Discover, model and combine energy leverages for large scale energy efficient infrastructures

Issam Raïs

Numéro National de Thèse : 2018LYSEN051

THÈSE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée par

l'École Normale Supérieure de Lyon

École Doctorale N°512
Informatique et Mathématiques de Lyon

Discipline : Informatique

présentée et soutenue publiquement le 28/09/2018, par :

Issam RAIS

Discover, model and combine energy leverages for large scale energy efficient infrastructures

Découvrir, modéliser et combiner des leviers énergétiques pour des infrastructures de calculs basse consommation

Devant le jury composé de :

Pascal BOUVRY	Professeur, University of Luxembourg	*Rapporteur*
Sébastien MONNET	Professeur, Université Savoie Mont Blanc	*Rapporteur*
Alba Cristina DE MELO	Professeure, University of Brasilia	*Examinatrice*
Frederic DESPREZ	Directeur de Recherche Inria	*Examinateur*
Jean-Marc PIERSON	Professeur, Université Paul Sabatier	*Examinateur*
Laurent LEFEVRE	Chargé de Recherche, Inria	*Directeur*
Anne BENOIT	Maître de Conférences, ENS de Lyon	*Co-directrice*
Anne-Cécile ORGERIE	Chargée de Recherche CNRS, Irisa	*Co-encadrante*

Contents

Remerciements

Je remercie toutes celles et ceux qui ont échangé avec moi, m'ont encouragé, soutenu et supporté lors de cette aventure de 3 ans. Tout particulièrement tout l'écosystème qui m'a permis de m'épanouir et grandir en tant que chercheur.

Je suis également reconnaissant envers l'ensemble des membres du jury, en particulier mes deux rapporteurs pour avoir lu ce manuscrit en détail mais aussi le reste du jury pour avoir accepté d'en faire partie.

À ma famille qui a littéralement tous les jours été là pour moi, des moments de doutes aux réussites, qui réussit à me faire prendre de nouveaux points de vue pour me faire relativiser et réaliser ce qui est accompli, et ce bien avant mon idée de me lancer dans l'informatique. Aux excellents et pertinents conseils sans lesquels je ne serais pas là aujourd'hui. Jamais mes choix ont été questionnés, toujours poussés vers l'avant à "faire ce que je veux". Mes points de vues brutes et non réfléchis ont toujours été écoutés et discutés. Vous le savez, je ne vous remercierai jamais assez.

Pour le pousse, un socle solide est base et gage de réussite valide

Cette thèse s'est effectuée au croisement de nombreuses équipes, qui m'ont accueillies au sein de leurs environnements. Ce croisement des chemins a permis une prise de recul inestimable, une analyse et de nombreuses rencontres autrement impossibles.

Cette thèse a été portée par un triple encadrement. Je remercie forcement et fortement celui qui m'a toujours considéré comme un collègue et m'a surtout supporté durant ces trois longues années (et demi, pour lui). Celui qui arrive le matin en disant qu'il a une nouvelle collaboration possible, avec l'autre bout du monde. Celui qui a écrit avec moi le sujet du DS à M-15. Celui qui met tout dans le sujet du mail, qui est toujours occupé mais a quand même du temps pour tous ses doctorants, qui évite et est précis avec les conflits, qui est constamment de bonne humeur, même quand c'était compliqué, qui a toujours su trouver les mots justes pour répondre à mes incessantes demandes de points supposés durer "5 minutes, t'inquiètes pas ..." mais qui finissaient par durer 2 heures (en moyenne), les prises de choux sur les détails, la terminologie des termes utilisés et les changements d'avis et retours en arrière, toujours pour le bien du travail en cours. Pour avoir écouté mes idées les plus folles et ne jamais m'avoir stoppé dans l'exécution de l'une d'elles, m'avoir toujours fait confiance et toujours pousser vers l'avant.

À celle qui a su m'accueillir et me mettre à l'aise dans une équipe lointaine, avec qui les affinités et discussions ont été tout de suite très simples et saines, qui m'a accueillie dans son bureau lors de mes déplacements dans l'équipe. Qui a tout le temps été à l'écoute et donné d'excellents conseils, sur le présent ou le futur. Qui a toujours répondu avec franchise aux questions. Qui m'a aidé à structurer mes idées d'une manière remarquable à l'écrit. Qui a été précise, parfois incisive mais toujours très pertinente.

À celle qui a été très précise lors des relectures, des discussions et avec qui j'ai partagé des repas et des idées lors des fameux "points sandwich du vendredi", qui finissaient par une discussion sur l'article ou le sujet de recherche en cours, qui avait le recul nécessaire sur la situation.

Je remercie ce trio d'encadrement d'avoir su me laisser le choix et la liberté de tenter des choses depuis le premier jour de cette thèse, d'avoir répondu avec précision à certaines des questions par mails, d'être perdu mais quand même de suivre tous mes *gits*, de m'avoir laissé proposer ma méthodologie et mes idées, d'avoir écouté mes innombrables présentations, lu et relu tous mes écrits, de l'idée sur pdf non formatée aux articles de journaux, de me laisser me tromper et ensuite m'expliquer mes erreurs et m'aider à fixer le tir.

Un tuteur aide le pousse à grandir droit, essayez donc avec trois !

À l'ensemble de l'équipe avec laquelle j'ai d'abord été accueillie comme stagiaire et ensuite comme collègue. Les pauses café animées du début de thèse, les départs au RU à 11:30. Équipe où j'ai grandi en tant que chercheur, en ayant reçu beaucoup de connaissances, intra ou extra recherche. Sans oublier le pôle d'Orléanais au labo. À celui qui a toujours trouvé du temps pour m'expliquer la situation actuelle de la recherche, toujours de bon conseil et m'aider pour les choix futurs. À celui qui m'a aussi toujours considéré comme collègue, toujours à blaguer sur ... tout ce qui était pour lui exploitable, en étant présent pour avoir les échanges les plus drôles. À celle qui a toujours été présente, de ma signature de contrat en stage, à mes nombreuses missions faites au dernier moment (désolé), toujours avec le sourire, la bonne humeur et une pédagogie irréprochable.

À ma grande sœur de la recherche, qui depuis sa thèse et jusqu'à aujourd'hui encore, écoute et supporte mes questions et pensées alambiquées, mais qui m'a toujours répondu avec énormément de gentillesse (même quand elle *en a gros*) à deux bureaux ou à l'autre bout de la France, et qui m'a fait découvrir la recherche mais aussi le sport le plus cool ... après le futsal. Avec qui je pouvais parler de tout, intra et extra recherche, qui me permet un prisme sur le futur dans la recherche. Avec qui j'ai effectué ce qui restera tout de même ma collaboration la plus vivante de cette thèse.

À celui qui a compté mes *collaborations* (et qui compte encore...), les discussions autour de mets raffinés même si le service était à désirer, les schémas explicatifs sur post-it, sur mes rapports doubles colonnes, ma créativité doxygen sur mes sites (j'aime bien, moi), qui comprend pas la vie que je mène, qui me laisse écrire sur les vitres du bureau, sans prêt d'espace Veleda (*Mais c'est cool*), les remplacements à la Doua, les *bars to bars sessions* improvisées, *My Salsa*, *Martin*, les *diss* gratuits mais toujours bien placés, les suggestions bien trouvées. La liste est longue donc je vais m'arrêter. Le meilleur co-bureau, *How Sway ?*, parce qu'il reste tard pour travailler, *on aura quand même bien rigolé.*

C'est en voyant l'aîné s'étendre que le pousse va comprendre.

Mes nombreux et variés co-bureau, qui ont dus me voir coloniser le tableau (d'une précédente expérience frustrante ?), voir mes étudiants et mes collaborateurs débouler dans le bureau, donc me supporter durant ces 3 années. À ceux qui m'ont toujours accueillis en pause, au 2eme comme au 3eme étage, que ce soit pour discuter, de recherche mais surtout du reste, ou pour jouer au basket ou au foot.

À tous ceux et celles qui ont partagés un repas avec moi, avec une mention très spéciale au petit rayon de soleil de la salle passerelle, qui a grandi en même temps que ce manuscrit.

À celle qui a partagée mon bureau lors de cette troisième année de thèse, pour sa bonne humeur infinie et son point de vue toujours positif sur le monde, les *délices*, les soirées de partage dans ou en dehors du travail.

À celle qui a partagée avec moi ses points de vue sur la recherche, l'écriture et l'industrie et son stagiaire. Qui m'a encore plus donné envie de faire un post-doc, qui a essayé avec succès de faire deux équipes se rencontrer en dehors du travail, qui fait des journées de folie sans montrer signe de fatigue. *blush*

À celle qui a partagée avec moi ces matchs et ces discussions autour du foot, l'île aux chien, les nombreuses pizza (et autres) englouties, les récits sur ses voyages et ses rencontres, la complicité et le shi fu mi où elle excelle encore, toujours avec le sourire et une joie de vivre contagieuse.

À celle qui a partagée mes points de vue parfois compliqués lors de déjeuners et balades. Qui a toujours su être à l'écoute, me faire prendre du recul que ce soit lors de discussions durant ces repas, post pièces de théâtre ou sur les quais, avec qui je peux parler de tout. Aux analyses précises de toiles, à tous ces *Theads* vivants ou morts sur des sujets extrêmement variés. Merci pour ton temps, ton envie de partager sans fin, même avec ton agenda bien plein.

Merci à tous ces collègues qui pour certains sont devenus bien plus. Tous les cours donnés, les TPs et TDs donnés, tous les repas partagés, toutes les discussions, toutes les réunions, toutes les collaborations, tous les événements, toutes les conférences, tous les lieux. Tout le monde s'y reconnaîtra.

Une plante avec une seule source, aussi parfaite soit elle, ne sera pas satisfaite.

Résumé en français

Répondre aux problématiques et s'attaquer à l'inconnu a conduit à une croissance exponentielle des besoins informatiques. Les humains ont repoussé les limites des connaissances chaque jour grâce aux efforts des informaticiens. Ces derniers sont sans cesse sollicités pour développer et améliorer les capacités de calcul des machines actuelles. Impactant divers domaines de recherche comme la finesse de résolution météorologique ou la simulation du cerveau, ces efforts permettent de traiter des données plus fines et plus importantes chaque jour.

Pour répondre à la demande croissante en puissance de calcul et en analyse de données, les architectures actuelles mettent en œuvre simultanément de nombreuses techniques de parallélisme. Les architectures deviennent de plus en plus complexes. Ces évolutions et améliorations sont utilisées dans les machines HPC (de l'anglais, High Performance Computing).

Différents paradigmes ont émergé dans la littérature pour construire de telles infrastructures de calcul à grande échelle. Les plus communes sont :

- Les clusters regroupant des nœuds de calculs, reliés entre eux par un réseau dédié [84];
- Les grilles informatiques, qui sont une collection de grappes géographiquement distantes les unes des autres [47];
- Les Clouds, qui sont des ressources informatiques abstraites utilisées comme des services à la demande. De telles infrastructures masquent des informations architecturales telles que la topologie réseau ou l'emplacement des nœuds pour les utilisateurs finaux [17];
- Finalement, les supercalculateurs sont de larges architectures haute performance avec un réseau haute vitesse dédié construit comme une seule machine, pour réduire la latence et le temps de calcul.

La taille et la complexité de tels systèmes ralentissent la gestion efficace et rapide de telles architectures. Toutes ces améliorations ne sont pas suffisantes pour répondre aux demandes croissantes des applications, en quête de plus de puissance de calcul par opération. Les constructeurs de supercalculateurs sont principalement axés sur la performance, toujours à la recherche de vitesse d'exécution plus élevée. Cette vision n'est plus viable avec une informatique haute performance allant vers l'Exascale, supposé grand consommateur d'énergie.

La prochaine étape vers le calcul haute performance à grande échelle est l'Exascale, soit atteindre 10^{18} opérations par secondes, un facteur 10 comparé à la meilleure machine publique actuelle listée dans le Top500 [76].

L'Exascale permettra aux prochaines générations d'affronter et de répondre à des problématiques inaccessibles actuellement telles que de meilleures réactions aux changements climatiques, une conception efficace des énergies renouvelables, une accélération de la guérison du cancer, une meilleure compréhension de l'univers et de beaucoup d'autres mystères. Construire et faire

fonctionner de manière permanente et efficace la première machine Exascale est un véritable défi.

Plusieurs challenges doivent être abordés à différents niveaux tels que [12] :

- Accroître le passage à l'échelle des applications qui devront faire face et exploiter une nouvelle ampleur de parallélisme;
- Faire face à une forte hétérogénéité des nœuds composés d'unités de calculs de différents types (CPU, GPU, MICs, etc) impliquant une grande portabilité des applications;
- Produire des méthodologies de programmation pour faciliter l'utilisation et l'extension des modèles de programmation actuels pour exploiter de telles machines et leurs hétérogénéités;
- Augmenter la mémoire et la capacité de stockage en parallèle des capacités de calculs;
- Une forte résilience aux erreurs, qui peut se produire à une plus grande échelle, inconnue jusqu'à présent;
- Fonctionner autour de l'enveloppe de 20 à 30 MW. En augmentant seulement le pic de puissance du premier et second supercalculateurs du Top500, il faudrait entre 150 MW et 340 MW pour construire une machine Exascale à partir de la même configuration, ce qui est intenable. En effet, il représente respectivement 1/7 et 1/3 de la puissance maximale pouvant produire une centrale nucléaire de 1000MW [42]. Ainsi, d'énormes efforts sur l'efficience énergétique doivent être faits.

La première machine Exascale est encore à venir, mais l'architecture actuelle des 10 machines les mieux classées du Top500 et l'architecture annoncée des premières machines Exascale [38] supportent l'idée qu'une machine Exascale devrait avoir une architecture proche de la suivante :

- Des nœuds de calculs et de stockages de données séparées;
- Des milliers de cœurs dans un nœud de calculs avec une mémoire locale plus grande et plus rapide que celle actuellement disponible dans les nœuds de calculs;
- Des centaines de milliers de nœuds composés de processeurs et d'accélérateurs hautement parallèles tels que les MIC (Many Integrated Cores) et les GPU (Graphical Processing Units), avec une énorme quantité de mémoire partagée à grande vitesse;
- Des milliers de nœuds de stockage de données atteignant des centaines de Pbytes cumulés;
- Un réseau très haut débit et dédié pour connecter étroitement tous les composants précédents.

Ces changements s'accompagnent de coûts, venant des technologies de pointe utilisées dans ces architectures mais aussi de leurs consommations d'énergie.

De tels systèmes à grande échelle mettent l'accent sur les problèmes présents mais qui ne sont pas l'objectif principal des chercheurs. L'Exascale implique de nombreuses problématiques comme la distribution de données, le développement et la réutilisation d'applications, la résilience et la consommation d'énergie. Cette dernière problématique est liée à toutes les précédentes. Par exemple, stocker et déplacer des données loin du nœud de calculs ou communiquer trop de données serait désastreux du point de vue de la consommation énergétique. Un autre exemple

est l'exécution de services tels que la résilience ou l'ordonnancement, où le choix d'un algorithme peut avoir un impact majeur sur l'énergie consommée. Cela fait de la problématique énergétique un enjeu majeur, centrale et critique.

La consommation d'énergie est une préoccupation croissante et à la confluence de toutes ces problématiques. En 2017, l'empreinte énergétique des systèmes informatiques dans le monde est estimée à environ 7% de la demande mondiale en électricité. Elle est également responsable de 2% des émissions mondiales de carbone [26]. Ces chiffres inquiétants ont des conséquences financières et environnementales directes sur les gestionnaires d'infrastructure, comme les fournisseurs de Cloud et les opérateurs de supercalculateurs. Avec la multiplication des dispositifs connectés par personne à travers le monde, la réduction de la consommation d'énergie des systèmes informatiques grande échelle est une étape obligatoire à franchir pour construire une société numérique durable. Pour atteindre une machine Exascale durable, les informaticiens doivent d'abord comprendre et être économes en énergie à tous les niveaux possibles sur les supercalculateurs actuels, de l'infrastructure utilisée à l'application exécutée.

La somme des pics de consommation d'énergie des cinq premiers supercalculateurs au cours du premier mois de cette thèse (Top500 datant de novembre 2015) était d'environ 50.512 MW, alors que la dernière mise à jour de la somme de la liste (Top500 datant de Novembre 2017) est d'environ $45,01$ MW. Une évolution et une prise de conscience de l'importance de la métrique du pic de consommation par les constructeurs de supercalculateurs peut être remarquée. Bien que cette métrique soit intéressante, elle ne suffit pas pour comprendre la consommation complète de l'ensemble de l'installation.

La puissance utilisée (P) à un temps donnée (t) par tout dispositif est composée de deux parties :

- P_{static} est la partie fixe qui est due à une perte progressive d'énergie des condensateurs chargés, habituellement appelés courant de fuite, présente dans tout dispositif composé de transistors.

- $P_{dynamic}$ est la partie variable, résultant de l'usage actif des composants :

$$P(t) = P_{static} + P_{dynamic}(t).$$

La puissance est la vitesse à laquelle le travail est effectué par le système étudié. Elle est exprimée en $Watts$. Les métriques liées à la puissance, telles que la puissance maximale ou la puissance moyenne, représentent le stress subit par les composants du système étudié. La consommation d'énergie E d'un composant dépend de sa consommation d'énergie P au cours du temps t. Pour un intervalle donné T_1 à T_2, avec $T_1 < T_2$, sa consommation énergétique est donnée par la formule suivante:

$$E(T_1, T_2) = \int_{T_1}^{T2} P(t)dt.$$

Ainsi, l'énergie, exprimée en $Joules$, est la quantité totale de travail effectué par un système sur une période donnée. Les métriques liées à l'énergie représentent le travail nécessaire pour accomplir une tâche ou un ensemble de tâches. Ces deux familles de métriques (liées à la puissance et à l'énergie) permettent d'évaluer l'impact d'une architecture ou d'une application du point de vue de la consommation. C'est la base de toutes les métriques liées à l'énergie.

Les installations informatiques, telles que les supercalculateurs et les centres de données grande échelle, sont incontestablement de grands consommateurs d'énergie dont l'efficacité énergétique doit être d'avantage améliorée. Plusieurs techniques ont été développées afin de réduire

la consommation électrique de telles installations. Pour faire face à cette préoccupation croissante, de nombreuses solutions ont été développées à plusieurs niveaux : infrastructure, matériel, intergiciel et application. Les exemples de leviers matériels sont le "Dynamic Voltage et Frequency Scaling" (DVFS) [101] ou encore l'allumage et l'extinction de composants [80]. Au niveau des intergiciels, les politiques de planification de tâches peuvent également être étiquetées comme des leviers [53]. Enfin, au niveau de l'application, la façon dont elle est implémentée a des effets importants sur sa consommation de puissance et d'énergie [1].

Il est donc urgent de considérer l'efficacité énergétique comme un enjeu majeur de nos installations informatiques modernes, d'où l'utilisation obligatoire de ces leviers. Ces derniers sont disponibles en grand nombre dans les centres de calculs grande échelle. En dépit de leurs gains potentiels, les utilisateurs et les administrateurs ne considèrent pas tous les leviers disponibles pour une meilleure efficacité énergétique. Néanmoins, l'utilisation de ces leviers, seuls mais surtout combinés, pourrait être compliquée et contre-productive.

Cette thèse étudie la découverte, l'évaluation et l'usage efficace en énergie des leviers disponibles dans les infrastructures de calculs grande échelle. La problématique de la modélisation et de l'usage d'un levier énergétique unique dans de vraies conditions d'utilisation est abordée. Nous répondons à la problématique de la combinaison de leviers, faisant partie de la même ou de différentes familles, et proposons des solutions génériques à l'usage dynamique de leviers combinés devant faire face à des contraintes variables ou fixées. Enfin, le monitoring, la combinaison, et l'analyse comparative générique de leviers combinés et l'extraction de connaissance sont étudiés.

Des précédents challenges, les objectifs suivants ont été fixés :

1. Fournir une méthodologie générique pour évaluer le gain potentiel d'un levier en tenant compte des coûts éventuels;
2. Fournir une méthodologie pour modéliser et utiliser un levier seul, tout en prenant en considération les multiples contraintes et coûts qui peuvent subvenir durant l'usage d'un levier;
3. Fournir un modèle abstrait qui permet de détecter, combiner et d'évaluer automatiquement les leviers existants;
4. Fournir une solution logiciel pour extraire une connaissance compréhensible depuis une étude multi-leviers.

Sur la base des objectifs précédemment définis, nous présentons dans cette thèse les contributions suivantes :

1. Nous proposons une définition de levier et d'acteur, ainsi qu'une première classification des leviers habituellement disponibles dans un centre de calcul;
2. Nous décrivons notre méthodologie pour évaluer et modéliser un levier et l'appliquons à l'allumage et l'extinction de machines sur des traces conséquentes de centres de calcul (de quelques semaines à quelques années);
3. Nous proposons Green Factory, un intergiciel fortement extensible qui permet la découverte, la combinaison et l'évaluation automatique des leviers tout en recueillant les métriques choisies pour construire une table de score, la table des leviers. Nous combinons des leviers de différentes familles en recueillant des métriques relatives à la puissance et à l'énergie consommée;

4. De la table des leviers précédemment définie et obtenue, nous décrivons la connaissance pouvant être extraite de cette table en utilisant des prédicats définis;
5. Nous proposons une première combinaison de leviers pour répondre à des contraintes relatives à l'énergie consommée pour une application HPC utilisée en production.

