, tandis que `dist_min` et `dist_mean` contiennent respectivement les valeurs minimale et moyenne desdites distances. `sample_max` contient quant à elle la valeur maximale prise par les échantillons contenus dans la portion de la série temporelle analysée délimitée par la fenêtre glissante. Pour finir, `t_sample_max` et `t_dist_mean` sont des coefficients intervenant dans la définition de seuils permettant de déterminer si une valeur de k est une période candidate.

Maintenant que les paramètres de `determine_candidate_periods` ont été passés en revue, intéressons-nous au cœur de l'algorithme 1. L'ensemble des valeurs possibles du facteur de décalage k sont soumises à trois tests pour déterminer si elles correspondent à des périodes candidates. La première condition à vérifier pour que k soit une période candidate est que f admette un minimum local en k. Cette condition découle du résultat présenté par [Freitag 2001], et se comprend plutôt trivialement : si une portion de série temporelle est constituée d'une séquence d'occurrences d'un même motif, lorsque l'on décale cet extrait de série temporelle d'un multiple de la longueur du motif en question, la version décalée peut se superposer à la version originale. Cela implique que la distance point à point entre ces deux versions soit nulle dans le cas idéal, et relativement proche

de 0 dans le cas général d'une série temporelle aléatoirement bruitée. Par contre, lorsque la valeur du décalage k n'est pas un multiple de la longueur du motif qui se répète, l'originale et la version décalée ne se superposent pas, ce qui implique une distance point à point entre les deux non nulle, et dans tous les cas supérieure à celle obtenue lorsqu'il y a superposition. Ainsi, que la série temporelle soit idéale ou bruitée, lorsque le décalage k est un multiple de la longueur du motif récurrent, f admet un minimum local en k.

La seconde condition est que la distance associée à k soit inférieure à un seuil dépendant de la valeur maximale des échantillons contenus dans la fenêtre glissante, à savoir $(t\_sample\_max \cdot sample\_max)$.

Algorithme 1 Détermination des périodes candidates

```
1: procedure DETERMINE_CANDIDATE_PERIODS(in dist_vect, in sample_max, in
   dist_min, in dist_mean, in t_sample_max, in t_dist_mean, out candidate_periods)
2:
3:     vect_init(candidate_periods);
4:
5:     for k ∈ ⟦1; L⟧ do
6:         if not has_a_local_min_in(dist_vect, k) then
7:             continue;
8:         end if
9:
10:        if dist_vect[k] > sample_max · t_sample_max then
11:            continue;
12:        end if
13:
14:        if dist_vect[k] > dist_min + t_dist_mean · (dist_mean − dist_min) then
15:            continue;
16:        end if
17:
18:        vect_add(candidate_periods, (k, dist_vect[k]));
19:    end for
20:
21:    return candidate_periods;
22: end procedure
```

En effet, comme précisé quelques lignes auparavant, l'existence d'un comportement localement périodique impliquera que la distance résultant d'un décalage k égal à la longueur de la périodicité associée sera **relativement** proche de 0. Ce « relativement » dépend de la série temporelle : par exemple, si ses échantillons prennent valeurs dans $[10000; 20000]$, alors une différence relative de 1%, induite par un bruit aléatoire, entre deux échantillons « logiquement identiques » impliquera une distance locale de l'ordre de 100. En reprenant cet exemple avec des valeurs dans l'intervalle $[100; 200]$, la distance locale associée est de l'ordre de 1. On comprend ainsi la nécessité de faire en sorte que le seuil associé à la notion de « relativement proche de 0 » dépende des valeurs prises par la

série temporelle analysée [2]. On peut également remarquer que cette condition est plus exigeante pour les périodicités longues que courtes, étant donné que plus il y a d'échantillons potentiellement bruités, plus la distance associée à un facteur de décalage k est potentiellement importante. C'est pourquoi la valeur par défaut de `t_sample_max` est fixée à 0.40, heuristique qui a d'ailleurs été confortée expérimentalement. Avec un coefficient relativement élevé, le seuil permet de détecter les longs motifs récurrents en autorisant la présence d'un bruit aléatoire plutôt soutenu, tout en demeurant une condition difficile à remplir en l'absence de périodicité.

Si la seconde condition vise à séparer les cas où il y a présence de comportements localement périodiques de ceux où il n'y en a pas, la troisième condition, exprimée par la ligne 14 de l'algorithme 1, cherche à séparer les minima locaux associés à des périodes **réellement** candidates de ceux qui ne le sont pas, dans le cas où la portion de série temporelle analysée exhibe des régions périodiques. En effet, dans le cas où un comportement localement périodique est exhibé au niveau de la fenêtre glissante analysée, il se peut que des valeurs du facteur de décalage non-multiples de la longueur de la périodicité induisent des distances satisfaisant la seconde condition, tout en étant des minimum locaux [3]. Cela peut notamment se produire lorsque le motif associé à la périodicité contient des sous-motifs récurrents : même si le recouvrement entre la fenêtre glissante et sa version décalée n'est que partiel, il peut être suffisant pour que la distance associée passe sous le seuil de la seconde condition. En exigeant que la distance pour la valeur de k considérée soit proche de la valeur minimale de $f(k)$, qui est **nécessairement** [4] associée à une valeur de k multiple de la longueur de la périodicité, on cherche à filtrer les valeurs parasites de k qui remplissent les deux premières conditions mais qui ne sont pas associées à des périodes réellement candidates. La valeur par défaut de `t_dist_mean` est fixée à 0.40, suite à une série d'expérimentations de mise au point : un compromis permettant d'écarter une majorité des valeurs parasites de k sans pour autant écarter des périodes candidates abondamment bruitées. L'algorithme 2 constitue une deuxième protection contre ces valeurs parasites, de sorte à ce que la période de base sélectionnée corresponde bien à la longueur du motif périodique associé à la fenêtre glissante analysée.

En ce qui concerne l'algorithme 2, on peut commencer par préciser que parmi ses trois paramètres d'entrée, `candidate_periods` contient les périodes candidates identifiées par l'algorithme 1, et que `t_per_min` et `t_per_family` sont des paramètres de configuration exposés à l'utilisateur, possédant des valeurs par défaut, fixées expérimentalement.

La première étape de l'algorithme consiste à filtrer les périodes candidates qui sont inférieures à `t_per_min`. Cela est nécessaire parce qu'il est possible qu'une valeur du coefficient de décalage proche de 1 induise une distance qui satisfasse les conditions de l'algorithme 1. En effet, étant donné que la conception de `Phase-TA` doit permettre de prendre en compte une certaine quantité de bruit, et qu'un faible décalage peut être assimilé à du bruit, il devient nécessaire d'écarter ces petites valeurs de k de la liste des périodes candidates [5], puisqu'elles sont fréquemment des diviseurs de la période de base.

2. Une solution alternative, et couramment employée, est de normaliser la série temporelle. Néanmoins cela implique une étape supplémentaire, au coût algorithmique non négligeable.

3. En tout cas, la probabilité associée était suffisamment élevée pour que ce phénomène soit observé plusieurs fois pendant la phase de mise au point de `Phase-TA`.

4. Une brève démonstration de cette affirmation est proposée à la fin de cette sous-section.

5. Durcir les deuxième et troisième conditions de l'algorithme 1 permettrait de les écarter, mais

Expérimentalement, ce phénomène n'a été observé que pour $k \in [\![1; 3]\!]$, et la valeur par défaut de `t_per_min` est 10.

Algorithme 2 Détermination de la période de base

1: **procedure** DETERMINE_BASE_PERIOD(**in** *candidate_periods*, **in** *t_per_min*, **in** *t_per_family*, **out** *base_period*)
2:
3: **for** $p \in candidate\_periods$ **do**
4: **if** $p.period < t\_per\_min$ **then**
5: vect_remove($candidate\_periods, p$);
6: **end if**
7: **end for**
8:
9: **if** is_empty($candidate\_periods$) **then**
10: **return** -1; ▷ Il n'y a plus de période éligible, donc pas de période de base.
11: **end if**
12:
13: $base\_period \leftarrow \left(\underset{p}{\operatorname{argmin}} \left(candidate\_periods\,[p]\,.dist \right) \right) .period$;
14:
15: **for** $p \in candidate\_periods$ **do**
16: **if** $p.period > base\_period$ **then**
17: **continue**;
18: **end if**
19:
20: $dist\_to\_multiple \leftarrow \lfloor \frac{base\_period}{p.period} + \frac{1}{2} \rfloor$;
21:
22: **if** $dist\_to\_multiple < t\_per\_family$ **then**
23: $base\_period \leftarrow p.period$;
24: **end if**
25: **end for**
26:
27: **return** *base_period*;
28: **end procedure**

La seconde étape de l'algorithme consiste, quant à elle, à déterminer la période de base parmi les périodes candidates. En effet, si k_{bp} est la longueur du motif périodique associé à la région périodique dans laquelle est contenue la fenêtre glissante analysée, la liste des périodes candidates contient notamment tous les multiples de k_{bp} inférieurs à L. De plus, du fait du bruit, le minimum global en fonction du facteur de décalage de la distance entre la fenêtre glissante et sa version décalée n'est pas nécessairement associé à la période de base. Il est néanmoins nécessairement associé à l'un des multiples de k_{bp}. L'approche retenue consiste alors à sélectionner la période candidate k_0 pour laquelle f

diminuerait drastiquement la quantité de bruit que `Phase-TA` autorise. Ainsi, cela serait contre-productif pour un large éventail de séries temporelles. D'où l'ajout de cette seuil concernant la longueur minimale d'une période candidate.

admet un minimum global, puis à déterminer, au sein de la liste des périodes candidates, son plus petit diviseur. Ladite valeur de k est alors retenue comme période de base. On comprend ainsi l'importance de filtrer les faibles valeurs de k à la première étape de l'algorithme (ainsi que les valeurs de k parasites de la liste des périodes candidates au niveau de l'algorithme 1). Afin de déterminer si une période candidate est un diviseur de la période initialement sélectionnée, on vérifie que le quotient de la division de la seconde par la première est « presque entier » : la différence à l'entier le plus proche en valeur absolue de la partie décimale du quotient doit être inférieure à `t_per_family`, qui vaut 0.05 par défaut. Cette notion de « presque entier » est nécessaire pour autoriser des perturbations temporelles (bruits d'insertion et de suppression, cf. sous-section 7.1.3), qui auront pour effet d'impacter la longueur des occurrences des motifs périodiques.

Heuristique et configuration par défaut Comme précisé lors de l'introduction de cette section, la méthodologie conçue pour détecter les comportements localement périodiques est une heuristique : elle a pour but de conduire à une extraction des instances périodiques exactes pour une majorité de séries temporelles, en proposant des performances satisfaisantes (la section 8.4 présente une analyse de la complexité algorithmique et une évaluation des performances de `Phase-TA` dans sa globalité). Il est donc normal que l'étape de détection des régions périodiques puisse échouer dans certains cas, et il est possible de construire des séries temporelles qui la mettent en défaut pour une configuration donnée. Néanmoins, ces échecs sont marginaux : expérimentalement et statistiquement, ils ne concernent que 2 à 3% des fenêtres analysées. D'autant plus que, comme cela sera démontré par la suite (cf. section 8.3), ce qui est important pour que les motifs représentatifs construits par `Phase-TA` soient pertinents, c'est qu'ils soient inférés à partir de **suffisamment** d'instances périodiques. À cela s'ajoute le fait que la tolérance au bruit, et la manière dont elle est intégrée à l'algorithme de détection des régions périodiques, sont configurables via les paramètres exposés par `Phase-TA`. Ainsi, il est possible d'adapter le comportement de l'algorithme de détection pour qu'il analyse correctement une série temporelle qui met en échec sa configuration par défaut.

Un des objectifs des travaux de recherche était précisemment de faire en sorte que la configuration par défaut de `Phase-TA` soit adaptée à une famille de séries temporelles, et qu'elle permette d'analyser avec succès une vaste majorité de ces dernières. La configuration par défaut mentionnée dans le cadre de ce chapitre a été élaborée expérimentalement, en analysant des profils applicatifs de certaines applications présentées par la section 4.2, dans le but de construire une configuration adaptée à la majorité des profils applicatifs `HPC`. En ce qui concerne la partie de la configuration par défaut associée à la détection des comportements localement périodiques, les applications utilisées ont été `CG`, `EP`, `FT`, `MG`, et `SP-MZ` des `NPB`. Et cette configuration par défaut a été utilisée, sans modification ni adaptation, pour l'ensemble des expériences menées avec `Phase-TA` (incluant celles présentées par le chapitre 8), sur un large panel d'applications de calcul à haute performance. En d'autres mots, cela tend à **prouver empiriquement** qu'il est possible de construire une configuration par défaut **adaptée à une famille de séries temporelles**. L'élaboration d'une méthodologie de construction de la configuration par défaut pour un type de séries temporelles donné, à l'image de ce qui a été fait pour `BDPO` (cf. section 6.1), constituerait un sujet de recherche très intéressant, qui n'a malheureusement pas pu être traité dans le cadre de la thèse.

Une petite démonstration Démontrons que, lorsque la fenêtre glissante analysée est incluse dans une région périodique, la valeur du facteur de décalage k minimisant $f(k)$ est nécessairement égale à un multiple de la longueur de la périodicité associée. Supposons que la fenêtre glissante analysée soit effectivement incluse dans une région périodique associée à une périodicité de longueur k_0. Raisonnons alors par l'absurde, et supposons qu'il existe une valeur k_m non multiple de k_0 telle que f exhibe un minimum global en k_m. Alors, $f(k_m) \leq f(k_0)$. Or, $f(k_0)$ est « proche de 0 » puisqu'il existe une périodicité de longueur k_0. Ainsi, $f(k_m)$ est « proche de 0 », ce qui implique que k_m est la longueur d'une périodicité qui a engendré la région périodique dans laquelle la fenêtre glissante analysée est contenue. Or, cela est impossible, puisque ladite fenêtre glissante est constituée d'occurrences d'un motif périodique de longueur k_0, et que k_m n'est pas un multiple de k_0. Notons que l'hypothèse où ledit motif périodique de longueur k_0 serait constitué de plusieurs occurrences d'un motif périodique de longueur k_m est invalidée par la définition d'une périodicité (cf. sous-section 7.1.2), qui est associée à un motif de **taille minimale**. CQFD.

Digression sur le thème des distances Pour terminer cette section, une petite digression concernant le choix de la distance de Manhattan, et non pas de la distance euclidienne, dans le cadre de la détection de comportements localement périodiques. Le raisonnement sous-jacent à l'approche peut être reformulé de la sorte : si la portion de série temporelle analysée exhibe un comportement localement périodique, il existe une valeur de décalage telle que la version décalée de ladite portion puisse être superposée presque exactement à la version initiale[6]. Cette notion de superposition implique que chaque point de la version décalée trouve son alter ego dans la version initiale. En d'autres termes, chaque point de la version décalée est quasiment similaire au point de la version initiale auquel on le superpose. Ce qui appelle à une comparaison point à point des deux versions. Et la distance de Manhattan est précisément une « distance point à point », là où la distance euclidienne permet que les différences point à point puissent partiellement se compenser, et « applatit » leur somme via l'application de la racine carrée (cf. sous-sections 2.3.1 et 2.3.2).

7.4.2 De l'importance de la largeur de la fenêtre glissante

Comme cela a été observé quelques paragraphes auparavant, la longueur $2 \cdot L$ de la fenêtre glissante a un impact significatif sur l'étape de détection de comportements localement périodiques. En effet, avec une fenêtre glissante de taille $2 \cdot L$, il n'est possible que de détecter des périodicités de longueur au plus L. À cela s'ajoute le fait que la valeur optimale de L dépend à priori de l'application, et que la déterminer n'est pas une tâche triviale. Pour répondre à cette problématique, `Phase-TA` implémente l'algorithme 3 d'auto-tunage de la valeur de L. Cet algorithme repose notamment sur la notion d'**optimalité**, c'est-à-dire la proportion de la série temporelle imputable aux instances périodiques détectées, que nous allons maintenant définir mathématiquement. Soit $\mathcal{S}$ une série temporelle contenant $N_{\mathcal{S}}$ échantillons, et soit $\mathcal{I}$ l'ensemble des instances périodiques

6. Cette interprétation découle directement de la défintion mathématique du caractère p-périodique, avec $p \in \mathbb{R}^*$, d'une fonction réelle g, définie sur $\mathbb{R}$ et à valeurs réelles, à savoir g est p-périodique $\Leftrightarrow \forall x \in \mathbb{R}, f(x+p) = f(x)$.

détectées par `Phase-TA` lors de l'analyse de $\mathcal{S}$ avec une fenêtre glissante de longueur L. Notons, $\forall i \in \mathcal{I}$, N_i le nombre d'échantillons de l'instance périodique i. Alors, l'optimalité de L pour l'analyse de $\mathcal{S}$ ayant conduit à l'extraction de la collection d'instances périodiques $\mathcal{I}$ est définie par :

$$optim_{\mathcal{S},\mathcal{I}}(L) = \frac{1}{N_{\mathcal{S}}} \sum_{i \in \mathcal{I}} N_i \tag{7.1}$$

Ainsi, l'optimalité est comprise entre 0 et 1, et plus elle se rapproche de 1, plus les instances périodiques détectées avec une fenêtre glissante de longueur L recouvrent une portion significative de la série temporelle complète. Le raisonnement sous-jacent est qu'une périodicité est d'autant plus significative que la fraction de la série temporelle qui lui est associée est grande, en faisant fi du nombre d'occurrences de ladite périodicité.

Algorithme 3 Auto-tunage de la longueur L de la fenêtre glissante pour l'analyse de la série temporelle $\mathcal{S}$

```
1: procedure AUTOTUNE_SW_LENGTH(in S, in L_sup, in nb_sub_intervals, in nb_iter_dichotomy, out L )
2:
3:     array_init(ibound_optims, nb_sub_intervals − 1);
4:
5:     sub_int_w ← L_sup / nb_sub_intervals;
6:
7:     for i ∈ [[1; nb_sub_intervals − 1]] do
8:         L ← i · sub_int_w;
9:         I ← detect_perarea_extract_perinst(S, L);
10:        ibound_optims ← optim_{S,I}(L);
11:    end for
12:
13:    middle_bound ← sub_int_w · argmax_k (ibound_optims[k]);
14:
15:    left_bound ← middle_bound − sub_int_w;
16:    right_bound ← middle_bound + sub_int_w;
17:
18:    L_LEFT, optim_LEFT ← dichotomic_process(
19:            S, nb_iter_dichotomy, left_bound, middle_bound
20:    );
21:    L_RIGHT, optim_RIGHT ← dichotomic_process(
22:            S, nb_iter_dichotomy, middle_bound, right_bound
23:    );
24:
25:    return (optim_LEFT > optim_RIGHT ? L_LEFT : L_RIGHT);
26: end procedure
```

Apportons des précisions concernant l'algorithme 3. L'intervalle de recherche L est spécifié via sa borne supérieure L_{sup}, sa borne inférieure étant 0. Il est ensuite parcouru en suivant une procédure inspirée d'un processus dichotomique. Dans un premier temps, l'intervalle de recherche est subdivisé en `nb_sub_intervals` sous-intervalles dont L va visiter les bornes, en excluant les bornes de l'intervalle de recherche global. Pour chaque valeur de L visitée, la détection des périodicités et l'extraction des instances périodiques est effectuée. Ces dernières sont rassemblées dans la collection $\mathcal{I}$. Cela permet de calculer l'optimalité associée à la valeur considérée pour L.

Une fois l'optimalité de l'ensemble des bornes des sous-intervalles calculée, il est possible de sélectionner celle exhibant la plus haute valeur, ainsi que les deux bornes l'encadrant, bornes de l'intervalle de recherche global incluses. Les trois bornes sélectionnées permettent de définir deux intervalles qui vont être explorés via `nb_iter_dichotomy` itérations d'un processus dichotomique, ce qui constitue la seconde étape de l'algorithme 3. Soit $n \in \mathbb{N}^*$ le numéro d'itération du processus dichtomique, soit $I_n = [\![i_n, s_n]\!]$ l'intervalle de recherche considéré à l'itération n, et soit m_n le point milieu de I_n. Notons qu'au début de l'itération n, les optimalités de i_n et s_n sont connues. Le processus dichotomique peut alors être résumé en deux étapes :

1. La détection des régions périodiques et l'extraction des instances périodiques sont effectuées pour le point milieu de I_n, c'est-à-dire pour $L = m_n$, ce qui permet de calculer l'optimalité associée ;
2. L'intervalle de recherche suivant, I_{n+1}, est défini en sélectionnant ses bornes parmi m_n, i_n, et s_n, de sorte à ce que :
   (a) m_n soit une des bornes de I_{n+1} ;
   (b) La valeur de L, parmi i_n, m_n, et s_n, associée à la plus haute optimalité soit sélectionnée ;
   (c) La somme des optimalités des bornes choisies pour I_{n+1} soit maximisée, dans la mesure où les contraintes 2a et 2b sont satisfaites.

L'application du processus dichotomique conduit à l'identification d'une valeur de L « localement optimale » pour chacun des deux intervalles sélectionnés suite à la première étape de l'algorithme. La valeur de L retenue est celle à laquelle est associée la plus haute optimalité. L'approche de cet algorithme d'autotunage peut ainsi être résumée comme suit : (1) balayer à gros grain un large intervalle de recherche pour tenter d'identifier la portion dudit intervalle contenant la valeur optimale de L, puis (2) explorer la portion identifiée de manière dichotomique pour tendre vers ladite valeur optimale de L.

Les trois paramètres d'entrée de l'algorithme 3 permettent à l'utilisateur de définir le fonctionnement de `Phase-TA` en ce qui concerne le compromis entre performances et caractère optimal de la valeur de L choisie. Chaque paramètre possède une valeur par défaut, résultante des expérimentations initiales menées avec l'outil. Ces valeurs par défaut sont globalement neutres du point de vue du compromis précédemment cité, ce qui conduit à des valeurs satisfaisantes de l'optimalité associée à L, pour un impact sur les performances raisonnable. Par exemple, lesdites expérimentations menées avec `Phase-TA`, pour l'ensemble des applications et des conditions expérimentales considérées, ont toutes conduit à la détection de périodicités dont la longueur est inférieure à 2000 échantillons. C'est pour cela que la valeur par défaut pour L_{sup} a été fixée à 10000, ce qui représente

un compromis entre impact raisonné sur les performances de `Phase-TA` et faible probabilité de rencontrer une périodicité plus longue que L_{sup}. Quant à `nb_sub_intervals` et `nb_iter_dichotomy`, ils sont par défaut respectivement égaux à 10 et 5.

Pour conclure, il convient de remarquer que la valeur de L retenue à l'issue de l'exécution de l'algorithme 3 n'est pas, dans le cas général, la valeur optimale de L. Implémenter une procédure de recherche s'appuyant sur de l'optimisation bayésienne (en adaptant [Miyazaki 2018]), ou encore utiliser une valeur adaptative pour L en s'inspirant des arbres quaternaires [7] a été envisagé. Néanmoins, le « gain » aurait été plus que marginal par rapport au coût algorithmique, d'autant qu'il n'est pas nécessaire de détecter toutes les instances périodiques associées à une périodicité pour la caractériser correctement, il en faut juste suffisamment, comme la section 8.3 le démontre. Ainsi, l'algorithme d'auto-tunage actuellement implémenté par `Phase-TA` s'avère être un bon compromis entre proximité à l'optimalité et coût en termes de performances.

7. https://fr.wikipedia.org/wiki/Quadtree

7.5 Clustériser les instances périodiques

La deuxième étape de la méthodologie implémentée par `Phase-TA` consiste à appliquer un algorithme de clustering à la collection d'instances périodiques détectées et extraites du profil applicatif analysé lors de la première étape de ladite méthodologie. Cette étape est fondamentale dans l'optique de la construction de motifs représentatifs par moyennage. En effet, si des instances périodiques aberrantes se glissaient parmi les instances périodiques à moyenner pour construire un motif représentatif, ce dernier s'en retrouverait parasité, et perdrait en pertinence en tant que substitut de la périodicité associée. Le but recherché est donc de regrouper les instances périodiques en fonction des périodicités qui les ont engendrées, comme cela est illustré par la figure 7.6.

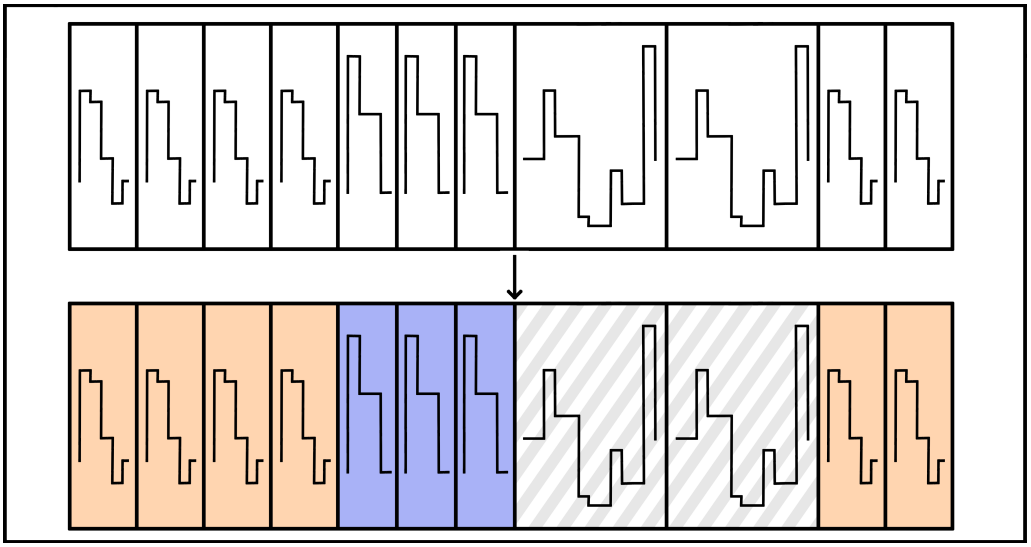


FIGURE 7.6 – Schéma de principe associé à l'étape de clustering des instances périodiques. On peut observer que les instances périodiques sont regroupées en fonction de la périodicité qui les a engendrées.

Avant d'être clustérisées, les instances périodiques subissent deux pré-traitements : elles sont triées par longueur et alignées. Ces deux étapes préliminaires au clustering sont respectivement détaillées par les sous-sections 7.5.1 et 7.5.2. La démarche associée à la sélection de l'algorithme de clustering approprié est détaillée par la sous-section 7.5.3. Pour finir, la sous-section 7.5.4 aborde en profondeur une des spécificités de l'algorithme de clustering retenu.

7.5.1 Pré-traitement : regrouper les instances périodiques par longueur

Avant de clustériser les instances périodiques détectées, deux pré-traitements sont appliqués. Le premier d'entre-eux consiste à regrouper les instances périodiques par paquets, en fonction de leurs longueurs. Un paquet peut être modélisé comme un intervalle centré sur une longueur L, et dont la largeur est égale à un pourcentage β de la longueur L. On note alors un tel groupe $G\,(L, \beta) = [\![L \cdot\ (1 - \beta/2)\,; L \cdot\ (1 + \beta/2)]\!]$. Par exemple, $G\,(1000; 10\%) = [\![950; 1050]\!]$ regroupera toutes les instances périodiques dont la longueur

est comprise entre 950 et 1000. Ce pré-traitement trouve sa justification dans une observation simple : pour que deux instances périodiques puissent avoir été engendrées par une même périodicité, il est nécessaire qu'elles aient des longueurs similaires, dans le voisinage de la longueur de ladite périodicité. Une fois le regroupement par longueur effectué, les groupes dont les cumuls des longueurs des instances périodiques qu'ils contiennent couvrent moins de $\alpha = 5\%$ de la longueur totale de la série temporelle analysée sont considérés comme marginaux, et sont donc ignorés par la suite. Filtrer les groupes marginaux a deux avantages principaux :

— Un groupe marginal ne comporte que peu d'instances périodiques, qui sont donc soit associées à une périodicité peu significative soit des instances périodiques aberrantes (e.g. faux-positif de détection). Dans les deux cas, les ignorer permet de se concentrer sur les périodicités significatives exhibées par la série temporelle analysée ;

— La complexité des algorithmes de clustering croît, dans l'extrême majorité des cas, de manière polynomiale avec le nombre d'objets à clustériser. En éliminant les instances périodiques des groupes marginaux avant l'étape de clustering, on diminue le temps d'exécution associé à cette dernière. À noter qu'en procédant à un clustering par parquet, on met également en place une approche « diviser pour régner » qui permet une optimisation supplémentaire du temps d'exécution associé à l'étape de clustering.