Le manuscrit est organisé comme suit :

- Le Chapitre 2 présente notre classification en quatre familles des leviers énergétiques les plus utilisés dans la littérature et comment ils affectent la consommation énergétique.
- Dans le Chapitre 3, nous proposons notre méthodologie pour détecter, étudier et évaluer les usages possibles d'un levier énergétique. Nous appliquons ensuite partiellement cette méthodologie sur les Thermo-Electric-Generators (TEG).

  L'application de cette méthodologie sur les TEG est issue de [130]. Elle est aussi appliquée dans [125, 129].
- Dans le Chapitre 4, nous appliquons entièrement la précédente méthodologie pour créer des acteurs génériques pour le levier d'allumage et d'extinction, qui constitue une solution intéressante à l'usage dynamique d'un centre de calculs grande échelle.

  Ce Chapitre 4 est dérivé de [121, 124, 123, 128].
- Le Chapitre 5 propose une combinaison de leviers : le nombre de threads utilisés, le nombre de processus utilisés ainsi que la version de codes utilisés pour une application spécifique. Cette étude souligne la variabilité apportée par une combinaison de leviers et propose des acteurs pour répondre à l'usage d'une telle combinaison de leviers.
- Le Chapitre 6 présente la construction de la table des leviers avec Green Factory, notre intergiciel qui découvre, combine et évalue automatiquement les leviers combinés de même que l'extraction de connaissances depuis la table des leviers.

  Ce Chapitre 6 est issue de [126, 127].
- Enfin, le Chapitre 7 présente les conclusions et perspectives de ces travaux, avec un rappel des contributions et résultats principaux.

Chapter 1

Introduction

Contents

Answering problematics and tackling the unknown have led to the exponential growth of computing needs. Humans have pushed boundaries of knowledge further every day, thanks to efforts of computer scientists to grow and improve computing capabilities. These improvements have impacted various research fields, from meteorology capabilities at very fine grain to brain simulation at the neural level, allowing finer and bigger data to be treated. To face growing computing needs, multiple techniques have been developed and implemented in the hardware to improve computing capabilities and throughput of architectures, by treating several data at the same time, or executing several tasks at the same time.

1.1 Short history of parallelism capabilities

Computer scientists tend to divide work among available resources where many calculations can be done concurrently and large problems can be divided into smaller ones. These smaller tasks can then be coordinated and solved at the same time.

In 1964, Cray's CDC 6600 is the first machine to use *Out-Of-Order* [28] execution, where processors choose instructions to execute from an analysis of the availability of input data and execution units. By processing next instructions that are ready to run immediately and do not depend on previous ones, the processor avoids idle periods.

In parallel, also in 1964, a concurrent to the CDC 6600, the IBM System/360 Model 91 proposes an architecture with *Pipelining*, where instructions are divided into sub-steps so the instructions can be executed partly at the same time, thus keeping the processor busy all the time [27].

In 1964, the CDC 6600 is the first machine to use a *Super-Scalar* processor, where multiple functional units receive work at the same time [24].

In 1966, ILLIAC IV is the first machine based on vectorial processor, following the first *Vectorial* processor designed in 1963, named SOLOMON. Thus, it is the first machine to have the status of "massively parallel machine" [99].

In 1993, Intel Paragon XP/S is the first machine to integrate multi-core computing CPU's, namely the i860/XP processor [41]. Multi-core consists in at least two independent processing units, in the same processor. Thus, it adds the possibility of executing multiple programs or multiple instructions from a same program, to increase the overall execution time.

From *OutOfOrder* to *Multiprocessing*, actual processors composing large scale computing facilities, such as supercomputers, implement all the previously exposed parallelism techniques, making it complicated to exploit every possible parallelism at the same time.

1.2 HPC facilities, a brief taxonomy

To answer the growing demand for computing power and data analysis, today's architectures implement all the previously exposed parallelism techniques, at the same time. Thus, they become more and more complex. These evolutions and improvements are used as High Performance Computing (HPC) facilities.

Different paradigms have emerged in the literature to build such large-scale computing facilities. The most common ones are:

- Clusters are collections of computing nodes, linked together with a dedicated network [84];
- Computing grids are a collection of clusters geographically distant from one another [47];
- Clouds are computing resources abstracted as on-demand services. Such infrastructures hide architectural information like network topology or node location to end users [17];
- Supercomputers are very large, high-performance architectures with a dedicated high-speed network built as a single machine, with a unique design to reduce latency and time to compute.

The size and the complexity of such systems slow the efficient and fast management of such architectures. All these improvements are still not enough to answer growing demands of applications, always seeking for more computing power per operations. Supercomputer builders are mainly focused on performance, always seeking for more and more performance, which is not sustainable anymore, towards Exascale computing.

1.3 The Exascale challenges and predicted architecture

The next milestone to very large scale computing capabilities is Exascale computing, reaching 10^{18} operations per seconds, 10 times the actual best public supercomputers listed in Top500 list [76], named *Sunway TaihuLight*. This supercomputer reaches 93TFlops/s for a peak power consumption of 15 MW (Mega Watt). Exascale will enable next generations to face and answer problematics that are unattainable to ours such as better reactions to climate changes, efficient design of cost efficient renewable energy, speeding up cancer curing, better understanding of the universe, and a lot of other mysteries that humanity still has to reveal. Building and running

permanently and efficiently the first Exascale machine is a real challenge. Several open challenges have to be faced at different levels such as [12]:

- Achieving ultra large scale scalability, with applications that will have to face and exploit a new magnitude of parallelism;
- Facing high heterogeneity, with nodes composed of computing units from various kinds (CPU, GPU, MICs, etc) implying high portability of applications;
- Providing programming methodology to ease the usage and extent of current programming models to exploit such machines and heterogeneity;
- Consistently increasing memory and storage, along with Exascale computing capabilities;
- Providing strong resiliency to errors, that may happen at a larger unseen scale, so far;
- Operating under the power envelope of 20 to 30 MW. From only scaling up the power peak of *Sunway TaihuLight* and *Tianhe-2*, respectively the actual first and second supercomputers from the Top500's list, it would require between 150MW to 340MW to build an Exascale machine from the same setup, which is untenable. Indeed, it represents respectively 1/7 and 1/3 of an average 1000 MW nuclear power plant at full throttle [42]. Thus, tremendous energy saving efforts have to be done.

The first Exascale machine is yet to come, but the actual architecture and setup of the top 10 public supercomputers from top500 and the announced architecture of machines towards Exascale [38] support the idea that an exascale machine should have an architecture close to the following one:

- Separated computing and data storage nodes;
- Thousands of computing cores in a computing node with larger and quicker than actual local memory;
- Hundred thousands of nodes composed of highly parallel CPUs and accelerators like MICs (Many Integrated Cores) and GPUs (Graphical Processing Units), with huge amount of high-speed shared memory;
- Thousands of memory nodes reaching hundreds of Pbytes cumulated;
- Very high speed and dedicated network to tightly connect all the previous components at multiple levels (intra and extra node).

This comes with a cost, either from state of the art technologies used in these architectures but also from their energy consumptions. Such large-scale systems put an emphasis on problematics that were present but not the main focus of researchers. Exascale implies a lot of problematics like data distribution, application development and re-usability, resilience, and energy consumption. This last problematic is linked to the previous ones. For instance, storing and moving data far from the computing node or communicating too much data will be disastrous from an energy consumption perspective. Another example is the execution of services like resiliency or scheduling, where choosing an energy consuming algorithm for this given service

can have major impact on the overall consumed energy. That brings energy problematics as a major, central and critical focus.

Energy consumption is a growing concern and on the verge of all these problematics. In 2017, the energy footprint of IT systems around the world is estimated around 7% of the global electricity demand. It also is responsible of 2% of global carbon emission [26]. This worrying consumption and carbon footprint have direct financial and environmental consequences on facility managers, such as Cloud providers and supercomputer operators. With the multiplication of connected devices per person around the world, reducing the energy consumption of large-scale computing systems is a mandatory step to address in order to build a sustainable digital society. To reach a sustainable Exascale machine, computer scientists first have to be energy efficient at every possible level on nowadays supercomputers, from infrastructure to application level.

The sum of power peak consumption of top 5 supercomputers during the first month of this thesis (Top500 November 2015) was around 50.512 MW, while the last update of the list's sum (Top500 November 2017) is around 45.01 MW. An evolution and a realization of the importance of the power peak metric from supercomputer builders can be noticed. Although this metric is relevant, it is not enough to estimate the complete consumption of the whole facility.

1.4 Energy efficiency in large-scale computing facilities

From large-scale facilities responding to high demand to connected end to end devices in our pockets, IT systems and services are nowadays everywhere. These systems are composed of various components to connect you to others (network related), to understand and answer quickly your requests (computing related), to save and store all your data (storage related), all consuming large amount of energy. Several metrics have been proposed and discussed for evaluating the energy efficiency of computing facilities. In this section, we propose an overview and discussion of existing metrics.

1.4.1 Power and Energy

The power used (P) at a given time (t) by any system is composed of two main parts:

- P_{static} is the fixed part that is due to a gradual loss of energy from charged capacitors, usually called leakage current, present in any transistor-composed device;
- $P_{dynamic}$ is the variable part, resulting from the active usage of components, thus:

$$P(t) = P_{static} + P_{dynamic}(t).$$

The power is the rate at which the work is performed by the studied system. It is expressed in $Watts$. Power-related metrics, like max power or average power, usually represent the stress put on components of the studied system. The energy consumption E of a component depends on its power usage P over time t. For a given time interval T_1 to T_2, with $T_1 < T_2$, its energy consumption is given by:

$$E(T_1, T_2) = \int_{T_1}^{T2} P(t)dt.$$

Thus, the energy, which is expressed in *Joules*, is the total amount of work performed by a system over a time period. Energy-related metrics represent the necessary work to complete a task or set of tasks. These two metric families (energy and power related) are the de-facto metrics to evaluate the impact of an architecture or application from its consumption perspective. It is the basis of all energy-related metrics.

1.4.2 Evaluation of energy efficiency

Energy efficiency has been recognized as a major problematic of computer science. A lot of metrics related to the energy efficiency of large-scale computing systems has emerged in the literature.

PUE

Power usage effectiveness (PUE) is the most widely used metric for computing facilities for an overall clue about energy efficiency of a facility. PUE is a ratio that represents how efficiently a computing facility uses energy:

$$PUE = \frac{TotalFacilityEnergy}{ITEquipementEnergy}.$$

As suggested by its name, the *TotalFacilityEnergy* represents the sum of all the energy consumed by all equipments in the facility. Thus, IT equipments, cooling components, and other infrastructure-related elements are included in that factor. The *ITEquipmentEnergy* only considers the energy consumed by computing-related components such as computing nodes, networking devices, and storage units.

Although it is named "Power Usage Effectiveness", it actually measures the energy usage of a computing facility. PUE was introduced in 2006 and promoted by the Green Grid, a non-profit organization of IT professionals. It has became the most commonly used metric for reporting the energy efficiency of computing facilities.

The minimum possible score is 1.0 when $TotalFacilityEnergy$ and $ITEquipementEnergy$ are equal, meaning that all the power provided is used only by IT equipment. So, the closer the value is to 1.0, the more efficient the facility is. A PUE close to such a value indicates that a greater portion of the power going to a facility is used for its computing and data treatment. For instance, a PUE of 1.5 means that a facility needs half as much additional power as needed for the IT equipment to operate, whereas a PUE of 2.0 means that a facility needs as much additional power for non-IT elements as it does for IT hardware.

Despite being a standard, PUE has several limitations:

- A facility might lower its PUE when being under high load. A high IT load increases power consumption of IT systems but cooling systems do not always scale accordingly, thus raising the $ITEquipementEnergy$ without scaling up the $TotalFacilityEnergy$ factor accordingly, resulting in a better PUE.
- The power dedicated to cooling systems will depend on location, duration and period of the evaluation. Thus, a same facility computing the PUE at two different periods, cold and hot weather, might output a completely different PUE.

Thus, a lot of other metrics have emerged through time to perfect the flaws of PUE and compute energy efficiency differently.

Compute Power Efficiency

CPE is a measure of the efficiency of a facility. Each Joule consumed in a computing facility is not used to do useful work. Components consume energy even in idle state, while others use it to perform computations, thus useful work.

$$CPE = \frac{ITEquipementUtilization}{PUE}.$$

$ITEquipementUtilization$ represents the fraction of energy usefully used by components. Although being a good improvement to the PUE metric, it is quite complicated to know at every second and for every component if the energy is being used usefully. Thus, it is the responsibility of the user of this metric to determine if a given device is using energy properly.

Performance per Watt

Performance per Watt is a measure of the energy efficiency for a particular component, architecture or facility. For every $Watt$ consumed, it represents the amount of computation that can be executed. A common measure is the FLOPS (Floating Point Operations Per Second) per $Watt$, noted $FLOPS/Watt$. To fairly compare one architecture to the other, the same high computing intensity benchmark should be implemented, such as LINPACK, used to rank all supercomputers for the Top500.

Performance per Watt is the chosen metric for the Green500 list [46]. This list ranks the same public supercomputers than the Top500 list, but instead of peak computing power delivered (Flop/s), it focuses on an energy efficient perspective with performance per Watt (Flop/Watt) as the main ranking metric.

Thermal Dissipated Power

Due to resistance in electronic circuits and the (very) high density of computing components such as CPUs, the energy dissipated in form of heat is almost equal to all the energy consumed. Thermal Dissipate Power is the maximum amount of heat generated by a component that a cooling system has to dissipate. Thus, it also is a good hint on the maximal power consumption of a component. An average CPU that populates facilities (Xeon E5-2690) has a TDP around 115 W, while the TDP of the Xeon Phi 7290, which populates a lot of supercomputers, is around 220 W.

Recently, manufacturers have been pointed out to evaluate TDP on non-intensive workloads, thus not stressing the CPU at its rupture point as this metric is supposed to represent [61]. Thus, Intel introduced a new metric called *Scenario Design Power* (SDP) that is supposed to represent the mainstream usages of such devices.

Reusable and carbon dioxide related metrics

Finally, a few metrics are related to the impact that the facility has on the environment. The Green efficiency coefficient (GEC) is a measure of energy that comes from renewable sources that is used by components of the supercomputer. It is defined as:

$$GEC = \frac{GreenEnergyConsumed}{TotalEnergyConsumed}.$$

This metric can be used to evaluate the environment friendly nature of a computing facility. However, this metric does not reveal the carbon emissions of a given facility.

CUE, for carbon usage effectiveness, represents the carbon dioxide emission in the environment:

$$CUE = \frac{EmissionCO_2}{ITEquipementEnergy},$$

where $EmissionCO_2$ represents the total dioxide emissions from total energy absorbed by components of a computing facility. It includes all green house gases such as CO_2 and CH_4, carbone dioxide and methane respectively, that are emitted in the atmosphere, for a whole year.

Finally, the water usage effectiveness (WUE) is one of the few metrics that takes into account the water used in a facility:

$$WUE = \frac{WaterUsedAnnualy}{ITEquipementEnergy}.$$

The previously exposed and described metrics can be expressed as a ratio between a given important focus and energy consumed by IT equipment or the total energy consumption of the complete facility. PUE is the most popular and widely spread metric across computing facility vendors. It is usually used as a sale argument when it comes to new facilities. However, its usability and computation are restricted. Important amount of information can be hidden. For instance, equipment specifications and conditions of evaluation (duration, location, effective load) are not expressed in this metric. As a direct consequence, most of research works focus on the computing units themselves to measure the energy efficiency of an infrastructure and base their analysis on energy and power related metrics.

Computing facilities such as data centers and supercomputers became through years more and more complex eco-systems. For instance, water is used more and more often in such facilities, from cooling components to humidification. Thus, using a combination of metrics to gather different focuses is necessary. With growing energy consumption and worrying focuses such as carbon emission and water outage, it is mandatory to use metrics related to such focuses like WUE, CUE and GEC for water, carbon emission and green energy usage, respectively. These metrics give good large-scale clues but not coarse-grain ones to detect and learn how to use components and various possibilities of usage of different components.

1.5 Research problems and objectives

Computing facilities, such as supercomputers and large-scale data centers, are major energy consumers whose energy efficiency is still to improve. Several techniques have been developed in order to lower the electrical consumption of such facilities. To face this growing concern, many solutions have been developed at multiple levels of computing facilities: infrastructure, hardware, middleware, and application. Examples of hardware leverages are Dynamic Voltage and Frequency Scaling (DVFS) [101] or Shutdown Techniques [80]. At the middleware level, energy-efficient policies for task, jobs and resource managers can also be labeled as leverages [53]. Last, at the application level, the way an application is implemented has important effects on its energy and power consumption [1].

It is urgent to embrace energy efficiency as a major concern of our modern computing facilities. Using these leverages is mandatory to improve energy efficiency. A lot of leverages are

available on large-scale computing centers. In spite of their potential gains, users and administrators do not always consider using all available leverages to improve energy efficiency. However, using these techniques, either alone or combined, could be complicated and counterproductive if they were not wisely used.

This thesis investigates the discovery, evaluation and energy efficient usage of leverages available on a large-scale computing facility. The problematic of modeling and using a single energy leverage on real-life scenarios is tackled. We also confront the problematic of combined energy leverages, from same or different families, and tackle the issue of a generic solution to the dynamic usage of combined leverages answering various fixed or dynamic constraints. Finally, the generic combination, monitoring and benchmarking of combined leverages and extraction of knowledge from this combination is investigated.

From the previous challenges, the following objectives have been set:

1. Provide a generic methodology to evaluate the potential gain of a leverage by taking into account possible costs;

2. Provide a methodology to model and use a single leverage while taking into account multiple constraints and costs that could happen during the usage of this leverage;

3. Provide an abstract model that permits to automatically detect, combine and benchmark existing leverages;

4. Build a solution and software framework to extract humanly understandable knowledge about combinations of leverages.

1.6 Contributions

Based on the objectives previously defined, we present in the thesis the following contributions:

1. We propose a definition of leverages and actors, and a first classification of usually available leverages in a computing facility;

2. We describe our methodology to evaluate and model a leverage, and apply it to the shutdown and wake-up leverages on consequent traces (from weeks to years of traces) of computing facilities;

3. We propose Green Factory, an expendable framework that permits to automatically discover, combine and benchmark leverages while gathering chosen metrics to construct a scoring table, the table of leverages. We combine leverages from different families while gathering energy, power and performance metrics;

4. From the previously obtained and defined table of leverage, we describe a couple of representative applications showing understandable knowledge that could be extracted from this table of leverages;

5. We propose a selected usage of combined leverages to answer energy and power related constraints for an HPC application used in production.

1.7 Organization of the manuscript

The remainder of the manuscript is organized as follows.

- Chapter 2 presents a classification in four families of the most used leverages in the literature, together with their relative literature and how they affect energy consumption.
- In Chapter 3, we propose a methodology to detect, study and evaluate the possible usage of a leverage as an energy and power leverage. We then apply part of the methodology to the Thermo-Electric-Generator (TEG) leverage.

  The application of the proposed methodology on Thermo-Electric-Generators is derived from [130]:

  – Issam Raïs, Laurent Lefèvre, Anne Benoit, Anne-Cécile Orgerie. "An Analysis of the Feasibility of Energy Harvesting with Thermoelectric Generators on Petascale and Exascale Systems." In *Workshop Optimization of Energy Efficient HPC and Distributed Systems (OPTIM 2016), in conjunction with the 2016 International Conference on High Performance Computing and Simulation (HPCS 2016)*, 2016.

  This methodology has also been used to study other leverages [125, 129, 122]:

  – João Vicente Ferreira Lima, Issam Raïs, Laurent Lefèvre, Thierry Gautier. "Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms" In *The International Journal of High Performance Computing Applications* 2018

  – João Lima, Issam Raïs, Laurent Lefèvre, Thierry Gautier. "Performance and Energy Analysis of OpenMP Runtime Systems with Dense Linear Algebra Algorithms." In *SBAC-PAD 2017 Workshops, the International Symposium on Computer Architecture and High Performance Computing*, 2017.

  – Pierre-François Dutot, Yiannis Georgiou, David Glesser, Laurent Lefèvre, Millian Poquet, Issam Raïs. "Towards Energy Budget Control in HPC." In *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2017.

- In Chapter 4, we fully apply the previously exposed methodology to create generic actors and usage of these actors for Shutdown, a leverage that constitutes an relevant solution to the dynamic usage of large-scale computing facilities.

  Chapter 4 is derived from [121, 124, 123, 128]:

  – Issam Raïs, Anne-Cécile Orgerie, Martin Quinson, Laurent Lefèvre. "Quantifying the Impact of Shutdown Techniques for Energy-Efficient Data Centers." In *Concurrency and Computation: Practice and Experience*, 2018.

  – Anne Benoit, Laurent Lefèvre, Issam Rais and Anne-Cécile Orgerie. "Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints." In *International Journal of High Performance Computing Applications*, 2017.

  – Anne Benoit, Laurent Lefèvre, Anne-Cécile Orgerie, Issam Raïs. "Shutdown Policies with Power Capping for Large Scale Computing Systems." In *Euro-Par: International European Conference on Parallel and Distributed Computing*, 2017.

  - Issam Rais, Anne-Cécile Orgerie, and Martin Quinson. "Impact of Shutdown Techniques for Energy-Efficient Cloud Data Centers." In *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2016.

- Chapter 5 proposes a combination of multiple leverages: number of threads, number of processes and code version, for a specific type of application to underline the variability proposed by the combined leverages. It also proposes actors to answer the usage of such combined leverages.

- Chapter 6 presents the construction of the table of leverages with Green Factory: the Automatic Benchmark and Discovery of Energy Leverages, our framework that automatically discovers, combines and benchmarks leverages along with the extraction of understandable knowledge.

  Chapter 6 is derived from [126, 127]:

  - Issam Rais, Laurent Lefevre, Anne-Cécile Orgerie, Anne Benoit. "Exploiting the Table of Energy and Power Leverages" In *ICA3PP 2018 : 18th International Conference on Algorithms and Architectures for Parallel Processing*, Guangzhou, China, November 15-17, 2018
  - Issam Raïs, Mathilde Boutigny, Laurent Lefèvre, Anne-Cécile Orgerie, Anne Benoit. "Building the Table of Energy and Power Leverages for Energy Efficient Large Scale Systems" In *HPCS2018 : IEEE The 16th Annual Meeting of International Conference on High Performance Computing & Simulation*, 2018.

- Finally, Chapter 7 presents conclusions and perspectives, with a summary of main contributions and findings.

Chapter 2

Leverages: state of the art

Contents

The elements composing a computing facility are not all used at full throttle all at the same time, thus necessitating modulation of their activity on the fly. To answer that possible variability on different possible usages, a lot of techniques having different states of usage exists with various focuses.

We call these techniques *Leverages*.

Definition 1 *A* leverage L *is a triplet* $L = (S, s_c, f_s)$*, where* $S = \{s_0, s_1, \ldots, s_n\}$ *is the set of available valid states of* L*,* s_c *is the current state of* L*, and* f_s *is a function to update the current state to a new state* $s'_c \in S$*.*

Energy consumption is currently recognized as a major issue. This is also true on the computing level where IT systems and connected devices per person are multiplied. Thus, our overall consumption of energy of our IT devices keep rising. To deal with this growing concern, from data-center building and management to the end connected device connected in our pocket, such leverages had been developed to reduce / regulate directly or indirectly energy consumption.

An energy or power leverage is a leverage that has a high impact on the power or energy consumption of a device through its various states or through the modification of its current state. Switching from one state to another can have a cost in terms of time and energy.

Thus using these leverages at the right moment at their right state without deteriorating the benefit of other leverages while respecting user constraints is a real challenge. In this thesis, we recognize four families of leverages: infrastructure, hardware, middleware, and application.

2.1 Infrastructure leverages

In this section, leverages from the infrastructure family are developed. These leverages are relative to the scope of the elements physically composing the data center or supercomputer.

2.1.1 Cooling systems

Servers composing supercomputers give off almost every Joule consumed as heat. This generated heat could be dangerous to elements composing the facility if not well extracted or recycled. Various ways of extracting this heat have been developed through the years.

- Air cooled systems, which is the most used state of this leverage in the majority of existing supercomputer. Here, server racks are arranged into cold and hot aisles. In cold aisles, the front sides of the server racks face each other. The aisle provides the cool intake air to each server. Cold air is generated, usually from the floor, in cool aisles by the CRAC unit (Computer Room Air Conditioner units). In hot aisles, the rear sides of the racks face the aisles. Hot air is captured and returned, in a closed loop, to the CRAC input.

- Water-cooling systems are usually made of direct on-chip cooling. It consists in applying water pipes on high density elements rejecting important amount of heat (like computing units).

- Direct fluid immersion systems are immersing servers in liquids that can transport high quantity of heat, such as water, oil or combined solutions.

- Free cooling is using the tempered air to cool the data center.

Usually, these states are atomic in a supercomputer, in the way that only one of these solutions will be chosen during the setup of the supercomputer.

From an energy efficient perspective, it is clear that the cooling system, whatever choice is made, has a great influence on the overall consumption of the complete facility. Literature evaluates the part of the energy consumption of cooling systems to be between 30% to 50% of the complete energy consumption of a facility [32, 83].

2.1.2 Recycled energy: eligible harvesters

New ways to generate or at least recycle energy have to be explored. Despite technological and software breakthrough to increase computational capabilities while reducing energy consumption, it became a real limiting factor for large-scale computing centers. This is especially true for supercomputers reaching petascale performance and aiming for exascale. Nowadays, sustained one megaWatt per year costs around one million dollars per year for a datacenter usage [39]. None of this lost energy is reversed in a direct usable electricity for a direct re-use. Energy harvesting is the process of extracting energy from surrounding environment [22]. Even if this recovered energy concerns a few Joules, it is recycled from a previous consumption, and it is thus increasing the energy efficiency. Every recycled Joule is a non produced one, thus avoiding CO_2 generation.

Several energy harvesting solutions exist. Among them, photovoltaics is the one in vogue: it aims at transforming solar beams into electricity [71]. Piezoelectric generators are built to recycle pressure expressed on a component and generate an amount of energy from the distortion of the component [69]. Thermoelectricity aims at recycling wasted energy under heat form. Under a delta temperature, a thermoelectric generator converts lost energy from heat to electricity form [100]. Pyroelectricity follows the same principle but at higher delta temperature [87].

Thermo-electricity was discovered during the 19th century by Thomas Johann Seebeck. It is a physical phenomenon where a thermal delta seen through a component results in an electrical current [108]. A Thermo Electric Generator (TEG) can be used as a generator (when it is exposed to a delta temperature) or as a cooler (when a current passes through it). Here, we focus on the generator feature.

For years now, TEGs are used on spacial scientific programs like Voyager. TEGs are used to provide energy in space where solar panels are too far from the sun to generate energy. TEGs are not auto sufficient. Heat generated by plutonium isotope and cold space temperature (2 or 3 degrees above absolute zero) create a suitable environment for TEG power generation [10].

Renewable energy are still not as effective as fossil or nuclear generators, in their efficiency. Although, it will be exploring the feasibility of such a leverage at the Exascale level can reveal its relevancy, especially in a context where every recovered Joules could be critical for the overall energy consumption.

2.1.3 Computing units

The core elements of computing facilities are the wide range of components, populating the nodes, that perform computations. We call Computing Unit (CU) all ways that a data-center or supercomputer have to compute a given task. The most widely spread are:

- CPU: for central processing unit is the core element of a node. It is the component that carries instructions of a program.
- GPU: for Graphical Processing Unit, originally for graphical processing and rendering, they turn out to be very efficient for embarrassingly parallel computations.
- MIC: for Many Integrated Cores, is a hybrid between CPU and GPU, where cores have the same architecture that general purpose bootable CPUs ones, but combine a lot of small cores, like in GPUs.
- ASIC: for Application Specific Integrated Circuit, is a customized circuit built specifically to answer a specific task.

- FPGA: for Field Programmable Gate Array, can be seen as a reconfigurable ASIC. In fact, it is a dedicated circuit for an application but that can be reconfigured.

As computing nodes can be a combination of these CUs, this leverage could be split into two families: bootable (with CPU and MIC) and accelerators CUs (with GPU, ASIC and FPGA). The internal current state of such families is then the chosen architecture and model vendor for the chosen family state.

Each one of these previous computing integrated circuits has its own field of action. Moving towards Exascale means being able to run a wide range of applications, thus ASIC and FPGA look inappropriate. Heterogeneity between general purpose CPUs bounded with accelerators like GPUs and MICs seems to be the right compromise. Their performance per Watt is different from one application to the other. Thus, choosing the wrong CU for a given task could lead to a waste of energy [5].

2.1.4 Memory

Because computing is done on large amount of data, memory also has a high impact on energy consumption. In [32], authors evaluated the impact of memory on energy consumption on a node level. Depending on the usage, it varies from 19% to 48% of the total energy consumed, on a node with a multi-core Intel Xeon architecture and on a node with a dual core Intel Atom, respectively. Memory rises in various forms and at various points of the architecture: from small sized very fast cache memory bounded to the CPU, to very large slow memory such as hard drives. Thus, answering memory efficiency is one of the important challenges to build a sustainable exascale system.

Nowadays current universal memory technology, DRAM for Dynamic Random-Access Memory, consists in a random access semiconductor memory. Each bit of data is stored in a separate capacitor within a circuit. It is a Volatile Memory (VM) as the electric charge on the capacitors leaks off gradually. Thus, a periodic refresh is needed to rewrite the data in the capacitors, restoring them to their original charge. Otherwise, due to the gradual leak, the data on the chip would be lost. Thus, it is a dynamic memory as opposed to Static Random-Access Memory (SRAM) that is Non Volatile Memory (NVM). DRAM is usually used for big sized low speed memory like RAM, when SRAM is used for high speed small size memory like cache.

Due to the high cost of SRAM and low efficiency of DRAM, memory manufacturers will have to push the boundaries of actual memory capacity, energy efficiency, and performance to reach a sustainable exascale system. The strongest contender to meet these requirements and to replace the DRAM is the 3D XPoint (3D Cross point), a non-volatile memory (NVM) technology by Intel and Micron Technology. It consists in a stackable cross-gridded data access array capable of keeping the state of the information with no refreshments needed, even when turned off. Thus, it combines the advantages of both SRAM and DRAM [19].

Memory components and computing components do not progress at the same speed. If this trend is not reversed, future exascale systems may have a dramatical reduction of memory per core with a direct effect of increasing communication and lowering computational efficiencies and thus lowering the energy efficiency.

2.2 Hardware leverages

In this section, leverages from the hardware family are exposed. These leverages are relative to the scope of available hardware techniques composing a computing node.

2.2.1 Sleep states

To dynamically answer the changing load of a data-center or supercomputer, nodes and computing units possess sleep states. It consists in turning on and off resources depending on platform usage.

ACPI, the Advanced Configuration and Power Interface [62], is a widely used open standard that operating systems use to configure components and to perform power management. It permits to put unused components to sleep through defined interfaces. Because it is not a requirement, all of the possible states defined in this specification are not implemented in all architectures. We focus on the ACPI sleeping states. They consist in various types of node configurations including different shutdown approaches and protocols. According to the ACPI specification, there are 5 possible sleeping states:

- S1 (weak): this is a low wake latency sleeping. No system context is lost and no hardware is turned off in this state.
- S2 (weak): Similar to S1, except that CPU and system cache is lost.
- S3: low wake latency sleeping state where all context is lost except system memory, i.e., CPU, caches, and chip set context are lost. Hardware restores CPU and L2 configurations and maintains memory context.
- S4: Lowest power but longest wake latency where only platform context is maintained.
- S5: System shutdown state. Similar to S4, except that the OS does not save any context.

Given this theoretical specification, we focus on the Linux implementation of this system power management. The available sleep states on the Linux kernel are [64]:

- S0 or "Suspend to Idle": freezing user space and putting all I/O devices into low-power states.
- S1 or "Standby / Power-On Suspend": same as S0, adding the fact that non boot CPUs are put in offline mode and all low-level systems functions are suspended during transitions into this state. The CPU retains power meaning operating state is lost, so the system easily starts up again where it left off.
- S3 or "Suspend-to-RAM": Everything in the system is put into low power state mode. System and device state is saved and kept in memory (RAM).
- S4 or "Suspend-to-disk": Like S3, adding a final step of writing memory contents to disk.

On the top of our knowledge, most of facility servers do not implement or allow S3 (Suspend-to-RAM) sleep state, because of numerous errors when resuming (especially errors due to network connections with Myrinet or Ethernet protocols). Typically, only S0, S4, and S5 (not described as a linux sleep state in the kernel, but S5 corresponds to shutting down the system) are available for operational use. Thus, sleep states is a relevant leverage to deal with idle periods seen on nodes and can provide energy savings by reducing the static consumption of the nodes (P_{Static}). However, previously mentioned states are not transition free. Indeed, they have a cost in time and in energy, to go from one state to the other. This leverage has to be applied carefully to avoid counterproductive effects on energy efficiency.

2.2.2 DVFS

From smartphones to supercomputer nodes, DVFS, for Dynamic Voltage and Frequency Scaling, raised as a popular leverage to modulate their working frequency and thus improve the energy efficiency. DVFS is nowadays commonly available on recent HPC nodes present in supercomputers, such as the overly dominating CPU on top 500's list: the intel Xeon e5-2690. This CPU has a range of frequency from $2.90GHz$ to $3.80GHz$.

DVFS is a leverage that modulates the voltage and/or frequency of a computing system. Dynamic power ($P_{Dynamic}$) of CMOS circuits is a function of voltage (V) and frequency (f) as:

$$P_{Dynamic} = C * f * V^2,$$

where C is the capacitance, a measure of how long it takes for a given current to produce a given voltage change. As it is defined here, $P_{Dynamic}$ is dominated by the V^2 factor. Hence, by reducing the frequency, which is highly correlated to the voltage, energy can be saved.

Despite its popularity, the DVFS has several limitations [9, 67, 77]:

- It only acts on the dynamic part of power consumption ($P_{Dynamic}$), although for supercomputer nodes, the static part (P_{Static}) represents a big part of the consumption;
- It reduces the performance as it may increase execution time and lead to missed deadlines;
- Voltage transition requires time, around tens of microseconds, and thus implies energy consumption;
- Due to the increase in leakage energy with high frequencies, constructors tend to build multi-core architectures with lower ranges of frequencies;
- Power usage may raise the temperature, thus, increasing the voltage or frequency may raise the system power demand, faster than the previous formula indicates.

In summary, the DVFS leverage can provide energy savings. These savings can be done during the dynamic part of the consumption of a node, thus on non-idle periods. However, because of its several limitations, it has to be applied carefully.

2.2.3 Simultaneous multi-threading

Simultaneous multi-threading (SMT) or hyper-threading for the Intel technologies, is a technique to improve the overall efficiency of CPUs with hardware multi-threading at the core level. In other words, multiple threads can run concurrently in a unique core of a given CPU. Even when running on the same core, the threads are separated from a context perspective, from one another. It allows multiple independent threads to keep the resources provided by processor busy all the time (like avoiding bubbles in the pipelining process). SMT can be seen as a multiple preemptive thread execution. It permits to reveal and extract additional instruction-level parallelism from multiple threads [72].

Although SMT seems to be a good way of increasing the performance, it has several critical drawbacks such as a danger of surcharging a core, where the CPU will spend all its time switching context between threads or processes running concurrently. Also, the limit concerning the number of possible threads per core is not well defined, even from a manufacturer perspective, because it highly depends on how the threads are using the core capabilities.

Thus, this leverage could highly deteriorate performance and energy efficiency of an application. These drawbacks push the HPC application developers not to use such a leverage, by usually allocating only one thread per core, because of its high variability concerning performance metrics. This leverage is to use with great parsimony.

2.3 Middleware leverages

In this section, leverages from the middleware family are developed. These leverages are in the scope of available techniques helping the usage and setup of the supercomputer from a software point of view.

2.3.1 Compiler

The compiler is the program that converts the code of the developer to binary understandable by the machine through multiple passes.

Multiple compilers could be used to generate a given binary from a given source code. GCC, The Gnu Compiler Collection, is the most common one. It is a front-end for multiple languages developed and open source since 1987. Clang is an open source compiler for multiple languages developed since 2005. ICC is the Inter C++ Compiler and it is proprietary. A given source code compiled on a given machine with these three compilers is unlikely to generate the same binary file. Thus, a variability from a performance and/or energy consumption metric is inevitable.

On top of our knowledge, no paper in the literature crosses the problematics of compilation with energy efficiency, and focuses directly as a choice to generate various binary that can have various impact on energy consumption. Most of the papers tackling these problematics use DVFS and dynamic compilation to manage energy and performance trade-off, as for instance in [63, 115], where authors developed a dynamic compilation phase where they automatically detect sections of codes where the frequency could be reduced to gain energy. It can be summed up as dynamic compiler techniques for microprocessor voltage and frequency control.

2.3.2 OpenMP runtime: tasks

OpenMP is the most widely used implementation of multi-threading. It is a method of parallelizing where a master thread forks a specified number of slave threads. The work is then equally divided among them. The role of the runtime is mainly to create and allocate threads to different processors. There are various OpenMP implementations of the OpenMP API such as libGOMP, libOMP and XKaapi.

The differences raised here are mainly focused on task models, which is an emerging solution to answer multi-threading. OpenMP dependent task model allows us to define dependencies between tasks using declaration of accesses to memory with *in*, *out*, or *inout*. Two tasks are independent (or concurrent) if and only if they do not violate the data dependencies of a reference sequential execution order.

libGOMP is the OpenMP runtime that comes with the GCC compiler. Since version 4.9, it implements dependent tasks. Dependencies between tasks are computed thanks to a hash table that maps data to the last task writing data. Ready tasks are pushed into several scheduling deques. The main deque stores all the tasks generated by the threads of a parallel region. Tasks are inserted after the position of their parent tasks in order to keep an order close to the sequential execution order. Because threads share the main deque, serialization of operations is

guaranteed by a pthread mutex, which is a bottleneck for scalability. To avoid overhead in task creation, libGOMP implements a task throttling algorithm that serializes task creation when the number of pending tasks is greater than a threshold.

libOMP manages dependencies in the same way that libGOMP by using a hash table. Nevertheless, memory allocation during task creation relies on a fast thread memory allocator. libOMP task scheduling is inspired from the Cilk almost non blocking work stealing algorithm [49] but where deque operations are serialized using locks. It implies distributed deques management with high throughput of deque operations. libOMP also implements a task throttling algorithm by using bounded size deque.

XKaapi [52] is a task library for multi-CPU and multi-GPU architectures, which provides binary compatible library with libGOMP [18]. Task scheduling is based on the almost non-blocking work stealing algorithm from Cilk [49] with an extension to combine steal requests in order to reduce overhead in stealing [105]. The XKaapi-based OpenMP runtime also has support to many OpenMP extensions such as task affinity [112] that allows us to schedule tasks on NUMA architectures, and to increase performance by reducing memory transfer and thus memory energy consumption.

With most of the architectures becoming more and more parallel at the node level with more and more computing cores, it is necessary to fully exploit this availability. From the literature, the OpenMP runtime leverage is an appealing candidate to better energy efficiency. Although at our knowledge, there is no work in the literature comparing the efficiency of runtimes on an energy consumption level.

2.3.3 Shutdown policies

Here, we argue about the various algorithms or policies that the literature offers to deal with shutdown techniques. Pioneering work on studying the energy-related impacts of shutdown techniques started in 2001 [23, 85]. These early works did not consider any transition cost for switching between the two state of this leverage: on and off. These works nonetheless showed the potential impact of such techniques. Demaine et al. examine the power minimization problem where the objective is to minimize the total transition costs plus the total time spent in the active state [35]. They develop a $(1 + 2\alpha)$-approximation algorithm, with α the transition cost.

However, the parameters considered for the transition costs of such policies highly vary across the literature. Gandhi et al. take into account the energy cost of switching on servers (no switching off cost as it is estimated to be negligible in comparison with the switching on cost) [51]. This energy cost is assumed to be equal to the transition time multiplied by the power consumption while in the on state while Lin et al. take into account the energy used for the transition and the delay in migrating connections or data [73].

Shutdown techniques do not only impact energy consumption, they also affect temperature and consequently cooling systems [119]. They can also be used for limiting the dark silicon effect, i.e., the under-utilization of the device integration capacity due to power and temperature effects [44]. This issue has lead to the introduction of user-specified, dynamic, hardware-enforced processor power bounds, as for the Intel's Sandy Bridge family of processors for instance [89]. At a facility level, it translates into power budgeting, where the total power budget is partitioned among the cooling and computing units, and where the cooling power has to be sufficient to extract the heat of the computing power. Given the computing power budget, Zhan et al. propose an optimal computing budgeting technique based on knapsack-solving algorithms to determine the power limit for the individual servers [118].

Thus, even on a leverage that is well studied, like shutdown techniques, high impact on a data-center, from various point of views like energy or temperature, can be witnessed. It is thus necessary to create such policies that answer the need for the leverage to switch state in order to meet an objective. Due to the non-proportionality of actual electrical components [111] and due to their high switch cost, it became necessary to use such shutdown policies.

2.3.4 Scheduling policies

Scheduling policies represent the way the jobs are delivered to the machines composing the facility. Many papers focus on controlling the power and energy consumption through such leverage that could have different focus or objective, either to minimize, or maximize. It could depend on various objective of the belligerents (user, owner, electrical provider) of the facility. For example, the owner can have the objective of maximizing the usage of its machines while a green user can have the objective of minimizing the energy consumption of its runs. Thus, a policy could maximize the throughput of computation, thus time spent executing the set of job, or the overall energy spent to do the complete set. Here, we focus on energy and power related scheduling policies.

Patki et al. [81] argue that thanks to the control of power consumption, one can buy a bigger cluster for the same annual price. The objective is to control the final energy cost of the cluster while keeping good performances even under power constraints (or *power capping*). In [45], authors defend *MaxJobPerf*, a scheduling policy that maximizes the throughput of a fixed set of jobs while maintaining a strong power capping constraint. Gholkar et al. in [54] presented a 2-level hierarchical power capping solution based on intel's RAPL technology, which is adapted to guarantee a local power capping.