Notons que l'étape de clustering est tout de même nécessaire puisqu'il est possible qu'une série temporelle exhibe deux périodicités distinctes de longueurs similaires, ce qui a notamment été observé lors de l'analyse de profils applicatifs associés à `NAMD`.

7.5.2 Aligner les instances périodiques

Une fois les instances périodiques rassemblées par longueur, il faut encore les aligner avant de les clustériser. En effet, il est possible que les instances périodiques extraites n'aient pas la même « phase à l'origine » : lors du balayage de la série temporelle analysée par la fenêtre glissante, le début de cette dernière ne coïncide pas nécessairement avec le même point de la première instance périodique qu'elle contient. Dans ce cas de figure, bien qu'engendrées par la même périodicité, les instances périodiques concernées ne sont pas similaires, et il est donc impossible de construire un motif représentatif à partir de ces dernières. Cette situation est illustrée par la figure 7.7.

Néanmoins, étant donné que les instances périodiques sont les occurrences d'une périodicité, elles demeurent **identiques à un décalage circulaire près**, même lorsqu'elles sont déphasées. Afin d'aligner les instances périodiques contenues dans un des paquets formés suite au premier pré-traitement, on procède comme suit. Pour commencer, l'une d'entre elle est sélectionnée comme référence. Elle est notée i_r, et sa longueur l_r. Ensuite, chacune des instances périodiques restantes, celle considérée étant notée i_a et sa longueur l_a, est décalée circulairement pour l'ensemble des valeurs de décalage λ possibles (i.e. $\lambda \in [\![1; l_a - 1]\!]$). On note i_a^λ l'instance périodique i_a décalée circulairement de λ échantillons. Pour chaque valeur de λ, la distance de Manhattan entre i_r et i_a^λ est calculée. Elle est notée $d_1\left(i_r; i_a^\lambda\right)$. La valeur de décalage retenue pour l'instance périodique à aligner est alors $\lambda_a = \operatorname{argmin}_\lambda \left(d_1\left(i_r; i_a^\lambda\right)\right)$. L'instance périodique i_a est finalement décalée

circulairement de λ_a échantillons. Cette transformation est effectuée en place. À noter qu'il est possible que l'instance périodique de référence et l'instance périodique à aligner n'aient pas la même longueur. Dans ce cas, la distance de Manhattan est calculée sur les $min\,(l_a; l_r)$ premiers échantillons.

Pour finir, notons que le choix de l'instance périodique de référence n'est pas déterminant. En effet, les occurrences d'un motif périodique, même bruitées, demeurent très similaires. En conséquence, les valeurs de décalage circulaire retenues pour des instances périodiques engendrées par une même périodicité seront identiques ou presque, puisque les distances à l'instance périodique de référence associées évolueront de façon similaire. C'est également pour cela que cette méthode d'alignement des instances périodiques reste pertinente dans le cas où les instances périodiques contenues dans le paquet considéré sont engendrées par plusieurs périodicités distinctes.

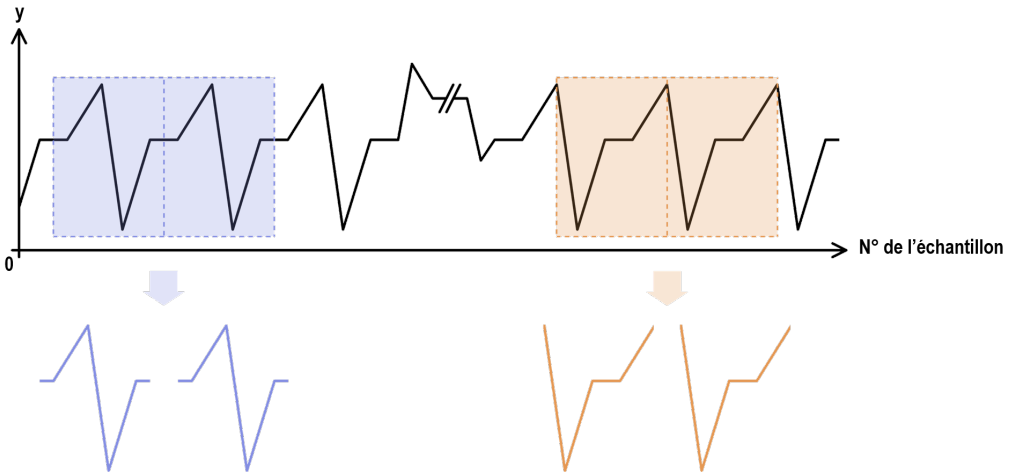


FIGURE 7.7 – Schéma illustrant la différence de phase que peuvent présenter des instances périodiques engendrées par la même périodicité en raison des positions initiales des fenêtres glissantes associées. Les deux positions, en violet et orange, de la fenêtre glissante conduisent à l'extraction d'instances périodiques engendrées par la même périodicité, mais déphasées.

7.5.3 Quel algorithme de clustering ?

Une fois les deux pré-traitements mis en place, la question a été de savoir quel algorithme de clustering appliquer. Avant toute chose, définissons deux concepts qui seront utilisés par la suite. Un algorithme de clustering est **adéquat** s'il conduit à des clusters d'instances périodiques pertinentes de structures similaires à ceux attendus. Un exemple afin d'illustrer cette définition : si l'analyse manuelle des instances périodiques d'un profil applicatif se conclut par la construction de 2 clusters, chacun associé à une périodicité distincte, notées $\mathcal{P}_1$ et $\mathcal{P}_2$, alors un algorithme de clustering est jugé adéquat s'il conduit à la formation de deux clusters, l'un contenant majoritairement des instances périodiques engendrées par $\mathcal{P}_1$, et l'autre contenant majoritairement des instances périodiques engendrées par $\mathcal{P}_2$. On définit alors la **pureté** d'un cluster comme la proportion de ce dernier associée à la périodicité majoritairement représentée. Par exemple, si un cluster contient 20 instances périodiques engendrées par $\mathcal{P}_1$ et 5 instances périodiques engendrées par

$\mathcal{P}_2$, alors la périodicité majoritairement représentée est $\mathcal{P}_1$ et la pureté du cluster est $20/25 = 0.80$.

Maintenant que les notions d'adéquation et de pureté sont définies, il est possible d'établir une liste de critères à remplir, d'importance décroissante, afin d'évaluer les algorithmes de clustering considérés:

1. Il faut que l'algorithme de clustering soit adéquat, et propose la pureté moyenne la plus élevée possible pour les clusters qu'il construit. En effet, afin que l'analyse d'un profil applicatif par `Phase-TA` soit représentative de ce dernier, il faut qu'elle retranscrive fidèlement sa structure. Or, le nombre et la prépondérance des périodicités détectées par `Phase-TA` découle directement de la structure des clusters construits lors de cette seconde étape. C'est pourquoi il est fondamental que l'algorithme de clustering soit adéquat. De plus, il est capital de minimiser la présence d'instances périodiques « aberrantes » dans les clusters formés, de sorte à ne pas perturber la construction des motifs représentatifs, présentée par la section 7.6. D'où l'importance d'avoir des clusters de puretés élevées ;
2. Il faut, dans la mesure du possible, que les hyper-paramètres de l'algorithme de clustering permettent de définir une configuration par défaut valable à l'échelle d'une famille de séries temporelles. Cette condition est primordiale si l'on souhaite éviter que `Phase-TA` ne nécessite une configuration spécifique pour chaque analyse ;
3. Il faut que la complexité de l'algorithme de clustering permette d'atteindre des performances compatibles avec une analyse à chaud des données ;
4. Il faut que la précision et le rappel de l'algorithme de clustering soient aussi élevés que possible, ou au moins suffisants pour permettre d'inférer des motifs représentatifs pertinents. En effet, comme la section 8.3 le montre, la construction d'un motif représentatif ne nécessite pas l'intégralité des instances périodiques engendrées par la périodicité associée. Il faut simplement en considérer suffisamment, ce qui rend la contrainte sur la précision et le rappel plutôt lâche.

Il convient également de préciser que, dans le but d'évaluer les algorithmes de clustering envisagés, les instances périodiques extraites par `Phase-TA` pour 4 profils applicatifs ont été clustérisées manuellement et étiquettées, de sorte à notamment pouvoir calculer précision et rappel. Les 4 profils applicatifs en question exhibent des instances périodiques aux caractéristiques variées : (1) le profil associé à `NAMD` exhibent peu d'instances périodiques (19 au total), réparties entre deux périodicités longues (environ 1500 échantillons) et très similaires, (2) le profil associé à `OpenFOAM` pour le solveur `simpleFOAM` appliqué au cas d'utilisation `motorBike` présente une unique périodicité de longueur environ égale à 300 échantillons et ayant engendré 69 instances périodiques, (3) pour le profil associé à `HPCG`, `Phase-TA` a détecté 293 instances périodiques réparties non équitablement entre deux périodicités très différentes (58 pour la première et 235 pour la seconde), de longueurs respectives 293 et 238, et, enfin, (4) le profil associé `NEMO` n'exhibe qu'une périodicité d'une longueur d'une centaine d'échantillons, mais engendrant 874 instances périodiques.

Algorithme des k-moyennes Jouant peu ou prou le rôle de figure de proue des algorithmes de clustering, l'algorithme des k-moyennes [Lloyd 1982] a été le premier à être

considéré. Ce dernier repose sur un processus itératif qui part de clusters initiaux arbitraires et les raffine dans le but de minimiser les WGSS intra-clusters par rapport à leurs centroïdes respectifs. En conséquence, cela implique de devoir calculer le centroïde d'un ensemble d'instances périodiques, pour chaque cluster, à chaque itération. Bien que des algorithmes permettent de le faire, à l'image de celui utilisé par Phase-TA pour construire les motifs représentatifs, leurs complexités algorithmiques sont bien trop élevées pour remplir le prérequis 3. Et même les variations sur le thème de l'algorithme des k-moyennes, telles que l'utilisation d'espace de Hilbert à noyaux reproduisants [Dhillon 2004] pour se soustraire à l'obligation de calculer les centroïdes, induisent une complexité algorithmique prohibitive en raison des nombreux calculs de DTW nécessaires. De la même manière, les nombreuses heuristiques qui ont vu le jour dans le but d'accélérer à la fois la convergence et le cœur d'une itération du processus de raffinement de l'algorithme des k-moyennes, à l'image de [Elkan 2003] et [Gribel 2019] ne compensent pas la complexité algorithmique élevée, inhérente au fait d'utiliser l'algorithme des k-moyennes sur des séries temporelles.

Toutefois, dans un espoir désespéré, l'algorithme des k-moyennes a été évalué en substituant chaque instance périodique par un ensemble de grandeurs statistiques lui étant associées (e.g. valeurs médiane, moyenne, minimale, maximale, etc.). Les résultats sont présentés par le tableau 7.1.

TABLE 7.1 – Résultat du clustering avec l'algorithme des k-moyennes pour les 4 profils applicatifs dont les instances périodiques ont été annotées.

k-moyennes	Adéquation	Pureté	Précision	Rappel
NAMD	Non	0.714	0.538	0.373
OpenFOAM	Non	0.827	0.728	0.611
HPCG	Non	0.972	0.997	0.446
NEMO	Oui	0.989	0.804	0.754

Si l'on fait fi de la nécessité de définir à priori le nombre de clusters, ce qui va à l'encontre du prérequis 2, l'algorithme des k-moyennes appliqué à des grandeurs statistiques représentatives des instances périodiques s'est avéré uniquement adéquat pour NEMO, ce qui est rédhibitoire. Cela s'explique probablement par le fait que les instances périodiques exhibées par le profil applicatif de NEMO analysé sont courtes et très similaires, il n'y a donc que peu de place pour des variations statistiques qui ont perturbé largement le clustering pour les autres profils applicatifs, conduisant à la formation de plusieurs clusters associés aux mêmes périodicités. À cela s'ajoutent des puretés, précisions, et rappels qui sont loin d'être aussi élevées que celles exhibées par les autres algorithmes envisagés. L'algorithme des k-moyennes a donc été écarté.

Algorithme OPTICS Présenté par [Ankerst 1999], OPTICS est un algorithme de clustering par étude de la densité spatiale. En d'autres termes, c'est en s'appuyant sur la concentration du nombre d'échantillons dans l'espace vectoriel auxquels ces derniers appartiennent que l'algorithme les clustérise. C'est le second algorithme de clustering qui a été considéré.

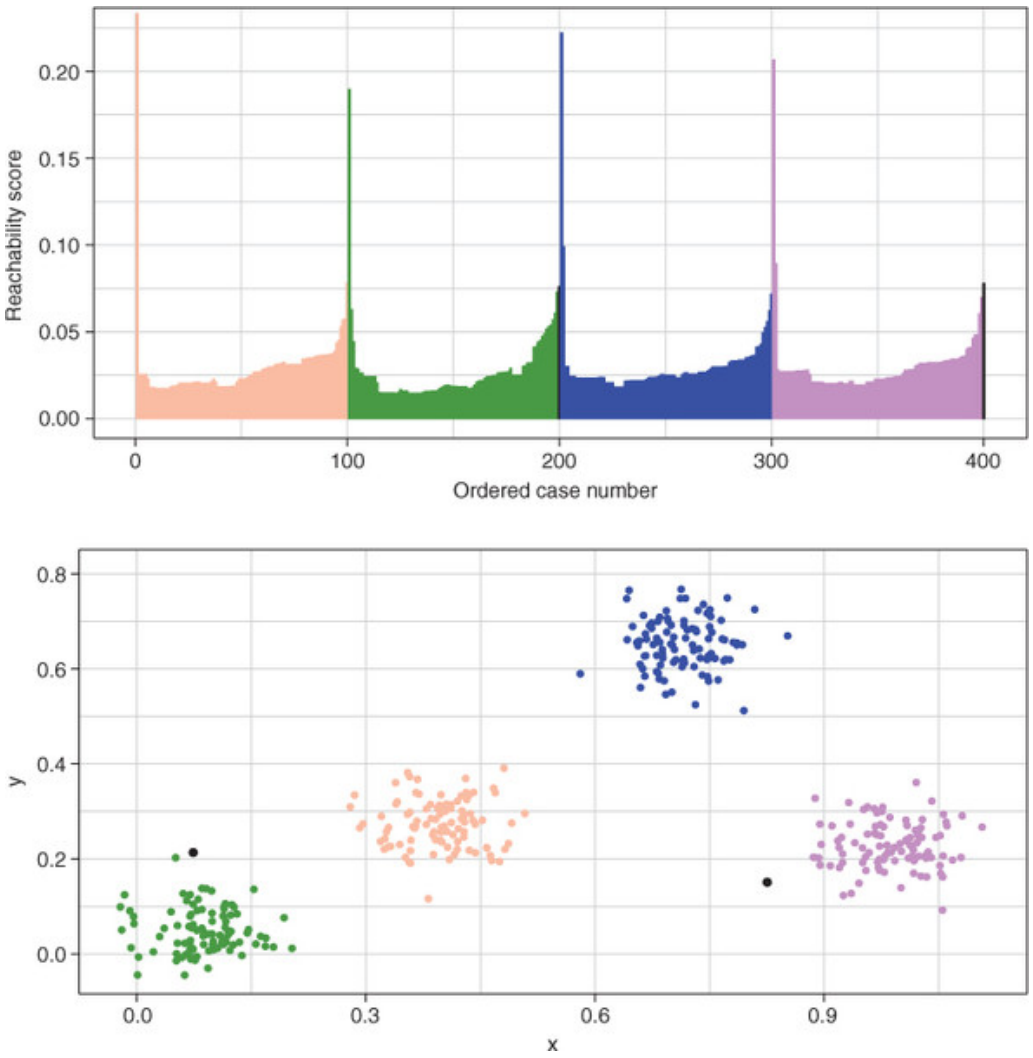


FIGURE 7.8 – Exemple de graphe d'accessibilité résultant de l'application d'`OPTICS` à un problème de clustering. Source : https://livebook.manning.com/book/machine-learning-for-mortals-mere-and-otherwise/chapter-18/86

Pour ce faire, `OPTICS` définit notamment la notion de ϵ-voisinage des éléments à considérer, c'est-à-dire l'ensemble des éléments restant à clustériser qui sont à une distance de l'élément considéré inférieure à ϵ. Pour qu'un élément à clustériser devienne un élément « cœur », il faut que l'ϵ-voisinage du dit élément contienne au moins $MinPts$ éléments. On peut alors définir la « distance d'accessibilité » qui représente grossièrement la distance qui sépare un élément d'un élément cœur. Une file de priorité $\mathcal{A}$, indexée sur l'ordre de traitement, va accueillir les éléments traités, et une autre file de priorité, notée $\mathcal{B}$ et indexée sur la distance d'accessibilité va accueillir les éléments à traiter par la suite. Initialement, l'ensemble des éléments constitue une collection $\mathcal{S}$. Tant que $\mathcal{S}$ n'est pas vide et que $\mathcal{B}$ l'est, les éléments de $\mathcal{S}$ vont être traités et ajoutés à $\mathcal{A}$. Lors de ce traitement, on évalue si l'élément considéré est un point cœur, et si c'est le cas, son ϵ-voisinage est ajouté à $\mathcal{B}$. Ce sont alors les éléments de $\mathcal{B}$ qui seront traités, et les ϵ-voisinages de ces derniers, s'ils existent, alimenteront à leurs tours $\mathcal{B}$. Si un élément appartient à l'ϵ-voisinage de plusieurs éléments cœur, sa distance d'accessibilité, ainsi que son rang dans $\mathcal{B}$, peuvent

être mis à jour. Une fois l'ensemble des éléments traités, leur ordre de traitement (et donc d'ajout à $\mathcal{A}$) permet de tracer un graphe d'accessibilité, qui représente l'évolution de la distance d'accessibilité en fonction de l'ordre de traitement, tel que celui présenté en exemple par la figure 7.8. Chaque « creux » est associé à un cluster, et une analyse des tangentes au graphe d'accessibilité permet globalement d'extraire les différents clusters automatiquement.

Lorsque les hyper-paramètres d'`OPTICS`, à savoir ϵ et $MinPts$, ont été finement choisis, les résultats obtenus pour le clustering des instances périodiques annotées extraites des 4 profils applicatifs sont plutôt prometteurs, comme le montre le tableau 7.2 : l'algorithme s'est avéré adéquat, et a démontré d'excellents puretés, précisions, et rappels pour l'ensemble des 4 profils applicatifs considérés. Ce à quoi s'ajoutent de très bonnes performances découlant de sa complexité algorithmique en $\mathcal{O}(n \cdot \log(n))$ lorsque son implémentation est optimisée.

TABLE 7.2 – Résultat du clustering avec l'algorithme `OPTICS` pour les 4 profils applicatifs dont les instances périodiques ont été annotées.

`OPTICS`	Adéquation	Pureté	Précision	Rappel
`NAMD`	Oui	1.00	1.00	1.00
`OpenFOAM`	Oui	0.946	0.879	0.856
`HPCG`	Oui	0.971	0.987	1.00
`NEMO`	Oui	0.994	0.933	0.981

Néanmoins, les valeurs « optimales » des hyper-paramètres pour chacun des 4 profils applicatifs considérés sont différentes, sans qu'aucune heuristique ne semble se dessiner. Dans une moindre mesure, l'extraction automatique des clusters à partir du graphe d'accessibilité introduit au moins un nouvel hyper-paramètre qui a nécessité des ajustements pour `NAMD` et `HPCG`. Bien qu'une évolution d'`OPTICS`, nommée `DeLi-Clu` et présentée par [Achtert 2006], permet de s'abstraire de ϵ, le choix de $MinPts$ demeure spécifique à une application et avait un impact trop significatif sur les résultats de l'algorithme. En conséquence, l'évaluation des algorithmes de clustering envisageables se poursuit, et `OPTICS` aurait pu être retenue si aucune meilleure solution n'avait été trouvée par la suite.

Clustering hiérarchique ascendant avec saut minimal Commençons par décrire le principe général des algorithmes de clustering hiérarchique ascendant. Pour commencer, les distances entre toutes les paires possibles d'éléments à clustériser sont calculées. S'il y a $n \in \mathbb{N}^*$ éléments à clustériser, cela fait un total de $n \cdot (n-1)$ distances à calculer, généralement rassemblées dans une matrice triangulaire inférieure appelée « matrice des distances ». Dans le cas où les éléments à clustériser sont des séries temporelles, `DTW` est fréquemment utilisée en guise de distance. C'est le cas pour `Phase-TA`.

Une fois la matrice des distances calculée, il est possible de déterminer les deux éléments à clustériser les plus proches. Ils sont alors sélectionnés et constituent un premier ensemble d'éléments. La matrice des distances est alors mise à jour : les distances aux éléments qui viennent d'être regroupés sont invalidées, et remplacées par la distance au groupe d'éléments nouvellement formé. Plusieurs définitions de la distance à un groupe

d'éléments existent, et chacune a des caractéristiques qui lui sont propres et induit des propriétés spécifiques. Dans notre cas, la distance à un groupe d'éléments retenue est le saut minimal, défini par [Sibson 1973], comme suit : soient $\mathcal{A}$ et $\mathcal{B}$ deux ensembles d'éléments à clustériser, alors $dist\,(\mathcal{A}, \mathcal{B}) = \min_{a \in \mathcal{A}, b \in \mathcal{B}} (dist\,(a, b))$. Autrement dit, la distance entre deux ensembles d'éléments est la distance minimale entre deux points appartenant respectivement à l'un et l'autre des ensembles. À titre indicatif, l'algorithme de clustering hiérarchique ascendant avec saut minimal est généralement appelé `SLINK`.

On détermine alors quelle est la plus petite distance figurant dans la matrice des distances mise à jour, et on réunit les groupes d'éléments associés. Trois cas de figure sont possibles : (1) les deux groupes sont des éléments isolés et un nouveau groupe d'éléments est créé, (2) un des deux groupes est un élément isolé, et ce dernier est greffé à l'autre groupe, ou (3) deux groupes d'éléments sont fusionnés. La matrice des distances est à nouveau mise à jour, et on itère de la sorte jusqu'à ce qu'il ne reste qu'un seul groupe, contenant tous les éléments.

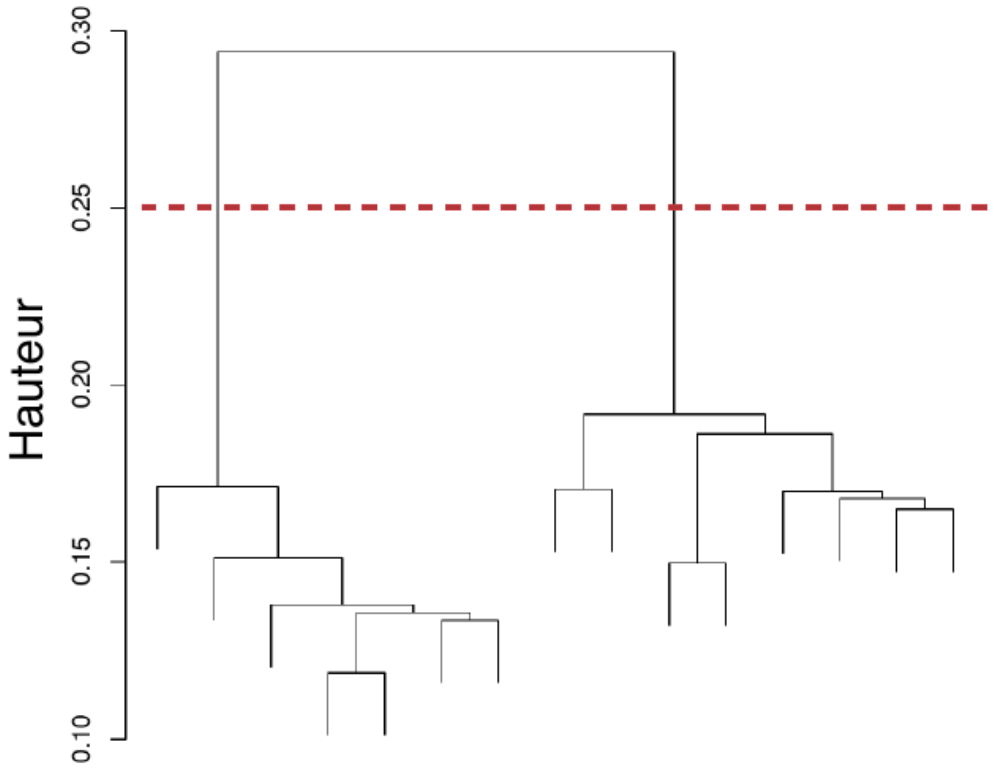


FIGURE 7.9 – Dendrogramme résultant de l'exécution de l'algorithme de clustering hiérarchique ascendant avec saut minimal pour un profil applicatif associé à `NAMD`. La ligne pointillée rouge représente un critère de coupe classique : définir une hauteur de coupe. Ici, cela conduit à la formation de deux clusters.

Le processus de clustering est généralement représenté par un dendrogramme, tel que celui présenté par la figure 7.9 qui résulte de l'exécution de `SLINK` pour `NAMD`. Chaque terminaison correspond à un des éléments à clustériser, et les traits horizontaux qui relient deux groupes d'éléments ayant été fusionnés se voient attribuer une hauteur, qui

correspond à la distance entre les groupes concernés. Un « critère de coupe » doit ensuite être appliqué pour extraire les clusters du dendrogramme. Cela revient à définir quels segments verticaux doivent être coupés pour définir les groupes d'éléments qui formeront des clusters. Le critère de coupe le plus simple est de définir une hauteur de coupe, comme l'illustre la figure 7.9. Un critère de coupe spécifique au domaine des profils applicatifs HPC a été conçu et implémenté. Il est détaillé par la sous-section 7.5.4.

L'utilisation de SLINK avec le critère de coupe spécifique mentionné à l'instant pour clustériser les instances périodiques exhibées par les 4 profils applicatifs considérés a conduit à d'excellents résultats, rassemblés dans le tableau 7.3.

TABLE 7.3 – Résultat du clustering avec l'algorithme des SLINK pour les 4 profils applicatifs dont les instances périodiques ont été annotées.

SLINK	Adéquation	Pureté	Précision	Rappel
NAMD	Oui	1.00	1.00	1.00
OpenFOAM	Oui	1.00	0.959	0.844
HPCG	Oui	0.991	0.998	1.00
NEMO	Oui	1.00	0.994	0.824

En effet, l'algorithme s'est révélé non seulement adéquat pour l'ensemble des profils applicatifs considérés, mais a également affiché une pureté proche de la perfection : seule une instance périodique aberrante s'est glissée dans un des clusters que l'algorithme a construit pour HPCG. En contrepartie, ce dernier a écarté plus d'instances périodiques pertinentes des clusters significatifs que OPTICS, ce qui s'est traduit par un rappel inférieur. Néanmoins, comme précisé en introduction de cette sous-section, adéquation et pureté ont plus de valeur que précision et rappel, même si ceux exhibés par SLINK étaient tout à fait satisfaisants. Les clusters construits contenaient suffisamment d'instances périodiques pour envisager la construction de motifs représentatifs. Du point de vue des performances, SLINK s'est également montré satisfaisant même si légèrement plus lent qu'OPTICS. En effet, comme démontré dans [Sibson 1973], la complexité de l'algorithme de clustering est en $\mathcal{O}(n^2)$, et ne nécessite qu'un calcul de la matrice des distances, à l'instar de OPTICS. Par contre, une seule et unique configuration des hyper-paramètres impliqués dans le critère de coupe a conduit à l'ensemble des résultats expérimentaux. En d'autres termes, il a été possible de traiter un panel de profils applicatifs avec une unique configuration par défaut. Du point de vue du critère 2, c'est un énorme avantage de SLINK par rapport à OPTICS. Et, de mon point de vue, le fait de raisonner sur le dendrogramme produit par l'algorithme de clustering permet, et facilite, l'élaboration d'une configuration par défaut permettant de couvrir une large proportion des séries temporelles d'un domaine spécifique. Tout cela mis bout à bout conduit à la conclusion suivante : l'algorithme de clustering qui a été retenu est SLINK.