An energy budget policy has been studied in [43]. The algorithm was implemented upon LSF, which is a proprietary resource and job management system, and it makes use of CPU Frequency scaling technique. The authors claim that while their policy manages to control the cluster's energy budget, they did not observe any energy reduction. In [78], Murali et al. study a meta-scheduler that controls multiple HPC centers. In [66], Khemka et al. maximize a "utility" function in a cluster with daily energy budget. They solve the problem thanks to an offline heuristic. The objective is to reduce the overall cost by adapting the energy consumption to the electricity price of each different cluster by allocating a subset of the energy budget to each of them. Yang et al. [117] consider the scheduling problem with 2 periods: one during which an energy limit is set, and the other one without energy limit. While this approach is relevant, the algorithm they used is not scalable and is hardly usable with other constraints.

Job scheduling has a major impact on the utilization of data and computing components of facilities and thus, a high impact on how energy is going to be consumed. It is a perfect candidate to smooth the energy consumed during time through energy budget or controlled power capping. Thus, scheduling policy is a relevant leverage to combine to other leverages to control energy consumed during time of a studied facility.

2.3.5 Compression

Data transfer is also time and energy consuming, especially on large scale infrastructure where memory per core keeps getting lower and where every node level consumption could have major impact on the overall energy consumption, because of the scaling effect. Lossy and lossless compression are relevant leverages to tackle these problematics. The impact of lossless compression techniques on energy consumption has been studied in the literature. Barr et al. demonstrate

that, using several typical compression algorithms, there is an energy consumption increase when compression is applied before wireless transmission [8]. On the contrary, in [116], authors investigate the need of compression to reduce the energy consumed to save battery on hand-held devices. Their results show that on key cases for devices downloading files, the gain of energy consumption when using lossless compression can be significant. Welton et al. allocate idle CPU resources to compress network traffic, thus reducing the amount of data transferred over the network and increasing the effectiveness of network bandwidth [114]. In [68], the authors explore register file data compression for GPUs to improve power efficiency. Compression reduces the width of the register file during read and write operations, which in turn reduces dynamic power.

Satish et al. show that current cluster implementations suffer from high latency data communication with large volumes of transfers across nodes, leading to inefficiency in performance and energy consumption [93]. Authors conclude that these constraints can be overcome by using a combination of efficient low-overhead data compression techniques to reduce transfer volumes along with latency-hiding techniques.

Very recent papers [37, 48, 103, 104] have studied lossy compression as a key leverage to reduce the data to process. It consists in preprocessing data using detectors to predict if data is pertinent and should be sent to the computing kernel.

Despite several investigations on using lossy or non lossy compression as leverage to consume less energy, none of the previous papers presented a simple model that could answer, at a given time, if the compression could be beneficial from a current state of the system (size of file currently exchanged, actual available bandwidth, energy consumption of nodes) and applied it to the complexity of datacenter, supercomputer or other systems.

2.4 Application leverages

In this section, we develop the most used leverages in the application family. These leverages are relative to the scope of the binary being generated and executed.

2.4.1 Number of threads and processes leverage

Multi-core architectures are nowadays the de-facto standard in modern architectures. The first studied application leverage permits the usage of multiple cores during computation through the spawn of multiple threads. OpenMP [31], a well-known application programming interface abstraction for multi-threading, can be used to exploit this intra-node parallelism of multi-cores. It consists of a set of directives that modify the behavior of the executed code, where a master thread forks a specific number of slave threads that run concurrently. This multi-thread leverage increases the CPU utilization of the node. Consequently, because of the non-power proportionality of current hardware architectures [111], this leverage can improve the energy efficiency of the node.

The second studied leverage is the multi-process leverage. It is a de-facto standard to exploit the large scale distributed and massively parallel capabilities of modern architectures. It consists in using distributed nodes across the facility to divide the work usually by passing messages through the high speed network connecting the computing nodes. Such a shared distributed memory process permits a treatment and a throughput that a single node cannot deliver at once. MPI (Message Passing Interface) is a standardized and portable message passing library. It is available in various languages used in high performance computing such as C, C++ and

Fortran. At the end of 1993, the first normalization is standardized by researchers from academia and industry [58, 113]. It consists in a set of primitives that permit both point-to-point and collective communication.

As the two standards of both multi-threaded and multi-process computing, these two leverages are tightly coupled in the literature, that usually uses the number of threads and processes as leverage states to vary the energy used for computation.

In [70], the leverage is the number of OpenMP threads for a fixed number of MPI processes, where this leverage is combined to the DVFS leverage. The contribution of the work is to predict performance to reduce the energy consumption of an application. In [75], the leverages used to reduce the consumption of an application are also the number of OpenMP threads and the DVFS. However, in this paper, these leverages are used to recover the waiting time of MPI processes. In [6], authors provide a mathematical formulation of the multi-objective performance tuning problem. The work shows that energy-aware applications are possible through the usage of such leverages. The number of MPI processes and the number of OpenMP threads are again both used as leverages.

Thus, the choice of the number of OpenMP threads and the number of MPI processes are two tightly coupled leverages in the literature. These leverages permit a precise control of $P_{Dynamic}$. Thus, previous related works about number of threads and processes leverages have shown that the choice of current states for both leverages help modulate the energy consumption of nodes.

2.4.2 Code Version Variability

From a given algorithm to a complete application, it exists various ways of reaching the same output, either for example from implementation logic, usage of parallelism or chosen compilation environment. The generated variety of code version could be a combination of leverage to be energy efficient.

In [5], authors present a predictive model to estimate power consumption and computation performance (execution time). The prediction is made for a given device. A single version of code is given for each device from CPU to GPU. Thus, in this work, the selected version of code is actually used to choose the best type of hardware to save energy. In [6], authors explore the variability of the energy consumption of multiple CPU while using DVFS. Code versions are provided as different binaries of the same application. Thus, the work points out the possibility of obtaining multiple energy consumption behaviors by selecting a version of code while varying the frequency of the processor.

Green programming (or coding) is used to reduce the consumption of a specific application or service by iteratively modifying its source code by hand. For example, in [1], authors explore the differences in terms of energy consumption between iterative and recursive versions of the same application, thus different parallel code versions are not explored.

In [3], authors have built a framework to reduce the number of iterations in expensive sequential loops and functions through approximations. These reductions lead to energy savings. In this work, the generation of a new version of the code is done through low level compilation analysis. In [106], authors use an auto-tuning framework to study different code versions under energy concerns. This work does not consider production scenarios and knowledge construction time. The auto-tuning framework used in the work generates different code transformation strategies specific to numerical simulation.

Considering a real-case production application and multiple parallel implementations of this real-case production application, hand-written codes are very long and difficult to produce.

For this reason, green programming as it is experimented nowadays is difficult to conceive for production codes. Simplifying the usage and generation of various versions of code is important, even more to produce energy-efficient versions of codes.

However, none of the previous papers have contributed to an automation process of the usage of code version leverage. Moreover, none of these works have studied the feasibility of such Green Programming (GP) concept for production HPC numerical simulations.

2.4.3 Computation precision

Current architecture allow a wide variety of precision for computation. This leverage represents this collection of possibilities (i.e., int, float, double). Such a leverage alters the precision of the results computed by the application, but lower precision translates into shorter data representation and so, less computation and less energy consumption.

In [60], authors have shown that high precision of many applications, like audio and video processing, is not always needed and that a lower precision can be tolerated to reach almost the same output. In [59], authors study the effect of limited precision data representation and computation on neural network training. Within the context of low-precision fixed-point computations, they observe that the rounding plays a crucial role in the output. They show that deep networks can be trained using only 16-bit wide fixed-point number representation without deteriorating the output.

By relaxing the need for fully precise or completely deterministic operations, approximate computing techniques allow substantial improvements of energy efficiency. Thus, precision of computation is a relevant leverage for a wide range of applications to lower the energy consumption of applications at the detriment of precision.

2.4.4 Vectorization

Current CPUs allow the usage of vectorization capabilities to exploit intra-core parallelism. On Intel architectures, it started with MMX instruction in Pentium P5 architectures in 1997 [82]. It was then extended to SSE [50]. SSE was then extended to SSE2, SSE3, SSSE3 and finally SSE4. AVX [74] then introduces new instructions, followed by AVX2 and finally AVX512 available in XeonPhi architecture. These instruction sets permit single instruction on multiple data (SIMD) at application level.

Vectorization is also a possible leverage to use towards energy efficiency. In [20, 21], authors have evaluated and showed that the i5 and i7 generations of Intel CPUs' vectorization help to consume less energy for computation intensive benchmark such as PARSEC. They also show that vectorization should be used on specific cases (computation intensive algorithms with strict data alignment), and using it when these conditions are not met could be counterproductive from an energy perspective. Thus, using such a leverage combined with others has to be taken cautiously.

2.5 State of the art: lessons learned

Scientists and application developers are always asking for more computation capabilities to deal with finer grain and highly parallel applications. We showed, in the previous chapter, that nowadays and future computing architectures keep getting complex and heterogeneous. The next long awaited milestone is the Exascale computing, reaching 10^{18} operations per seconds.

Nowadays, architectures are not able to reach this magnitude of computation, mainly because it would not be sustainable from an energy perspective.

To answer the energy efficiency problematic and to have a better control over the power and energy consumed, a lot of techniques have emerged at different levels of the computing facility. These techniques were not all made to directly be energy efficient but to improve the throughput of the entity on which it is working. In this manuscript, we call these techniques *leverages*, entities that possess multiple states of activity, a way to switch from one state to the other, and a current state that permits, to modulate a metric on which it is working. Thus, energy and power leverages, which are our main focus in this thesis document, are leverages that directly modulate the power or energy that it consumes. We categorize the existing leverages in four different families: infrastructure, hardware, middleware, and finally application.

Many energy leverages such as shutdown policies or DVFS are well studied and possess an important literature on their potential gain as energy leverages. Other potential leverages have been exploited as performance leverages, such as OpenMP runtime, but they still do not have a study on their potential status as an energy leverage. These leverages can be combined to improve the energy efficiency of the facility. The hard objective to achieve is to reduce the energy consumption and meet imposed constraints, while not deteriorating the computing efficiency or resource utilization. All previously exposed leverages can play a key part in improving the energy efficiency of a supercomputer, if used with caution. For example, literature shows that the vectorization, as well as the compression leverage, can help towards energy efficiency if properly used, in right conditions and specific cases, thus depending on other leverage usages and environment setup.

Despite that, and on top of our knowledge, no methodology to evaluate the influence of a leverage on its environment exists. In other words, no coarse grain methodology exists to say if the usage of a leverage would be beneficial to the overall system's metric focus, here energy and power related metrics. Also, there is no simple model to say if it should be beneficial for a leverage or a state of leverage to be used. There is no simple analysis tool to interpret the interactions between a leverage and another during a leverage combination. Finally, there is no such thing as an understandable knowledge or hint extractor from leverages combination. To sum-up, no methodology to detect, evaluate and model a potential leverage exists nor permits a direct analyze of the impact of a leverage on another one when combined.

Chapter 3

Underline, evaluate and model a single power and energy leverage

Contents

As shown in the previous chapter, a lot of techniques that we call leverages exist at different levels of the facility, to modulate the usage of components. Leverages are most of the time easy to use, but their impact on a given or a set of metrics is usually unknown. Most leverage studies have been made manually without extractable methodology to discover, evaluate and use a given leverage.

In this chapter, we give our focus and methodology that we will follow all along this manuscript to study a leverage. This chapter presents the followed methodology to detect, evaluate and model a leverage, to estimate its possible usage as an energy and power leverage. First, leverages will be taken independently, before being combined later.

Computing facilities, such as datacenters and supercomputers, users and administrators are facing various problematics in their day to day usage or administration of the given infrastructure, either from just a node or for a complete infrastructure usage level. We believe that a fine grain knowledge about leverages is necessary to properly answer a given problematic. A leverage can impact various measured metrics (like energy consumption, flops, execution time,

...), so it can be used to answer multiple problematics. It is hard to use multiple leverages at the same time to answer a given problematic. Thus, we first study an independent leverage. In this manuscript, we mainly focus on how energy, power and performance related problematics are impacted by the usage of key leverages from different families on real-case studies and scenarios. All studies are done on real machines with real measurements.

From the literature, we can see that a lot of leverages can help answer a given problematic like the reduction of energy consumption, but have to be carefully used. The controlled configuration of a leverage can be a partial solution and a start of solution to a given problematic. Because we focus on energy, power and time related metrics, we evaluate the potential gain of studied leverages on these related metrics by taking into account their costs in time and energy.

3.1 Leverages and actors

In this section, we formally define *Leverages* and *Actors*, two terminologies that will be paired and used all along our methodology description.

3.1.1 Leverage

We formally define a *leverage* with the following definition:

Definition 2 *A* leverage L *is a triplet* $L = (S, s_c, f_s)$*, where* $S = \{s_0, s_1, \ldots, s_n\}$ *is the set of available valid states of* L*,* s_c *is the current state of* L*, and* f_s *is a function to update the current state to a new state* $s'_c \in S$*.*

The type of a state $s \in S$ of a leverage $L(S, s_c, f_s)$ can be any type such as frequency, application version of code, or even working state of a node (on, off). A leverage is a way to offer a choice to a user, as well as a way to modify this choice through the function f_s. Day life examples of leverages could be the gearbox of a car for which S is the set of available gears, where f_s is the clutch pedal (for a manual transmission vehicle), and s_c is the current selected gear. Note that a function is the object that permits to change the current state to another possible state.

In the case of a computing server, we can for instance consider the DVFS capability (Dynamic Voltage and Frequency Scaling) as a leverage. Assuming that the available DVFS frequencies for a given CPU are 1.8 GHz, 2.3 GHz, and 2.6 GHz (the current one, so $s_c = 2.6$ GHz), the associated DVFS leverage is $L(\{1.8\ GHz, 2.3\ GHz, 2.6\ GHz\}, 2.6\ GHz, f_{DVFS})$, where f_{DVFS} is the DVFS system configuration file.

Thus, a leverage can be seen as a tool that possesses a fixed number of internal states S, that helps a user to change its current state s_c easily through its function f_s. As a tool, the leverage is not able to change its current state by itself. Indeed, changing the state of a leverage is the aim of an actor.

3.1.2 Actor

We call *Actor* the smart entity that configures one state from S offered by one or multiple leverages as the current one. More formally, let $\mathcal{L}$ be the set of possible leverages, and $\mathcal{C}$ a set of constraints to fulfill. These constraints could be of any type. We also denote $states(L)$ the function that returns the set of states S of a leverage L.

Definition 3 *An* actor A *is a function that for a subset of leverages* $\mathcal{L}_{sub} \subset \mathcal{L}$ *and a set of constraints* $C \in \mathcal{C}^n$ *returns a set of new chosen states* S_{res}*, one for each leverage of* $L \in \mathcal{L}_{sub}$*,* $s'_c \in states(L)$*. Each new state* s'_c *returned by an actor* A *is called a* choice*.*

An actor aims at fulfilling constraints by choosing a new state $s'_c \in states(L)$. For example, the *OnDemand* linux governor chooses the DVFS current state depending on the current system load[1]. Most of the time, as the ideal solution is not available in $states(L)$, a trade-off has to be found between all constraints and objectives of C.

By its definition, we can realize that computing facilities such as data-centers and super-computers are filled with leverages and potential energy and power leverages at different levels of the facility. Because of that specificity, a user of this kind of facility is always using multiple leverages at the same time, with or without being aware of their usage. A better knowledge of available leverages, a better knowledge of their interactions and a smart usage through actors that answers a set of constraints are then mandatory.

3.2 Mono leverage study

This section presents the proposed steps to follow to understand how a leverage operates and how a leverage influences a given metric.

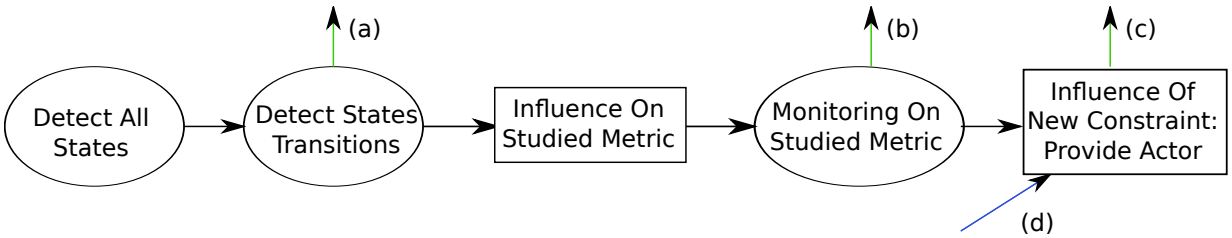


Figure 3.1: Followed flow to evaluate, model and monitor a single energy and power leverage.

Figure 3.1 presents the followed flow that we are presenting in this section to discover the set of states, the possible state transitions, the impact of states and transitions, their costs in the studied context(s), and how to provide an actor to answer new constraints for a given leverage. A squared step represents a non-automatic step where a study has to be done, while a rounded one represents a step that can be automated. A green arrow represents a possible exploitable output from the methodology, and a blue arrow represents an input.

3.2.1 How a leverage operates

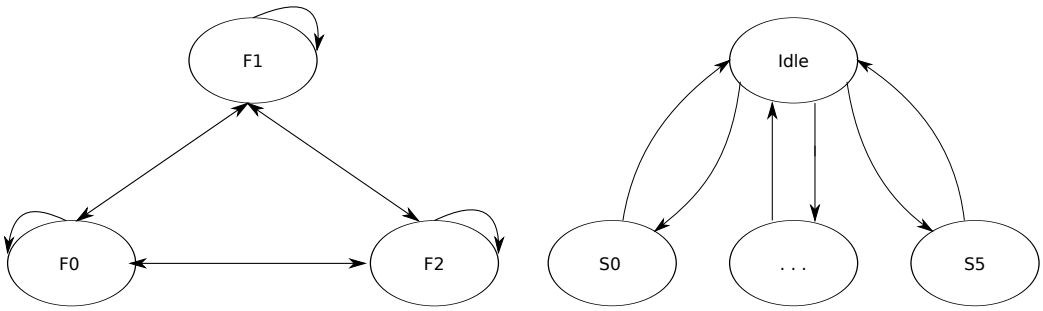
The first thing to understand about a leverage is how it works. In other words, detecting all states and how to go from one valid state to another is an inevitable step that has to be done to start the study of a leverage. Depending on the leverage, this step can be done through a study of the literature, by manual or automatic exploration of the studied infrastructure.

As previously exposed in subsection 3.1, a leverage has a set of possible states. These states represent all the possibilities of configurations offered by the studied leverage. To discover the set of all possible states, an exhaustive actor with no constraints that explores all possible

[1]https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

states is needed. This actor can either be a physical user or an automated script manually or automatically exploring possible configurations.

Depending on the leverage, all states cannot be reached from all other states. The states can have precedences, meaning that a state cannot be reached if a previous state is not attained yet. Understanding the interaction between states is also a mandatory. On Figure 3.1, this study is represented by the two first circles. From this study, a graph of reachable states per current state can be obtained, with a recorded way of switching from one state to the other ((a) output in Figure 3.1).



(a) DVFS (b) Sleep States

Figure 3.2: State transitions, the example of DVFS and sleep state leverages.

Such an output is displayed in Figure 3.2. It presents the obtained graphs of transitions for both DVFS and sleep state leverages on subfigures 3.2a and 3.2b, respectively. In this illustrative example, F_n represents a possible frequency for the CPU, and S_n represents a possible sleep state of a computing node. An arrow represents a possible transition between two states. Every transition can have various ways to be activated and various associated costs. Thus, every leverage is unique in its transition between states. DVFS, for example, can go from any frequency to the other showing a fully connected graph between possible states, whereas it is impossible to go from one sleep state to the other without passing by the idle state.

3.2.2 How a leverage influences the studied metric

A given studied problematic is usually related to a metric or set of metrics. The influence of a leverage on a metric is linked to the understanding of the direct influence of changing the current state and its influence on the studied metric, either during the switch of state or while operating on the new state.

This study can first of all be done on a given problematic from a theoretical point of view. For instance, shutdown is a relevant leverage to save the P_{Static} part of the energy consumed by a component that is currently idle, while DVFS is a leverage that can modulate the energy consumed dynamically ($P_{Dynamic}$), but does not change the power statically consumed by a component. Therefore, even if these two leverages influence the same metric, the energy consumed on a node level, they do not do it the same way.

Also, a studied leverage metric can be located on only one component or distributed between many. For example, the Shutdown leverage has a mono-node energy consumption influence. However, the compression / decompression leverage (previously exposed in section 2.3.5) has a distributed metric between the sender and the receiver (energy consumed on the node that

compresses the file and energy consumed on the node that decompresses the file). Another example is the number of processes used to execute an application, that can be on one or multiple nodes.

Thus, understanding how a leverage and all its states influence the studied metric is also a mandatory step to evaluate how a leverage influences a used facility.

3.2.3 Monitoring a leverage

An aspect of the study of one leverage is to evaluate the real cost of a leverage in the multiple contexts where it will be used. It can be done through theoretical study but also through monitoring.

Monitoring the leverage can be done once transitions between states have been explored, in a given context. Usually, the configuration of a leverage is not free depending on studied metrics, here time and energy. Also, choosing a given state can highly impact the power and/or energy consumed by a component facility.

Monitoring the leverage costs (transitions and states) on various contexts where it will be used is mandatory to get an idea of how much the configuration of the leverage will cost, for given metrics. The configuration of a leverage on its future context can reveal what are the effects of a leverage on its environment.

To have the complete leverage knowledge, monitoring and storing costs of transition from one state to the other, as well as a monitoring of the component metric throughput during a given work (such as a given representative application or workload) is necessary. To do so, an exhaustive actor with knowledge of the state transitions and with no constraints to fulfill is needed to loop over possible leverages states (output (b) on workflow Figure 3.1).

3.2.4 Providing actors

Once that we know the set of states, the possible state transitions, the impact of states and transitions and their costs on the studied context(s), an actor answering a new constraint for the studied leverage can be provided.

An actor tells if the current state is beneficial to the studied metric. It could also tell us if choosing a new configuration (choosing another state for the studied leverage) is beneficial to answer the new constraint (input (d) in Figure 3.1), in a binary way (i.e., through a *yes* or *no* answer). This answer is based on what would happen if the state was changed to another one, thus tacking into account transition costs as well as the cost of the new current state. Such an actor based on previously explored costs permits to quickly answer and combine actors to answer multiple constraints at the same time.

Of course, a complete facility is facing other constraints. In our studies, energy consumption is the main focus. But the complete facility will have other constraints during its usage, such as power capping, cooling, electrical, maintenance, performance, etc. They can have influence on one another, thus a trade-off has to be found. These constraints will either be for the complete infrastructure or for a subset of nodes. Thus, an actor can be provided to answer these constraints. In this methodology, we propose to provide one actor to every constraint ((c) in Figure 3.1). With a set of actors being able to answer a given constraint each, we can compose these actors to respect a set of chosen constraints at the same time.

3.3 Mono leverage study applied to Thermo-Electric-Generators (TEG)

In this section, we apply the previously proposed methodology to show that Thermo-Electric-Generator (TEG) can be a relevant energy leverage for large-scale infrastructures. We chose an infrastructure leverage for a purely theoretical application of our methodology to underline the potential gains of such a leverage without providing any new actor.

Most of the energy consumed by resources in a computing facility is converted to heat. This heat, induced by computing resources, is generally a waste of energy in supercomputers. This is especially true in very large scale supercomputers, where the produced heat has to be compensated with expensive and energy-consuming cooling systems. Energy is a critical aspect for future supercomputing trends that currently try to achieve exascale, without having its energy consumption reaching an important fraction of a nuclear power plant. Thus, new ways of generating or recovering energy have to be explored. Energy harvesting consists in recovering wasted energy. ThermoElectric Generators (TEGs) aim to recover energy by converting wasted dissipated energy into usable electricity. By combining computing units (CU) and TEGs at very large scale, we spotted a potential way to recover energy from wasted heat generated by computations on supercomputers.

3.3.1 TEG leverage, how it operates

Energy could be produced from transformation of raw materials (oil, gas, coal, petrol, etc). Every produced kiloWatt is accompanied with a generation of CO_2 [90]. Because energy generators are not 100% efficient, a significant part of energy is lost during the production and transport [34]. Only a slight percentage of the conversion of raw materials is transformed into electricity. For example, the ratio between useful energy output and energy input (as known as energy conversion efficiency) of a coal plant is 33-35% [25].

The growing demand for processing data and computing implies improvements of electronic components. At the level of computing capabilities or energy consumption, computational components of current machines are becoming more and more efficient. Despite technological and software breakthrough to increase computational power while reducing energy consumption, it became a real limiting factor for computers. This is especially true for supercomputers reaching petascale performance and aiming for exascale, where sustained one megaWatt per year costs around one million dollars per year [39].

To face such problematics and as previously mentioned in Section 2.1.2, new ways to generate or at least recycle energy have to be explored. Energy harvesting is the process of extracting energy from surrounding environment and thermoelectricity aims at recycling wasted energy under heat form.

In a computing facility, a great part of consumed energy is lost in an exothermic way. For the safety of components, this lost energy must be carried away. To do so, expensive air or water cooling systems are built upon supercomputers. None of this lost energy is reversed in a direct usable electricity for a direct re-use. Thermoelectricity seems to be the proper energy harvester for this problem.

Binding CPUs and TEGs has been mentioned in [120]. It deals with common computers built with only one computing unit (CU), and it experiments only around the idle part of a CPU. Not stressing the CU implies lowering the delta temperature. Thus, the CPU is not under deep stress as it would be in an intensive supercomputer scenario.

The actual increase concerns about climate and energy efficiency force researchers to consider all possible solutions to consume better. One of the main concerns is energy lost in an exothermic way. Indeed, this energy enforces expensive costs in cooling systems. By transforming a part of this dissipated energy, TEGs may reduce costs.

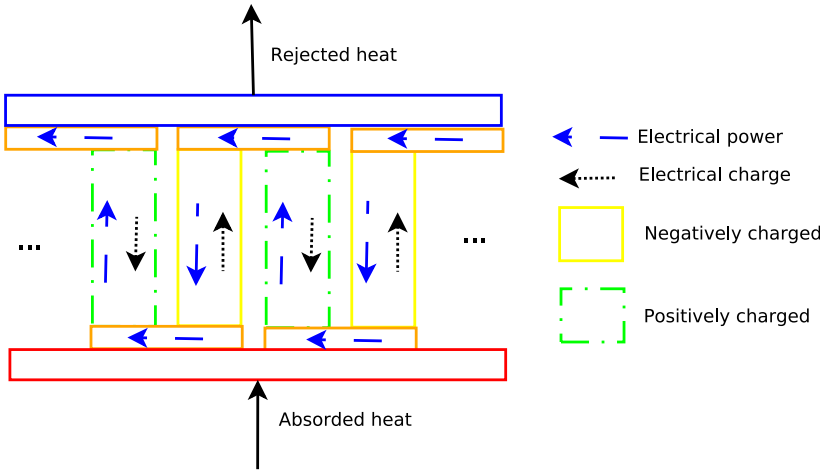


Figure 3.3: ThermoElectric Generator (TEG).

A thermo-electrical material transforms a thermal delta in electricity. TEGs are composed by positively (p-type) and negatively (n-type) doped connected semiconductor couples (see Figure 3.3). N-P couples are the charge carriers that can freely move through the metal. These carriers start to move under a delta temperature, according to the N-P couple properties. The delta temperature creates an excitation of the doped charge carrier, thus it creates a movement of the charge carrier, creating an electric current.

Ideally, a larger delta temperature creates a larger electrical current, but this statement is highly coupled with the fact that a semiconductor is effective only on a range of temperature, making the TEGs operational only on a limited delta temperature [100].

Other fields can have potential use of such a technique. A lot of heat is wasted from automotive, specially from exhaust pipes. A lot of exhaust pipes accompanied with TEG prototypes have emerged. Only 25 percent of the applied fuel energy made from combustion is used for mobility and accessories. About 40 percent of this energy is lost in exhaust pipe. Even if conversion percentage is low, it could lead to a nice ratio of CO_2 emission reduction [57]. Ultra-low power wireless sensors used for various measurements can also be combined with energy harvesting techniques. Thanks to the energy harvesting part of the TEGs and the ultra-low needs of power, it is nowadays possible to produce wireless sensors that are almost autonomous systems [92].

A TEG can be composed from different alloys. Every alloy has a temperature where it could be applied. The possible range of action on a TEG depends on a minimal and a maximal temperature. In fact, under this minimum temperature, no power can be generated and over this maximum temperature, the chosen TEG has an indeterminate behavior [100].

Here, we discuss the feasibility of using thermoelectric generators for petascale and future exascale machines. How much harvested energy can we hope for? Is recovered energy sufficient enough to be directly injected in CPUs or to be stored? How much time will it take to compensate the cost of combining thermoelectric generators and computational units?

3.3.2 Context and influence on studied metric

For the safety of the machine composing the supercomputer, the heat generated by CUs must be carried away from CUs. This results in a huge amount of wasted energy, when taking into account the dissipated energy from computation and from the expensive cooling system.

Generated energy from a TEG depends on the temperature on the cold and hot side. Thus, it depends on the delta temperature between the two parallel sides. Table 3.1 presents the extreme thermal context that could be witnessed on a supercomputer environment.

Table 3.1: Supercomputer extreme delta temperature in Celsius degree

Characteristic	Temperature
hotspot	95 - 110
cooling system	15 - 25
delta	95

We can assume that, because of the intensity of the usage, every CU is close to the maximum temperature. The hotspot informations correspond to the maximum reachable temperature by a Xeon CPU and a Nvidia GTX Titan Z GPU. The cooling system temperature represents the recommended range of temperature expected in a room containing computing components [56]. The delta is the difference between these two extreme temperatures. We make the hypothesis that a TEG is placed on every CU and that the temperature on top of this TEG is equal to the cooling system temperature. In other words, we consider that the cooling system in on top of every CU (like with water cooling system), thus creating the needed delta temperature for a TEG to generate electricity.

The maximum efficiency, noted η_{max}, represents the maximum percentage that a TEG can convert from a thermal Watt to an electric Watt. The formula of η_{max} is described as follows:

$$\eta_{max} = \frac{\Delta T}{T_h} \frac{\sqrt{1+ZT}-1}{\sqrt{1+ZT}+\frac{T_c}{T_h}}, \tag{3.1}$$

where

- T_c is the temperature on the cold side;
- T_h is the temperature on the hot side;
- ΔT is the difference between the hot and cold side, i.e., $\Delta T = T_h - T_c$;
- ZT is the figure of merit of the TEG.

The maximum efficiency of TEG is therefore governed by the figure of merit ZT, that is a parameter depending on the materials used to build the TEG, and by temperature of the hot and cold side of the TEG. It also depends on the Carnot efficiency (the left factor on Equation (3.1), i.e., $\Delta T/T_h$). This metric is needed to estimate the maximum usable power generated from a TEG on a specific environment.

Applying this metric to the previously described environment (Table 3.1) allows us to obtain the maximum efficiency reference, which depends on the ZT of the chosen TEG.

Figure 3.4 presents the evolution of the maximum efficiency, η_{max}, as a function of the ZT of TEGs with $T_c = 15$ Celsius degree, for three delta temperatures ($\Delta T = 50, 75, 100$). We consider

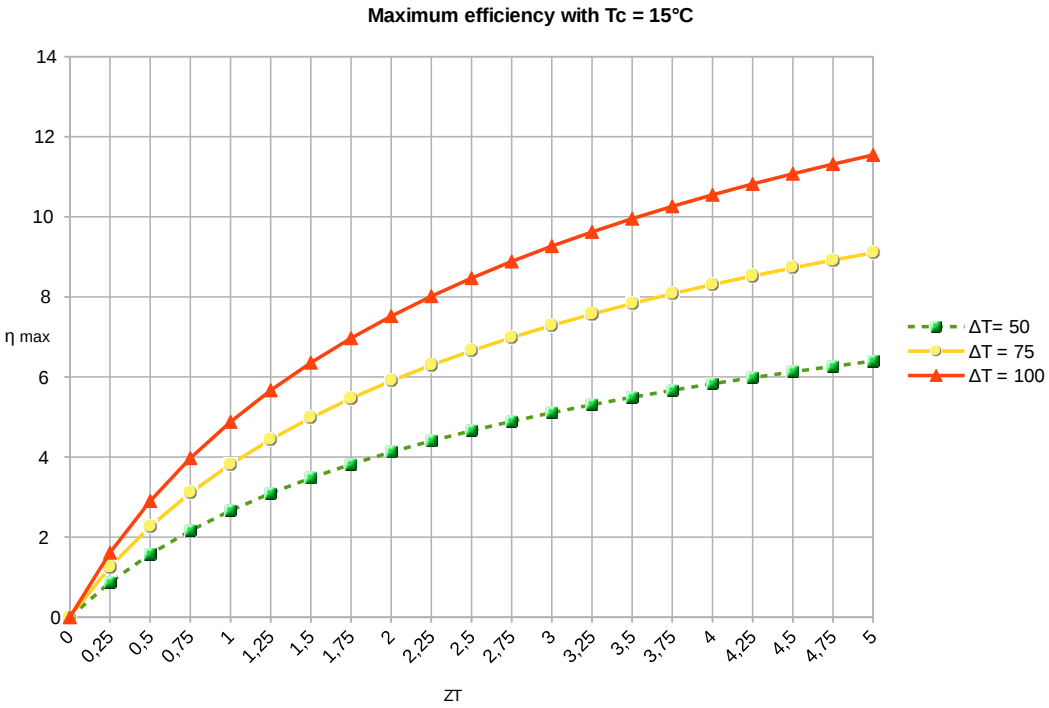


Figure 3.4: Maximum TEG efficiency in the supercomputer scenario.

a ZT between 0 and 5, given that the state of the art TEG figure of merit is nowadays around 2.2 [15]. In a very optimistic setting with $ZT = 5$ and $\Delta T = 100$, the maximum efficiency is around 12%.

The most widespread alloy for thermal exchanges between computing units and TEGs is the Bismuth Telluride (Bi_2Te_3) [55]. Literature [100] validates the fact that it is the only alloy that can be applied in an environment described in Table 3.1. In fact, in a range between 15 and 100 Celsius degree, only one N-P couple can be built to be used in a TEG (respectively Bi_2Te_3 and Sb_2Te_3). A TEG with these alloys has a device figure of merit ZT between 0.7 to 1.0 [91].

Considering Figure 3.4 and the known ZT of a compatible TEG, we can now say that the maximum efficiency in our environment will be between 3 and 5%.

3.3.3 Theoretical context and evaluation metrics

In order to figure out if the usage of TEGs is relevant in a petascale or exascale environment, we consider several scenarios.

For this analysis, we consider the state of art and potential CU candidates to an exascale machine: the in vogue KNL XeonPhi [94] with a peak around 3.0 TFLOPS and a TDP of $200W$, the very promising RexNeo[2] announced around 0.256 TFLOP with a TDP only around $4W$, and the mainstream Xeon CPU [79] around 0.506 TFLOPS and $145W$. We determine how many megaWatts a machine reaching one exaFLOPS composed only with one of these computing units will consume (see Table 3.2).

Following the Department of Energy (DOE) requirements, the expected exascale machine

[2]http://www.rexcomputing.com/

Table 3.2: Hypothetical wattage and number of computing units (in million units) for an exascale machine

Machine	MegaWatts	# CU
100% RexNeo	15	3.9
100% XeonPhi	66	0.3
100% Xeon	286	1.9

should achieve one exaFLOPS, with a consumption of 20 MW (or below) [98]. The machines in Table 3.2 are the hypothetical machines made with only one of the computing units previously presented.

The first observation is the fact that an exascale machine made only with RexNeo computing units can achieve one ExaFLOPS with 15 MW, which is under the recommended limit fixed by the DOE. The second observation is that a machine reaching one ExaFLOPS made only with KNL XeonPhi or Xeon is unconceivable when considering power consumption and the required limit imposed by the DOE. However, because these CUs are very common in the supercomputer field, we will keep them in the study.

Along with overall recovered energy from the usage of TEG, the return on investment is an important parameter to consider. In fact, as motivated earlier, TEGs offer a way to recover wasted energy. Therefore, this recovered energy is an amount of energy that will not be asked by the machine, thus it does not have to be produced elsewhere and bought. To compute the return on investment, we consider a new metric $yReturn$: it represents the number of years after which having installed TEGs on all CUs becomes profitable. It is defined as:

$$yReturn = \frac{TEGsOverallCost}{SustainedWattCostOneYear \times recoveredEnergy} ,$$

hence by dividing the overall cost of TEGs by the cost saved within one year thanks to the generated energy. In fact, the TEG costs will be amortized on the recovered energy costs. This generated energy could be directly used in the supercomputer or sold back to the provider, and it corresponds to an amount of energy that would have been asked to the provider. Recovered energy becomes a direct benefit past $yReturn$ years.

3.3.4 Theoretical calibration to evaluate the impact of TEG

The followed hypothesis is to put a TEG on every computational unit and to estimate how much energy would be recovered with the previous described environments (Table 3.1) extended to these exascale machines (Table 3.2) and on current supercomputers. Table 3.3 shows the hypothetical gain with a TEG with an efficiency η_{max} between 3 and 5% on every CU on the hypothetical machines that we chose for our exascale machines. In addition to the realistic values, we study the best hypothetical efficiency (ZT= 5, η_{max} around 12 % on Figure 3.4) in the supercomputer case. Even if the percentage is low, the energy recovered is equivalent to a non negligible computing unit number for every exascale machine.

Such an installation is beneficial only if the price of the deployment of TEGs is quite accessible and if the return on investment is under the lifetime of the supercomputer on top of which it was built.

In Table 3.4, we display the price of applying TEGs on every CU. Prices are from websites

Table 3.3: Energy recovered in MegaWatts and equivalent computing unit consumption compensation

Machine	3% (MW)	5% (MW)	12 % (MW)
100% RexNeo	0.47	0.78	1.87
100% XeonPhi	2	3.3	8.0
100% Xeon	8.5	14	34.38
Machine	3% (#CU)	5% (#CU)	12 % (#CU)
100% RexNeo	117 188	195 313	468 750
100% XeonPhi	10 000	16 667	40 000
100% Xeon	59 289	98 814	237 154

for common usage and TEG vendors ($4[3] and $10[4]). We could imagine that the cost for such a machine, per TEG, will be lower. As a lower bound, we consider an extreme case where a TEG costs $1. Even with common usage prices, applying a TEG on every CU stays reasonable, considering the price of a supercomputer such as Tianhe-2 as a reference. Note however that this cost only integrates the raw cost of TEG, without considering manpower and other deployment costs.

Table 3.4: TEG on CU - overall cost in million dollars and return on investment

Machine	TEG $4	TEG $10	TEG $1
100% RexNeo	15.6	39	3.9
100% XeonPhi	1.3	3.3	0.3
100% Xeon	7.9	19.7	1.97
	$4, 3%	$10, 5%	1$,12%
100% RexNeo	33.3	50	2
100% XeonPhi	0.6	1	0.04
100% Xeon	0.9	1.3	0.05

Based on the hypothetical exascale machines described in Table 3.2, the cost of having a TEG on every CUs of this machine is described in Table 3.4. The cost of one sustained Watt per year is fixed at $1 in our case. The bottom part of this table shows the return on investment of the different scenarios (i.e under various TEG cost and η_{max}).

Because of the low FLOPS of RexNeo, a very high number of CUs are needed to achieve one exaFLOPS. Thus, the return on investment for this machine is drastically high, overcoming the usual life time of a supercomputer. Except for this specific configuration, other hypothetical exascale machines benefit from TEGs between 8 months and 1 year and 4 months.

One can ask if the actual return on investment does not overcome the life time of a TEG. In fact, TEGs could last more than 15 years [11], so there should be no need to replace them during the lifetime of the supercomputer. In comparison, the return on investment of an exascale machine made only with XeonPhi or Xeon computing unit is very low.

To see the potential impact of TEGs on a real architecture, we chose to focus on the second machine on the top500, Tianhe-2 [107]. This supercomputer has been the number one machine in terms of peak performance for three years (from 2013 to 2016).

[3] $4 TEG example
[4] $10 TEG example

As exposed in [40], every CU of Tianhe-2 is composed of two Xeon processors and three Xeon Phi accelerators. There are 16000 computational nodes accompanied with 4096 front end nodes. It costs around 2.4 billion yuan, i.e., around 390 million US$ [107].

Tianhe-2 is composed of 32000 Xeon E5-2690 processors with each a TDP of $115W$, along with 48000 Intel Xeon Phi 31S1P accelerators, with a TDP of $270W$. Finally, 4096 Galaxy FT-1500 with a TDP of $65W$, are composing the front end nodes of Tianhe-2.

The total facility energy and IT equipment energy exposed consumption of Tianhe-2 in [40] is around 24 and 17 MW, respectively. The Power usage effectiveness (PUE) of such an infrastructure is around 1.41.

As explained in Section 3.3.2, the maximum efficiency in a supercomputer environment and with current TEGs technology is estimated to be between 3 and 5%, and the most optimistic efficiency is around 12%.

Table 3.5: Energy recovered in MegaWatts and equivalent computing unit consumption compensation in the Tianhe-2 case

Machine	3% (MW)	5% (MW)	12 % (MW)
XeonPhi	0.38	0.648	1.55
Xeon	0.11	0.184	0.44
Galaxy	0.0079	0.0133	0.032
Machine	3% (#CU)	5% (#CU)	12 % (# CU)
XeonPhi	960	1600	5760
Xeon	1440	2400	3840
Galaxy	122.88	204,8	491.2

Even if η_{max} is equal to 3% of 5%, Table 3.5 shows that the energy recovered is equivalent to a non negligible number of CUs. Such an installation could be beneficial if and only if the return on benefit is inferior to the supercomputer lifetime.

Table 3.6: TEG on all Tianhe-2 CUs - Profitability (years)

TEG $10			
	3%	5%	12%
XeonPhi	2,89	1,73	0,72
Xeon	1,23	0,74	0,30
Galaxy	5,12	3,07	1,28
TEG $4			
	3%	5%	12%
XeonPhi	1,15	0,69	0,28
Xeon	0,49	0,29	0,12
Galaxy	2,05	1,23	0,51
TEG $1			
	3%	5%	12%
XeonPhi	0,28	0,17	0,07
Xeon	0,12	0,07	0,03
Galaxy	0,51	0,30	0,12

Table 3.6 presents how much time, in years, it would take to recover the cost of one TEG on

every CU for Tianhe-2. This table presents various TEG configuration results. The worst case concerns the front end node with a TEG that costs $10. If we consider only CUs, the worst case is still the 3% TEG that costs $10, which will become profitable after approximately 3 years of usage.