7.5.4 Clustering hiérarchique ascendant avec saut minimum : détails du critère de coupe spécifique

Comme précisé dans la sous-section 7.5.3, un critère de coupe spécifique pour `SLINK` a été conçu et implémenté. Ce dernier s'appuie sur la phénoménologie associée à la famille de séries temporelles représentant des profils applicatifs `HPC`. En effet, les analyses couplées d'un ensemble de profils associés à des applications `HPC`, et des dendrogrammes résultant du clustering des instances périodiques qu'ils exhibent ont conduit aux observations suivantes :

— Les instances périodiques associées à une même périodicité sont très similaires, bien que bruitées, ce qui induit que les distances les séparant sont faibles relativement à la distance entre deux instances périodiques décorrelées. Ainsi, un cluster associé à une périodicité a tendance à être dense ;

— Même une infime différence entre deux motifs périodiques se traduit par une distance élevée entre les instances périodiques qu'ils engendrent respectivement, par comparaison à la distance entre deux instances périodiques issues de la même périodicité. En conséquence, la distance, au sens de `SLINK`, entre des clusters associés à des périodicités différentes tend à être importante. Cela a notamment été mis en lumière par les profils associés à `NAMD`, qui présentent deux périodicités très similaires, mais dont les occurrences respectives sont séparées par de grandes distances ;

— Les instances périodiques aberrantes ont tendance à être situées à la racine des dendrogrammes, du fait de leurs distances élevées aux clusters d'instances périodiques engendrées par des périodicités.

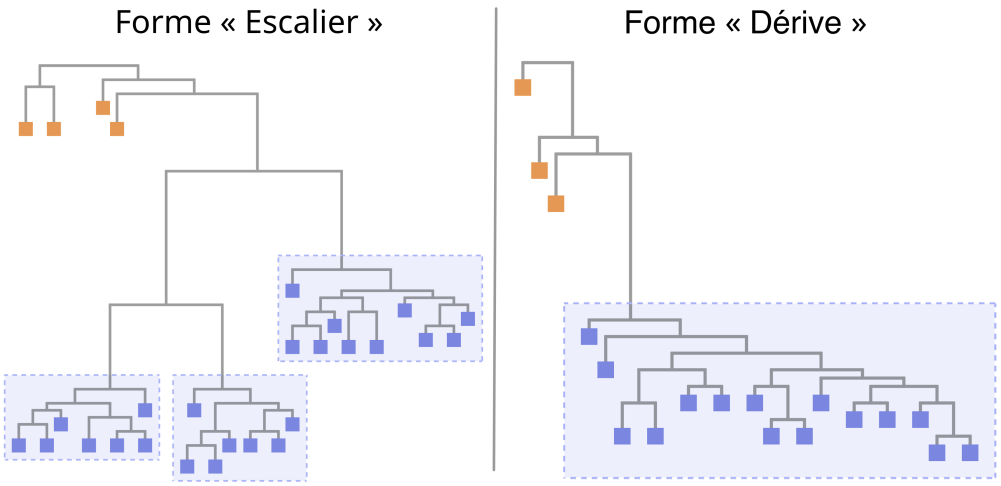


FIGURE 7.10 – Les deux structures récurrentes qui ressortent de l'étude des dendrogrammes associés au clustering d'instances périodiques extraites d'un profil applicatif `HPC`. La forme « escalier » qui est représentative des profils exhibant plusieurs périodicités distinctes, et la forme « dérive » qui est typique des profils exhibant une unique périodicité.

Une des conséquences de l'ensemble de ces observations est que les dendrogrammes semblent ne pouvoir prendre qu'une des deux structures « types » présentées par la figure 7.10.

Lorsque le profil applicatif analysé n'exhibe qu'une seule périodicité, le dendrogramme associé au clustering des instances périodiques qu'elle a engendrées ressemble à la forme « dérive ». En effet, les distances séparant lesdites instances périodiques étant faibles, ces dernières s'amoncèlent, potentiellement par petits groupes, en dérivant vers la gauche (ou vers la droite en fonction de la manière selon laquelle le dendrogramme est construit) par petits sauts de hauteur. Les instances périodiques aberrantes étant éloignées du cluster ainsi formé, un grand saut de hauteur les en sépare.

Dans le cas où le profil applicatif analysé exhibe plusieurs périodicités, la forme « escalier » est représentative de l'allure que le dendrogramme associé prendra. Comme cela a été observé précédemment, chaque périodicité engendre un cluster d'instances périodiques dense, avec de faibles distances entre lesdites instances périodiques. Par contre, la distance entre deux occurrences de motifs périodiques différents étant élevée, les clusters associés aux périodicités seront significativement séparés et distincts. Ils fusionneront donc en formant des paliers de hauteurs croissantes, assimilables à un escalier. Encore une fois, les instances périodiques aberrantes se trouvent en haut de cet escalier du fait de leurs distances élevées aux clusters formés.

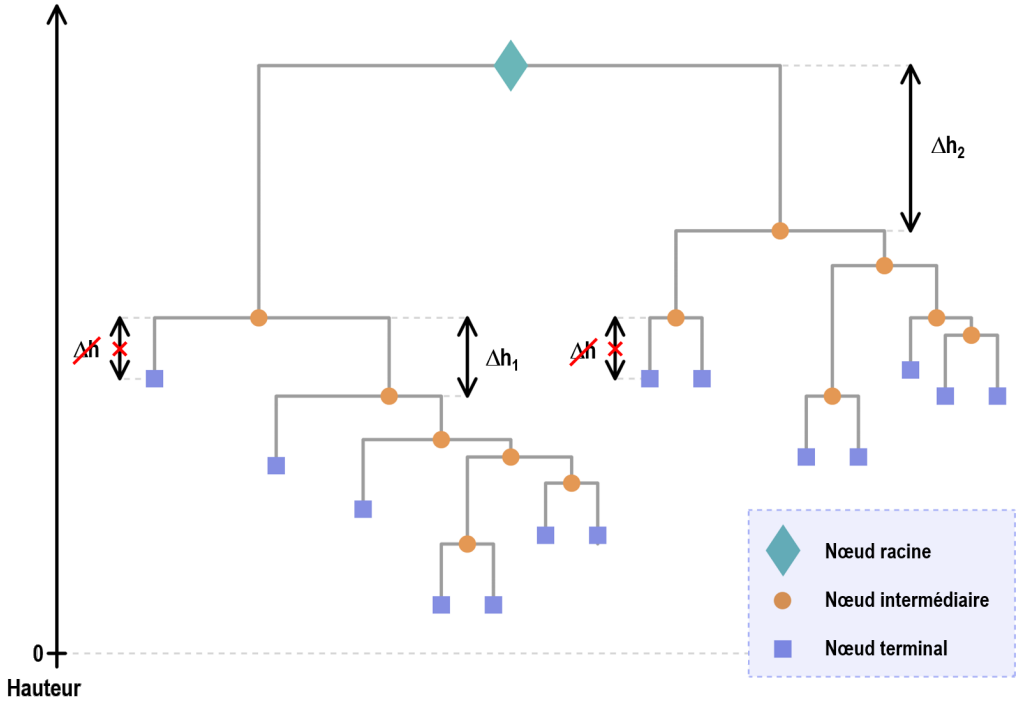


FIGURE 7.11 – Schéma illustrant la définition d'un saut de hauteur au sein d'un dendrogramme, c'est-à-dire la différence de hauteur entre deux nœuds intermédiaires, ou entre le nœud « racine » du dendrogamme et un de ses nœuds intermédiaires.

L'idée sous-jacente au critère de coupe est alors d'exploiter la différence d'échelle entre les sauts de hauteur au sein d'un cluster d'une part, et les sauts de hauteur entre clusters, ou séparant un cluster d'une instance périodique aberrante d'autre part. Avant

de spécifier l'algorithme associé au critère de coupe, définissons clairement ce qu'est un saut de hauteur en nous appuyant sur la figure 7.11.

Ainsi, un **saut de hauteur**, notée Δh est la différence de hauteur, représentée par un segment vertical, existant entre deux nœuds intermédiaires d'un dendrogramme, ou entre son nœud « racine » et un de ses nœuds intermédiaires. Par contre, les segments verticaux reliant les nœuds terminaux aux paliers horizontaux représentant la distance de ces derniers aux groupes avec lesquels ils fusionnent ne sont que des artifices graphiques de présentation. Ce ne sont pas des sauts de hauteur. À noter que, au sein d'un saut de hauteur, le nœud associé à la plus faible hauteur est appelé **nœud inférieur**, et celui associé à la hauteur la plus élevée est appelé **nœud supérieur**.

Le pseudo-code 4 décrit l'algorithme associé au critère de coupe pour `SLINK` spécifique au domaine des profils applicatifs `HPC`. Il est configurable via deux hyper-paramètres α et δ dont l'impact sur son fonctionnement sera explicité par la suite, et il retourne la liste des clusters (via `cluster`) qu'il a extraits du dendrogramme sur lequel il a été appliqué, spécifié via `dendro`.

Pour commencer, $\mathcal{N}$ représente le nombre de nœuds terminaux appartenant au dendrogramme, et donc le nombre d'instances périodiques à clustériser. Ensuite, un parcours du dendrogramme permet de déterminer l'ensemble ΔH des sauts de hauteur qu'il exhibe. La structure de données associée à un saut de hauteur contient les informations relatives au segment vertical associé, également appelé **lien**, c'est-à-dire ses nœuds inférieur et supérieur, ainsi que la différence de hauteur entre ces deux nœuds. Il est alors possible de déterminer quelle est la valeur médiane, notée `med`, des différences de hauteur associées à l'ensemble des sauts de hauteur ΔH. L'ensemble des sauts de hauteur est alors analysé pour déterminer lesquels répondent aux deux critères suivants :

— La différence de hauteur associée au saut de hauteur considéré est au moins δ fois supérieure à la différence de hauteur médiane exhibée par le dendrogramme ;

— Le sous-dendrogramme dont la racine est le nœud inférieur du saut de hauteur considéré, qui représente un potentiel cluster, contient au moins $\alpha\%$ des instances périodiques contenues dans le dendrogramme complet.

Le premier critère découle des observations faites en introduction de cette sous-section. En effet, la différence de hauteur médiane est vraisemblablement, pour ne pas dire obligatoirement, représentative de la distance typique entre deux instances périodiques engendrées par la même périodicité [8]. Et les distances séparant différents clusters, ou bien encore des instances périodiques aberrantes desdits clusters, sont élevées par rapport à cette distance typique. En sélectionnant les sauts de hauteur associés à des différences de hauteurs élevées par rapport à la médiane desdites différences de hauteur, on vise les liens inter-clusters et les liens séparant les clusters des instances périodiques aberrantes.

Le second critère sert un double rôle. Pour commencer, il permet de ne pas créer une constellation de clusters contenant quelques instances périodiques aberrantes chacun. En effet, ces dernières sont fréquemment très différentes les unes des autres, ce

8. Si l'on arrive à l'étape de clustering, c'est que des instances périodiques ont été détectées et extraites lors de la première étape de la méthodologie associée à `Phase-TA`. De plus, `SLINK` fusionne les groupes d'éléments exhibant la distance la plus courte à chaque itération, ce qui implique que la majorité des sauts de hauteur sont intra-clusters, et donc entre des groupes d'instances périodiques engendrés par les mêmes périodicités.

qui implique parfois des sauts de hauteur entre des multiples petits groupes d'instances périodiques aberrantes satisfaisant le premier critère. Ensuite, il n'est pas rare qu'un cluster soit constitué d'un ensemble de sous-groupes très denses d'instances périodiques [9]. Il est alors possible que la différence de hauteur médiane soit associée à la distance entre deux instances périodiques appartenant à un tel sous-groupe. Dans ce cas, bien que les distances entre ces différents sous-groupes soient faibles lorsque comparées par exemple aux distances inter-clusters, il est envisageable que les différences de hauteur associées satisfassent le premier critère. En imposant un nombre minimal d'instances périodiques pour être éligible à une coupe dans le dendrogramme, on évite la fragmentation des clusters associés à des périodicités en sous-groupes très denses contenant seulement quelques instance périodiques.

Une fois les liens éligibles à une coupe identifiés, on peut procéder à la coupe en question et former les sous-dendrogrammes associés aux différents clusters. En appliquant de nouveau le critère sur la cardinalité minimale que doit avoir un cluster, on peut alors filtrer les instances périodiques aberrantes et les sous-dendrogrammes associés à des périodicités peu significatives, de sorte à se concentrer sur celles qui dominent le temps d'exécution de l'application analysée.

Algorithme 4 Critère de coupe pour `SLINK` spécifique au domaine des profils d'applications de calcul à haute performance.

```
1: procedure SLINK_TREE_CUT(in dendro, in α, in δ, out clusters)
2:     N ← compute_nb_term_nodes(dendro.root);
3:
4:     ΔH ← compute_hjumps(dendro);
5:     med ← determine_median_hdiff(ΔH);
6:
7:     vect_init(links_to_cut);
8:
9:     for Δh ∈ ΔH do
10:        n_TS ← compute_nb_term_nodes(Δh.link.inf_node);
11:
12:        if (Δh.hjump > δ · med) and (n_TS > α · N) then
13:            vect_add(links_to_cut, Δh.link);
14:        end if
15:    end for
16:
17:    clusters ← cut_links(dendro, links_to_cut);
18:
19:    for cluster ∈ clusters do
20:        if compute_nb_term_nodes(cluster.root) < α · N then
21:            vect_remove(clusters, cluster);
22:        end if
23:    end for
24: end procedure
```

9. Typiquement parce que le bruit appliqué à la série temporelle est localement très homogène.

Comme cela a été mentionné par la sous-section 7.5.3, l'utilisation du critère de coupe décrit par le pseudo-code 4 a permis d'extraire des clusters adéquats et purs, avec la configuration par défaut associée, à savoir : $\delta = 5$ et $\alpha = 0.15$. Ces deux valeurs ont été déterminées empirement, et se sont révélées adaptées pour **l'intégralité** des profils applicatifs analysés dans le cadre des travaux de recherche. Il est ainsi possible de construire une configuration par défaut adaptée à une famille de séries temporelles. Pour conclure, ajoutons que le dendrogramme résultant du clustering avec `SLINK` se prête particulièrement à la définition d'un critère de coupe spécifique à un type de séries temporelles, avantage certain dans le cadre de `Phase-TA`.

7.6 Construire les motifs représentatifs

Cette dernière étape de la méthodologie implémentée par `Phase-TA` a pour but de construire les motifs représentatifs associés aux périodicités engendrant les comportements localement périodiques détectés par la première étape. Pour ce faire, `Phase-TA` implémente l'algorithme `Dynamic time warping Barycentric Averaging` (`DBA`), qu'il applique aux clusters d'instances périodiques construits lors de la seconde étape. Les instances périodiques appartenant à un cluster sont alors moyennées pour construire le motif représentatif associé à la périodicité ayant engendré lesdites instances périodiques, comme cela est illustré par la figure 7.12.

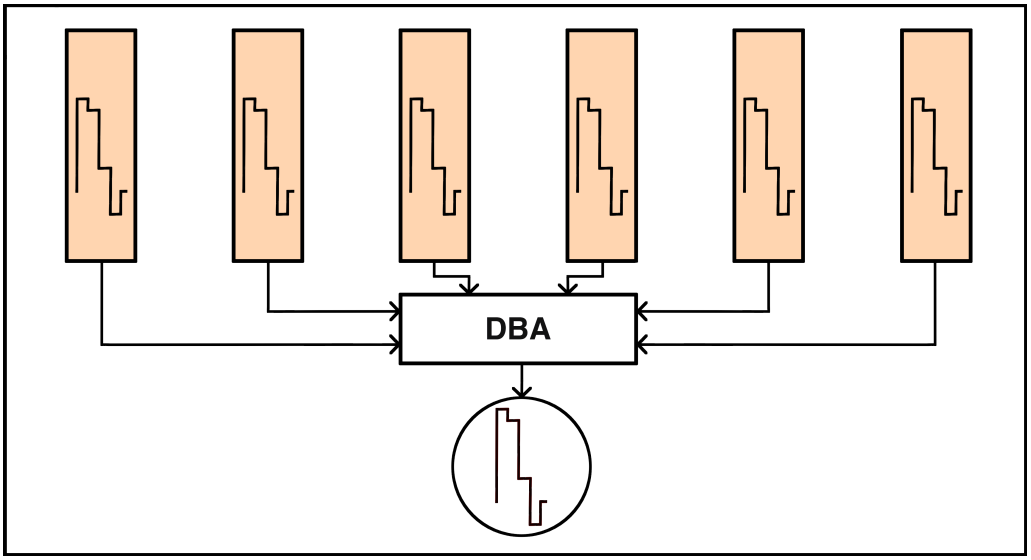


FIGURE 7.12 – Schéma de principe associé à l'étape de construction des motifs représentatifs associés aux périodicités qui ont engendré les comportements localement périodiques détectés par `Phase-TA`. Pour ce faire, les instances périodiques appartenant à un cluster issu de la seconde étape de la méthodologie sont moyennées en appliquant l'algorithme `DBA`.

Après avoir présenté l'algorithme `DBA` dans la sous-section 7.6.1, nous nous attarderons sur son initialisation dans la sous-section 7.6.2. Enfin, la sous-section 7.6.3 détaillera la manière dont `DBA` a été parallélisé dans le cadre de l'implémentation de `Phase-TA`.

7.6.1 Présentation du `Dynamic time warping Barycentric Averaging`

`DBA` est un processus de raffinement itératif présenté par [Petitjean 2011]. Il utilise `DTW` (cf. sous-section 2.3.3) comme mesure de similarité entre les séries temporelles qu'il moyenne, profitant notamment du fait que `DTW` appaire les points des séries temporelles dont il calcule la similarité. Le principal avantage à utiliser `DTW` comme mesure de similarité est que cela permet de s'abstraire d'une partie des perturbations temporelles (bruits d'insertion et de suppression, cf. sous-section 7.1.3) inhérentes aux séries temporelles affectées par des variations aléatoires. En effet, comme le montre la figure 7.13, les distances

classiques, telles que la distance euclidienne, sont fortement impactées par la présence de perturbations temporelles, et les algorithmes de moyennage de séries temporelles les utilisant ont tendance à produire des motifs moyens très peu pertinents, relativement aux séries temporelles moyennées.

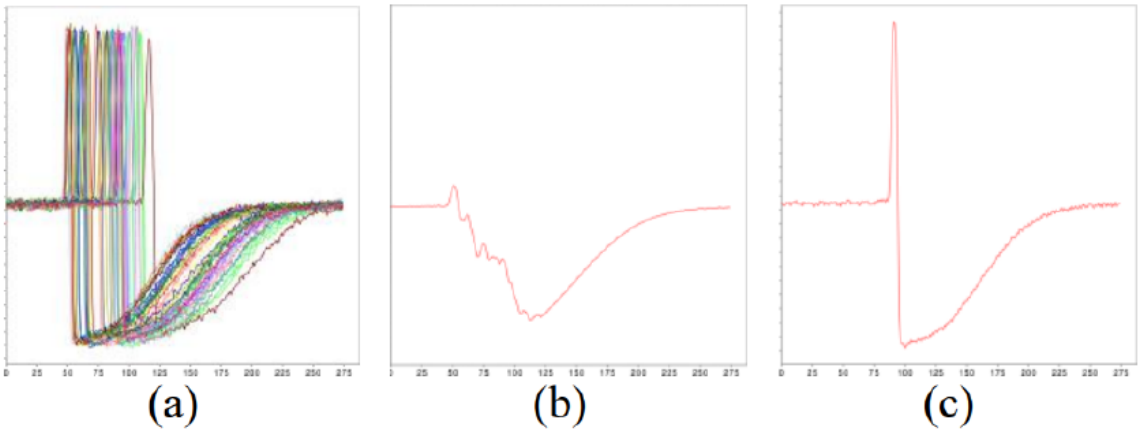


FIGURE 7.13 – (a) Ensemble de séries temporelles à moyenner. (b) Motif moyen construit en calculant la valeur moyenne de chaque échantillon à l'aide de la distance euclidienne. (c) Motif moyen construit par `DBA`, qui utilise `DTW` comme mesure de similarité.
Source : [Petitjean 2016].

`DBA` est initialisé avec une série temporelle arbitraire, qu'il va modifier à chaque itération pour améliorer sa pertinence en tant que motif moyen des séries temporelles dont il cherche à construire la représentation moyenne. Pour évaluer la pertinence du motif moyen, il calcule son `WGSS` par rapport aux séries temporelles moyennées (cf. sous-section 2.3.4). Dans [Petitjean 2011] et [Petitjean 2014], deux résultats fondamentaux concernant le processus itératif implémenté par `DBA` sont démontrés :

— Il est garanti que chaque itération du processus de raffinement du motif moyen ne peut que faire décroître le `WGSS` entre ledit motif moyen et les séries temporelles moyennées ;

— La série temporelle dont le `WGSS` par rapport aux séries temporelles moyennées est minimal est unique. De plus, le motif moyen construit par `DBA` tend asymptotiquement vers cette dernière, lorsque le nombre d'itérations du processus de raffinement appliquées augmente.

Ainsi, il est certain que chaque itération de `DBA` rendra le motif moyen plus pertinent, relativement aux instances périodiques moyennées. Chaque itération du processus de raffinement implémenté par `DBA` est composée des étapes suivantes :

1. Calcul de `DTW` entre le motif moyen et chaque série temporelle de l'ensemble de séries temporelles à moyenner ;

2. Pour chaque point $\mathcal{P}$ du motif moyen, construction de l'ensemble $\mathcal{S}$ des points des séries temporelles à moyenner qui sont alignés avec $\mathcal{P}$, d'après `DTW` ;

3. Calcul du barycentre de $\mathcal{S}$, qui devient la valeur raffinée de $\mathcal{P}$ (i.e. la nouvelle valeur de $\mathcal{P}$), à la fin de l'itération du processus de raffinement.

Comme pour n'importe quel processus de raffinement, l'initialisation du motif moyen (abordée par la sous-section 7.6.2) et le critère de terminaison sont primordiaux, et ont un impact fort sur la qualité du motif moyen construit et les performances de l'algorithme. En ce qui concerne le critère de terminaison du processus de raffinement itératif implémenté par `Phase-TA`, ce dernier s'arrête si au moins l'une des deux conditions suivantes est satisfaite :

1. N_{iter} itérations ont été exécutées (par défaut, $N_{iter} = 31$) ;
2. n_{cons} itérations consécutives ont induit une diminution relative du `WGSS` entre le motif moyen et les instances périodiques à moyenner inférieure à t_{term} (par défaut, $n_{cons} = 5$ et $t_{term} = 2.5\%$).

Les valeurs par défaut des paramètres impliqués dans le critère de terminaison ont été définies empiriquement, à partir des expérimentations menées avec `Phase-TA`. Elles permettent l'inférence de motifs représentatifs pertinents, comme cela est démontré dans le chapitre 8, tout en maintenant le coût algorithmique du processus de raffinement sous contrôle. En effet, la condition de terminaison 1 garantit qu'un nombre conséquent d'itérations seront exécutées au cas où le processus de raffinement ne semblerait pas converger, de sorte à construire le motif représentatif le plus pertinent possible. Et c'est la condition de terminaison 2 qui évalue la convergence du processus de raffinement : dès que l'évolution du `WGSS` associé au motif représentatif se rapproche suffisamment d'un comportement asymptotique, le processus est stoppé, afin de limiter le nombre d'itérations. Empiriquement, toutes les analyses menées avec `Phase-TA` jusqu'à présent ont conduit à une terminaison anticipée via la condition 2, à l'image de l'évolution du `WGSS` pour l'analyse d'un profil applicatif de `NEKbone` présentée par la figure 7.14.

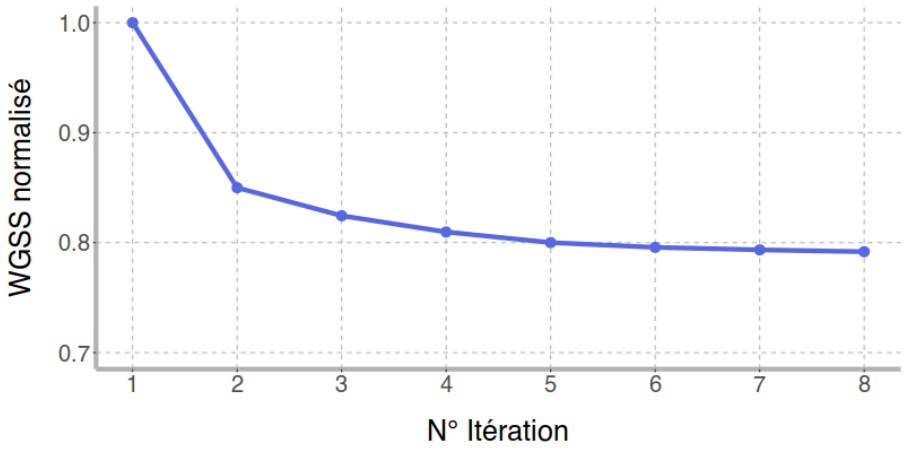


FIGURE 7.14 – Évolution du `WGSS` lors du raffinement itératif du motif représentatif associé à un profil applicatif de `NEKbone`. Un comportement asymptotique se dessine en moins de 10 itérations, traduisant une terminaison anticipée de `DBA`.

7.6.2 Initialisation du motif représentatif

Concernant l'initialisation du motif moyen, il est souligné dans [Petitjean 2011] que DBA a tendance à converger plus rapidement lorsque le motif moyen initial est l'une des séries temporelles à moyenner. Or, en ce qui concerne Phase-TA, l'étape de clustering des instances périodiques (décrite par la section 7.5) implique de calculer la matrice des distances associée au groupe d'instances périodiques à moyenner. En conséquence, l'ensemble des calculs de DTW pour toutes les paires d'instances périodiques sont disponibles au moment de l'initialisation de DBA. Il est donc possible, avec un impact négligeable sur les performances de Phase-TA, de déterminer quelle instance périodique minimise la distance cumulée par rapport aux autres instances périodiques, et d'initialiser DBA avec ladite instance périodique. Cette initialisation du motif moyen, notée (1) par la suite, a été comparée empiriquement à l'initialisation proposée par [Petitjean 2011], et notée (2) par la suite, à savoir choisir aléatoirement une instance périodique parmi celles à moyenner pour initialiser le motif représentatif. Pour ce faire, parmi la collection de profils applicatifs disponibles au moment du développement de cette partie de Phase-TA, 15 ont été sélectionnés (3 pour chacune des applications suivantes : NEKbone, HPCG, NAMD, NEMO, et FT). Les deux initialisations ont été implémentées et les 15 profils ont ensuite été analysés 3 fois chacun, en utilisant les deux implémentations. Les nombres d'itérations du processus de raffinement et les WGSS associés aux motifs représentatifs construits ont ensuite été moyennés par profil et par initialisation. Les résultats obtenus par les deux initialisations ont ensuite été comparés par profil, ce qui a conduit aux conclusions suivantes :

— Avec l'initialisation (1), le processus itératif de raffinement converge plus rapidement que lorsque l'initialisation (2) est utilisée, nécessitant en moyenne 3.4 itérations de moins pour atteindre un comportement asymptotique ;

— Les WGSS des motifs représentatifs construits lorsque l'initialisation (1) est utilisée sont inférieures de 2.0% en moyenne à leurs homologues associés à l'initialisation (2).

En conséquence, le schéma d'initialisation de DBA retenu pour Phase-TA est le premier, c'est-à-dire la sélection de l'instance périodique qui minimise la somme des DTW calculées relativement aux autres instances périodiques à moyenner.

7.6.3 Parallélisation de l'algorithme DBA

Chaque itération du processus de raffinement implique de calculer DTW entre le motif représentatif en cours de raffinement et l'ensemble des instances périodiques. La complexité algorithmique associée est très élevée, et le temps d'exécution de DBA est clairement et largement dominé par ces calculs. Les implémentations disponibles de DBA au moment du développement de Phase-TA, y compris celle de référence proposée par les auteurs de [Petitjean 2011], étaient toutes séquentielles. Afin d'améliorer les performances de Phase-TA en vue d'être capable de faire des analyses à chaud, une version parallèle de DBA a été implémentée. Le pseudo-code 5 présente la parallélisation de l'algorithme de raffinement appliqué lors d'une itération de DBA.