Tianhe-1A is up since 2010, Titan since 2012, while Tianhe-2 is up since 2013. Even in the worst case, the time in which our hypothetical TEG deployment on Tianhe-2 CUs will become profitable is way under the usage of a supercomputer. In fact, as shown in the Top500 list, a supercomputer usage could last a few decades.

With a reasonable and realistic TEG cost and a recycling efficiency, it becomes profitable, concerning CUs, only after 0.69 year, approximately 9 months (for example a $4 TEG with an efficiency of 5%).

3.4 Conclusion about mono leverage study

Through this chapter, we studied a methodology to study a leverage:

- We focused on a general problem to answer and look for a relevant leverage to answer or partially answer this question;
- We model the leverage, its various costs and influences, how it interacts with its environment and evaluate the various costs of the leverage;
- We understand the various contexts where the studied leverage could be applied and be beneficial, from an energy perspective;
- We then provide a way to answer a new constraint for the given leverage thanks to a new actor focusing and answering a unique constraint.

This methodology permits to evaluate and model a leverage, how it could be configured and how useful it could be. This methodology only focuses on one leverage at a time, which permits an analysis of the impact of a leverage on its environment. Thus it permits to evaluate and make sure that a leverage will be relevant on a given set of contexts. These kind of studies are necessary to assert that a given leverage is relevant and is significantly helping to answer a given problematic.

In our theoretical application of our proposed methodology, we show that the TEGs are an energy leverage in a supercomputer environment. Indeed, TEG usage could save energy and be quickly profitable, even if it is not the best environment to exploit the potential of TEGs. In fact, such an environment implies a low delta temperature between hot and cold side of the TEG, a low hot temperature on the computing unit, and thus a low maximum efficiency. However, theoretical applications show that even if only a small fraction of the heat could be transformed into energy, the return on investment could be short. Thus, at the petascale or exascale level, where every hypothetical energy gain is mandatory, it could be a new way to recover energy. We present the feasibility of energy harvesting on a petascale and exascale environment using the ThermoElectric Generators leverage.

Because this methodology is only focused on the study of one leverage, it does not allow us to evaluate the interaction and the combination of leverages, or the influence on other leverage usages. Also, it does not allow us to combine one leverage with the other and evaluate their interactions. It allows us to evaluate and model a leverage and make sure that using such a leverage will be beneficial in a given context.

Chapter 4

Modeling and proposing actors for a single leverage: focusing on the Shutdown leverage

Contents

In the previous chapter, we explored the feasibility of studying a unique leverage with our previously proposed methodology and applied it in a theoretical study of the Thermo Electric Generators. In this chapter, we apply that methodology from the discovery of the leverage to an expandable actor composition to fulfill several chosen constraints at the same time, with the example of the Shutdown leverage.

Multiple constraints have to be taken into account for such a leverage to be applied on real infrastructures: the time and energy cost of switching on and off, the power and energy consumption bounds caused by the electricity grid or the cooling system, and the availability of renewable energy. In this chapter, we propose actors translating these various constraints into different shutdown approaches that can be combined for a multi-constraint purpose. Our actors and their combinations are validated through simulations on a real workload trace.

4.1 Introduction

In Chapter 2, we exposed several leverages that have been proposed by system designers in order to help reducing electrical power consumption, as for instance shutdown and slowdown approaches. The first and most explored solution consists in turning on and off resources depending on platform usage. Nowadays, hardware components of a datacenter or supercomputer (servers, network switches, data storage, etc.) are not yet energy proportional. In fact, the static part (i.e., the part that does not vary with workload) of the energy consumed, for instance by computing units, represents a high part of the overall energy consumed by the nodes. Therefore, shutting unused physical resources that are idle and not expected to be used in a predicted duration could lead to non negligible energy savings, as previously exposed in Section 2.2.1. Several works that studied the energy-related impacts of the shutdown leverage, as exposed in Section 2.3.3, did not consider any transition cost for switching between on and off states, but they nonetheless showed the potential impact of such a leverage. Yet, aggressive shutdown policies are not always the best solution to save energy [80].

Shutdown seems to be a relevant leverage to save energy, but this leverage cannot be applied at large scale if no constraint is respected on the target system. This is especially true if three types of constraints are taken into account: the cost of shutdown and wake-up (in terms of time and energy), electric and thermal constraints imposed to the whole infrastructure. We can see the computing facility as a composition of *IT machines and cooling system, communicating with an electrical provider to deal with various electric related constraints.*

Supporting shutdown of large amount of resources can be risky as it impacts the whole infrastructure of supercomputers (electricity provision, cooling systems, etc.). Resource providers and managers can be human who are responsible of the administration of large supercomputers, but they can also be software components that deal with resources (schedulers, resource management frameworks, etc.). Actually, turning off too many nodes could cause the temperature to be too cold and the power used to be under the minimum power capping negotiated with the electrical provider. Likewise, if too many nodes are turned on and if the energy consumed during shutdown and wake-up sequences is taken into account (which is far from being free), limits fixed by the power provider can greatly be overcome and at the same time, could cause the temperature to raise drastically, creating hotspots. If such constraints are not taken into account, they can put into danger machines composing the studied computing facility.

This chapter addresses the question on how resource providers and managers can be helped to validate their constraints concerning the shutdown of large amount of physical computing, storage and networking resources. In the context of this chapter, the proposed actors and validations will focus on servers (called *nodes*). The study of the shutdown leverage and the proposed actors are done through the previously proposed methodology to underline, evaluate and model a single energy leverages exposed in Chapter 3.

The proposed solutions in this chapter aim at:

- Modeling the shutdown leverage that can be used under actual and future supercomputer constraints;
- Taking into account the impact of On→Off (from on to off state, corresponding to a shutdown operation) and Off→On (from off to on state, corresponding to a wake-up operation) sequences in terms of time, power and energy;
- Taking into account idle and off states observed after such sequences, since they deeply impact the electrical usage of resources;
- Allowing a mono or combined usage of actors in order to help resource managers and providers to respect several constraints at the same time.

This chapter explores the creation of shutdown actors for several constraints that can be handled by resource providers and that deal with infrastructure constraints:

- The *basic actors* allow comparisons with several related works where turning on and off can be free and immediate.
- The *sequence-aware actors* focus on the On→Off sequences when providers want to switch off several useless resources and to switch them on again when these resources are needed. These actors deal with the availability of scheduling On→Off sequences during gaps and their potential energy benefits.
- The *electricity-aware actors* deal with the electrical provision of supercomputers in order to avoid large-scale aggressive electrical demands (due to massive switch on of resources) and to respect power capping requirements.
- The *cooling-aware actors* respect the constraints imposed by the cooling infrastructure associated with the supercomputer. They follow the thermal constraints of the system by reducing the number of possible On→Off sequences.
- The *renewable-energy-aware actors* support selective shutdown policies by considering the electricity provenance (from renewable energy or from fossil-based energy sources).

The proposed actors are described one by one and their combined usage is illustrated. Such actors are validated through simulation on real trace log usage.

The chapter is organized as follows. Section 4.2 exposes how the shutdown leverage operates, following the first steps of the methodology presented in Section 3.2, from detecting all states (Section 3.2.1), to evaluating the impact of such a leverage on energy (Section 3.2.2). Section 4.3 presents the monitoring of the shutdown leverages on various architectures, the fourth step described in Section 3.2.3 of the proposed methodology. Section 4.4 presents the modeling of

the various shutdown (On→Off) constraints and creation of all actors introduced above. It also explains how the actors can be used and combined, which corresponds to the last step of the methodology presented in Section 3.2.4. The experimental setup is described in Section 4.5. Experimental results are analyzed in Section 4.6 and Section 4.7 for large scale and finer grain studies, respectively. Finally, Section 4.8 concludes this work.

4.2 How shutdown leverage works

In this section, we develop the first steps of the previous methodology: evaluate and model the states and state transitions for a given leverage, in this case the shutdown leverage.

4.2.1 Sequence definitions: example of Off→On sequences for nodes

For node i, $Seq_i = \{(t_0; AvgP_0), \ldots, (t_n; AvgP_n)\}$ is the set of timestamps and average power consumption measurements of an Off→On or On→Off sequence, where t_0 and t_n represent the starting and ending time respectively of sequence Seq_i on node i. The length of the sequence is therefore $t_n - t_0$. At timestamp t_k $(1 \leq k \leq n)$, $AvgP_k$ is the average power consumption of node i.

To monitor such sequences, we use an external power monitoring allowing us to trace power consumption of nodes at a rate of one averaged power value per second. Figure 4.1 illustrates the boot sequence (or Off→On sequence) on Linux based servers, widely used on supercomputer infrastructures. First of all, power is supplied to SMPS (Switched-Mode Power Supply), which converts AC to DC. The BIOS (Basic Input Output System) is bootstrapped and launches POST (Power on Self Test), a series of tests by the BIOS, that checks the proper functioning of different hardware components. Then MBR (Master Boot Record), the first or last bytes of the disk, is loaded. MBR permits to launch GRUB (Grand Unified Bootloader), which is responsible for choosing the kernel to be launched. INIT is the first executed process. It is in charge of running all runlevels ("/etc/rcX.d/").

Along with computing nodes, Grid'5000 provides management tools like kapower3, a utility that allows a user to have control on the power status of a reserved node[1]. On mulitple sites, it gives access to external wattmeters monitoring entire servers and providing one power measurement per second per server with a 0.125 Watts accuracy. We monitor the boot sequence (wake-up operation, Off→On) to detect when each event happens. Unfortunately, no information can be extracted between BIOS and GRUB operations. The first event that can be monitored in this sequence is the Kernel launch; this is the main reason of the aggregated sections of BIOS-MBR-GRUB in Figure 4.2, which shows how the power evolves with time during a monitored boot sequence, on a Taurus node (from Grid5000 experimental platform, previously explored in the previous chapter, the node characteristics are presented in Table 4.1).

We get the time where kernel starts with the "dmesg" tool (which is a logging of what happened during the launch of the kernel). The INIT monitoring is made by modifying the runlevel script.

Next, we detail the set of possible states accounting for these Off→On and On→Off sequences.

[1] `https://www.grid5000.fr/mediawiki/index.php/Power_State_Manipulation_commands`

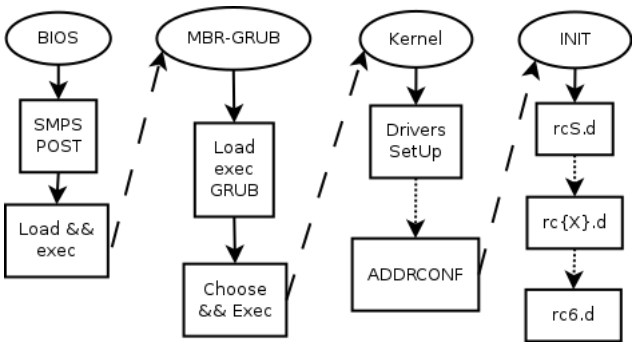


Figure 4.1: Linux monitored boot sequence.

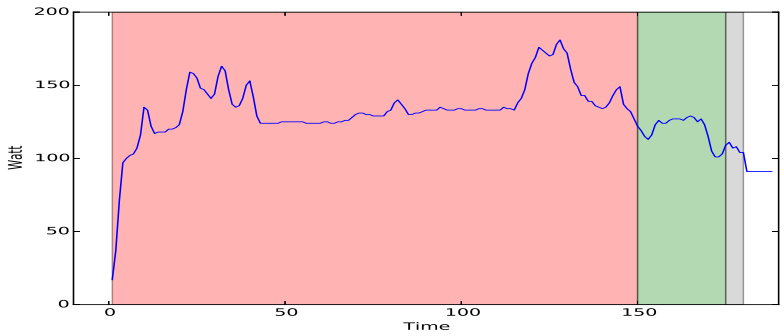


Figure 4.2: Averaged monitored Off→On sequence of a Taurus node running Linux: BIOS-MBR-GRUB sequence in red; Kernel in green; Init in gray (average of 50 runs).

4.2.2 Possible states and graph of transitions

Since we wish to account for Off→On and On→Off sequences, we partition the devices taken into account into four distinct sets as illustrated in Figure 4.3:

- ON in progress: Set of nodes in the Off→On sequence;
- ON: Set of nodes turned on, able to receive computation. This state is divided into two sub-states: Idle and Run;
- OFF in progress: Set of nodes in the On→Off sequence;
- OFF: Set of nodes turned off, unable to receive computation.

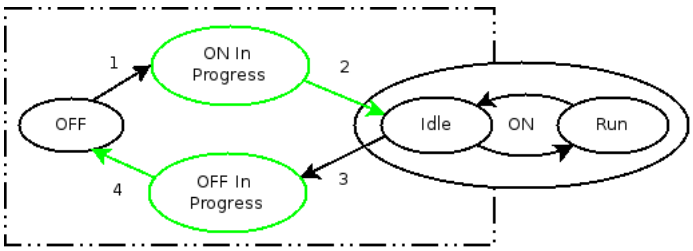


Figure 4.3: States and transitions during various sequences.

Furthermore, we denote by ALL the set of all nodes. We define our action scope only on the Idle state, i.e., nodes that are turned on but not currently computing. Nodes on the *Run*

state (i.e., currently computing) are not on the action scope of this actor, since it is rather the scope of the scheduler to decide to stop computation and turn nodes to the Idle state. We need however to be aware of the nodes in the *Run* state since we aim at ensuring a global power capping. Thus, a node goes from the *ON in progress* state to the *ON* state through the *Idle* state. It can leave the *ON* state only when it is in the *Idle* state.

In Figure 4.3, the dotted line square therefore represents the scope of the actors described here. We aim at allowing a set of nodes to switch from one state to another, by taking one of the four numbered transitions. Transitions 2 and 4 are automatically taken at the end of the *ON in progress* or *OFF* in progress states, while we may decide to trigger transitions 1 or 3.

A node in the *ON in progress* state could be in several sub-states, according to the Linux boot sequence: BIOS-MBR-GRUB, Kernel, Init, or whatever boot up sequence is defined on the node. We consider other states as atomic.

We use the actors as follows: we decide what can be done at the current time-step T_c, knowing that there is an idle interval of length T_{gap} on a given node. In our case, the actor decides whether the node should be shut down, given the enforced constraints.

4.3 Shutdown energy calibration

This section presents the energy calibration concerning the shutdown leverage done on two types of architectures: actual and promising architecture to reach energy proportionality. We imposed a strict protocol for calibration: an idle period of 20 seconds, followed by a calibrated On→Off, then a 20 seconds Off section finally followed by a Off→On sequence. Thus, every energy and time calibration sequence is followed and previewed with idle periods to avoid noise.

4.3.1 Actual architectures

Grid'5000 servers are representative of typical architectures that can be found in usual datacenters. As it is an experimental testbed mainly for distributed systems research, it presents a high variety of servers (currently 24 different clusters). We used servers from three clusters which are power-monitored (one server per cluster). These servers presents heterogeneous characteristics described on top of the Table 4.1.

To obtain the needed calibration values, we monitored the three servers while performing switching off and on operations. These nodes are running a standard Debian Jessie (Debian GNU/Linux 8.0 for x64 architectures). The results presented on the bottom part of Table 4.1 show averaged values over 100 experiments for these operations with the S5 mode (regular shutdown). Note that the standard variation of every chosen node for every chosen calibration is negligible.

We performed similar experiments for the S4 mode (Suspend-to-Disk). However, our experiments show that S4 mode takes more time to switch between On and Off states (T_{OnOff}) than the S5, the same time for switching between Off and On (T_{OffOn}), and a similar Off power consumption (P_{Off}). Consequently, this mode is useless from an energy point of view. The nodes do not support the S3 mode (Suspend-to-RAM).

4.3.2 Future energy proportional architectures

The ARM big.LITTLE processor is an example of promising architecture from an energy point of view. It combines a low-power processor with a high-performance one to offer an heterogeneous

Table 4.1: Calibration nodes' characteristics and energy parameters for On-Off and Off-On sequences (average and standard deviation on 100 experimental measurements)

Features	Orion		Taurus		Paravance	
Server model	Dell PowerEdge R720		Dell PowerEdge R720		Dell PowerEdge R630	
CPU model	Intel Xeon E5-2630		Intel Xeon E5-2630		Intel Xeon E5-2630v3	
# of CPU	2		2		2	
Cores per CPU	6		6		8	
Memory (GB)	32		32		128	
Storage (GB)	2 x 300 (HDD)		2 x 300 (HDD)		2 x 600 (HDD)	
GPU	Nvidia Tesla M2075		-		-	
Parameters	Orion		Taurus		Paravance	
	Average	Std dev.	Average	Std dev.	Average	Std dev.
E_{OffOn} (J)	23,386	215.45	19,000	169.6	19,893	1,571.2
E_{OnOff} (J)	775.79	125.6	616.08	75.23	1,115	82.3
T_{OffOn} (s)	150	1.73	150	1.49	167.5	16.6
T_{OnOff} (s)	6.1	1.0	6.1	0.7	13	1.9
P_{idle} (W)	135	0.5	95	0.4	150	0.9
P_{off} (W)	18.5	0.4	8.5	0.3	4.5	0.6

architecture closer to power proportionality than other processors even with dynamic frequency scaling [65]. The idea consists in activating one processor at a time: either the low-power one during low workload or the powerful one during high activity.

This concept has been extended to data centers in [109] in order to build power-proportional servers. In this approach named BML (Big, Medium, Little), a computing node is composed of three processing units aiming at different levels of workload and energy consumption. It is assumed that each processing unit is able to be turned on and off independently from the others.

We take the same BML configuration that the calibration measurements presented in [110]. A Big unit corresponds to Graphene node on Grid'5000 platform, a Medium unit corresponds to a Chromebook and finally, a Little unit corresponds to a Raspberry node. We then assume, as in [110], that it exists a computing node composed with these three processing units. The required energy values for BML nodes are provided in Table 4.2. The Medium unit presents a behavior different from the two others with a $T_{OffOn} < T_{OnOff}$.

Table 4.2: Initial calibration values for independent BML units from [110]

Parameters	Big	Medium	Little
E_{OffOn} (J)	4,940	49.3	40.5
E_{OnOff} (J)	760	77.6	36.2
T_{OffOn} (s)	71	12	16
T_{OnOff} (s)	16	21	14
P_{idle} (W)	47.7	4	3.1
P_{off} (W)	8	1.9	2.2

We consider several configurations where BML components can be turned off separately or simultaneously. The considered configurations are the following:

- *AtomicBML*: the three processing units composing a BML node are turned off simultane-

ously, behaving as a single node. For this configuration, the energy values E_{OffOn} and E_{OnOff} are the sum of the three units' values, the times T_{OffOn} and T_{OnOff} are the maximum as we assume that components can be switched in parallel, and the power values P_{idle} and P_{off} are the sum.

- *OnlyB, OnlyM, OnlyL*: only one of the processing unit composing the BML node is turned off. In this case, P_{off} corresponds to the sum of the P_{off} of the turned off processing unit and the P_{idle} of the others. P_{idle} corresponds to the sum of the three P_{idle}. T_{OffOn}, T_{OnOff}, E_{OffOn} and E_{OnOff} are equal to the values in Table 4.2 of the component which is switched on or off.

- *FlexibleBML*: all possible computing units are turned off individually: if an idle period does not allow for all processing units to be turned off, only the possible ones are turned off.

These use cases represent possible configurations of future processing architectures which should get closer to energy proportionality than current ones. One can wonder if shutdown techniques can be beneficial for such architectures from an energy point of view.

4.4 Actors

In this section, we introduce and define all the actors and the constraints they are respecting in Section 4.4.1. We then explain how to combine the actors in Section 4.4.2.

4.4.1 Actor definitions

In this section, we derive several actors, assuming that we have a given knowledge about the node reservations, i.e., for each node, we have a list of intervals during which the node is in the idle state. We aim at deciding whether this node can be turned off and then back on, while respecting the constraints of the system and improving the goals. These are actor-dependent and are detailed in the next sections.

We therefore provide an entity giving advice on changing the state of a (set of) node, making sure that the overall system responds to the described constraints. This entity is called an actor. It is, here, acting on the OnOff leverage.

Basic actors

Two basic actors are used by most works in the literature: either the nodes are never shut down (No-OnOff actor), or there is no cost (time, energy, thermal) to turn on or off a node (LB-ZeroCost-OnOff actor: *Lower Bound Zero Cost OnOff actor*), making it very simple to shutdown a node (but very far from reality). In this context, the node consumes nothing when executing an On→Off or Off→On sequence. Thus, there is no cost nor time spent to switch state, and no power peak observed during the sequence. Therefore, switching on or off nodes has no impact on the system. This LB-ZeroCost-OnOff actor hence provides a theoretical lower bound on the gains that can be achieved by shutting down nodes.

Sequence-aware actors

The sequence-aware actors make sure that the sequence observed on a node or set of nodes during On→Off or Off→On sequences does not overcome the fixed constraints (time, energy, etc). Therefore, we need to record a few data for every node composing the studied case, in particular a record of the Off→On sequence and of the On→Off sequence.

Time constrained actor The SEQ-AW-T (*Sequence-Aware Time*) actor checks whether there is enough time to perform an On→Off followed by an Off→On sequence on a node, given the available time slot where the node is idle. Let T_{gap} be the size of the "gap", i.e., the interval of idle time of the node. Then, SEQ-AW-T allows us to turn off the node in this time slot if and only if $T_{\text{OnOff}} + T_{\text{OffOn}} \leq T_{\text{gap}}$, where T_{OnOff} (resp. T_{OffOn}) is the time spent by the node during an On→Off (resp. Off→On) sequence.

Energy constrained actor The SEQ-AW-E actor (*Sequence-Aware Energy*) further refines SEQ-AW-T by checking whether turning off the node is beneficial in terms of energy. The minimum time T_s of the gap is now further constrained by the energy savings:

$$T_s = \max\left(T_{\text{OnOff}} + T_{\text{OffOn}}, \frac{E_{\text{OnOff}} + E_{\text{OffOn}} - P_{\text{off}}(T_{\text{OnOff}} + T_{\text{OffOn}})}{P_{\text{idle}} - P_{\text{off}}}\right),$$

where:

- P_{idle} is the power consumption when the node is in the Idle state (unused, but powered on);
- P_{off} is the power consumption when the node is switched off (typically not null and lower than P_{idle});
- E_{OnOff} is the energy consumed during the On→Off sequence;
- E_{OffOn} is the energy consumed during the Off→On sequence.

The first term states, as for SEQ-AW-T, that at least a time $T_{\text{OnOff}} + T_{\text{OffOn}}$ is needed to turn off the node (and back on) during the idle interval. The second term ensures that there will be gains in energy: the energy saved by running at P_{off} rather than P_{idle} is $T_s(P_{\text{idle}} - P_{\text{off}})$ during the interval, but the additional energy due to the On→Off and Off→On sequences is $E_{\text{OnOff}} + E_{\text{OffOn}} - P_{\text{off}}(T_{\text{OnOff}} + T_{\text{OffOn}})$. Therefore, if $T_s < T_{\text{gap}}$, where T_{gap} is the size of the "gap", i.e., the interval of idle time of the node, then it is beneficial to turn off (at the beginning of the gap) then on (at the end of the gap) the node, in terms of energy consumption.

Electricity-aware actor

The electricity-aware actor, ELEC-SF (*Electrical Scalability Factor*), aims at ensuring the safety of the computing facility through its electrical provisioning, given that the following information is provided: how many Watts could be added (ESF-Up) or retrieved (ESF-Down) in the facility in a given duration? We call this the electrical scalability factor (ESF). For instance, between $0\ W$ and $1,000\ W$ of power usage of IT equipment (W_{IT}), 10 Watts can be added in the facility overall usage during one second, as illustrated in Table 4.3.

For IT power (W_{IT})	ESF_{Up}
$0\ W \rightarrow 1,000\ W$	$+10\ W$ during $1s$
$1,000\ W \rightarrow 10,000\ W$	$+50\ W$ during $1s$
$10,000\ W \rightarrow 100,000\ W$	$+100\ W$ during $1s$

Table 4.3: Electrical Scalability Factor illustration.

From this information, we can define the function $electricalScalabilityFactor(X)$, which returns true if the addition or removal of all nodes in set X will be supported energetically by the infrastructure and the electrical provider, i.e., if the ESF is respected.

In this context, the actor allows us to turn off and then on nodes during an idle interval if and only if the global ESF is respected for all nodes at the time of the On→Off and Off→On sequences.

Power-capping-aware actor

The POWER-CAP actor (*Power-Capping-Aware*) aims at maintaining an average power budget and guaranteeing minimal or maximal electrical power consumption. Indeed, turning on and off components could lead to hard power capping disruption.

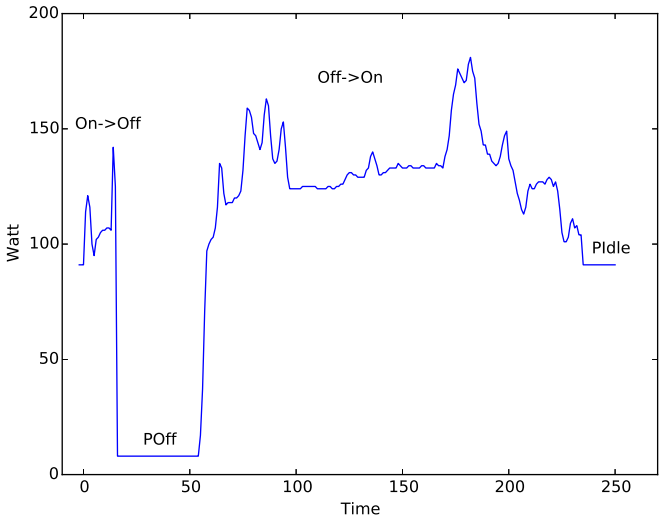


Figure 4.4: On→Off followed by P_{off}, Off→On, and finally P_{idle} section for Taurus (Grid'5000) calibrated node (average of 50 runs).

Figure 4.4 shows a set of consecutive sequences: an On→Off sequence, a section in the OFF state, an Off→On sequence, and finally a section in the idle state. These experiments represent the shutdown and boot of a node during a gap (i.e., idle interval in the schedule). All the previous actions energetically stress the node, whether it is in an upper or lower way.

A minimum power capping ($POWERCAP\_Min$) represents a constraint set by the electrical supplier, defined by providing a lower bound on power. A maximum power capping ($POWERCAP\_Max$) represents power limit fixed by the electrical provider, defined by an upper bound on power. These minimum and maximum power capping values may be a function of the time, i.e., the requirements may change in time.

We introduce the function $PowerSum(X)$, which returns the sum of the power consumed by nodes in X.

We can turn *Off* or *On* nodes in set X if and only if $PowerSum(ALL) \geq POWERCAP\_Min$ and $PowerSum(ALL) \leq POWERCAP\_Max$ at all time during the sequence.

Cooling system-aware actor

The Cool-Aw actor (*Cooling System-Aware*) accounts for the cooling system in use. Therefore, we need to record basic informations about the chosen cooling system: the instantaneous needed IT power (W_{IT}), and the Cooling Scalability Factor (CSF_{up} and CSF_{down}) for every level of cooling system, similar to the electrical scalability factor defined in Section 4.4.1.

We make the assumption that the cooling system has several working *levels*. Thus, several power levels for cooling are available in function of the IT power needed by the cooling system. For instance, between 0 W and $1,000$ W of power usage of IT equipment, one Watt can be added in the facility overall usage during one second, as illustrated for CSF_{up} in Table 4.4.

Renewable energy-aware actor

The last defined actor, Renew-E (*Renewable Energy-Aware*), assumes that we have the knowledge of the provenance of energy (green or brown) at actual time and near future (predicted). Green energy is provided with specific sources (sun, wind, etc.), while brown energy is mainly provided with fossil materials (coal, oil, etc.). The aim of this actor is to minimize the usage of brown energy. Hence, it decides to turn off nodes when brown energy can be saved. We assume that the green energy production is done on-site, for instance through photovoltaic panels. Furthermore, the facility does not sell its generated green energy. Therefore, no gain can be obtained by turning off nodes when using the green energy. A consequence of this strategy is that it will reduce the number of On→Off sequences for a same waste of usable energy.

At time t, $EnergyProv(Src, t, X)$ checks if the provenance of energy on node X is Src, where Src can be G (for green) or B (for brown). Then, at the beginning of an idle interval (time-step t_{start}, interval of duration T_{gap}), we check whether there exists a time-step t such that $t_{start} < t < t_{start}+T_{\text{gap}}$, and $EnergyProv(B, t, X)$ is true. If this is the case, then we turn off the node at time-step t_{start}.

4.4.2 Combining actors

The proposed actors can be implemented through several software components and organized in a "workflow" of pipelined components. When an On→Off possibility happens in the system, due to a gap in activity, this possibility is analyzed by each actor one by one. If each actor gives an acceptance due to the observed constraints, the On→Off sequence can be scheduled. We provide an example of combination of actors in Figure 4.5. It works as follows for a given idle interval on a node:

For IT power (W_{IT}) between	CSF_{up}	Level
$0\ W \rightarrow 1,000\ W$	$+1\ W$ during $1s$	1
$1,000\ W \rightarrow 10,000\ W$	$+10\ W$ during $1s$	2
$10,000\ W \rightarrow 100,000\ W$	$+100\ W$ during $1s$	3

Table 4.4: Cooling Scalability Factor illustration.

- SEQ-AW-T advises the provider whether there is time to turn off the node and then back on before it is in use again;
- RENEW-E may tell the provider to turn off the node, because the current energy source is brown;
- COOL-AW may prevent the provider to turn off the node if it would stress it too much in terms of temperature;
- Finally, ELEC-SF may prevent the provider to turn off the node if it would stress it too much in terms of electric power.

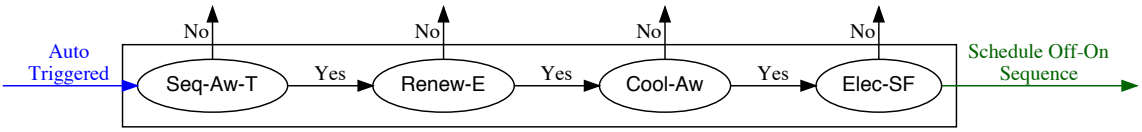


Figure 4.5: Example of auto-triggered actor combination.

From the described combination of actors in Figure 4.5, a lot of possibilities are "consistent" through the usage of the described actors, in their scope, their concerns and their combinations.

4.5 Experimental setup

To instantiate our actors in various configurations, we developed a simulator capable of replaying a real datacenter trace, with real calibrations of nodes and jobs (time, power, energy).

4.5.1 Calibration of jobs, nodes and proposed simulation

Grid'5000[7], was used as a testbed. On the Lyon site, the energy consumption of all nodes from all available clusters (Nova, Orion, Sagittaire, Taurus) is monitored through a dedicated wattmeter, exposing one power measurement (Watt) per second with a 0.125 Watts accuracy per node. Therefore, we can obtain detailed traces giving the energy consumption of jobs at any time step. We extract an average power consumption of each job. Thanks to these traces, we are able to replay in a realistic way the jobs and to simulate their corresponding energy consumption.

We monitored Taurus nodes to calibrate in time, energy and power the Off→On and On→Off sequences, as explained in Section 4.2; the results are detailed in Table 4.1.

The experimental setup must be close to the real environment where the actors want to be applied, in order to be relevant during the analysis of the usage of such actors in a possible given context. The following sections present our used experimental setups to show the relevance of the shutdown leverage.

4.5.2 Experimental setup: large-scale study

Our first experimental setup represents large scales usage of the shutdown leverage to evaluate its impact on large amount of nodes and for long periods of time. In order to provide a fair

comparison among actors, we simulate their behavior on real workload traces. The simulation tool is using calibration measurements that we performed on several servers representing different hardware architectures. Simulations combine the workload traces and the energy calibration values to compare actors.

The used workload traces come from two kinds of data centers: an experimental small-size data center of an experimental testbed and a supercomputer for bioinformatics. They provide two different utilization scenarios which exhibit different workload patterns and utilization levels.

Operational Cloud platform: E-Biothon

The E-Biothon platform is an experimental Cloud platform to help speed up and advance research in biology, health and environment. It is based on four Blue Gene/P racks and a web portal that allow members of the bioinformatics community to easily launch their scientific applications. Overall, the platform offers 4096 4-cores nodes, reaching a peak power of 56 Travel [33].

We obtained a workload trace for this platform covering from the 1st of January 2015 to the 1st of April 2016, so roughly 15 months of resource utilization. In this trace, the average size of idle periods is around 2.8 hours while the overall usage is 98%.

Experimental testbed: Grid'5000

Grid'5000 is a large-scale and versatile testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data [7]. Since 2005, the testbed offers distributed computing resources which are highly reconfigurable. Thus, it is a unique operational platform dedicated to experiments. In 2016, it consists of about 1,000 servers embedding 8,000 overall, geographically distributed on 9 sites. For our evaluation, we took the workload trace of the Rennes site from the 1st of April 2010 to the 1st of April 2016, thus representing 6 years of resource utilization on this site. During this period, the weighted arithmetic mean of the number of nodes is 149 and the average size of idle periods is around 6.17 hours. The overall usage is around 33%.

4.5.3 Experimental setup: fine grain study

To show the impact of combined shutdown actors, a second experimental setup is used. For the detailed evaluation of our various proposed actors, we extracted the real workload usage of the Grid'5000 Lyon site from October 24, 2016 to November 1, 2016, thus representing approximately one week of resource utilization on this site.

The trace only contains nodes that were used during this period, which is up to 76. We consider that all nodes in the trace have similar P_{idle}, P_{off}, Off→On and On→Off sequences. This work is focused on shutting down nodes, thus we consider that the scheduled jobs cannot be moved.

We always combine a simulation of Seq-Aw-T with all other actors in order to allow a correct execution of the actors when an On→Off followed by an Off→On sequence should occur. Therefore, the evaluation of actors can be applied on the same workload.

Figure 4.6 represents the profile of accumulated power consumption of nodes in Lyon for the extracted trace replayed with our previously exposed hypothesis. Table 4.5 presents statistics for this trace, day by day in various points of interest: number of jobs, average job consumption, and average job size. This week was chosen because of its representativeness of the workload variability that we observed overall on this platform by looking at larger traces. Indeed, for

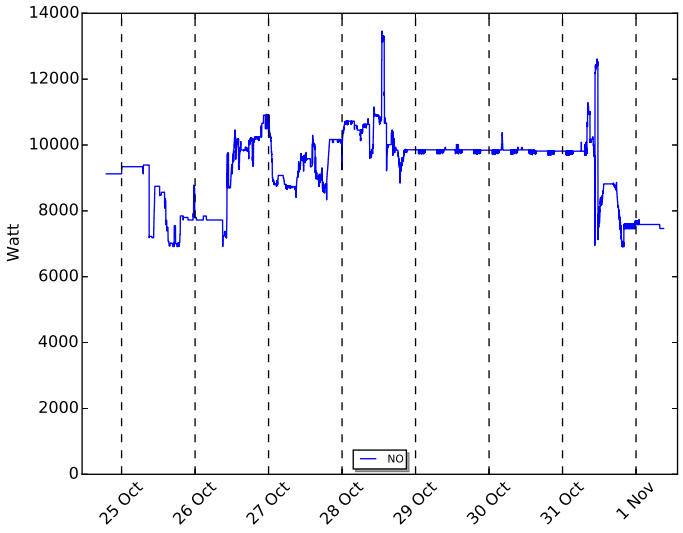


Figure 4.6: Trace replay with No-OnOff (NO).

Day	#Jobs	Average job power cons. (W)	Average job size (s)
Oct. 24 (7PM to 12AM)	33	157.91	50,401.24
Oct. 25 (Full day)	144	155.08	23,002.74
Oct. 26 (Full day)	277	159.79	12,299.06
Oct. 27 (Full day)	353	154.11	13,819.43
Oct. 28 (Full day)	318	159.96	27,286.17
Oct. 29 (Full day)	171	174.11	41,525.71
Oct. 30 (Full day)	180	174.04	39,453.67
Oct. 31 (Full day)	563	173.39	12,821.24
Nov. 1 (12AM to 8AM)	48	179.25	17,179.17

Table 4.5: Grid5000 trace statistics.

this week, the power consumption trace exhibits important peaks (Oct. 28), short peaks (Oct. 31), short climbs (Oct. 24 to 25), important climbs (Oct. 26) and sustained stable sections (Oct. 29), as shown in Figure 4.6 Variability of the trace can also be witnessed in Table 4.5 for various usages either concerning number of jobs (for instance, the differences between Oct. 25 and 28), average job consumption (for instance, Oct. 27 vs 31) or average job size variability (for instance, Oct. 26 vs 29) witnessed from one day to another.

The following sections present the results of the simulator for large scale and finer grain scale on the extracted traces with calibrations of Orion, Taurus and Paravance nodes (Table 4.1) while applying previously defined actors. Note that for the fine grain simulations, we always combine the actors (except No-OnOff and LB-ZeroCost-OnOff) with Seq-Aw-T to ensure that the node is in the On state when it starts computing on the trace (and hence that we decide to turn off the node only if it can be turned on before the end of the interval).

4.6 Impact of energy-aware actor: large-scale study

This section explores the simulation results of the shutdown actor that focuses on energy (Seq-Aw-E) with the various hardware calibrations and the large scale workload traces de-

scribed in Section 4.5.2. For every trace replay, the nodes are assumed to be homogeneous. Thus, every node of the trace is respecting the configuration of one of the calibrated nodes for each run. Such a study permits to evaluate and give tenancies about expected energy saving at large scale.

4.6.1 Experiments on actual architectures

We first examine the case of current architectures based on the calibration made on the Grid'5000 nodes and described in Table 4.1. While actor SEQ-AW-E always performs exact prediction of the future workload in order to adequately switch on and off the nodes, Aggressive actor (*Aggr.*) does not attempt to foresee the future and switches off a node as soon as it is unemployed. It does not consider the future and tries to switch off a server as soon as it is in idle state without any prediction attempt. Such an approach is expected to result in a higher energy consumption than SEQ-AW-E because many idle periods may be lower than T_s. In such cases, switching off increases the energy consumption compared to staying idle. The LB-ZEROCOST-ONOFF actor assumes that state transitions have no cost in terms of energy and time, but switched off nodes are still consuming a bit ($P_{off} \neq 0$), so the energy gains of this actor are not 100%.

Table 4.6 shows the percentage of energy that could be saved during idle periods with each actor compared to the energy consumed if nodes are never switched off. The last two columns present the average number of On-off cycles per node for the entire duration of the workload (respectively 6 years and 15 months for the two workload traces).

Table 4.6: Energy gains on idle periods and number of on-off cycles per node for current servers

Calibration	% Energy saved on idle periods			# On-Off cycles per node	
	SEQ-AW-E	*Aggr.*	LB-ZEROCOST-ONOFF	SEQ-AW-E	*Aggr.*
Grid'5000 trace, 6 years, 149 nodes on average					
Orion	85.87%	85.59%	86.29%	3,080	5,690
Taurus	90.56%	90.22%	91.05%	2,980	5,690
Paravance	96.66%	96.46%	97.00%	3,333	5,690
E-Biothon trace, 15 months, 4096 nodes					
Orion	85.18%	84.56%	86.29%	33	70
Taurus	89.83%	89.07%	91.05%	33	70
Paravance	96.03%	95.61%	97.00%	38	70

The results show that by turning off nodes, even when considering On-Off and Off-On costs, consequent energy gains can be made on real platforms. In the most unfavorable configuration (ie. Orion configuration), by using shutdown techniques, we can theoretically save up to 86% of the energy consumed while being in idle state. In the case of Grid'5000 trace, this percentage represents around 706,000 kWh for the 6 years, so roughly a cost of 70,600 euros (at a cost of 0.10 euros per kWh). For the E-Biothon trace, we can also save up to 86% of the energy consumed in the idle case, this represents 109,000 kWh for 15 months, roughly 10,900 euros of loss to keep servers idle.

The number of On-Off cycles per node reaches at the maximum $5,690$ for the 6-year Grid'5000 traces, so 2.59 per day, far less than the 50,000 start/stop cycles typically allowed by HDD manufacturers during their 5-year lifetime under warranty [96, 97]. This clearly states that even aggressive shutdown actors have no impact on disk lifetime.

It is worth noticing that significant energy gains can be performed for both traces even though they present completely different use cases. In particular, the E-Biothon trace comes from an operational bioinformatics supercomputer and although energy savings are smaller than for the Grid'5000 trace in comparison with the infrastructure size, they are still not negligible, representing around 73,680 kWh per year for the Orion case (most unfavorable case) with a basic shutdown actor like *Aggr.* (without prediction algorithm).

The energy saved with actors Seq-Aw-E and *Aggr.* are very close to the LB-ZeroCost-OnOff actor (around 2% difference in the worst case). Even without knowledge about the future (actor *Aggr.*), energy gains are quite similar. This means that even simple shutdown actors – not including workload predictions – can save consequent amounts of energy, close to the optimal bound. These results show that the energy gains of Seq-Aw-E and *Aggr.* is too close (for Orion 0.28% of difference between the actors, roughly 2,000kWh over 6 years) to justify the elaboration of a prediction algorithm: such a complex algorithm to design would only bring negligible benefits from an energy point of view.

4.6.2 Experiments on promising future architectures with improved shutdown modes

After this analysis on current hardware, we study the impact of shutdown techniques on envisioned future architectures: regular nodes with an S3 mode (Suspend-to-RAM) and power-proportional nodes. For the S3 mode, it was not available on the Grid'5000 servers used for our calibration measurements. However, one can assume that when this technology will become more mature and used in hardware composing datacenters, it could present an appealing trade-off between energy consumption (for switching off nodes) and reactivity (for their short switching time T_{OnOff} and T_{OffOn}).

Table 4.7: Assumed energy calibration on envisioned nodes with S3 mode

Parameters	Values
E_{OffOn} (Joules)	2,300
E_{OnOff} (Joules)	2,300
T_{OffOn} (seconds)	10
T_{OnOff} (seconds)	10
P_{idle} (Watts)	135
P_{off} (Watts)	37
T_s (seconds)	20

After discussing with people from the Leibniz Supercomputing Centre operating the SuperMUC HPC system [2] on which S3 is available, we get quantitative indications stating that, the power consumption on S3 mode is about twice bigger than when regularly switched off, and that the On-Off and Off-On sequences are close in terms of duration. So, based on an Orion calibration from our measurements (as presented in Table 4.1), we assume that an envisioned node with S3 mode would present the energy calibration parameters shown in Table 4.7.