Algorithme 5 Parallélisation d'une itération du processus de raffinement de `DBA`

```
1: procedure REFINEMENT_PROCESS_ITERATION(in periodic_instances, in out
   rep_pattern)
2:     init_global_association_array((g_assoc);
3:
4:     for inst ∈ periodic_instances do
5:         init_dtw_matrix(dtw_matrix);
6:         init_partial_association_array(array_assoc, p_assoc);
7:         compute_dtw(dtw_matrix, rep_pattern, inst);
8:         fill_partial_association_array(p_assoc, dtw_matrix);
9:     end for
10:
11:    fuse_partial_association_arrays(g_assoc, array_assoc);
12:
13:    for sample_id ∈ ⟦0;length(rep_pattern) − 1⟧ do
14:        rep_pattern [sample_id] ← compute_barycenter(g_assoc [sample_id]);
15:    end for
16: end procedure
```

Avant de détailler le pseudo-code 5, précisons que les boucles `for` ont été parallélisées en utilisant `OpenMP 4.5`. En entrée de la procédure de raffinement, on trouve la collection des instances périodiques que l'on cherche à moyenner (`periodic_instances`), et le motif représentatif (`rep_pattern`) qui sera mis à jour en place. L'idée sous-jacente à la parallélisation est de construire un tableau d'association partiel par instance périodique : un tableau qui pour chaque point du motif représentatif contiendra l'ensemble des points de l'instance périodique considérée qui sont associés au dit point d'après `DTW`[10]. L'ensemble des tableaux d'association partiels sont rassemblés dans une collection, de sorte à pouvoir être fusionnés par la suite. Cette fusion est exécutée séquentiellement, par un unique thread, et conduit à la création d'un seul tableau d'association contenant, pour chaque point du motif représentatif, l'ensemble des points des instances périodiques qui lui sont associés d'après `DTW`. La principale raison pour laquelle l'étape de fusion n'est pas parallélisée est qu'il aurait été nécessaire d'introduire des éléments de synchronisation entre les différents threads, et que le gain en termes de performances aurait été marginal, voire inexistant. Une fois le tableau d'association global construit, il est possible de mettre à jour le motif représentatif en calculant, pour chacun de ses points, le barycentre des points qui lui sont associés d'après `DTW`. Cette étape est également exécutée en parallèle. On peut d'ailleurs remarquer que les étapes de l'algorithme qui sont parallélisées ne nécessitent aucune synchronisation ente les différents threads.

Étant donné que la parallélisation du code rend son exécution mono-threadée sous-efficace par rapport à sa version séquentielle, cette dernière a également été implémentée pour couvrir le cas où `Phase-TA` serait exécuté avec un seul thread. En conséquence, il a été possible de mesurer l'accélération (« speedup » pour les anglophones) associée à la parallélisation de `DBA`. Pour ce faire, le code source des deux implémentations a été

10. Dans les faits, c'est donc un tableau de vecteurs, puisque les points de l'instance périodique associés au point considéré du motif représentatif sont stockés dans un vecteur.

annoté pour mesurer leurs temps d'exécution à l'aide de `clock_gettime` avec l'horloge `CLOCK_MONOTONIC_RAW`. Pour chacune des 5 applications impliquées dans l'évaluation de l'impact du critère d'initialisation de `DBA` (cf. sous-section 7.6.2), 1 profil applicatif a été analysé 5 fois, en faisant varier le nombre de threads, sur un nœud de `dahu` (cf. sous-section 4.1.2). Ces temps d'exécution ont été moyennés par implémentation (séquentielle vs parallèle), et les facteurs d'accélération associés ont pu être calculés. Ils sont présentés par le tableau 7.4.

TABLE 7.4 – Étude du facteur d'accélération engendré par la parallélisation du processus de raffinement de `DBA`, par rapport à l'implémentation séquentielle, en fonction du nombre de threads.
T_{mono} correspond aux temps moyens pour les exécutions mono-threadées.

Nombre de threads	T_{mono}	2	4	8	16	32
Accélération - `NEKbone`	3.14 s	1.76	3.58	7.22	14.50	29.13
Accélération - `HPCG`	19.9 s	1.87	3.84	7.74	15.34	30.23
Accélération - `NAMD`	2.03 s	1.92	3.91	7.85	15.64	31.09
Accélération - `NEMO`	4.21 s	1.81	3.69	7.60	14.89	29.33
Accélération - `FT`	2.68 s	1.89	3.84	7.72	15.10	30.41

On peut observer que le facteur d'accélération lié à la parallélisation du processus de raffinement de `DBA` passe globalement à l'échelle avec le nombre de threads, ce qui permet de réduire substantiellement le temps d'exécution associé à la construction du motif représentatif. La principale raison sous-jacente à ce passage à l'échelle est le fait que les tâches effectuées en parallèle sont homogènes : toutes les instances périodiques ont (quasiment) la même longueur, et tous les calculs de `DTW` et toutes les constructions des vecteurs d'association partiels induisent des charges de travail sensiblement similaires. De même, la résidence en mémoire est à la fois homogène, et favorable au passage à l'échelle puisqu'elle se situe en mémoire cache L_3, avec des structures de données stockées linéairement (le `prefetcher` peut donc jouer son rôle et prévenir une majorité des défauts de cache). De plus, chaque thread alloue les structures de données qu'il modifie, ce qui permet d'éviter les latences associées à l'accès en écriture à des données hébergées sur d'autres nœuds `NUMA` [11].

Les légères différences d'accélération constatables entre les différentes applications trouvent vraisemblablement leur explication dans l'étape de fusion des vecteurs d'association partiels. En effet, cette étape est exécutée séquentiellement, et son temps d'exécution dépend à la fois du nombre d'instances périodiques et de leurs longueurs. Ainsi, les différentes applications exhibant des périodicités de longueurs variées et des nombres de régions périodiques différents, la fusion des vecteurs d'association partiels s'exécute en un temps différent d'une application à l'autre, expliquant les différences légères entre les facteurs d'accélération. Pour finir, notons que les différences de performance d'une application à l'autre pour les cas mono-threadés s'expliquent également par le fait que

11. [Trahay 2018] donne un bon aperçu de l'impact que peuvent avoir ces latences sur les performances, et du déséquilibre entre les différents threads qu'elles peuvent engendrer.

les applications exhibent des nombres différents d'instances périodiques de longueurs différentes. L'impact de ces deux « paramètres » sur les performances de `Phase-TA` est notamment discuté dans la section 8.4.

7.7 Récapitulatif des points clés concernant Phase-TA au fil d'une analyse illustrée étape par étape d'un profil applicatif HPC réel

Cette section passe en revue les points clés concernant Phase-TA en s'appuyant sur une analyse d'un profil applicatif HPC réel, associé à l'application HPCG. Chaque étape importante de la méthodologie est ainsi illustrée en spécifiant ses états d'entrée et de sortie.

Il s'agit d'analyser un profil applicatif exhibant des comportements localement périodiques engendrés par des périodicités. Deux extraits du profil en question sont présentés par la figure 7.15, on peut notamment y observer des occurrences de trois motifs périodiques distincts. La première étape de la méthodologie consiste à détecter les régions périodiques pour en extraire les instances périodiques. Ces dernières sont regroupées dans une unique collection, dont un sous-ensemble est présenté par la figure 7.16.

Le but de la seconde étape de la méthodologie est de clustériser les instances périodiques en fonction des périodicités qui les ont engendrées. Avant de procéder au clustering à proprement parler, deux pré-traitements sont nécessaires. Tout d'abord, les instances périodiques sont triées par paquets en fonction de leurs longueurs. Dans le cas du profil applicatif de HPCG analysé, deux paquets sont formés, comme le montre la figure 7.17. Le second sous-traitement consiste quant à lui à aligner les instances périodiques, au sein d'un paquet, ce qui est illustré par la figure 7.18. Il est alors possible d'appliquer le clustering à chaque paquet considéré individuellement pour construire des clusters qui regroupent les instances périodiques en fonction des périodicités qui les ont engendrées. Comme la figure 7.19 le montre, deux clusters ont été construits à partir du premier paquet d'instances périodiques, et un troisième cluster a été construit à partir du second paquet.

Il est alors temps d'appliquer la troisième étape de la méthodologie sous-jacente à Phase-TA afin de construire les motifs représentatifs des périodicités exhibées par le profil applicatif analysé. Cela est fait en appliquant DBA à chaque cluster d'instances périodiques, et les motifs ainsi obtenus sont présentés par 7.20.

Pour être complet concernant les caractéristiques clés de Phase-TA, précisons que l'optimisation de son code source a été une des considérations premières lors de son implémentation, dans le but de pouvoir proposer des analyses à chaud des profils applicatifs. La section 8.4 est dédiée à l'étude de la complexité algorithmique et des performances de Phase-TA, afin de déterminer si ces dernières permettent justement de conduire de telles analyses.

Pour finir, insistons sur un point fondamental : la conception de l'approche sous-jacente à Phase-TA implique que le comportement de ce dernier soit finement paramétrable, via l'exposition de nombreuses variables de configuration. Cela permet d'ajuster l'analyse à la série temporelle considérée, pour pouvoir en traiter une large gamme sans modifier le code source de Phase-TA. Néanmoins, les multiples algorithmes mis en jeu ont été développés et/ou modifiés de sorte à permettre la construction de configurations adaptées à des familles de séries temporelles spécifiques. Il est alors possible d'analyser une majorité des séries temporelles appartenant à ladite famille sans modifier la configuration de Phase-TA. Dans le cadre de ce manuscrit, les séries temporelles d'intérêt sont

les profils applicatifs HPC. Une part des travaux de recherche autour de Phase-TA ont consisté à construire sa configuration par défaut de sorte à ce qu'elle permette d'analyser un large éventail de profils applicatifs[12]. Cela rend l'usage basique de Phase-TA suffisamment simple et évident, même pour un utilisateur de supercalculateur lambda, ce qui est une condition sine qua non de son adoption dans le domaine du calcul à haute performance.

12. Cela a été confirmé empiriquement, puisque l'intégralité des expériences menées avec Phase-TA et présentées par le chapitre 8 s'appuient sur sa configuration par défaut, sans adaptation.

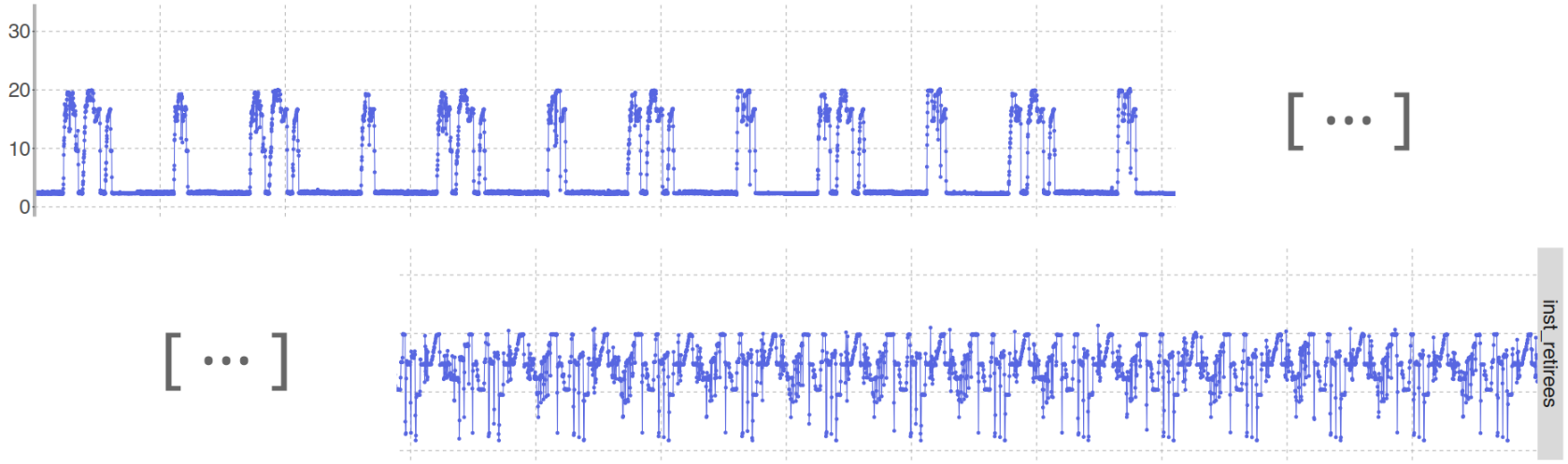


FIGURE 7.15 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : extraits du profil à analyser.
On peut observer des occurrences de trois motifs périodiques, deux d'entre elles sont très similaires et interviennent dans la première partie du profil, alors que les occurrences de la troisième consistuent le second extrait.
À noter que les échelles temporelles des deux morceaux ne sont pas identiques à des fins de présentation, une des périodicités étant largement plus courte que les deux autres.

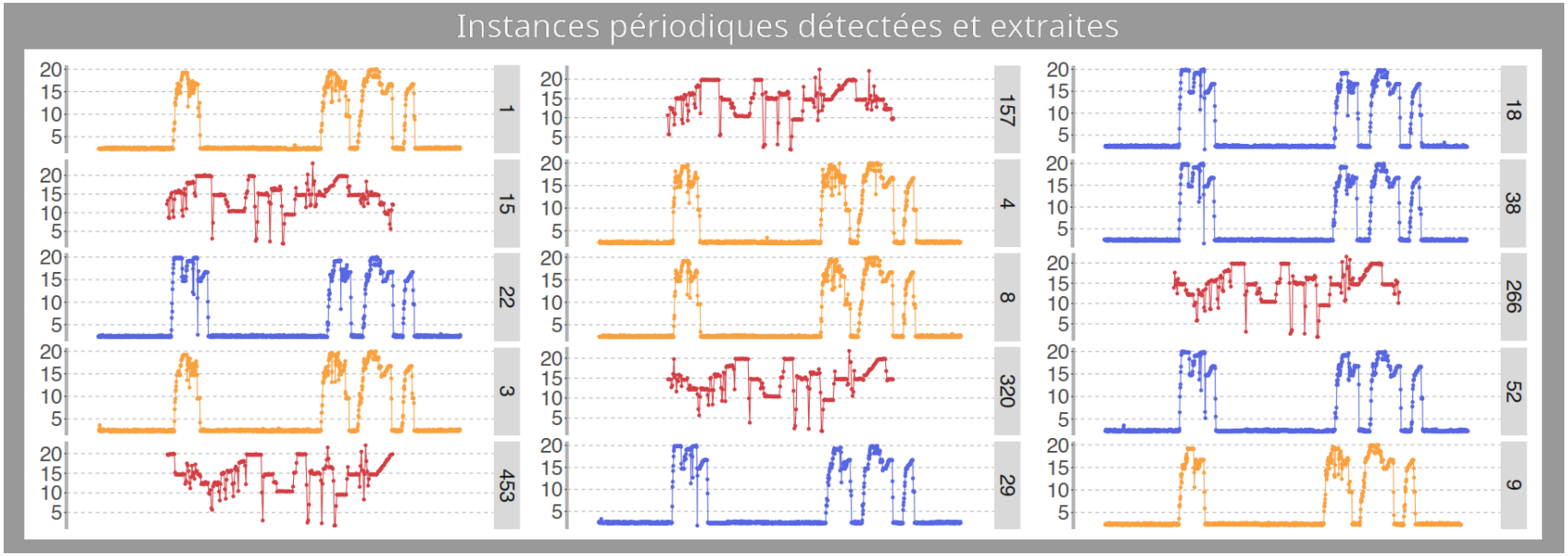


FIGURE 7.16 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : sous-ensemble des instances périodiques détectées et extraites.
Suite à la première étape de la méthodologie implémentée par `Phase-TA`, l'ensemble des instances périodiques sont rassemblées dans une unique collection.
À noter qu'une couleur a été associée à chaque périodicité, et que ce code couleur sera conservé tout du long de cet exemple.

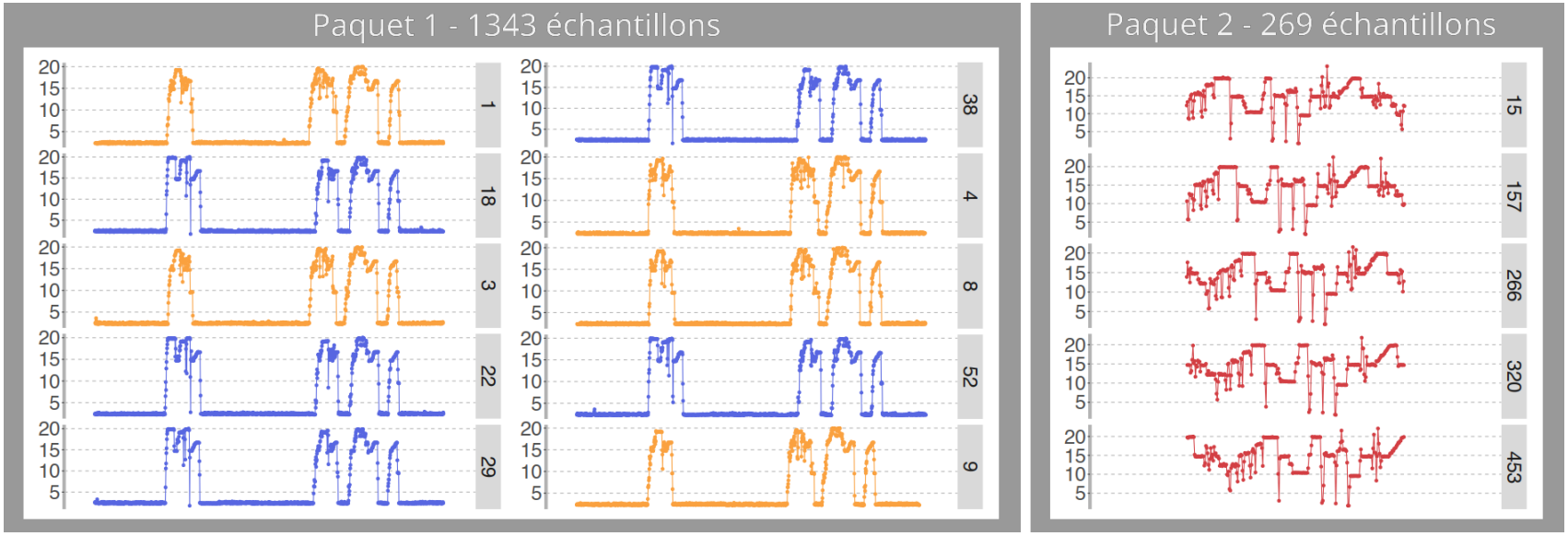


FIGURE 7.17 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : tri par paquets en fonction la longueur des instances périodiques en guise de pré-traitement pour l'étape de clustering.
Chaque paquet est centré sur une longueur, respectivement 1343 et 269, et d'un rayon égal à 5% de ladite longueur. Ainsi, les paquets accueillent les instances périodiques dont les longueurs sont respectivement comprises dans les intervalles $[\![255; 283]\!]$ et $[\![1275; 1411]\!]$.
Les nombres associés aux instances périodiques correspondent à leurs indices d'insertion au sein du paquet.

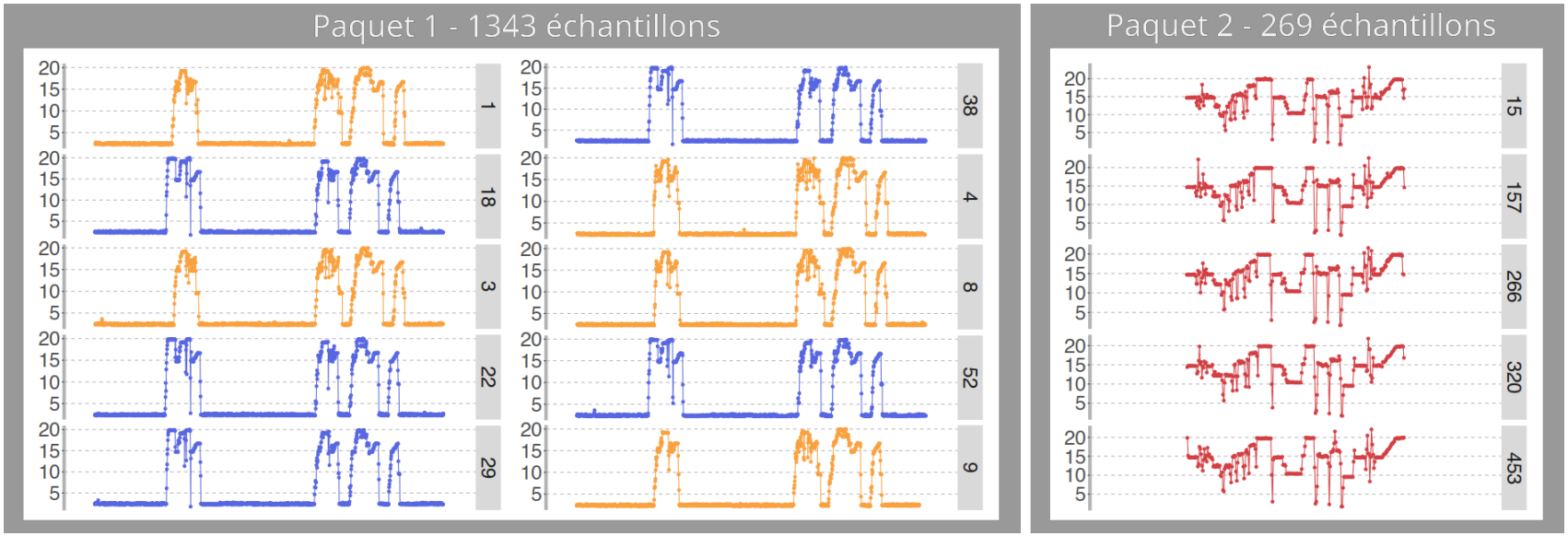


FIGURE 7.18 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : alignement, par paquet, des instances périodiques en guise de pré-traitement pour l'étape de clustering.

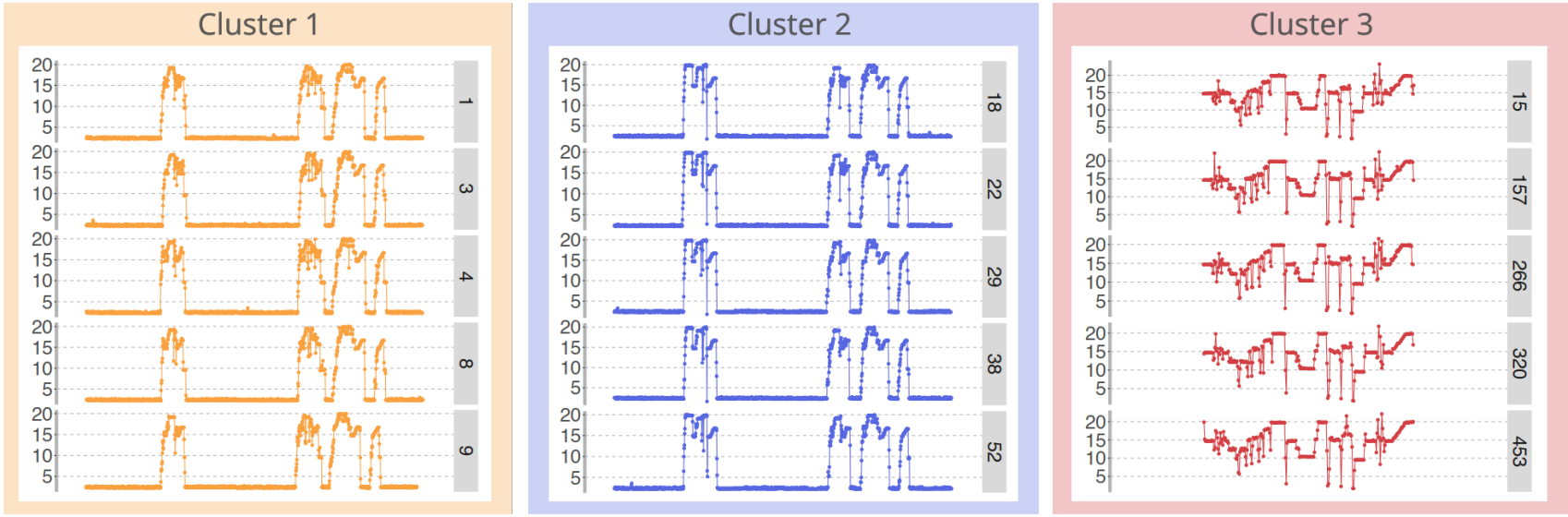


FIGURE 7.19 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : clustering, par parquet, des instances périodiques.
La seconde étape de la méthodologie implémentée par `Phase-TA` conduit à la formation de trois clusters : les clusters 1 et 2 sont issus du paquet 1, alors que le cluster 3 est issu du paquet 2. Chaque cluster regroupe les instances périodiques associées à une périodicité spécifique.

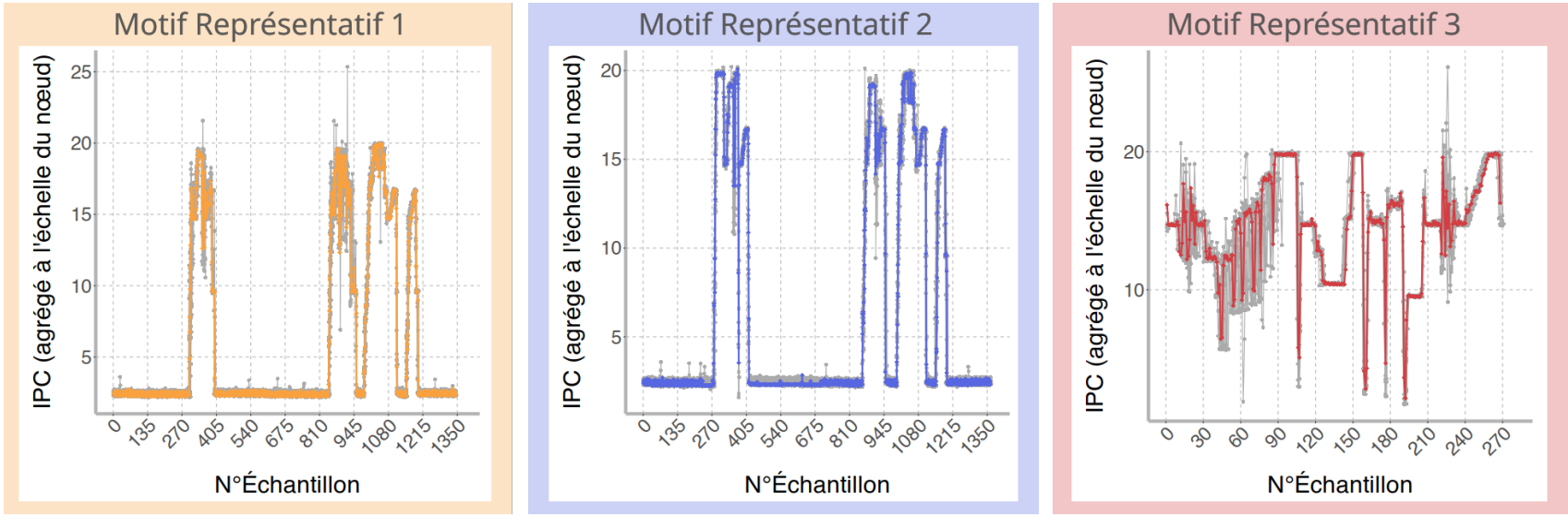


FIGURE 7.20 – Exemple d'analyse étape par étape d'un profil applicatif de `HPCG` par `Phase-TA` : construction des motifs représentatifs associés aux périodicités détectées, en appliquant `DBA` à chaque cluster formé à l'étape précédente.
Les motifs représentatifs, tracés en couleurs, sont superposés à une sélection quelconque de 10 des instances périodiques à partir desquelles ils ont été construits.

Expérimentations

8

Ce chapitre présente un ensemble d'expérimentations menées avec `Phase-TA`, et les différents résultats et conclusions qui en découlent. L'infrastructure sur laquelle l'ensemble de ces expériences ont été conduites est présentée en détail par la sous-section 4.1.2, il s'agit de `dahu`, un cluster `HPC` dédié à la recherche mis à disposition par `Grid'5000`. De la même manière, les applications de calcul à haute performance qui ont servi de support à ces travaux sont : (1) `HPCG`, (2) le composant `TOP/PISCES` de `NEMO 4.0` appliqué au cas d'utilisation `GYRE`, et (3) le solveur `icoFOAM` de `OpenFOAM 20.06` appliqué au cas d'utilisation `cavity`. La section 4.2 présente ces applications plus en profondeur, qui seront désignées par la suite comme le *panel expérimental*.

Profitons de ce paragraphe introductif pour spécifier les lignes directrices de la méthodologie expérimentale suivie, et insistons sur le fait qu'elles **s'appliquent à l'ensemble des expérimentations** présentées par ce chapitre.

Pour commencer, les profils applicatifs analysés ont été générés par l'exécution du panel expérimental sur un total de 16 nœuds de calcul, pour un total de 512 cœurs physiques. `NEMO` et `OpenFOAM` ont été exécutées avec un total de 512 rangs `MPI`, chaque rang étant assigné à un cœur physique spécifique. Quant à l'application `HPCG`, elle a été exécutée selon un modèle hybride `MPI+OpenMP`, avec un rang `MPI` par processeur, et 16 threads `OpenMP` par rang `MPI`, tous affins de cœurs physiques spécifiques. Pour être complets, rappelons qu'à cet ensemble de 16 nœuds de calcul était adjoint un système de fichiers `Lustre` dédié, exposé via le réseau d'interconnexion rapide.

Les profils applicatifs analysés présentent l'évolution temporelle de l'`IPC`. D'autres métriques ont été monitorées et analysées, par exemple le nombre de défauts en cache `L3` par cycle de référence du processeur, et ont conduit à des résultats similaires. Lesdits profils applicatifs ont été obtenus via le monitoring de compteurs de performance, à l'échelle du nœud de calcul complet, mis en place par `BDPO`, avec une période d'échantillonnage de 5 ms. Pour ce faire, une instance de `BDPO` a été exécutée sur chaque nœud, et il y a donc un profil applicatif par nœud.

Insistons sur le fait que **l'ensemble des analyses** ont été faites avec **la même** configuration de `Phase-TA`, à savoir sa configuration par défaut, dimensionnée pour la famille des séries temporelles que constituent les profils applicatifs. Enfin, précisons que toutes les expériences ont été reproduites au moins 3 fois, et ont conduit, sur l'ensemble des nœuds, à des profils applicatifs et des résultats similaires.

Les différentes expérimentations présentées par ce chapitre ont des visées différentes. Pour commencer, la section 8.1 cherche à évaluer la prépondérance des comportements localement périodiques dans les profils applicatifs `HPC`, et leurs relations avec les différentes phases des applications associées. La section 8.2 a quant à elle pour objet l'évaluation de la pertinence des motifs représentatifs construits par `Phase-TA` en tant que substituts des périodicités associées. Dans la même veine, la section 8.3 s'intéresse à l'influence du nombre d'instances périodiques détectées sur la pertinence desdits motifs représentatifs. Une évaluation empirique des performances de `Phase-TA`, accompagnée d'une étude théorique de sa complexité algorithmique, constituent le cœur de la section 8.4. Pour finir, la section 8.5 propose un bilan des expérimentations, et présente brièvement les cas d'utilisation que ces dernières permettent d'envisager pour `Phase-TA`.

8.1 Évaluation de la prépondérance des comportements périodiques dans les profils d'applications `HPC`

Les premières expérimentations qui ont été menées avec `Phase-TA` ont eu pour but de démontrer que les applications de calcul à haute performance itératives exhibent des comportements localement périodiques. Cela a également été l'occasion de quantifier la prépondérance desdits comportements localement périodiques à l'échelle des temps d'exécution complets des applications concernées, tout en validant le fait que `Phase-TA` permet de détecter les périodicités associées.

Pour ce faire, les profils applicatifs associés aux trois applications du panel expérimental ont été étudiés manuellement, codes sources à l'appui, puis analysés par `Phase-TA`, de sorte à confirmer que les résultats présentés par ce dernier étaient en accord avec les résultats attendus.

Les résultats obtenus suite à l'analyse des profils applicatifs par `Phase-TA` sont d'ailleurs résumés dans le tableau 8.1, où T_{exec} désigne le temps d'exécution moyen pour les trois exécutions en secondes, $\#per$ désigne le nombre de périodicités détectées, et $\#inst$ désigne le nombre d'instances périodiques détectées, ces dernières étant réparties entre $\#rp$ régions périodiques, et représentant une proportion $\%period$ du profil applicatif complet.

TABLE 8.1 – Résumé (valeurs moyennes pour 3 exécutions) des analyses par `Phase-TA` des profils applicatifs du panel expérimental.
T_{exec} représente le temps d'exécution moyen en secondes, et $\#per$ représente le nombre de périodicités détectées. Ces dernières ont engendré $\#rp$ régions périodiques, dont $\#inst$ instances périodiques ont été extraites. Lesdites instances périodiques couvrent $\%period$ pourcents des profils applicatifs dont elles sont issues.

	T_{exec}	$\#per$	$\#inst$	$\#rp$	$\%period$
`HPCG`	$1129.59s$	3	521	43	89.20%
`NEMO`	$659.696s$	1	451	27	76.73%
`OpenFOAM`	$659.391s$	1	169	44	67.48%

La première observation qu'il convient de faire est que la totalité des applications exhibent des périodicités, détectées par `Phase-TA`, et qui sont associées à une proportion significative du temps d'exécution global : en moyenne 77.80%, et jusqu'à presque 90% pour `HPCG`. On peut également observer que les instances périodiques sont réparties en plusieurs régions périodiques, avec en moyenne 10 instances périodiques par région périodique. Cela tend à confirmer que les profils applicatifs `HPC` ne sont seulement que localement périodiques, et que les portions engendrées par des périodicités sont entrecoupées de portions apériodiques.

Confrontons maintenant les résultats proposés par `Phase-TA` aux observations réalisées lors de l'étude des trois applications du panel expérimental. Pour commencer, concentrons-nous sur `NEMO`, dont l'exécution commence par une brève phase d'initialisation apériodique, responsable d'un peu plus de 5% du temps d'exécution. Pendant cette phase d'initialisation, la configuration est lue et les données d'entrée sont partitionnées. Le reste du temps d'exécution de l'application est presque entièrement dédié à l'exécution d'un unique noyau de calcul, dont les itérations sont entrecoupées de phases d'écriture sur disque (i.e. des « checkpoints »). On peut grossièrement estimer que ces phases d'écriture sur disque sont responsables d'environ 10% du temps d'exécution total. Ainsi, l'analyse de `Phase-TA` est cohérente avec les observations effectuées pour `NEMO` : une unique périodicité est détectée, et 76.73% du temps d'exécution lui est attribué, lorsqu'un unique noyau de calcul est exécuté pendant environ 85% du temps d'exécution. Ajoutons, à titre indicatif, que le motif construit par `Phase-TA` pour représenter la périodicité détectée est présenté par les figures 8.2 et 8.4.

En ce qui concerne `HPCG`, `Phase-TA` détecte trois périodicités distinctes, couvrant un total de 89.20% du temps d'exécution global. En analysant le fonctionnement de `HPCG`, on s'aperçoit que l'application génère procéduralement le jeu de données d'entrée sur lequel elle applique par la suite un unique noyau de calcul construit autour de la méthode du gradient conjugué. Par la suite, en observant les profils applicatifs associés à `HPCG`, on s'aperçoit que :

— La phase de génération des données d'entrée occupe environ 40% du temps d'exécution ;

— La phase de génération des données d'entrée exhibe deux régions périodiques consécutives et ininterrompues, impliquant deux périodicités distinctes bien que de lon-

gueurs et d'allures similaires ;

— La phase de calcul suit la phase de génération des données d'entrée, et occupe les 60% restants du temps d'exécution. Une seule périodicité semble associée à cette phase de calcul.

Parmi les trois périodicités détectées par `Phase-TA`, il apparaît clairement que les deux premières sont associées avec la phase de génération des données d'entrée, et 31.8% du temps d'exécution leurs sont attribués. La troisième périodicité est quant à elle associée à la phase de calcul, et 57.4% du temps d'exécution lui sont attribués. Ainsi, comme pour `NEMO`, les résultats de l'analyse par `Phase-TA` sont cohérents avec les observations réalisées. Pour finir, précisons que l'exemple illustrant pas à pas la méthodologie implémentée par `Phase-TA` dans la section 7.7 est issu de l'analyse d'un des profils applicatifs associés à `HPCG` dans le cadre des expérimentations présentées par cette section. La figure 7.20 présente notamment les motifs représentatifs construits par `Phase-TA`.

Concernant `OpenFOAM`, `Phase-TA` ne détecte qu'une périodicité, à laquelle 67.48% du temps d'exécution total sont attribués. L'analyse détaillée de l'exécution révèle qu'après une très brève (d'une durée légèrement supérieure à 1% du temps d'exécution total) phase d'initialisation apériodique, pendant laquelle `OpenFOAM` charge le jeu de données d'entrée prétraité, un unique noyau de calcul est exécuté jusqu'à la fin de l'exécution de l'application. Les calculs sont entrecoupés de phases d'écriture sur disque apériodiques relativement courtes, puisqu'elles représentent, au cumulé, environ de 2% du temps d'exécution total de l'application (le nombre de ces écritures sur disque dépend de la configuration de `icoFOAM`). De plus, les phases de calcul se révèlent particulièrement intensives, avec plus de 2 instructions retirées par cœur physique par cycle de référence, et les profils applicatifs exhibent une forte variabilité durant ces dernières. Cette variabilité s'explique vraisemblablement par le fait que les cœurs de calcul sont surcadencés via `TurboBoost` dès que l'amplitude thermique nécessaire est disponible. Et le fait que l'utilisation des unités de vectorisation `AVX512` ajoute une dose de complexité à la gestion thermique des processeurs renforce fort probablement le caractère variable de l'activation de `TurboBoost`.

L'analyse des profils applicatifs par `Phase-TA` met en lumière cette variabilité, notamment lorsque l'on met en relation les nombres de régions périodiques détectées et d'instances périodiques qui en ont été extraites : en moyenne, une région périodique contient 4 à 5 instances périodiques. Autrement dit, les régions périodiques détectées sont entrecoupées d'occurrences de la périodicité suffisamment bruitées pour que `Phase-TA` les considère comme des cassures dans les comportements localement périodiques. En conséquence, seulement un peu plus de deux tiers des occurrences de la périodicité sont détectées par l'analyse des profils applicatifs de `OpenFOAM`. Néanmoins, il semble qu'une quantité suffisante d'instances périodiques soient extraites pour que les motifs construits par `Phase-TA` soient, visuellement, représentatifs de la périodicité en question, comme la figure 8.1 le laisse entrevoir [1].

Ainsi, malgré la part significative des occurrences de la périodicité qui ne sont pas détectées par `Phase-TA`, les résultats obtenus grâce à l'analyse des profils applicatifs associés à `OpenFOAM` demeurent néanmoins cohérents avec les observations réalisées.

1. La question de la pertinence des motifs représentatifs est abordée en détail par les sections 8.2 et 8.3.

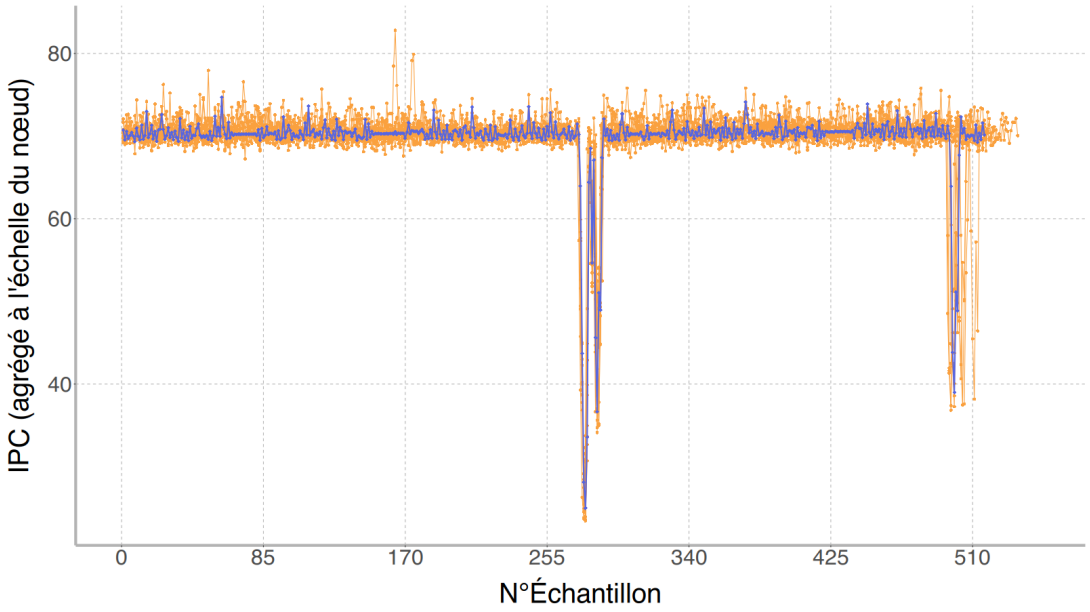


FIGURE 8.1 – Motif représentatif (en violet) de la périodicité associée au noyau de calcul principal de `icoFOAM`, superposé à un ensemble quelconque de 10 des instances périodiques (en orange) à partir desquelles il a été inféré.

Pour finir, il convient de s'intéresser aux portions périodiques des profils des trois applications du panel expérimental qui ne sont pas couvertes par des instances périodiques détectées par `Phase-TA`. Tout d'abord, remarquons que lesdites portions ne représentent qu'environ 10% des parties exhibant des comportements localement périodiques des profils applicatifs associés à `HPCG` et `NEMO`, et un peu moins de 30% de ceux associés à `OpenFOAM`.

Pour ce qui est de l'explication sous-jacente, elle est plutôt simple et recoupe ce qui a été dit pour `OpenFOAM` : certaines occurrences des périodicités sont suffisamment bruitées pour que `Phase-TA` les considère comme des portions apériodiques car trop différentes des occurrences les entourant. En effet, même si la méthodologie associée à `Phase-TA` permet de s'abstraire d'une quantité de bruit significative (aussi bien d'insertion et de suppression, que de modification), cette quantité demeure bornée et fixée par la configuration de l'outil. Et en adaptant finement ladite configuration, il est possible de faire en sorte que `Phase-TA` détecte la majorité de ces occurrences manquantes. Par exemple, avec des configurations spécifiques, il a été possible de détecter l'intégralité des instances périodiques exhibées par un profil applicatif de `OpenFOAM`, et plus de 97% de celles exhibées par un profil applicatif de `NEMO`. Néanmoins, comme précisé auparavant, cela nécessite une expertise que l'utilisateur lambda n'aura vraisemblablement pas. Occasion parfaite pour rappeler que la configuration par défaut de `Phase-TA` a été construite de sorte à couvrir une large gamme de profils applicatifs `HPC`, quitte à échouer à détecter une proportion acceptable d'occurrences d'une périodicité.

En conclusion, pour l'ensemble des trois applications du panel expérimental, l'analyse des profils applicatifs par `Phase-TA` est **cohérente** avec les observations résultantes de l'étude manuelle desdits profils, et des codes sources des applications concernées. Ainsi, Phase-TA est un outil capable de **détecter les comportements localement périodiques** des applications `HPC`, qui **dominent** largement les temps d'exécution de ces dernières.

8.2 Évaluation de la pertinence des motifs représentatifs inférés par `Phase-TA`

Juger de la pertinence des motifs construits par `Phase-TA` n'est pas chose aisée, pour plusieurs raisons. Premièrement, comme précisé dans la sous-section 7.1.2, les périodicités demeurent inobservables directement : seules leurs occurrences permettent une observation indirecte. Ensuite, se pose la question de la référence par rapport à laquelle évaluer la pertinence des motifs représentatifs. Deux approches ont été envisagées dans le cadre de ces travaux de recherche : par rapport aux instances périodiques, et par rapport à d'autres profils en substituant les instances périodiques par le motif représentatif. Enfin, on peut également se poser la question de la « portée » de la pertinence : lors de l'exécution d'une application sur plusieurs nœuds de calcul, est-ce que le motif représentatif construit à partir du profil applicatif généré sur un des nœuds en question est pertinent vis-à-vis du profil applicatif généré sur un autre de ces nœuds ? Et qu'en est-il lorsque l'on considère des profils associés à des exécutions différentes de la même application ?

Pertinence vis-à-vis des instances périodiques Pour commencer, il est possible de juger qualitativement, i.e. visuellement, de la pertinence des motifs représentatifs inférés par `Phase-TA`, relativement aux périodicités associées.

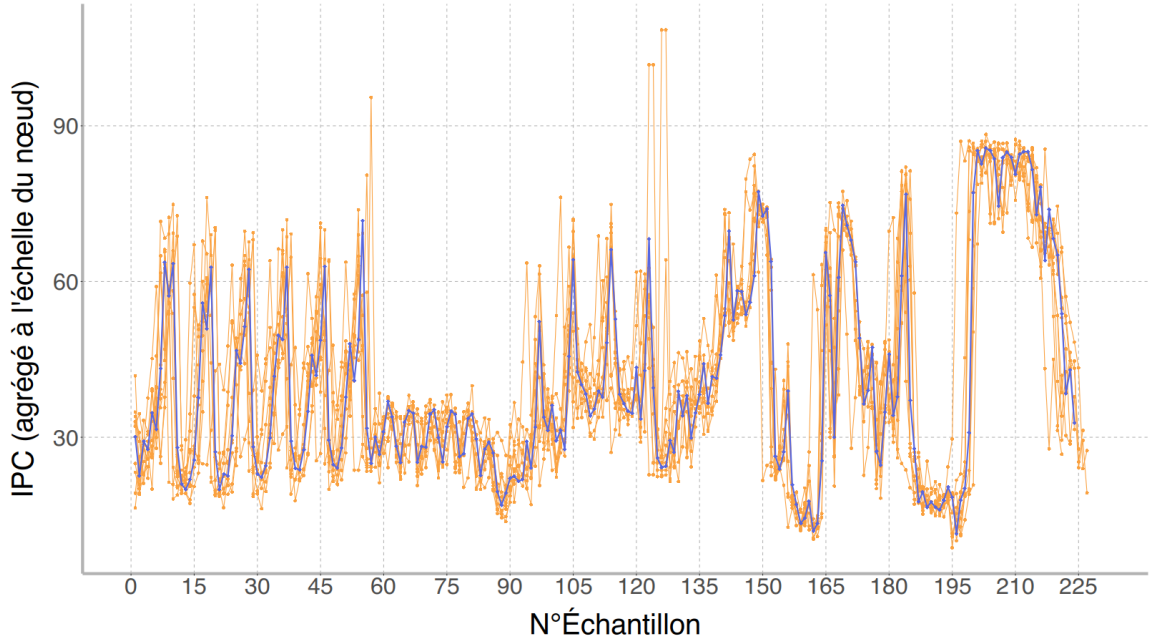


FIGURE 8.2 – Motif représentatif (en violet) de la périodicité associée au noyau de calcul principal de `NEMO`, superposé à un ensemble quelconque de 10 des instances périodiques (en orange) à partir desquelles il a été inféré.

Par exemple, la figure 8.2 montre le motif représentatif (en violet) inféré par `Phase-TA` pour un profil applicatif de `NEMO`, superposé à une sélection quelconque de 10 des instances périodiques à partir desquelles il a été construit. De la même manière, les figures 8.1 et 7.20 présentent les motifs représentatifs inférés pour `OpenFOAM` et `HPCG`, respectivement. Il

apparaît clairement que les motifs représentatifs sont des substituts vraisemblables pour les instances périodiques, qui constituent le seul moyen, bien qu'indirect, d'observer les périodicités exhibées par les profils applicatifs.

Et cette observation qualitative est vérifiée du point de vue quantitatif. Pour le démontrer, commençons par quelques définitions et notations. Soit p un profil applicatif, soit $\mathcal{I}_p$ l'ensemble des instances périodiques détectées par `Phase-TA` lors de l'analyse de p, soit l_p la longueur moyenne des instances périodiques appartenant à $\mathcal{I}_p$, soit m_p la valeur moyenne des échantillons des instances périodiques appartenant à $\mathcal{I}_p$, et soit $\mathcal{M}_p$ le motif représentatif inféré par `Phase-TA` à partir de $\mathcal{I}_p$. On peut alors définir Δ_p, qui représente approximativement la distance moyenne entre un point d'un motif représentatif, et un point d'une instance périodique avec lequel il est aligné :

$$\Delta_p = \frac{1}{card\left(\mathcal{I}_p\right) \cdot l_p} \cdot \sum_{i \in \mathcal{I}_p} DTW\left(\mathcal{M}_p, i\right) \tag{8.1}$$

Numériquement, pour l'ensemble des profils p analysés dans le cadre de ce chapitre [2], $\Delta_p \leq 0.04 \cdot m_p$. En d'autres termes, la distance moyenne entre un point d'un motif représentatif et un point d'une instance périodique avec lequel il est aligné est inférieur ou égale à 4% de la valeur moyenne d'un point d'une de ces instances périodiques. Cela vient confirmer quantitativement que les motifs représentatifs inférés par `Phase-TA` sont pertinents vis-à-vis des périodicités associées.

Pertinence par substitution des instances périodiques Le but premier d'un motif représentatif construit par `Phase-TA` est d'être un substitut à la périodicité à laquelle il est associé, et donc, par extension, aux occurrences de cette dernière. Afin de quantifier la pertinence d'un motif représentatif en tant que substitut, pour chaque profil applicatif considéré, un profil applicatif « substitué » a été généré. Il s'agit du profil applicatif « original » pour lequel toutes les instances périodiques détectées par `Phase-TA` ont été substituées par les motifs représentatifs associés. Par la suite, `DTW` est calculé entre chaque profil substitué et le profil original associé, dans le but de quantifier la pertinence des motifs représentatifs en tant que substituts pour les instances périodiques. La « pertinence » ici quantifiée est phénomènologiquement différente de celle quantifiée par le paragraphe précédent, qui met directement en relation un motif représentatif et les instances périodiques à partir desquelles il a été construit. En effet, le paragraphe précédent évalue la pertinence du motif représentatif en tant que motif moyen des instances périodiques, alors que la pertinence ici évaluée représente la capacité du motif périodique à se substituer à une instance périodique quelconque, à s'intégrer au profil applicatif.

Parallèlement, dans le but de construire une échelle de comparaison pour les valeurs de `DTW` entre profils substitués et originaux, deux autres profils applicatifs ont été considérés. Premièrement, `DTW` a également été calculé entre le profil original et un autre profil applicatif pour une autre exécution de l'application considérée dans les mêmes conditions expérimentales. Cet autre profil applicatif est appelé « autre » profil par la suite. Deuxièmement, un total de cinq profils aux valeurs aléatoires ont été générés : ils ont le même nombre d'échantillons que le profil original, et ces derniers sont générés via une loi

2. À titre indicatif, le panel expérimental compte 3 applications, exécutées au moins 3 fois, sur 16 nœuds. Soit un total minimal de 144 profils applicatifs.

normale dont les paramètres sont la valeur moyenne et l'écart relatif des échantillons du profil original. DTW a ensuite été calculé entre le profil original et ces 5 profils, par la suite désignés comme profil « aléatoires », et la valeur moyenne de ces 5 calculs est ensuite retenue.

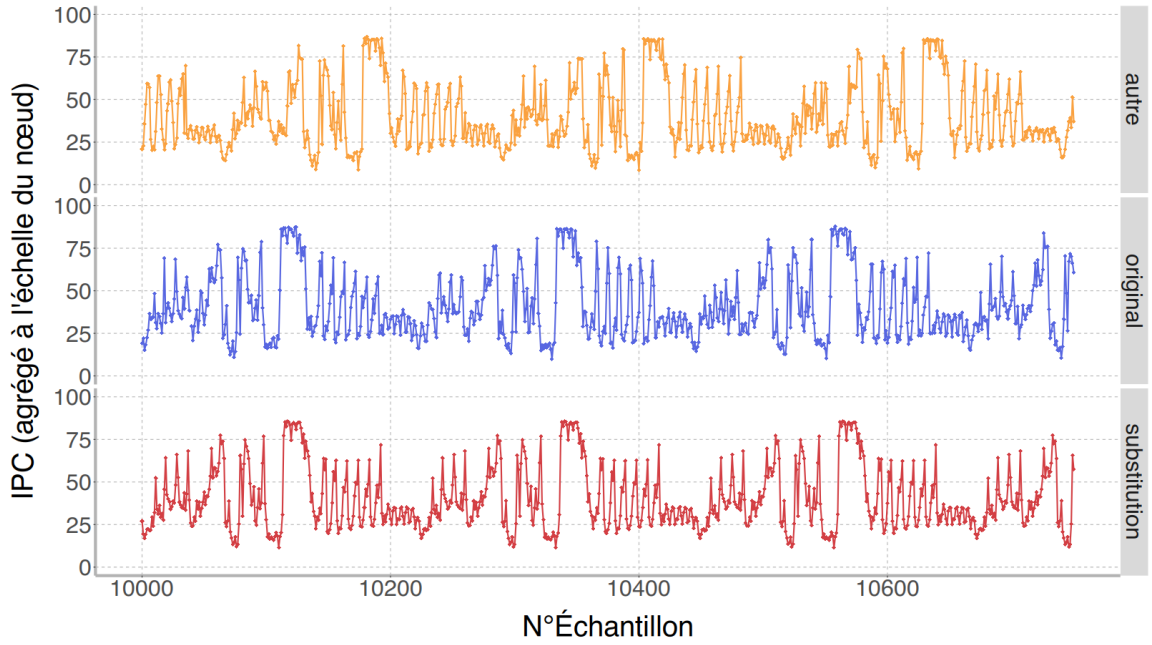


FIGURE 8.3 – Comparaison d'un profil applicatif « original » associé à NEMO (en violet) à : (1) un « autre » profil applicatif associé à une exécution de NEMO dans les mêmes conditions expérimentales (en orange), et (2) une version « substituée » (en rouge) qui correspond au profil « original » pour lequel toutes les instances périodiques détectées par Phase-TA ont été remplacées par les motifs périodiques associés.
Ainsi, le profil « substitué » exhibe trois occurrences du motif représentatif construit par Phase-TA suite à l'analyse du profil applicatif « original », qui se substituent à trois instances périodiques.

Le tableau 8.2 rassemble les résultats obtenus, en les moyennant sur l'ensemble des profils applicatifs originaux considérés, et en adjoignant à titre de contextualisation les nombres moyens d'échantillons et leurs écarts relatifs pour lesdits profils originaux. À noter qu'afin d'illustrer cette section, la figure 8.3 présente un extrait d'un profil applicatif « original » de NEMO, ainsi que les extraits associés des profils « autre » et « substitué ».

Les valeurs de DTW sont normalisées, ce qui veut dire qu'elles représentent grossièrement la distance moyenne qu'il y a entre un point d'un profil applicatif original, et un point associé du profil considéré (« autre », « substitué », ou « aléatoire »). On peut donc observer, notamment lorsque l'on considère les valeurs moyennes des échantillons, que les points des profils substitués sont très similaires à leurs homologues dans les profils originaux. De plus, d'après DTW, les profils substitués sont significativement plus similaires aux profils originaux que des profils issus d'autres exécutions de l'application concernée avec les mêmes paramètres expérimentaux. Conjointement, ces deux observations laissent donc penser que les motifs représentatifs construits par Phase-TA sont des **substituts per-**

tinents des périocités exhibées par les profils applicatifs, et donc de leurs occurrences.

Pour être complet, on peut remarquer que les valeurs de `DTW` entre les profils substitués et les profils originaux associés sont d'autant plus grandes que la variabilité relative des échantillons des profils applicatifs originaux est importante (cette dernière étant mesurée par le ratio entre l'écart relatif et la moyenne). Cela n'est pas étonnant, puisque cette variabilité signifie notamment que les bruits de modification sont plus intenses, comme c'est le cas pour `NEMO`. Et si `DTW` permet de compenser efficacement les bruits de suppression et d'insertion, sa tolérance aux bruits de modification demeure limitée. En conséquence, les valeurs de `DTW` pour les profils applicatifs exhibant une variablité intrinsèque importante ont tendance à être plus élevées.

TABLE 8.2 – Valeurs normalisées de `DTW`, par rapport aux profils applicatifs « originaux », des différents types de profils. En particulier, « aléatoire » présente la valeur moyenne pour les 5 profils générés aléatoirement.
$\overline{samples}$ représente les valeurs moyennes des échantillons des profils applicatifs « originaux », et σ les écarts relatifs associés, à titre de contextualisation.

	substitué	autre	aléatoire	$\overline{ipc}$	σ
`HPCG`	0.106	0.201	2.37	11.5	6.56
`NEMO`	0.963	1.86	7.41	38.8	22.3
`OpenFOAM`	0.081	0.495	2.72	70.2	7.11

Pertinence inter-nœud pour une même exécution Étant donné qu'un profil applicatif est généré par nœud de calcul, les motifs représentatifs inférés sont également spécifiques à un nœud. Il est alors légitime de s'interroger quant à la pertinence d'un motif représentatif vis-à-vis du profil applicatif associé à un autre des nœuds impliqués dans l'exécution d'un job `HPC`. La figure 8.4 présente les motifs représentatifs inférés pour l'ensemble des 16 profils applicatifs issus d'une exécution de `NEMO` sur les 16 nœuds de calcul de `dahu`. On s'aperçoit que les 16 motifs représentatifs sont très fortement similaires. Et cela se confirme numériquement. Notons $\mathcal{S}$ l'ensemble des 16 motifs représentatifs. Soient a et b deux motifs représentatifs quelconques appartenant à $\mathcal{S}$, dont les longueurs sont respectivement notées l_a et l_b. Soit m la valeur moyenne des échantillons de l'ensemble des motifs représentatifs appartenant à $\mathcal{S}$. Alors :

$$\frac{1}{max\,(l_a, l_b)} \cdot DTW(a, b) \leq 0.01 \cdot m \tag{8.2}$$

Autrement dit, la distance moyenne entre deux points alignés de deux motifs représentatifs quelconques parmi les 16 motifs représentatifs considérés est inférieure à 1% de la valeur moyenne d'un échantillon appartenant à un des 16 motifs représentatifs appartenant à $\mathcal{S}$. En conclusion, un motif représentatif inféré pour un profil applicatif est pertinent vis-à-vis des périodicités exhibées par les profils associés aux autres nœuds impliqués dans l'exécution parallèle d'une application de calcul à haute performance.

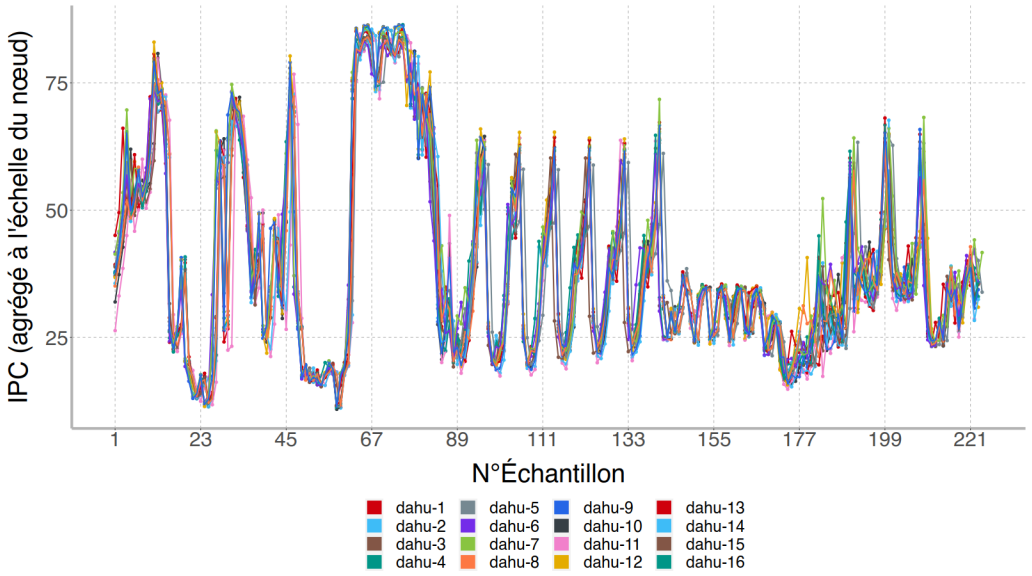


FIGURE 8.4 – Motifs représentatifs inférés pour les profils applicatifs associés aux 16 nœuds de calcul, pour une **même** exécution de `NEMO`.

Pertinence vis-à-vis d'un autre profil associé à une autre exécution La question de la reproductibilité mérite également d'être abordée : est-ce que des profils applicatifs générés par des exécutions différentes mais dans des conditions expérimentales (i.e. nœuds de calcul, données d'entrée, supports d'exécution, et configurations) identiques d'une même application exhibent des périodicités similaires ?

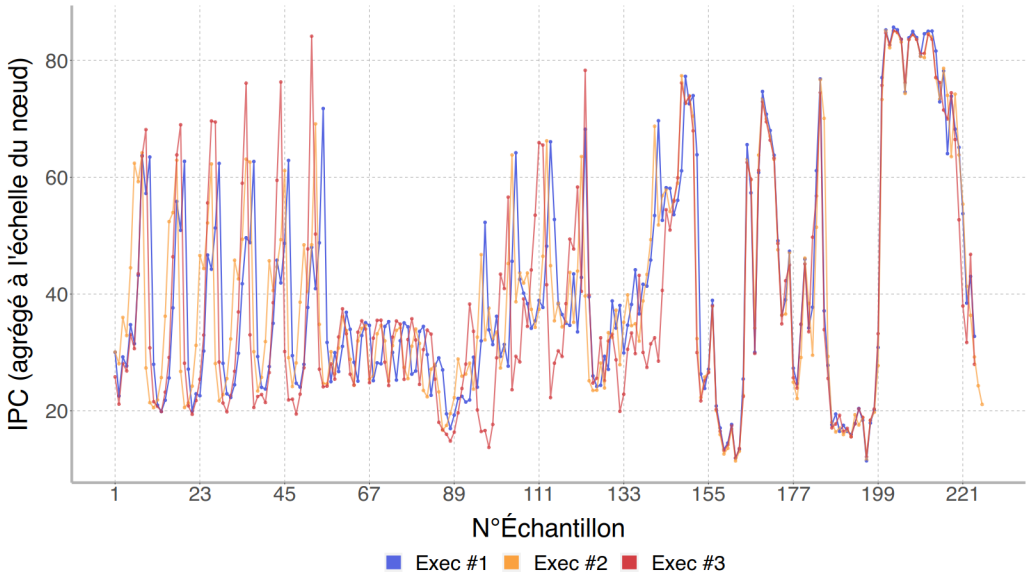


FIGURE 8.5 – Motifs représentatifs inférés pour les profils applicatifs associés à trois exécutions distinctes de `NEMO` avec les mêmes configurations expérimentales, sur un nœud de `dahu`.

Afin d'apporter une réponse, les motifs représentatifs construits par `Phase-TA` pour

chacune des trois exécutions des trois applications du panel expérimental ont été comparés, par application et par nœud. Pour une application donnée, une fois les comparaisons effectuées par nœud, les résultats de ces dernières ont été moyennées sur l'ensemble des nœuds.

Le tableau 8.3 regroupe lesdits résultats : (1) #1 et #2 représentent les identifiants des deux exécutions pour lesquelles les motifs représentatifs ont été comparés, (2) la valeur normalisée de `DTW`, moyennée sur l'ensemble des nœuds, entre ces deux motifs représentatifs est donnée par $\overline{DTW_{paire}}$, (3) $\overline{DTW_{total}}$ moyenne les valeurs par « paire d'exécution », et (4) $\overline{ipc}$ et σ sont les valeurs moyennes et écarts relatifs des échantillons d'`IPC` appartenant à l'ensemble des 48 profils applicatifs associés à chaque application.

TABLE 8.3 – Résultats de la comparaison des motifs représentatifs construits pour des profils applicatifs générés par des exécutions différentes d'une même application dans des conditions expérimentales identiques.

	#1	#2	$\overline{DTW_{paire}}$	$\overline{DTW_{total}}$	$\overline{ipc}$	σ
`HPCG`	1	2	0.106	0.105	11.5	6.56
	1	3	0.100			
	2	3	0.109			
`NEMO`	1	2	0.859	1.38	38.8	22.3
	1	3	1.66			
	2	3	1.61			
`OpenFOAM`	1	2	0.325	0.296	70.2	7.11
	1	3	0.267			
	2	3	0.297			

Pour commencer, on peut remarquer que la distance moyenne entre deux points associés d'après `DTW` de deux motifs représentatifs représente moins de 1% des valeurs moyennes des échantillons des profils applicatifs associés pour `OpenFOAM` et `HPCG`, et environ 3.5% pour `NEMO`. De plus, les valeurs de $\overline{DTW_{paire}}$ sont très homogènes pour une application donnée. À y regarder de plus près, seul le motif représentatif associé à la troisième exécution de `NEMO` semble sortir du rang, comme la figure 8.5 le montre : il est clairement visible qu'entre les échantillons 85 et 145, le motif représentatif considéré exhibe un comportement différent de ceux associés aux deux premières exécutions. La cause de cette « aberration » est vraisemblablement que les conditions expérimentales n'étaient pas *réellement* identiques. En effet, `dahu` est un cluster dédié à la recherche, partagé par de nombreux chercheurs. Il est donc fréquent que des expérimentations différentes s'exécutent en même temps sur différents nœuds du même cluster. Le taux d'utilisation des ressources partagées, telles que le réseaux d'interconnexion rapide, résulte alors d'une superposition des expérimentations. Ainsi, il est vraisemblable qu'une charge du réseau d'interconnexion rapide par une expérimentation tierce ait pu impacter les phases de communication de `NEMO` lors de sa troisième exécution. Ce qui est également cohérent avec le fait que les autres portions du motif représentatif n'aient pas été perturbées.

Malgré le comportement « partiellement différent » pour l'une des exécutions de `NEMO`,

les résultats obtenus permettent de conclure assez clairement que les motifs représentatifs construits par `Phase-TA` sont **pertinents vis-à-vis d'autres exécutions** des applications, pour des conditions expérimentales identiques.

Ce qu'il faut retenir Les expérimentations présentées dans cette section ont permis de démontrer, aussi bien qualitativement que quantitativement, que les motifs représentatifs construits par `Phase-TA` sont **pertinents vis-à-vis des périodicités associées**, et qu'ils constituent donc des substituts adéquats pour ces dernières et leurs occurrences. Pour finir, soulignons à nouveau deux résultats importantes :

— Lorsqu'une application `HPC` est exécutée sur plusieurs nœuds de calcul, les motifs représentatifs inférés pour le profil applicatif associé à l'un des nœuds sont également pertinents pour les profils applicatifs générés sur les autres nœuds de calcul ;

— Lorsqu'une application `HPC` est exécutée plusieurs fois avec les mêmes conditions expérimentales (i.e. mêmes paramètres, jeux de données similaires, etc.), les motifs représentatifs inférés pour une des exécutions en question sont également pertinents pour les profils applicatifs associés aux autres exécutions.

La motivation principale sous-jacente au développement de `Phase-TA` étant de fournir à `BDPO` des capacités de prédiction, ces résultats ont deux implications majeures : (1) l'analyse d'un seul des profils applicatifs générés par les multiples instances de `BDPO` sera nécessaire pour mettre en place des reconfigurations prédictives sur l'ensemble des nœuds de calcul impliqués dans l'exécution du job considéré, et (2) l'analyse de profils applicatifs à froid permettra de doter `BDPO` de capacités de prédiction pour les exécutions à venir de l'application considérée, tant qu'elles sont effectuées dans les mêmes conditions expérimentales.

8.3 Impact du nombre d'instances périodiques détectées sur la pertinence des motifs représentatifs

La section 8.2 a montré que les motifs représentatifs inférés par `Phase-TA` sont pertinents vis-à-vis des périodicités auxquelles ils sont associés. Néanmoins, il est légitime de se demander quel est l'impact de la durée de la série temporelle analysée sur la pertinence des motifs représentatifs construits par `Phase-TA`. En effet, pour que les motifs représentatifs soient pertinents vis-à-vis des périodicités associées, il faut qu'ils soient construits à partir d'un nombre suffisant d'occurrences des périodicités, c'est-à-dire d'instances périodiques. Or, le nombre d'instances périodiques croît (et décroît) avec la longueur de la série temporelle analysée, et donc le nombre d'itérations des noyaux de calcul exécutées dans le cas des applications `HPC`. Pour répondre à cette question, un protocole expérimental spécifique a été mis en place et suivi :

— Des profils applicatifs « longs », d'une durée d'une heure, ont été générés pour les trois applications du panel expérimental. Pour ce faire, les mêmes jeux de données d'entrée que pour les expérimentations présentées dans le cadre de ce chapitre ont été utilisés, mais le nombre de pas de temps simulés (i.e. le nombre d'itérations) a été augmenté ;

— Des sous-profils ont été extraits des profils applicatifs longs. 7 durées ont été retenues pour ces sous-profils : 1 minute, 3 minutes, 5 minutes, 10 minutes, 15 minutes, 30 minutes, et 45 minutes ;

— Pour chaque durée, 11 sous-profils différents ont été extraits (certains se recouvrent donc partiellement) ;

— Il a été vérifié que l'ensemble des sous-profils extraits étaient contenus dans les parties des profils applicatifs longs associées à l'exécution des noyaux de calcul (ce qui exclut donc les phases d'initialisation des données, par exemple).

Quelques notations avant de poursuivre : pour chaque profil applicatif long $\mathcal{P}$, l'ensemble des instances périodiques qu'il exhibe est noté $\mathcal{I}_{\mathcal{P}}$. De plus, on note, pour $n \in \{1, 3, 5, 10, 15, 30, 45\}$ et $i \in [\![1, 11]\!]$, $\mathcal{P}_{n,i}$ le i-ème sous-profil extrait de $\mathcal{P}$ d'une durée de n minutes. Ensuite, le motif représentatif associé à un profil applicatif p est noté $\mathcal{M}_p$. Le `WGSS`[3] associé au motif représentatif $\mathcal{M}_{\mathcal{P}}$ inféré pour un profil applicatif long $\mathcal{P}$, relativement aux instances périodiques $\mathcal{I}_{\mathcal{P}}$ à partir desquelles il a été construit, est noté $WGSS_{\mathcal{P}} = WGSS\left(\mathcal{M}_{\mathcal{P}}, \mathcal{I}_{\mathcal{P}}\right)$. Dans la même veine, on note $WGSS_{\mathcal{P}_{n,i}} = WGSS\left(\mathcal{M}_{\mathcal{P}_{n,i}}, \mathcal{I}_{\mathcal{P}}\right)$ le `WGSS` associé au motif représentatif inféré pour le i-ème sous-profil d'une durée de n minutes extrait de $\mathcal{P}$, relativement aux instances périodiques $\mathcal{I}_{\mathcal{P}}$ détectées pour le profil applicatif long associé. Ainsi, en comparant $WGSS_{\mathcal{P}_{n,i}}$ et $WGSS_{\mathcal{P}}$, il est possible d'évaluer la pertinence du motif représentatif construit pour le i-ème sous-profil d'une durée de n minutes extrait de $\mathcal{P}$, par rapport au motif représentatif inféré pour le profil applicatif long complet, qui sert de référence.

Par la suite, l'ensemble des sous-profils, ainsi que les profils applicatifs longs ont été analysés avec `Phase-TA`, et les `WGSS` introduits ci-dessus ont été calculés. Les résultats sont compilés et présentés par la figure 8.6 :

3. La définition du `WGSS` est donnée par la sous-section 2.3.4.

— Chaque croix correspond à un WGSS pour une durée n et un sous-profil $\mathcal{P}_{n,i}$ donnés ;

— Les losanges correspondent aux WGSS moyens pour les 11 sous-profils associés à une durée n donnée (ce qui correspond mathématiquement à $1/11 \cdot \sum_{i \in [\![1;11]\!]} WGSS_{\mathcal{P}_{n,i}}$;

— À côté des losanges figurent les nombres moyens d'instances périodiques détectées pour une durée de sous-profil donnée ;

— Les lignes horizontales bleues correspondent aux WGSS associés aux motifs représentatifs inférés pour les profils applicatifs longs, qui ont été qualifiés de WGSS de référence, et notés $WGSS_{\mathcal{P}}$.

Pour commencer, on peut remarquer que même pour les sous-profils les plus courts (1 minute), la différence relative entre le WGSS moyen pour les sous-profils et le WGSS de référence est plutôt raisonnable puisqu'elle est de 37.0% en moyenne (et vaut jusqu'à 67.3% pour OpenFOAM). Essayons maintenant de définir un seuil sur le nombre d'instances périodiques au-dessus duquel les motifs représentatifs inférés par Phase-TA peuvent être considérés comme pertinents vis-à-vis des périodicités auxquelles ils sont associés. figure 8.6 :

En s'appuyant sur la figure 8.6, il semble qu'un seuil de 100 instances périodiques soit plus qu'envisageable. En effet, d'une part, lorsque l'on considère la durée pour laquelle le nombre d'instances périodiques détectées est le plus proche de 100, la différence relative entre le WGSS moyen pour les sous-profils et le WGSS de référence est de l'ordre de 12.1% (jusqu'à 24.4% pour NEMO), ce qui est communément considéré acceptable pour une « marge d'erreur » ou un « seuil de tolérance ». D'autre part, la durée nécessaire pour atteindre ce nombre d'instances périodiques, 6 minutes en moyenne et moins de 10 minutes pour l'ensemble des applications, est relativement faible, notamment lorsque comparée aux durées classiques des jobs HPC, qui peuvent atteindre plusieurs dizaines d'heures. Ainsi, on peut retenir l'heuristique suivante : pour qu'un motif représentatif construit par Phase-TA soit **pertinent** vis-à-vis de la périodicité associée, il faut qu'**il ait été inféré à partir d'au moins 100 instances périodiques**, seuil généralement atteint après 10 minutes de l'exécution d'une application HPC (pour une période d'échantillonnage de 5 ms lors de la génération du profil de l'application par BDPO).

Pour être complet, deux observations supplémentaires doivent être faites :

— Le WGSS associé au motif représentatif inféré pour certains sous-profils est inférieur au WGSS de référence ;

— Le WGSS moyen associé aux sous-profils d'une durée donnée ne décroît pas forcément avec l'augmentation de la durée.

Les explications sous-jacentes à ces deux observations trouvent leurs sources dans la définition même de l'algorithme DBA (cf. section 7.6 pour une présentation de DBA). En effet, bien qu'il y ait des garanties sur la convergence de l'algorithme, il n'y a aucune garantie concernant sa vitesse de convergence. En conséquence, en fonction du critère d'initialisation et du groupe d'instances périodiques à partir duquel le motif périodique est construit, la diminution du WGSS induit par une itération du processus de raffinement peut varier de manière erratique. De plus, le critère de terminaison de l'algorithme DBA implémenté dans Phase-TA est sensible à la vitesse de convergence de l'algorithme. Cela peut donc conduire à des nombres d'itérations du processus de raffinement différents entre les sous-profils, même s'ils ont la même durée et sont extraits du même profil applicatif

long. Ensemble, ces éléments expliquent ces deux observations supplémentaires.

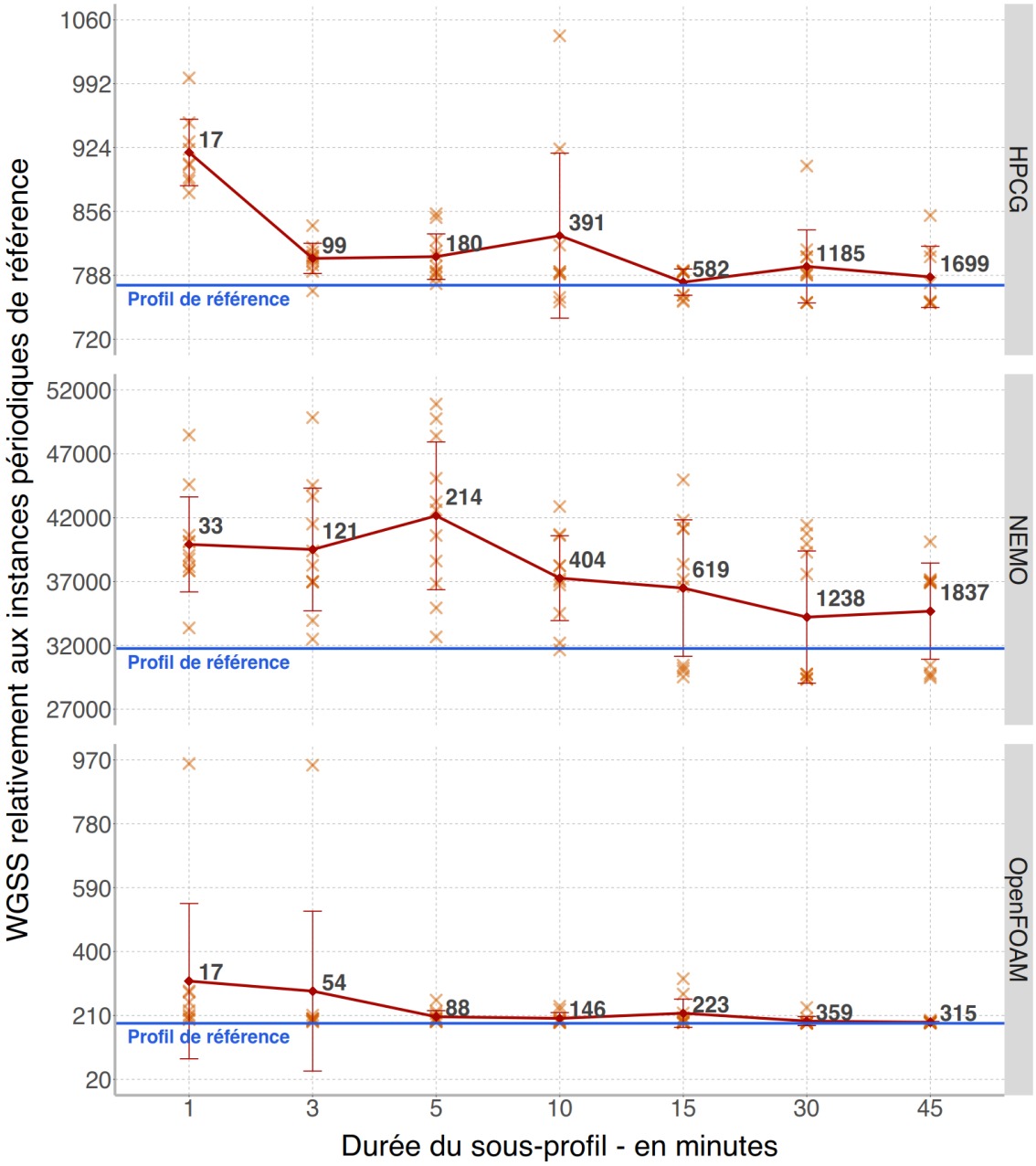


FIGURE 8.6 – Valeurs du `WGSS` associées aux motifs représentatifs produits par `Phase-TA` pour l'analyse de sous-profils, relativement à l'ensemble des instances périodiques détectées pour les profils applicatifs de référence associés.
Plus précisément, le graphe présente l'évolution du `WGSS` en fonction de la durée des sous-profils. Chaque croix représente le `WGSS` pour un sous-profil particulier, les `WGSS` moyens pour une durée de sous-profil donnée sont représentés par des losanges, et, accolés à ces derniers, on trouve les nombres moyens d'instances périodiques détectées associés. Les lignes horizontales bleues représentent les `WGSS` de référence.

8.4 Complexité et performances de `Phase-TA`

Cette section couple une étude théorique de la complexité algorithmique inhérente à la méthodologie implémentée par `Phase-TA`, présentée par la sous-section 8.4.1, à une évaluation empirique de ses performances, objet de la sous-section 8.4.2.

8.4.1 Analyse de la complexité de `Phase-TA`

Pour commencer, supposons qu'un profil applicatif contenant $\mathcal{L}$ échantillons soit analysé par `Phase-TA`, qui détecte et extrait N_{pi} instances périodiques de longueur moyenne L_{pi}. Ensuite, notons I_{AAVS} le nombre d'itérations [4] du processus d'auto-tunage de la longueur de la fenêtre glissante utilisée par la première étape de la méthodologie implémentée par `Phase-TA` (cf. section 7.4), et L_{SW} la longueur moyenne de ladite fenêtre glissante au cours de l'intégralité de l'exécution (i.e. auto-tunage et analyse à proprement parler). Notons également I_{DBA} le nombre maximum d'itérations du processus de raffinement implémenté par l'étape de construction des motifs représentatifs (cf. section 7.6). Continuons à définir les notations qui seront utilisées par la suite : soit C_{arith} la complexité d'une opération arithmétique généralisée (e.g. une addition, le calcul d'une valeur absolue, une multiplication, etc.), et soit C_{clust} la complexité d'une opération de clustering généralisée (typiquement constituée d'un ensemble fixe d'opérations arithmétiques généralisées et d'opérations liées à la gestion de structures de données). Enfin, notons C^{x}_{DTW} la complexité algorithmique associée au calcul de la `DTW` entre deux séries temporelles de longueur x, et $C^{x}_{d_1}$ celle associée au calcul de la distance de Manhattan entre ces deux mêmes séries temporelles. Alors, la complexité algorithmique associée à la méthodologie implémentée par `Phase-TA`, notée C_{PTA}, peut être modélisée de la manière suivante :

$$
\begin{aligned}
C_{PTA} = \mathcal{O}\left((I_{AAVS}+1)\cdot\left[\left(\mathcal{L}\cdot\frac{1}{L_{SW}}\right)\cdot\left(C_{d_1}^{L_{SW}}\cdot L_{SW}\right)\right]\right) \\
+ \quad \mathcal{O}\left(\left[\frac{N_{pi}\cdot(N_{pi}-1)}{2}\cdot C_{DTW}^{L_{pi}} + N_{pi}^{2}\cdot C_{clust}\right]\right) \\
+ \quad \mathcal{O}\left(\left[I_{DBA}\cdot\left(N_{pi}\cdot C_{DTW}^{L_{pi}} + L_{pi}\cdot C_{arith}\right)\right]\right)
\end{aligned}
$$

Le premier terme entre crochets est associé à la complexité résultant de l'étape de détection des régions périodiques, et du processus d'auto-tunage de L_{SW} qui lui est adjoint (cf. section 7.4). Son premier sous-terme représente le fait que la série temporelle est analysée une fenêtre glissante à la fois, et son second sous-terme représente la complexité associée au calcul des distances de Manhattan entre chaque position de la fenêtre glissante et ses versions décalées vers la droite. Le facteur multiplicatif qui précède ce premier terme rend compte du fait que l'analyse du profil applicatif implique I_{AAVS} parcours de ce dernier par la fenêtre glissante lors de l'étape d'auto-tunage de L_{SW}, puis un parcours effectif permettant de détecter et extraire les instances périodiques.

Le second terme entre crochets est associé à la complexité de l'étape de clustering des instances périodiques présentée par la section 7.5. Son premier sous-terme représente la complexité algorithmique liée au calcul de la matrice des distances, qui s'appuie sur

4. Plus précisément, la somme des itérations de chacune de ses deux étapes.

`DTW`. Quant à son deuxième sous-terme, il est associé à l'algorithme de clustering, à savoir `SLINK`, dont la complexité est formulée et démontrée dans [Sibson 1973]. À noter que les deux pré-traitements antécédant le clustering des instances périodiques, ainsi que l'application du critère de coupe une fois ledit clustering effectué, induisent des complexités algorithmiques largement négligeables devant ces deux sous-termes, et sont donc omis.

Le troisième terme entre crochets est associé à l'étape de construction des motifs représentatifs en exécutant `DBA` (cf. section 7.6), dont l'expression de la complexité algorithmique est donnée et démontrée par [Petitjean 2011] et [Petitjean 2014].

Sachant que $C_{d_1}^{L_{SW}} = \mathcal{O}\left(L_{SW} \cdot C_{arith}\right)$ (cela découle de sa définition, cf. sous-section 2.3.1), et que $C_{DTW}^{L_{pi}} = \mathcal{O}\left(L_{pi}^2 \cdot C_{arith}\right)$ (expression démontrée dans [Sakoe 1971]), la complexité algorithmique associée à la méthodologie implémentée par `Phase-TA` peut être réécrite comme suit :

$$\begin{aligned} C_{PTA} = \mathcal{O}([\mathcal{L} \cdot L_{SW} \cdot I_{AAVS} + N_{pi}^2 \cdot L_{pi}^2 + I_{DBA} \cdot N_{pi} \cdot L_{pi}^2 + \\ I_{DBA} \cdot L_{pi}] \cdot C_{arith} + N_{pi}^2 \cdot C_{clust}) \end{aligned} \tag{8.3}$$

De l'équation (8.3), il apparait que C_{PTA} dépend de trois paramètres de configuration, à savoir I_{AAVS}, L_{SW} et I_{DBA}, ainsi que de trois caractéristiques de la série temporelle analysée qui sont inter-dépendantes : $\mathcal{L}$, N_{pi}, et L_{pi}. De plus, on observe que la complexité algorithmique de `Phase-TA` est approximativement **globalement quadratique de la longueur de la série temporelle analysée**, étant donné que $\mathcal{L}$ est une borne supérieure de $N_{pi} \cdot L_{pi}$ qui, dans le cas idéal, tend vers $\mathcal{L}$.

Remarquons que, sachant que $\mathcal{L}$ est fixée, N_{pi} et L_{pi} sont liées puisque lorsque l'une croît, l'autre décroît proportionnellement, et inversement. L'évaluation des performances de `Phase-TA`, présentée par la sous-section 8.4.2, permettra notamment de déterminer lequel de ces deux paramètres s'avère être le facteur dominant de la complexité algorithmique pour des cas pratiques.

8.4.2 Étude des performances de `Phase-TA`

Afin d'évaluer les performances de `Phase-TA` de manière empirique, les temps d'exécution nécessaires à l'analyse de multiples profils applicatifs de durées variées ont été mesurés. Rappelons que l'étude de la complexité algorithmique de `Phase-TA` a mis en évidence que les grandeurs caractéristiques du profil applicatif analysé ayant la plus grande influence sur ladite complexité sont le nombre d'instances périodiques détectées et extraites, ainsi que leurs longueurs. Or, comme la section 8.1 l'a démontré, les trois applications du panel considéré dans le cadre de ces expérimentations exhibent des quantités différentes d'instances périodiques de longueurs variées : environ 220 échantillons pour `NEMO` contre plus de 520 échantillons pour `OpenFOAM`. Ainsi, cette évalutation empirique de `Phase-TA` permettra également d'étudier l'impact de ces deux grandeurs caractéristiques sur ses performances. Le protocole expérimental associé à ces mesures est le suivant :

— Les profils applicatifs considérés sont les sous-profils impliqués dans les expérimentations présentées par la section 8.3 : 77 sous-profils par application, pour un total de 231 sous-profils ;

— Chaque sous-profil a été analysé 5 fois par `Phase-TA`, et les temps d'exécution associés ont été mesurés ;

— Le temps d'exécution retenu pour chaque sous-profil est le temps d'exécution moyen pour ces 5 itérations ;

— Toutes les mesures ont été faites sur deux nœuds de calcul distincts, et ont conduit à des résultats similaires (la différence relative est inférieure à 1%).

Le tableau 8.4 présente les résultats obtenus, regroupés et moyennés en fonction de la durée des sous-profils (pour rappel, 11 sous-profils par durée et par application).

TABLE 8.4 – Temps d'exécution de `Phase-TA` (en secondes), en fonction de la durée du profil applicatif analysé (en minutes), pour `HPCG`, `OpenFOAM`, et `NEMO`.

	$1min$	$3min$	$5min$	$10min$	$15min$	$30min$	$45min$
`HPCG`	$0.44s$	$1.18s$	$2.67s$	$9.33s$	$19.60s$	$76.00s$	$168.84s$
`NEMO`	$0.45s$	$1.18s$	$2.67s$	$8.73s$	$18.38s$	$71.13s$	$162.76s$
`OpenFOAM`	$0.84s$	$1.40s$	$2.62s$	$6.95s$	$13.36s$	$46.98s$	$73.69s$

Tout d'abord, on peut observer que le temps d'exécution de `Phase-TA` croît de manière polynomiale avec la durée de la série temporelle analysée. Ou, plus précisément, avec le nombre d'instances périodiques détectées, qui a tendance à croître proportionnellement à la durée de la série temporelle, comme le montre la figure 8.6.

De plus, on peut observer que pour les sous-profils de courtes durées, lorsque le nombre d'instances périodiques détectées est faible, la longueur de ces dernières est le facteur dominant lorsqu'il s'agit des performances de `Phase-TA`. En effet, `OpenFOAM` exhibe moins d'instances périodiques, mais contenant plus d'échantillons, que `NEMO` et `HPCG`, et les temps d'analyse des sous-profils qui sont associés à `OpenFOAM` sont plus grands que pour ceux associés à `NEMO` et `HPCG`.

Au contraire, lorsque le nombre d'instances périodiques augmente, ce dernier devient clairement le facteur dominant concernant les performances de `Phase-TA`. En effet, pour des sous-profils d'une durée de 45 minutes, les temps d'analyse pour ceux associés à `HPCG` et `NEMO` sont respectivement 2.29 et 2.21 fois supérieurs aux temps d'analyse associés aux sous-profils de `OpenFOAM`. Ainsi, étant donné que la majorité des cas d'utilisation envisagés pour `Phase-TA` impliquent l'analyse de séries temporelles longues (plus précisément, contenant de grands nombres d'échantillons), on peut retenir que le facteur dominant lorsqu'il s'agit de ses performances est le nombre d'instances périodiques détectables exhibées par le profil applicatif analysé.

À la lecture des résultats présentés par le tableau 8.4, les performances de `Phase-TA` semblent plus que satisfaisantes, étant donné que l'analyse d'un profil de 45 minutes (qui compte 540000 échantillons) est conclue en moins de 3 minutes. Et lesdits résultats demeurent satisfaisants si l'on se concentre sur les profils dont la durée est de 10 minutes, durée remarquable d'après la section 8.3 car c'est à partir de cette dernière que l'on juge que les motifs représentatifs inférés par `Phase-TA` sont pertinents vis-à-vis des périodicités associées. En effet, les mesures effectuées montrent que le temps d'analyse d'une série temporelle d'une durée de 10 minutes (ce qui correspond à 120000 échantillons) est

inférieur à 10 secondes. Cela souligne que, du point de vue des performances, `Phase-TA` est **compatible avec l'analyse de données à chaud**, ce qui constitue un des objectifs de son cahier des charges initial.

Une dernière remarque pour conclure cette partie : un profil applicatif est propre à un nœud spécifique. En conséquence, la complexité algorithmique de `Phase-TA` est indépendante du nombre de nœuds exécutant l'application `HPC` considérée, et son temps d'analyse croît linéairement avec le nombre de nœuds : $\forall n \in \mathbb{N}$, $T_{exec}\,(n \text{ nœuds}) = n \times T_{exec}\,(1 \text{ nœud})$.

Cette remarque peut paraître triviale puisque `Phase-TA` a été conçu de sorte à analyser des séries temporelles, dont les profils applicatifs constituent une famille, et que le nombre de nœuds impliqués dans l'exécution d'une application n'a à priori aucune influence sur les facteurs dominant la complexité associée à l'analyse du profil généré par cette dernière. Néanmoins, dans le domaine du calcul à haute performance, les études de performances comportent systématiquement une évaluation de la capacité de passage à l'échelle des outils et/ou applications considérés. Ce qui implique de mesurer l'évolution de grandeurs caractéristiques du niveau de performance atteint en fonction du nombre de nœuds impliqués. Ce qui motive cette remarque est donc qu'il vaut mieux dissiper une incertitude en enfonçant une porte ouverte que risquer une incompréhension.

8.5 Bilan des expériences et cas d'utilisation envisagés

Après avoir dressé un bilan des expérimentations pour mettre en lumière leurs principaux résultats dans la sous-section 8.5.1, la sous-section 8.5.2 passera brièvement en revue certains cas d'utilisation que ces résultats permettent d'envisager. Enfin, la sous-section 8.5.3 proposera une comparaison de `Phase-TA` aux différents outils et méthodologies concurrents, présentés lors de l'étude de l'état de l'art.

8.5.1 Bilan des expériences

Le premier résultat important qu'il convient de rappeler concerne les applications de calcul à haute performance itératives. En effet, l'analyse de profils applicatifs associés au panel applicatif constitué de `HPCG`, `NEMO`, et `OpenFOAM` a permis de confirmer empiriquement que ces dernières exhibent des comportements localement périodiques qui représentent des portions très largement significatives de leurs temps d'exécution. Ces analyses ont également permis de démontrer que `Phase-TA` est capable de détecter fidèlement ces comportements localement périodiques, et de construire des motifs représentatifs des périodicités associées.

De plus, il a été démontré que lesdits motifs construits par `Phase-TA` sont des substituts pertinents à la fois des périodicités en question, et de leurs occurrences, dès lors qu'ils ont été inférés à partir d'un nombre suffisant d'instances périodiques (i.e. 100), quota généralement atteint lorsque la durée du profil applicatif analysé dépasse les 10 minutes. Un résultat connexe particulièrement intéressant est qu'un motif représentatif construit par `Phase-TA` à partir du profil applicatif associé à un nœud de calcul demeure pertinent pour les profils applicatifs associés aux autres nœuds impliqués dans l'exécution de l'application.

Si l'étude théorique de la complexité algorithmique de la méthodologie implémentée par `Phase-TA` a montré qu'elle était globalement quadratique de la longueur de la série temporelle analysée, les mesures pratiques de performance se sont révélées plus que satisfaisantes puisqu'il est possible d'analyser un profil applicatif de 10 minutes en moins de 10 secondes, ce qui permet d'envisager l'analyse de données à chaud.

Enfin, le fait que l'ensemble des expérimentations présentées par ce chapitre aient été réalisées avec la configuration par défaut de `Phase-TA` confirme que sa conception et son développement permettent de construire une configuration adaptée à une famille de séries temporelles spécifique.

8.5.2 Cas d'utilisation envisagés

Les résultats des expérimentations présentées par ce chapitre permettent d'envisager de multiples cas d'utilisation pour `Phase-TA`, dont la plus-value semble particulièrement élevée pour trois d'entre eux. Ces cas d'utilisation sont d'autant plus intéressants qu'ils s'inscrivent dans des domaines actuellement en pleine effervescence.

Fournir des capacités de prédiction aux outils de reconfiguration dynamique dans le domaine du calcul à haute performance Raison principale[5] pour laquelle `Phase-TA` a

5. Consulter la section 1.2 pour de plus amples précisions.

été développé, le fait de donner aux outils de reconfiguration un moyen de prédire le comportement à venir des applications monitorées est un enjeu fondamental du domaine du calcul à haute performance (cf. section 3.1). Une vaste majorité des outils de caractérisation du comportement des applications `HPC` nécessitent une modification de leurs codes sources, et/ou une instrumentation des environnements d'exécution. Au contraire, `Phase-TA` construit une modélisation pertinente d'une portion significative du comportement (passé, présent, et **futur**) de l'application monitorée grâce à une approche en **boîte noire**. Et cela constitue un avantage majeur à l'adoption des outils s'appuyant sur `Phase-TA` pour mettre en place des reconfigurations prédictives, aussi bien de la part de l'utilisateur final du supercalculateur qui n'a pas besoin de modifier son application que de l'administrateur du système qui peut donc mettre en place l'utilisation desdits outils sans avoir à impliquer les utilisateurs finaux.

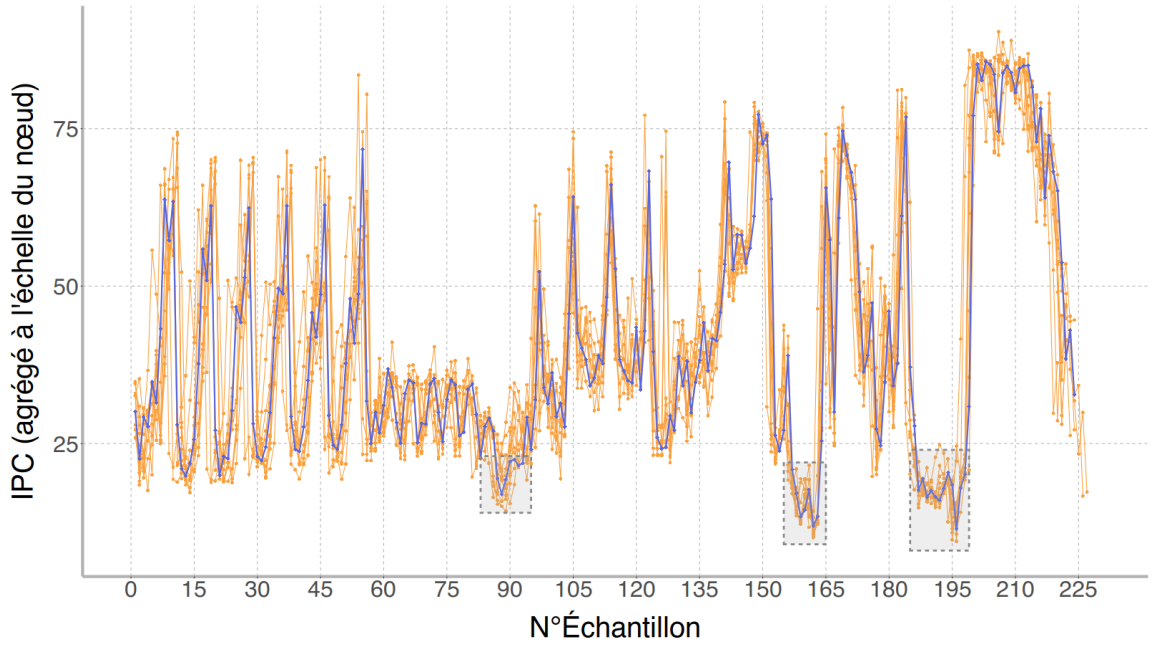


FIGURE 8.7 – Motif représentatif (en violet) de la périodicité associée au noyau de calcul principal de `NEMO`, superposé à un ensemble quelconque de 10 des instances périodiques (en orange) à partir desquelles il a été inféré.
Trois « zones d'intérêt » pour des reconfigurations du point de fonctionnement { tension ; fréquence } ont été identifiées (en gris).

Le premier outil de reconfiguration à bénéficier des motifs représentatifs construits par `Phase-TA` sera bien évidemment `BDPO`. Ces derniers serviront vraisemblablement dans un premier temps à l'élaboration automatique de configurations spécifiques aux applications monitorées (i.e. à la définition des seuils et des points de fonctionnement { tension ; fréquence }, cf. section 5.2). Dans un second temps, l'idée sera de faire évoluer le moteur de décision de `BDPO` pour que les reconfigurations qu'il met en œuvre soient associées à des « zones d'intérêt » qui auront été identifiées lors de l'analyse des motifs représentatifs.

Lorsque l'on considère une architecture matérielle similaire à celle utilisée dans le cadres des expérimentations avec `Phase-TA`, une « zone d'intérêt » est typiquement une

portion du profil applicatif présentant un `IPC` faible (e.g. moins d'une instruction retirée par cœur de calcul par cycle de référence) pour une durée suffisamment longue pour s'assurer qu'une reconfiguration ne soit pas contre-productive (e.g. au moins une dizaine d'échantillons pour une période d'échantillonnage de $5ms$). À titre d'exemple, la figure 8.7 identifie trois zones d'intérêt sur un motif représentatif associé à un profil applicatif de `NEMO`.

Deux approches sont envisagées pour la détection des zones d'intérêt à l'exécution. La première consiste à utiliser la mesure de similarité `DTW` entre le profil applicatif et un extrait du motif représentatif antécédant la zone d'intérêt en question. La seconde consiste à définir une grammaire génératrice pour les profils applicatifs (e.g. portion ascendante, portion descendante, plateau, etc.), à l'image de ce qui est présenté dans [Madi Wamba 2017]. Il serait alors possible de représenter un motif construit par `Phase-TA` comme une succession de jetons de la grammaire, d'identifier la séquence de jetons précédant la zone d'intérêt considérée, puis de déclencher les reconfigurations lorsque ladite séquence est détectée à l'exécution. Ces deux approches s'avèrent similaires, mais diffèrent notamment par les complexités algorithmiques qu'elles induisent, et la probabilité de faux-positifs de détection de zones d'intérêt.

Compression des séries temporelles L'arrivée des objets connectés et de « l'Internet des Objets » depuis le courant des années 2000 s'est accompagnée d'une explosion des données produites et à traiter : profusion de capteurs connectés, nombre d'hôtes à monitorer toujours plus important, etc. Et cette tendance n'est pas prête de disparaître, puisqu'elle est l'un des nombreux facteurs ayant notamment annoncé l'ère du « Big Data » et de « l'Edge Computing ».

Parmi les nombreuses problématiques inhérentes à ce pan de l'informatique moderne, figurent celles du stockage et du transport de l'immense quantité de données produites. En effet, supports de stockage et réseaux de communication ont besoin d'énergie pour fonctionner, et il n'est pas envisageable d'augmenter indéfiniment espaces de stockage et bandes passantes pour permettre d'absorber cette incroyable masse de données. Ni du point de vue économique, ni du point de vue écologique.

Une des approches qui ont vu le jour pour répondre à cette problématique est de compresser, avec aussi peu de « pertes » (dont le sens varie en fonction du contexte) que possible, les données avant de les transférer ou de les stocker, afin de minimiser leur volume. La littérature scientifique contient plusieurs efforts de recherche traitant de la compression des séries temporelles, notamment issues de capteurs connectés, précisément aux fins qui viennent d'être mentionnées. On peut citer [Chowdhury 2021], [Blalock 2018], ou bien encore [Danieletto 2014], qui utilisent notamment le concept de motifs récurrents pour procéder à la compression des séries temporelles.

En ce qui concerne `Phase-TA`, son usage pour compresser les profils applicatifs en substituant les instances périodiques par les motifs représentatifs associés paraît tout à fait pertinent, notamment au regard des expérimentations présentées par la section 8.2. En effet, en supprimant les instances périodiques d'un profil applicatif, on réduit le volume de ce dernier d'une substantielle portion (au moins 2/3 pour le panel expérimental d'après la section 8.1), lorsque les motifs représentatifs et les listes des positions où il faut les insérer pour reconstituer le profil applicatif occupent un volume largement négligeable en comparaison. De plus, le profil ainsi reconstruit retranscrit fidèlement la phénoménologie

associée au profil applicatif initial. Cela permettrait par exemple de réduire drastiquement le volume occupé par un « Data Lake » de profils applicatifs destiné à nourrir des algorithmes d'apprentissage, sans perte significative d'information.

Représentation intermédiaire des séries temporelles pour les algorithmes d'apprentissage Transition toute trouvée entre le paragraphe précédent et celui-ci, puisque le troisième cas d'utilisation envisagé pour `Phase-TA` concerne les algorithmes d'apprentissage. Plus précisément, la qualité de représentations intermédiaires que peuvent revêtir les motifs représentatifs construits par `Phase-TA`. En effet, comme [Kusiak 2001], [Abedin 2021], [Zupan 1998], ou bien encore [Bhagoji 2018] le fait de transformer au préalable les données sur lequelles des algorithmes d'apprentissage seront exécutés peut avoir de nombreuses vertues : améliorer la robustesse desdits algorithmes, améliorer la précision des modèles inférés, ou bien encore diminuer les temps d'inférence.

En substituant des séries temporelles par les périodicités qu'elles exhibent, via les motifs représentatifs, `Phase-TA` offre une représentation intermédiaire qui peut potentiellement avoir de nombreuses vertues sur les algorithmes d'apprentissage qui seraient appliqués, notamment ceux traitant de la classification des applications et de leurs profils.

8.5.3 Comparaison à l'état de l'art

Maintenant que les travaux de recherche autour de `Phase-TA` ont été présentés en grande largeur, il est temps de comparer `Phase-TA` aux outils présentés dans le cadre de l'état de l'art proposé par la section 3.3.

`Sketches` **- présenté par [Indyk 2000]** Par définition de la méthodologie présenté par Indyk et al., toute série temporelle exhibe forcément un comportement périodique à laquelle sont associés une période et un motif représentatif. Ainsi, en l'absence d'une extension de la méthodologie pour définir sous quelles conditions la période et le motif représentatif sont pertinents, elle s'avère inutilisable pour caractériser des comportements **seulement localement** périodiques comme ceux exhibés par les profils des applications `HPC`. À cela s'ajoutent le fait que la version efficace de la méthodologie est probabiliste, et qu'elle ne permet de détecter qu'une seule périodicité. La méthodologie présentée par [Indyk 2000] ne peut donc pas décemment remplir les objectifs fixés par le cahier des charges pour `Phase-TA`.

`Periodicity detection in time series databases` **- présenté par [Elfeky 2005]** La méthodologie présentée par Elfeky et al. ne permet pas de construire un motif représentatif du comportement périodique détecté. De plus, la discrétisation, couplée au fait que les symboles doivent être égaux à des puissances de 2 (ce qui limite le nombre de symboles encodés sur un entier « classique » sur une machine « moderne » à 64) rend quasiment inenvisageable toute construction d'un motif périodique pertinent. Enfin, la distance de Hamming est sensible aux dilatations temporelles (i.e. des bruits d'insertion et de suppression), ce qui rend la détection de comportements périodiques peu fiable pour une grande majorité des profils applicatifs issus du domaine du calcul à haute performance.

`STNR` **- présenté par** [Rasheed 2010] L'approche de `STNR` souffre de deux principales limitations vis-à-vis des objectifs fixés pour `Phase-TA`. Tout d'abord, il est nécessaire de discrétiser les séries temporelles à valeurs réelles avant d'appliquer la méthodologie. En plus d'ajouter une étape non-négligeable à la méthodologie, à la fois du point de vue des performances et de la pertinence des résultats, le motif représentatif associé à une périodicité avérée n'est autre que la sous-chaîne considérée. En d'autres termes, la reconstruction du motif représentatif à valeurs réelles sera potentiellement fortement imprécise, et ne résultera pas d'un moyennage effectué sur l'ensemble des occurrences de la périodicité.

La deuxième limitation à mentionner est le fait que la résistance au bruit est contrainte. Tout d'abord, la quantité de bruit compensable est fixée avant l'application de la méthodologie, et a un impact fort sur les performances de cette dernière. De surcroît, la méthodologie ne permet aucune résistance au bruit de remplacement, qui affecte une majorité des séries temporelles à valeurs réelles issues de données réelles.

`AUTOPERIOD` **- présenté par** [Vlachos 2005] La première limitation de `AUTOPERIOD` concerne le fait que l'ensemble de la série temporelle doit présenter une périodicité pour qu'elle puisse être détectée. Ainsi, en l'état, `AUTOPERIOD` ne peut pas détecter des comportements seulement localement périodiques. À cela s'ajoute le fait que `AUTOPERIOD` ne permet pas d'inférer un motif représentatif pour les périodicités détectées.

`EAR` **- présenté par** [Corbalan 2020] Tout d'abord, `DynAIS` n'est conçu pour ne détecter que les périodicités d'une série temporelle discrète constituée de symboles associés aux appels `MPI` d'une application `HPC`. Première conséquence, `DynAIS` n'est, de fait, pas adapté à l'analyse de séries temporelles continues, comme le sont les profils d'applications de calcul intensif. Seconde conséquence, il ne peut être utilisé que pour les applications s'appuyant sur `MPI`, ce qui exclut par exemple `NAMD`, et nécessite de surcharger le module `PMPI` de l'implémentation de `MPI` utilisée.

De plus, il n'y a aucune garantie quant au fait qu'une périodicité locale des appels `MPI` se traduise par une périodicité locale des compteurs de performance agrégés. Enfin, bien que `EAR` infère une signature de l'application via les compteurs de performance mentionnés en sus, il n'infère aucunement les motifs représentatifs associés aux potentielles périodicités locales qu'il détecte. Ainsi, lorsque l'on cumule tous ces éléments, `DynAIS` paraît inadapté à la détection et à la caractérisation des comportements localement périodiques d'une application `HPC`.

En conclusion, il semble qu'aucun outil ne concurrence `Phase-TA` lorsqu'il s'agit de détecter les comportements localement périodiques exhibés par les profils des applications `HPC`, et de construire des motifs représentatifs des périodicités associées.

Conclusion

9

Afin de conclure ce manuscrit, la section 9.1 commencera par dresser un bilan global des différentes contributions des travaux de recherche. La section 9.2 listera alors les perspectives de poursuite desdits travaux.

9.1 Bilan des principales contributions des travaux de recherche

Le domaine du calcul à haute performance a vu la notion d'efficacité énergétique des supercalculateurs passer sur le devant de la scène avec la course vers l'ExaScale. Maintenir la consommation énergétique des futurs systèmes exaflopiques sous contrôle sera en effet nécessaire pour qu'ils soient viables, aussi bien économiquement qu'écologiquement parlant. Dans ce contexte, ce manuscrit présente des travaux de recherche articulés autour de `BDPO` et `Phase-TA`.

`BDPO` est un outil de reconfiguration dynamique qui met en place du `DVFS` pendant les phases mémoires exhibées par l'exécution d'une application `HPC` dans le but d'améliorer l'efficacité énergétique associée. Il a été démontré expérimentalement, pour deux applications de calcul intensif réelles, que l'action de `BDPO` permet de diminuer la consommation énergétique induite par les exécutions desdites applications d'au moins 15%, tout en gardant la dégradation de performance associée sous le seuil acceptable de 4%. Et cela en demeurant agnostique des applications dont les exécutions sont améliorées du point de vue de l'efficacité énergétique, notamment grâce à la méthodologie empirique de calibrage qui permet de déterminer des valeurs pour les paramètres de configuration de `BDPO` adaptées à l'architecture matérielle du nœud sur lequel elle est exécutée. Pour détecter les phases mémoires qu'il exploite via la mise en place de `DVFS`, `BDPO` s'appuie uniquement sur un monitoring à grain fin de deux métriques, l'`IPC` et le `L3RWC`. Les reconfigurations des points de fonctionnement { tension ; fréquence } des cœurs de calcul auxquelles procède `BDPO` sont donc purement réactives. Conséquence directe : il est possible que certaines reconfigurations correctement motivées s'avèrent en réalité contre-productives, participant significativement à la dégradation de performance induite par l'action de `BDPO`.

`Phase-TA` a été développé dans le but d'apporter une solution à cette problématique, en donnant à `BDPO` la capacité de mettre en place des reconfigurations prédictives. En effet,

Phase-TA est capable de détecter les comportements localement périodiques exhibés par les profils des applications HPC itératives, et de construire des motifs représentatifs des périodicités associées. Il a été démontré expérimentalement que les motifs représentatifs en question sont d'excellents substituts des périodicités, et donnent donc une connaissance des motifs qui se répètent selon des longues chaînes d'occurrences contigües. Cela revient à donner une connaissance du comportement à venir des applications HPC, ce qui permettra vraisemblablement à des outils de reconfiguration dynamique, à l'instar de BDPO, de mettre en place des reconfigurations prédictives.

Pour ce faire, Phase-TA implémente une méthodologie en trois étapes : (1) détecter les régions périodiques pour en extraire les instances périodiques, (2) clustériser les instances périodiques pour les regrouper en fonction des périodicités qui les ont engendrées, et (3) construire, pour chaque cluster d'instances périodiques, un motif moyen représentatif de la périodicité en question.

Parmi les points clés de l'implémentation de Phase-TA, on peut notamment mettre en valeur la conception, en s'appuyant sur l'existant, d'un algorithme de détection des comportements localement périodiques, muni de son processus d'auto-tunage. Ce à quoi s'ajoute la création d'un critère de coupe, pour le dendrogramme résultant du clustering hiérarchique ascendant, spécifique à l'analyse des profils applicatifs des applications HPC itératives. Sans oublier la parallélisation de l'algorithme DBA qui permet d'accélérer significativement son exécution.

Ce dernier point met en évidence le soin tout particulier qui a été porté à l'optimisation de Phase-TA, aussi bien du point de vue algorithmique qu'au niveau de son implémentation. Ce qui permet d'envisager l'analyse de séries temporelles à chaud, comme cela a été démontré via une étude des performances de Phase-TA, qui s'est révélé capable d'analyser un profil applicatif comptant 120 000 échantillons en moins de 10 secondes.

Il faut également souligner que l'implémentation de Phase-TA expose de nombreux paramètres qui permettent de configurer finement le comportement de ses trois différentes étapes. Un élément fondamental du cahier des charges de Phase-TA était qu'il soit possible de lui construire une configuration adaptée à une famille de séries temporelles donnée, c'est-à-dire permettant d'analyser de manière pertinente une vaste majorité des séries temporelles appartenant à ladite famille. La configuration par défaut de Phase-TA a été construite de sorte à être adaptée aux profils applicatifs des applications de calcul à haute performance itératives. Cette dernière a été utilisée pour analyser un large panel de profils applicatifs associés à un éventail d'applications HPC. Et les résultats de ces analyses se sont avérés pertinents, validant ainsi l'item correspondant du cahier des charges de Phase-TA.

Une contribution majeure des travaux de recherche présentés par ce manuscrit consiste précisément en la caractérisation de la pertinence des motifs représentatifs construits par Phase-TA, et en l'étude des limites dans lesquelles ladite pertinence s'exprime. Ainsi, une « règle du pouce » a été inférée : pour qu'un motif représentatif produit par Phase-TA soit pertinent, il faut qu'il ait été construit à partir d'au moins 100 instances périodiques. Et ce nombre d'instances périodiques est généralement atteint pour les profils applicatifs d'une durée supérieure ou égale à 10 minutes (lorsque générés par un échantillonnage à la période de 5 millisecondes). Lorsque cette condition est remplie, il a été démontré empiriquement que les motifs représentatifs construits par Phase-TA sont d'excellents substituts des périodicités engendrant les comportements localement périodiques exhibés

par les applications HPC itératives.

Pour conclure, on peut mettre en évidence le fait que les résultats des travaux expérimentaux présentés par ce manuscrit, autour de BDPO et de Phase-TA, laissent entrevoir de multiples cas d'utilisation. Parmi lesquels figure l'amélioration de l'efficacité énergétique associée à l'exécution d'une application HPC, raison principale pour laquelle ils ont été développés.

9.2 Perspectives de poursuite des travaux

De manière assez évidente, la première perspective de poursuite des travaux, qui sera concrétisée, consiste à concevoir et implémenter l'exploitation par BDPO des motifs représentatifs construits par Phase-TA, dans le but de mettre en place des reconfigurations prédictives. La sous-section 8.5.2 détaille notamment les pistes envisagées pour ce faire. Il s'agira également de valider empiriquement qu'une fois doté de capacité de prédiction du comportement des applications HPC itératives, BDPO permettra d'améliorer leurs efficacités énergétiques, tout en ayant un impact négligeable, et non plus « acceptable », sur leurs performances. Et il faudra le faire sur un panel d'applications de calcul à haute performance encore plus vaste que celui, déjà large, utilisé dans le cadre des travaux de recherche présentés par ce manuscrit.

À l'heure actuelle, comme cela est mentionné par le chapitre 7, Phase-TA est en mesure d'analyser les profils unidimensionnels, pour lesquels chaque échantillon est un scalaire. Or, de nombreuses séries de données sont multidimensionnelles, chacun de leurs échantillons étant des vecteurs. Bien que Phase-TA ait été développé en prenant en compte le fait qu'il serait amené à analyser des séries temporelles multidimensionnelles, il conviendra d'implémenter et d'optimiser le support de ces dernières.

Concernant BDPO, de nombreuses poursuites des travaux de recherche sont envisageables et envisagées. Pour commencer, l'approche de BDPO se concentre actuellement uniquement sur les phases mémoires, et considère les autres types de phases simplement comme étant sensibles à la fréquence des cœurs de calcul. De la même manière, les reconfigurations mises en place par BDPO ne font intervenir que deux points de fonctionnement { tension ; fréquence } et consistent à passer de l'un à l'autre. De ces deux remarques on peut extraire les deux axes d'amélioration suivants pour BDPO : (1) s'appuyer sur les travaux existants [1] pour construire empiriquement une taxonomie des différents types de phases caractéristiques de l'exécution d'applications HPC sur des architectures modernes, et (2) faire évoluer les organes de décision et de reconfiguration de BDPO pour permettre d'adapter finement les points de fonctionnement { tension ; fréquence } des cœurs de calcul aux différentes phases appartenant à la taxonomie mentionnée dans le premier point. Et cela va sans dire que la méthodologie de calibrage devra être adaptée en conséquence, et qu'il faudra s'assurer que BDPO reste léger et agnostique de l'application dont il cherche à améliorer l'efficacité énergétique. Ainsi, le raffinement de l'approche de BDPO constitue une piste, ou plutôt une collection de pistes, de poursuite des travaux de recherche.

La quête vers l'ExaScale, l'ère des Masses de Données, l'Intelligence Artificielle. La dernière décennie a été particulièrement riche en termes de progrès et avancées techniques, et le paysage de l'informatique de pointe s'en est retrouvé bouleversé. Le domaine du calcul à haute performance a notamment été témoin de modifications profondes des architectures des supercalculateurs : les architectures CPU+GPGPU sont devenues prépondérantes, la mémoire persistante PMem a fait son apparition, etc. Si les travaux liés au portage de l'approche de BDPO vers les architectures intégrant des accélérateurs graphiques ont déjà commencé, il reste encore beaucoup de travail à accomplir. Sans oublier que le support des nouveaux processeurs faisant leurs entrées dans le domaine du HPC, tels que les processeurs ARM et l'European Processor Initiative (EPI), constitue également une piste

1. À l'image de [Chetsa 2015].

de poursuite des travaux de recherche.

Enfin, l'industrialisation de `Phase-TA` selon les standards en vigueur au sein de la branche R&D `HPC` d'Atos permettra de le rendre utilisable par, et en collaboration avec, d'autres équipes. Cela ouvrira la voie à l'exploration d'autres cas d'utilisation, dont ceux présentés par la sous-section 8.5.2. À cette occasion, il pourrait être tout particulièrement indiqué de travailler à l'élaboration d'une méthodologie expérimentale dédiée à la construction d'une configuration de `Phase-TA` spécifique à une famille de séries temporelles.

Pour terminer ce manuscrit, soulignons une fois de plus l'importance capitale que revêt l'amélioration de l'efficacité énergétique associée à l'exécution d'une application de calcul à haute performance dans le cadre du passage de l'ExaScale. Et cette importance ne cessera d'être grandissante : les futurs systèmes exaflopiques qui naîtront de la convergence entre `HPC` et `IA` auront vraisemblablement des tailles démesurées, et les contraintes énergétiques auxquelles ils devront répondre seront fort probablement encore plus strictes que celles qui encadrent la quête de l'exaflops. De la même manière, l'importance de la boucle de rétroaction permettant l'auto-tunage de l'exécution d'une application va vraisemblablement se renforcer, notamment parce que les phases de mise au point des algorithmes d'apprentissage profond sont cruciales pour que ces derniers soient pleinement performants. Autrement dit, les problématiques abordées dans le cadre de cette thèse constituent vraisemblablement des thématiques de recherche florissantes pour les années, voire les décennies, à venir.

Bibliographie

[Abedin 2021] M. Z. Abedin, G. Chi, M. M. Uddin, S. Satu, I. Khan et P. Hajek. *Tax Default Prediction Using Feature Transformation-Based Machine Learning*. IEEE Access, vol. 9, 2021. *cité page 172*

[Achtert 2006] E. Achtert, C. Böhm et P. Kröger. *DeLi-Clu: Boosting Robustness, Completeness, Usability, and Efficiency of Hierarchical Clustering by a Closest Pair Ranking*. Dans Advances in Knowledge Discovery and Data Mining, 2006. *cité page 127*

[André 2020] E. André, R. Dulong, A. Guermouche et F. Trahay. *DUF: Dynamic Uncore Frequency scaling to reduce power consumption*. 2020. *cité page 53*

[Ankerst 1999] M. Ankerst, M. Breunig, H-P. Kriegel et J. Sander. *OPTICS: Ordering Points to Identify the Clustering Structure*. SIGMOD Rec., vol. 28, no. 2, 1999. *cité page 125*

[Ao 2018] Y. Ao, C. Yang, F. Liu, W. Yin, L. Jiang et Q. Sun. *Performance Optimization of the HPCG Benchmark on the Sunway TaihuLight Supercomputer*. ACM Trans. Archit. Code Optim., vol. 15, no. 1, 2018. *3 citations pages 34, 72, et 74*

[Archibald 2020] R. Archibald, E. Chow, E. D'Azevedo, J. Dongarra, M. Eisenbach, R. Febbo, F. Lopez, D. Nichols, S. Tomov, K. Wong et J. Yin. *Integrating Deep Learning in Domain Sciences at Exascale*. Dans Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI, 2020. *cité page 47*

[Barbosa 2018] J. Barbosa, F. Almeida, M. D. Assunção, V. Blanco, I. Brandic, G. da Costa, M. F. Dolz, A. C. Elster, M. Jarus, H. D. Karatza*et al.* *Energy monitoring as an essential building block towards sustainable ultrascale systems*. 2018. *cité page 49*

[Bhagoji 2018] A. N. Bhagoji, D. Cullina, C. Sitawarin et P.Mittal. *Enhancing robustness of machine learning systems via data transformations*. Dans 2018 52nd Annual Conference on Information Sciences and Systems (CISS), 2018. *cité page 172*

[Blalock 2018] D. Blalock, S. Madden et J. Guttag. *Sprintz: Time Series Compression for the Internet of Things*. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., vol. 2, no. 3, 2018. *cité page 171*

[Cesarini 2020] D. Cesarini, A. Bartolini, A. Borghesi, C. Cavazzoni, M. Luisier et L. Benini. *Countdown Slack: A Run-Time Library to Reduce Energy Footprint in Large-Scale MPI Applications*. IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 11, 2020. *4 citations pages 11, 54, 55, et 99*

[Chetsa 2012] G. L. T. Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf et G. Da Costa. *A Runtime Framework for Energy Efficient HPC Systems Without a Priori Knowledge of Applications*. Dans Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, 2012. *cité page 58*

[Chetsa 2013] G. L. T. Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf et G. Da Costa. *A User Friendly Phase Detection Methodology for HPC Systems' Analysis*. GreenCom/iThings/CPScom, 2013. *cité page 58*

[Chetsa 2014a] G. L. T. Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf et G. Da Costa. *Exploiting Performance Counters to Predict and Improve Energy Performance of HPC Systems*. Future Generation Computer Systems, vol. 36, 2014. *cité page 58*

[Chetsa 2014b] G. L. T. Chetsa, L. Lefèvre et P. Stolf. *A Three Step Blind Approach for Improving HPC Systems' Energy Performance*. Concurrency and Computation: Practice and Experience, vol. 26, 2014. *cité page 58*

[Chetsa 2015] G. L. T. Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf et G. Da Costa. *Application-agnostic framework for improving the energy efficiency of multiple HPC subsystems*. Dans 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015. *6 citations pages 11, 58, 71, 100, 108, et 177*

[Chowdhury 2021] M. R. Chowdhury, S. Tripathi et S. De. *Adaptive Multivariate Data Compression in Smart Metering Internet of Things*. IEEE Transactions on Industrial Informatics, vol. 17, no. 2, 2021. *cité page 171*

[Christodoulis 2019] G. Christodoulis. *Adapting a HPC runtime system to FPGAs*. PhD thesis, 2019. *cité page 39*

[Corbalan 2020] J. Corbalan, L. Alonso, J. Aneas et L. Brochard. *Energy Optimization and Analysis with EAR*. Dans 2020 IEEE International Conference on Cluster Computing (CLUSTER) - Energy Efficient HPC State of the Practice Workshop (EE HPC SOP), 2020. *5 citations pages 11, 57, 62, 99, et 173*

[Da Costa 2015] G. Da Costa et J. Pierson. *DVFS Governor for HPC: Higher, Faster, Greener*. Dans 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015. *2 citations pages 56 et 98*

[Danieletto 2014] M. Danieletto, N. Bui et M. Zorzi. *RAZOR: A Compression and Classification Solution for the Internet of Things*. Sensors, vol. 14, no. 1, 2014. *cité page 171*

[Dhillon 2004] I. Dhillon, Y. Guan et B. Kulis. *Kernel K-Means: Spectral Clustering and Normalized Cuts*. Dans Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004. *cité page 125*

[Di Matteo 2020] T. Di Matteo, R. Croft, Y. Ni, S. Bird, Y. Li et Y. Feng. *AI-assisted HPC simulations and application to astrophysics and cosmology*. 2020. *cité page 47*

[Eastep 2016] J. Eastep, S. Sylvester, C. Cantalupo, F. Ardanaz, B. Geltz, A. Al-Rawi, F. Keceli et K. Livingston. *Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration Toward Co-Designed Energy Management Solutions*. ACM/IEEE Supercomputing 2016 (SC16), 2016. *cité page 53*

[Eeckhout 2005] L. Eeckhout, J. Sampson et B. Calder. *Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation*. Dans Proceedings of the IEEE Workload Characterization Symposium, 2005. *cité page 50*

[Elfeky 2005] M. G. Elfeky, W. G. Aref et A. K. Elmagarmid. *Periodicity detection in time series databases.* IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 7, 2005. *2 citations pages 60 et 172*

[Elkan 2003] C. Elkan. *Using the Triangle Inequality to Accelerate K-Means.* Dans Proceedings of the Twentieth International Conference on International Conference on Machine Learning, 2003. *cité page 125*

[Ferreira Lima 2019] J. V. Ferreira Lima, I. Raïs, L. Lefèvre et T. Gautier. *Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms.* The International Journal of High Performance Computing Applications, vol. 33, no. 3, 2019. *cité page 52*

[Freitag 2001] F. Freitag, J. Corbalan et J. Labarta. *A dynamic periodicity detector: application to speedup computation.* Dans Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001, 2001. *3 citations pages 62, 111, et 112*

[Gholkar 2018] N. Gholkar, F. Mueller, B. Rountree et A. Marathe. *PShifter: feedback-based dynamic power shifting within HPC jobs for performance.* 2018. *cité page 53*

[Gholkar 2019] N. Gholkar, F. Mueller et B. Rountree. *Uncore power scavenger: A runtime for uncore power conservation on HPC systems.* Dans Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019. *2 citations pages 11 et 53*

[Gribel 2019] D. Gribel et T. Vidal. *HG-means: A scalable hybrid genetic algorithm for minimum sum-of-squares clustering.* Pattern Recognition, vol. 88, 2019. *cité page 125*

[Hackenberg 2014] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon et Y. Georgiou. *HDEEM: high definition energy efficiency monitoring.* Dans 2014 Energy Efficient Supercomputing Workshop, 2014. *2 citations pages 39 et 49*

[Hackenberg 2015] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart et R. Geyer. *An Energy Efficiency Feature Survey of the Intel Haswell Processor.* Dans 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, 2015. *2 citations pages 51 et 73*

[Halimi 2014] J. P. Halimi, B. Pradelle, A. Guermouche et W. Jalby. *FoREST-mn: Runtime DVFS beyond communication slack.* Dans International Green Computing Conference, 2014. *3 citations pages 11, 54, et 71*

[Huang 2020] L. Huang, E. Clee et N. Ranasinghe. *Improving Seismic Wave Simulation and Inversion Using Deep Learning.* Dans Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI, 2020. *cité page 47*

[Ibeid 2018] H. Ibeid, L. Olson et W. Gropp. *FFT, FMM, and Multigrid on the Road to Exascale: performance challenges and opportunities.* 2018. *2 citations pages 74 et 75*

[Indyk 2000] P. Indyk, N. Koudas et S. Muthukrishnan. *Identifying representative trends in massive time series data sets using sketches.* Dans VLDB, 2000. *2 citations pages 60 et 172*

[Isci 2006] C. Isci, G. Contreras et M. Martonosi. *Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management*. Dans 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), 2006. *2 citations pages 11 et 108*

[Kjeldsberg 2017] P. G. Kjeldsberg, A. Gocht, M. Gerndt, L. Riha, J. Schuchart et U. S. Mian. *READEX: Linking two ends of the computing continuum to improve energy-efficiency in dynamic applications*. Dans Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, 2017. *cité page 59*

[Kocher 2019] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz et Y. Yarom. *Spectre Attacks: Exploiting Speculative Execution*. Dans 2019 IEEE Symposium on Security and Privacy (SP), 2019. *cité page 28*

[Koskela 2018] T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan et L. Oliker. *A novel multi-level integrated Roofline model approach for performance characterization*. Dans International Conference on High Performance Computing, 2018. *cité page 42*

[Kudo 2020] S. Kudo, K. Nitadori, T. Ina et T. Imamura. *Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku*. Dans 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), 2020. *cité page 47*

[Kumaraswamy 2018] M. Kumaraswamy, A. Chowdhury et M. Gerndt. *Design-time analysis for the READEX tool suite*. Parallel Computing is Everywhere, vol. 32, 2018. *cité page 59*

[Kusiak 2001] A. Kusiak. *Feature transformation methods in data mining*. IEEE Transactions on Electronics Packaging Manufacturing, vol. 24, no. 3, 2001. *cité page 172*

[Legrand 2019] A. Legrand, D. Trystram et S. Zrigui. *Adapting Batch Scheduling to Workload Characteristics: What Can We Expect From Online Learning?* Dans 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019. *cité page 48*

[Li 2010] D. Li, B. R. de Supinski, M. Schulz, K. Cameron et D. S. Nikolopoulos. *Hybrid MPI/OpenMP power-aware computing*. Dans 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010. *3 citations pages 11, 54, et 55*

[Libri 2018] A. Libri, A. Bartolini et L. Benini. *Dwarf in a Giant: Enabling Scalable, High-Resolution HPC Energy Monitoring for Real-Time Profiling and Analytics*. arXiv preprint arXiv:1806.02698, 2018. *2 citations pages 49 et 50*

[Liu 2010] Y. Liu et H. Zhu. *A survey of the research on power management techniques for high-performance systems*. Software: Practice and Experience, vol. 40, no. 11, 2010. *cité page 72*

[Lively 2014] C. Lively, V. Taylor, X. Wu, H.C. Chang, C-Y. Su, K. Cameron, S. Moore et D. Terpstra. *E-AMOM: an energy-aware modeling and optimization methodology for scientific applications*. Computer Science - Research and Development, 2014. *3 citations pages 11, 56, et 99*

[Lloyd 1982] S. Lloyd. *Least squares quantization in PCM*. IEEE Transactions on Information Theory, 1982. *cité page 124*

[Lobet 2018] M. Lobet, M. Haefele, V. Soni, P. Tamain et J. Derouillat. *High-Performance Computing at Exascale: challenges and benefits.* Oral presentation, 2018. *cité page 47*

[Madi Wamba 2017] G. Madi Wamba, Y. Li, A-C. Orgerie, N. Beldiceanu et J-M. Menaud. *Cloud workload prediction and generation models.* Dans SBAC-PAD: International Symposium on Computer Architecture and High Performance Computing, 2017. *cité page 171*

[Mazouz 2013] A. Mazouz, A. Laurent, B. Pradelle et W. Jalby. *Evaluation of CPU frequency transition latency.* Computer Science - Research and Development, vol. 29, 2013. *2 citations pages 73 et 98*

[Mellanox 2019] Mellanox. *White Paper: Introducing 200G HDR InfiniBand Solutions.* Rapport technique, 2019. *cité page 36*

[Miyazaki 2018] T. Miyazaki, I. Sato et N. Shimizu. *Bayesian Optimization of HPC Systems for Energy Efficiency.* Dans High Performance Computing, 2018. *cité page 120*

[Nicola 2018] J. Nicola. *How to stop data centres from gobbling up the world's electricity.* Nature, vol. 561, 2018. *cité page 47*

[Oleynik 2015] Y. Oleynik, M. Gerndt, J. Schuchart, P. G. Kjeldsberg et W. E. Nagel. *Run-Time Exploitation of Application Dynamism for Energy-Efficient Exascale Computing (READEX).* Dans 2015 IEEE 18th International Conference on Computational Science and Engineering, 2015. *cité page 59*

[Olivry 2020] A. Olivry, J. Langou, L-N. Pouchet, P. Sadayappan et F. Rastello. *Automated derivation of parametric data movement lower bounds for affine programs.* Dans PLDI 2020 - 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2020. *cité page 40*

[Patel 2020] T. Patel, A. Wagenhäuser, C. Eibel, T. Hönig, T. Zeiser et D. Tiwari. *What does Power Consumption Behavior of HPC Jobs Reveal? : Demystifying, Quantifying, and Predicting Power Consumption Characteristics.* Dans 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020. *cité page 48*

[Pedretti 2018] K. Pedretti, R. E. Grant J. H. Laros III, M. Levenhagen, S. L. Olivier, L. Ward et A. J. Younge. *A Comparison of Power Management Mechanisms: P-States vs. Node-Level Power Cap Control.* Dans 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018. *2 citations pages 33 et 54*

[Petitjean 2011] F. Petitjean, A. Ketterlin et P. Gançarski. *A global averaging method for dynamic time warping, with applications to clustering.* Pattern Recognition, vol. 44, no. 3, 2011. *4 citations pages 135, 136, 138, et 166*

[Petitjean 2014] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen et E. Keogh. *Dynamic Time Warping Averaging of Time Series Allows Faster and More Accurate Classification.* Dans 2014 IEEE International Conference on Data Mining, 2014. *2 citations pages 136 et 166*

[Petitjean 2016] F. Petitjean, G. Forestier, G. Webb, A. Nicholson, Y. Chen et E. Keogh. *Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm.* Knowledge and Information Systems, vol. 47, 2016. *cité page 136*

[Platini 2021] M. Platini, T. Ropars, B. Pelletier et N. De Palma. *CPU overheating prediction in HPC systems.* Concurrency and Computation: Practice and Experience, 2021. *cité page 48*

[Raïs 2018a] I. Raïs, H. Coullon, L. Lefèvre et C. Pérez. *Automatic Energy Efficient HPC Programming: A Case Study.* Dans 2018 IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), 2018. *cité page 51*

[Raïs 2018b] I. Raïs, L. Lefèvre, A-C. Orgerie et A. Benoit. *Exploiting the Table of Energy and Power Leverages.* Dans International Conference on Algorithms and Architectures for Parallel Processing, 2018. *2 citations pages 13 et 52*

[Rasheed 2010] F. Rasheed et R. Alhajj. *STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases.* Applied Intelligence, vol. 32, no. 3, 2010. *2 citations pages 61 et 172*

[Rashti 2015] M. Rashti, G. Sabin, D. Vansickle et B. Norris. *WattProf: A Flexible Platform for Fine-Grained HPC Power Profiling.* Dans 2015 IEEE International Conference on Cluster Computing, 2015. *cité page 49*

[Rountree 2009] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh et T. Bletsch. *Adagio: Making DVS Practical for Complex HPC Applications.* Dans Proceedings of the 23rd International Conference on Supercomputing, 2009. *cité page 11*

[Sakamoto 2017] R. Sakamoto, T. Cao, M. Kondo, K. Inoue, M. Ueda, T. Patki, D. Ellsworth, B. Rountree et M. Schulz. *Production Hardware Overprovisioning: Real-World Performance Optimization Using an Extensible Power-Aware Resource Management Framework.* Dans 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2017. *cité page 48*

[Sakoe 1971] H. Sakoe et S. Chiba. *A Dynamic Programming Approach to Continuous Speech Recognition.* Dans Proceedings of the Seventh International Congress on Acoustics, volume 3, 1971. *2 citations pages 44 et 166*

[Schuchart 2017] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg et W. E. Nagel. *The READEX formalism for automatic tuning for energy efficiency.* Computing, 2017. *3 citations pages 11, 58, et 100*

[Schöne 2016] R. Schöne, T. Ilsche, M. Bielert, D. Molka et D. Hackenberg. *Software Controlled Clock Modulation for Energy Efficiency Optimization on Intel Processors.* Dans 2016 4th International Workshop on Energy Efficient Supercomputing (E2SC), 2016. *cité page 32*

[Schöne 2019] R. Schöne, T. Ilsche, M. Bielert, A. Gocht et D. Hackenberg. *Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance*. Dans 2019 International Conference on High Performance Computing Simulation (HPCS), 2019. *3 citations pages 33, 51, et 73*

[Servat 2018] H. Servat, J. Labarta, H-C. Hoppe, J. Giménez et A. J. Peña. *Understanding memory access patterns using the BSC performance tools*. Parallel Computing, vol. 78, 2018. *cité page 48*

[Shalf 2011] J. Shalf, S. Dosanjh et J. Morrison. *Exascale Computing Technology Challenges*. Dans High Performance Computing for Computational Science – VECPAR 2010, 2011. *cité page 11*

[Sherwood 2001] T. Sherwood, E. Perelman et B. Calder. *Basic block distribution analysis to find periodic behavior and simulation points in applications*. Dans Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques, 2001. *cité page 50*

[Sherwood 2002] T. Sherwood, E. Perelman, G. Hamerly et B. Calder. *Automatically Characterizing Large Scale Program Behavior*. ACM SIGPLAN Notices, 09 2002. *cité page 50*

[Shoga 2014] K. Shoga, B. Rountree, M. Schulz et J. Shafer. *Whitelisting MSRs with msr-safe*. Dans 3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14, 2014. *cité page 81*

[Shoukourian 2017] H. Shoukourian, T. Wilde, H. Huber et A. Bode. *Analysis of the efficiency characteristics of the first high-temperature direct liquid cooled petascale supercomputer and its cooling infrastructure*. Journal of Parallel and Distributed Computing, vol. 107, 2017. *cité page 11*

[Sibson 1973] R. Sibson. *SLINK: An optimally efficient algorithm for the single-link cluster method*. The Computer Journal, vol. 16, no. 1, 1973. *3 citations pages 128, 129, et 166*

[Silvano 2019] C. Silvano, G. Agosta, A. Bartolini, A. Beccari, L. Benini, L. Besnard, J. Bispo, R. Cmar, J. Cardoso, C. Cavazzoni*et al. Supporting the Scale-Up of High Performance Application to Pre-Exascale Systems: The ANTAREX Approach*. Dans 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2019. *2 citations pages 11 et 48*

[Sourouri 2017] M. Sourouri, EB. Raknes, N. Reissmann, J. Langguth, D. Hackenberg, R. Schöne et PG. Kjeldsberg. *Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures*. Dans Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017. *3 citations pages 11, 59, et 98*

[Spiliopoulos 2011] V. Spiliopoulos, S. Kaxiras et G. Keramidas. *Green governors: A framework for Continuously Adaptive DVFS*. Dans 2011 International Green Computing Conference and Workshops, 2011. *cité page 73*

[Stoffel 2018] M. Stoffel et A. Mazouz. *Improving Power Efficiency Through Fine-Grain Performance Monitoring in HPC Clusters*. Dans 2018 IEEE International Conference on Cluster Computing (CLUSTER) - HPCMASPA Workshop, 2018. *cité page 16*

[Stoffel 2021] M. Stoffel, F. Broquedis, F. Desprez et A. Mazouz. *Phase-TA: Periodicity Detection and Characterization for HPC Applications*. Dans HPCS 2020 - 18th IEEE International Conference on High Performance Computing and Simulation, 2021. *cité page 16*

[Strevell 2019] M. Strevell, D. Lambiaso, A. Brendamour et T. Squillo. *Designing an Energy-Efficient HPC Supercomputing Center*. Dans Proceedings of the 48th International Conference on Parallel Processing: Workshops, 2019. *cité page 11*

[Sundriyal 2018] V. Sundriyal, M. Sosonkina, B. Westheimer et M. Gordon. *Core and uncore joint frequency scaling strategy*. Journal of Computer and Communications, vol. 6, 2018. *cité page 53*

[Trahay 2018] F. Trahay, M. Selva, L. Morel et K. Marquet. *NumaMMA: NUMA memory analyzer*. Dans Proceedings of the 47th International Conference on Parallel Processing, 2018. *cité page 140*

[Vlachos 2005] M. Vlachos, P. Yu et V. Castelli. *On Periodicity Detection and Structural Periodic Similarity*. Dans Proceedings of the 2005 SIAM International Conference on Data Mining, 2005. *2 citations pages 61 et 173*

[Vysocky 2018] O. Vysocky, M. Beseda, L. Říha, J. Zapletal, M. Lysaght et V. Kannan. *MERIC and RADAR Generator: Tools for Energy Evaluation and Runtime Tuning of HPC Applications*. Dans High Performance Computing in Science and Engineering, 2018. *cité page 11*

[Williams 2009] S. Williams, A. Waterman et D. Patterson. *Roofline: An Insightful Visual Performance Model for Multicore Architectures*. Commun. ACM, vol. 52, no. 4, 2009. *2 citations pages 40 et 41*

[Zrigui 2020] S. Zrigui, R. de Camargo, A. Legrand et D. Trystram. *Improving the performance of batch schedulers using online job runtime classification*. IEEE Transactions on Parallel and Distributed Systems, 2020. *cité page 48*

[Zupan 1998] B. Zupan, M. Bohanec, J. Demšar et I. Bratko. Feature transformation by function decomposition. 1998. *cité page 172*