Table 4.8 compares the energy gains with this S3 mode to a regular shutdown (S5) for the Orion case as shown in previous results presented in Table 4.6. Results indicate that S5 mode allows for more energy savings than S3 mode on these traces. Indeed, idle periods are long enough to easily compensate for the energy and time costs of switching between states.

Table 4.8: Energy gains on idle periods and number of on-off cycles per node with an envisioned S3 mode

Calibration	% Energy saved on idle periods			# On-Off cycles per node	
	SEQ-AW-E	*Aggr.*	LB-ZEROCOST-ONOFF	SEQ-AW-E	*Aggr.*
Grid'5000 trace, 6 years, 149 nodes on average					
Orion	85.87%	85.59%	86.29%	3,080	5,690
S3	72.51%	72.48%	72.59%	3,343	5,690
E-Biothon trace, 15 months, 4096 nodes					
Orion	85.18%	84.56%	86.29%	33	70
S3	72.31%	72.26%	72.59%	52	70

However, the consumption while in S3 mode (P_{off}) is, in this case, too high for competing with the energy saving percentage reached with a regular shutdown. For the S3 mode to be beneficial for workloads with consequent idle periods, it is thus required to diminish its energy consumption (P_{off}) rather than reducing the switching costs (and thus T_s).

4.6.3 Experiments on promising future energy-proportional architectures

Concerning power-proportional nodes, results are provided by Table 4.9 based on the calibration values and configurations exposed in Section 4.3.2. As expected, for both actors, when only one component over the three units composing the processing node can be switched off (cases OnlyL, OnlyM and OnlyB), it consumes more than when the three can (AtomicBML and FlexibleBML).

Moreover, switching off only the Little or the Medium components result in little energy savings (less than 9%). This explains that FlexibleBML – able to switch off the three components independently or together – brings minor improvements compared to AtomicBML, where the three components are always switched jointly (0.6% difference on actor SEQ-AW-E). For actor *Aggr.* and configuration FlexibleBML, it gives the same results as configuration AtomicBML because this actor automatically switches down all the components whenever possible, so it produces the same behavior as AtomicBML in this case.

Table 4.9: Shutdown impacts with an energy-proportional architecture

Calibration	% Energy saved on idle periods			# On-Off cycles per node	
	SEQ-AW-E	*Aggr.*	LB-ZEROCOST-ONOFF	SEQ-AW-E	*Aggr.*
Grid'5000 trace, 6 years, 149 nodes on average					
AtomicBML	77.66%	77.51%	77.91%	3,495	5,690
OnlyL	2.00%	2.00%	2.007%	5,690	5,690
OnlyM	8.93%	8.93%	8.941%	5,690	5,690
OnlyB	72.19%	72.05%	72.44%	3,511	5,690
FlexibleBML	77.72%		77.91%	5,690	
E-Biothon trace, 15 months, 4096 nodes					
AtomicBML	77.22%	76.93%	77.91%	42	70
OnlyL	2.00%	2.00%	2.007%	70	70
OnlyM	8.93%	8.93%	8.941%	70	70
OnlyB	71.78%	71.50%	72.44%	42	70
FlexibleBML	77.72%	76.93%	77.91%	70	70

Similarly to previous results, we observe that actors Seq-Aw-E and *Aggr.* give comparable results (0.79% of difference at maximum), and they are close to LB-ZeroCost-OnOff actor (0.98% at most). Designing an accurate workload prediction algorithm has therefore little interest for energy saving purpose. In the same way as previous results also, the number of On-Off cycles is small enough not to modify the hardware life expectancy.

4.7 Impact of each actor: finer grain study

Previous large-scale simulations showed us that the Seq-Aw-E actor is relevant to save large amounts of energy in various contexts. This section presents the results of simulation for all actors presented in Actors definition section (Section 4.4). All graphs in this section represent a trace replay for one or multiple combined actors with specific inputs. The one week long chosen trace for this replay is detailed in section 4.5.3. Table 4.10 presents the energy consumption in Joules of all actors in the figures included in this section.

4.7.1 Sequence-aware actors: Seq-Aw-T and Seq-Aw-E

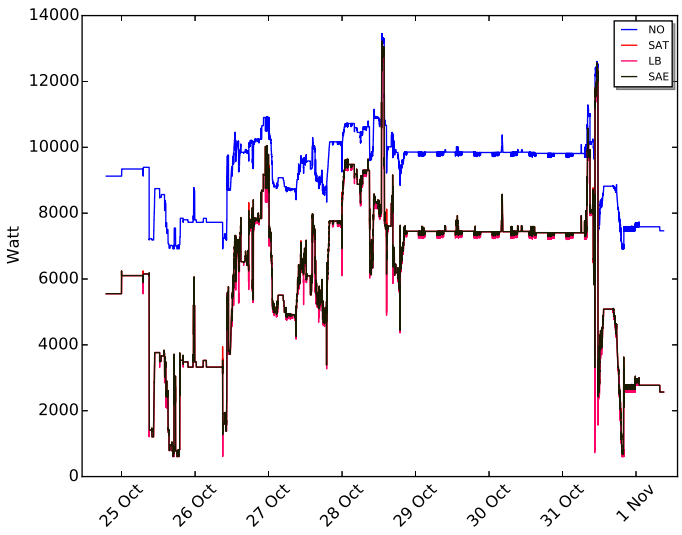


Figure 4.7: Trace replay No-OnOff (NO), Seq-Aw-T (SAT), Seq-Aw-E (SAE) and LB-ZeroCost-OnOff (LB).

Figure 4.7 shows results of No-OnOff, Seq-Aw-T, Seq-Aw-E, and LB-ZeroCost-OnOff actors, all previously defined section 4.4. Between the two sequence-aware actors, we can witness minor differences on the complete replay, for instance on Oct. 31 at 4:40, where Seq-Aw-T allows more Off→On sequences to be scheduled. This is the reason why the difference between the overall energy consumption of these actors is thin. Both of these actors lead to major energy savings, respectively 34.00% and 33.99% of energy savings compared to No-OnOff, as shown in Table 4.10. In comparison with the No-OnOff trace replay, major power peaks are witnessed because of the application of these actors. For instance, on Oct. 31, after a peak of work around $12000W$, a very low peak is witnessed around $1000W$. Such behaviors could lead to abrupt thermal changes and thus to hotspots and cool spots, so to possible deterioration of the computing nodes.

Actor	Total energy consumed (Joules)	# On→Off & Off→On	% Saved
No-OnOff	6,083,698,688	0	0,0
LB-ZeroCost-OnOff	3,983,408,384	1794	34.52
Seq-Aw-T	4,015,736,064	964	33.99
Seq-Aw-E	4,015,201,024	844	34.00
Elec-SF max	4,611,556,352	819	24.19
Elec-SF $max/2$	5,078,084,608	767	16.53
Elec-SF $max/4$	5,461,449,728	647	10.22
Elec-SF $max/8$	5,828,239,360	451	4.19
Power-Cap 2000 min	4,401,067,520	855	27.65
Power-Cap 4000 min	4,593,668,096	761	24.49
Power-Cap 6000 min	5,059,857,408	617	16.82
Renew-E	4,132,427,520	423	32.07
Cool-Aw split 2	4,927,842,304	851	18.99
Cool-Aw split 7	5,054,783,488	831	16.91
All	5,386,375,168	315	11.46

Table 4.10: Trace replay's energy consumption (in Joules), number of (On→Off, Off→On) sequences added with actors, and percentage of energy saved compared to No-OnOff.

We also compare with LB-ZeroCost-OnOff, the actor with immediate On→Off with zero cost, and we see that there is no significant difference in energy consumption observed when we accurately describe the cost of On→Off and Off→On sections. However, the number of On→Off that are effectively triggered is significantly lower, since we would not be able to resume the execution in practice if we were using the LB-ZeroCost-OnOff actor.

4.7.2 Electricity-aware actor: Elec-SF

This section presents the results for Elec-SF, the actor that aims at ensuring the respect of the electrical provider through scalability factors. ESF_{Max}, the maximal electrical scalability factor, is set to the maximum value witnessed during the No-OnOff replay (for ESF_{Up} and ESF_{Down}). For other Elec-SF replays, we divided ESF_{Max} by a factor to simulate more constrained electrical context.

Figure 4.8 presents No-OnOff, Seq-Aw-T and Elec-SF with ESF set to max values witnessed during the No-OnOff replay. We can note that Elec-SF does not give the same results as Seq-Aw-T, 33.99% and 24.19% of energy savings compared to No-OnOff as showed in Table 4.10, respectively. Thus, from extracted ESF factors from No-OnOff, we cannot get the same results as Seq-Aw-T.

Figure 4.9 presents No-OnOff, Seq-Aw-T and Elec-SF with ESF_{Max} divided by $(1, 2, 4, 8)$. One can note that the higher the ESF factors, the closest to Seq-Aw-T we can get. For instance, around Oct. 25 at 11:45, the one with the lowest overall power usage (thus with the highest number of Off→On sequences allowed) is the Seq-Aw-T replay, then we have ESF_{Max}; $ESF_{Max}/2$ comes third and so on until $ESF_{Max}/8$, which is merged with No-OnOff. The influence of Elec-SF could also be clearly witnessed around Oct. 31 at 4:40, where Seq-Aw-T is the only actor allowing such an important peak, while none of ESF actors can allow such a behavior, because of fixed constraints.

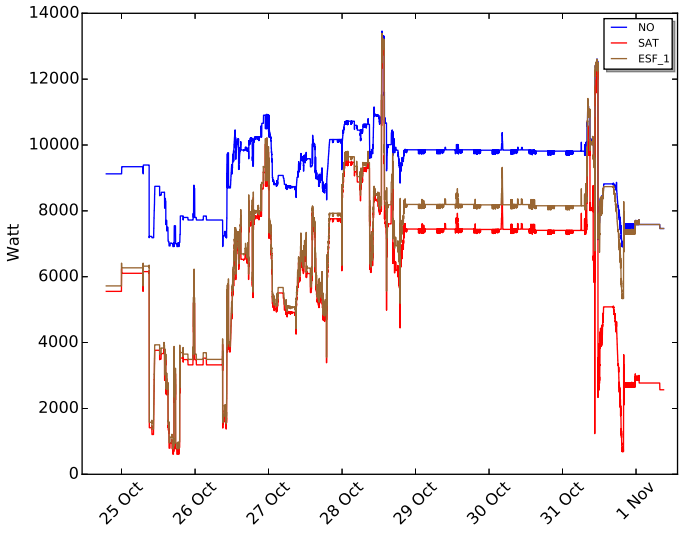


Figure 4.8: No-OnOff (NO), Elec-SF with max factor (ESF_1), and Seq-Aw-T (SAT).

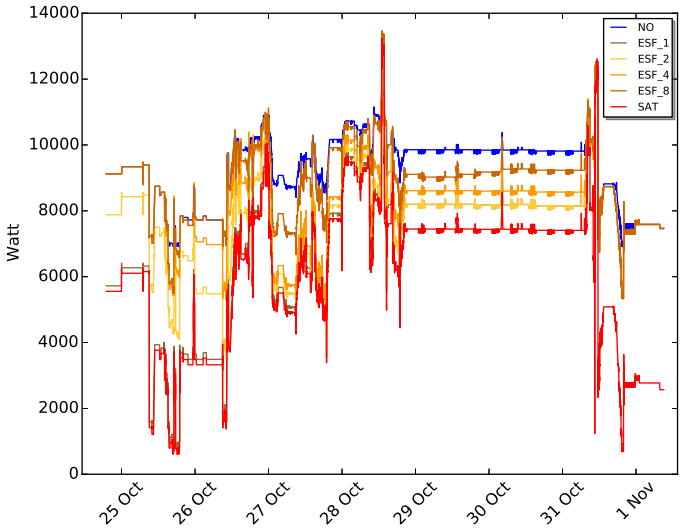


Figure 4.9: No-OnOff (NO), Elec-SF with all factors (ESF_1, ESF_2, ESF_4, ESF_8), and Seq-Aw-T (SAT).

4.7.3 Power-Cap

This section presents the results on the Power-Cap actor. We set a maximum and a minimum power cap throughout the simulation. We then modulate the minimal power cap to see how it acts with the trace replay. As a reminder, to only evaluate the shutdown leverage, scheduled jobs are fixed. Thus, we did not vary the maximal power cap because it highly depends on jobs and also because the difference between P_{idle} and P_{off} is more important than the difference between the peak witnessed during the Off→On or On→Off sequences and P_{idle}.

Figure 4.10 shows results of No-OnOff, Seq-Aw-T and Power-Cap with 2000, 4000 and 6000 as $POWERCAP\_Min$. Even with the highest minimum power cap, here $6000W$, we still make important energy savings (around 16.82 % compared to No-OnOff). The stratified

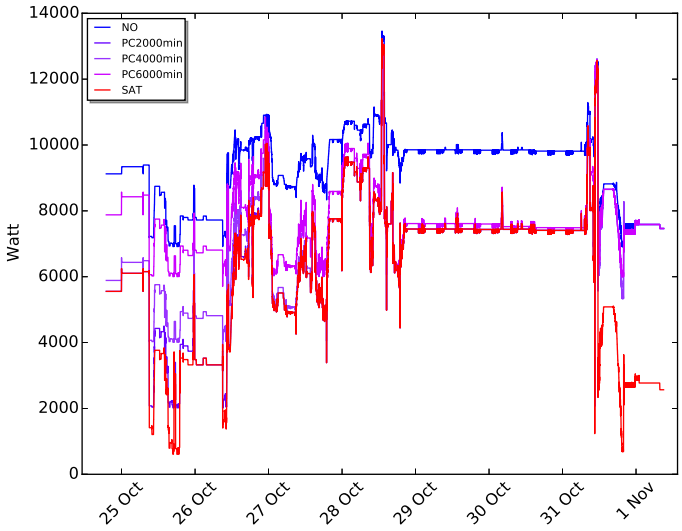


Figure 4.10: No-OnOff (NO), Power-Cap (with $POWERCAP\_Min = 2000, 4000, 6000$) and Seq-Aw-T (SAT).

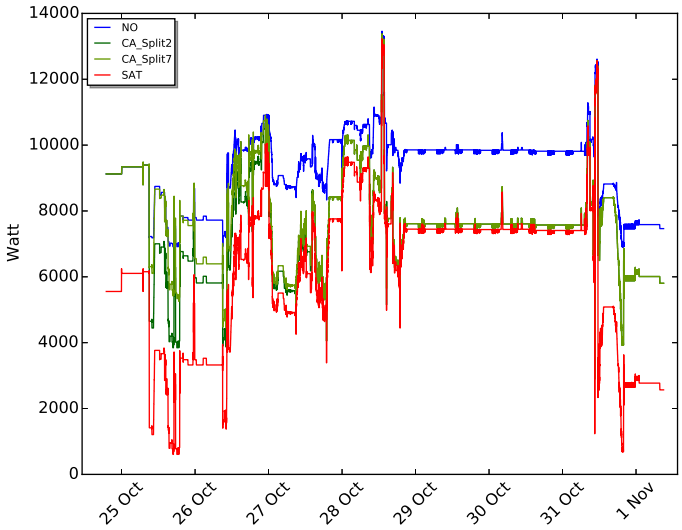


Figure 4.11: No-OnOff (NO), Cool-Aw (CA_Split2 and CA_Split7), and Seq-Aw-T (SAT).

power usage for every respected power cap was expected. In fact, a lower power cap permits more Off→On sequences to be scheduled and thus, more energy savings. The lowest cap constraint ($2000W$) shows that we can respect a minimum power capping and still have a close to minimum consumption.

4.7.4 Cool-Aw

Figure 4.11 represents the replay with No-OnOff, Seq-Aw-T, and Cool-Aw actors with two different set-ups. With *Split*7, we simulate a "smooth" scalability with 7 levels. We set the upper class (from 14000 W to 12000 W, noted [14000 : 12000], class 1) CSF_{Max} to ESF_{Max}.

Then, every 2000 W size class under it divides ESF_{Max} by i, with i the class number. For instance, the [12000 : 10000] class has a CSF_{Max} factor of $ESF_{Max}/2$, until the [2000 : 0] class with a CSF_{Max} factor of $ESF_{Max}/7$. Second, for *Split*2, two levels are set. CSF_{Max} of [14000 : 7000] is set at ESF_{Max} and [7000 : 0] is set at $ESF_{Max}/7$, which represents a more abrupt set-up. The logic is the same for CSF_{Min}.

*Split*2 allows more On→Off sequences to be scheduled, and thus gets better energy savings. *Split*2 stays longer with ESF_{Max} as the CSF_{Max} factor. For instance, note that from Oct. 25 at 7:00 AM to Oct. 27, *Split*2 is closer to Seq-Aw-T whereas *Split*7 is above both of them in Figure 4.11. Such a behavior is due to a less constrained setup in *Split*2 in the upper classes.

4.7.5 Renew-E

Figure 4.12 presents an example of the usage of Renew-E, Seq-Aw-T and No-OnOff. The provenance of energy is "Green" from start to Oct. 29 at 10:00 AM, "Brown" the rest of the time. As a reminder, this actor minimizes the usage of "Brown" energy by scheduling an On→Off sequence on a node if its current idle section contains "Brown" energy. This is why from start to Oct. 29 at 10:00 AM, almost no node is turned off (Renew-E very close to No-OnOff). The shift between Renew-E and No-OnOff at the beginning means that a few nodes are not used in the "Green" section. Around Oct. 29 at 10:00 AM, nodes start to shutdown due to the shift of the provenance (from "Green" to "Brown").

Figure 4.13 presents a typical usage of renewable energy. We set "Green" provenance during the day (from 7:00 AM to 7:00 PM) and "Brown" provenance at night. We can see that such an actor with this input is very close to Seq-Aw-T. "Green" periods can be witnessed for example Oct. 24 at 10:45 PM or Oct. 28 at 8:00 PM (basically where Seq-Aw-T is not fused to Renew-E). One can note that the energy benefits of Renew-E (32.07%) are very close to Seq-Aw-T (33.99%) with 2.2 times less On→Off sequences scheduled (it means that the Lyon site from Grid'5000 is extensively used during the day).

4.7.6 Combining the actors

Figure 4.14 presents all the actors previously exposed (Elec-SF with ESF_{Max}, Cool-Aw with *Split*7, Renew-E with DayNight, Seq-Aw-T and No-OnOff) and "All" is the combination of all of them. The combination of all the actors, previously described section 4.4.2, matches a behavior of one of the actors that is part of the combination. For instance, around Oct. 28 at 8:00 PM or Oct. 25 at 10:45 PM, we recognize the behavior of Renew-E where nodes stay up during green provenance. At the beginning, it matches the behavior of Cool-Aw with *Split*7 and it is very constrained at the beginning. Between Oct. 29 at 10:00 AM and Oct. 31 at 2:45 AM, we recognize the constraints set by Elec-SF with ESF_{Max} not being able to have the same gain as Seq-Aw-T. And finally, the behavior around the peak on Oct. 31 at 4:40 where "All" cannot go as low as Renew-E or Seq-Aw-T is similar to the behavior seen with Cool-Aw and Elec-SF.

While Figure 4.14 presents all actors independently in a defined configuration and their combination (All), Figure 4.15 progressively combines the actors together. For instance, "SAT_RE" represents the combination of Seq-Aw-T and Renew-E actors, and "SAT_RE_CA_ESF" corresponds to "All" in Figure 4.14. Table 4.11 represents the energy consumption and the number of On→Off sequences scheduled during the combined actors of Figure 4.15.

One can note that each added actor brings more constraints and thus allows less On→Off sequences to be scheduled, compared to the previous combination, as shown in Table 4.11. Thus,

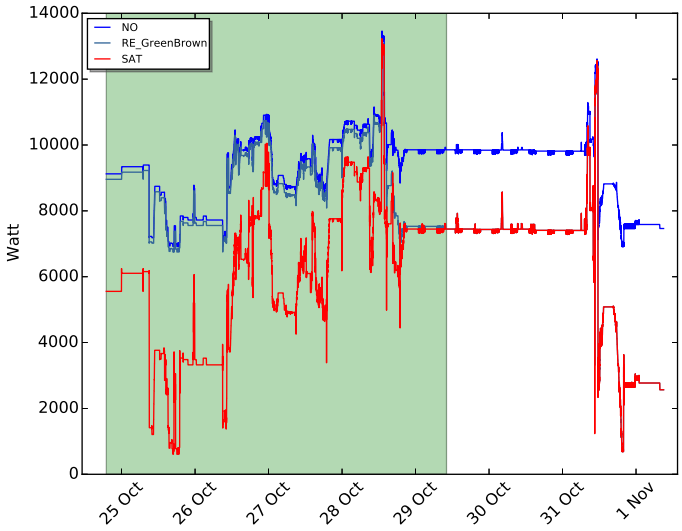


Figure 4.12: No-OnOff (NO), Renew-E (RE_GreenBrown, green then brown energy) and Seq-Aw-T (SAT).

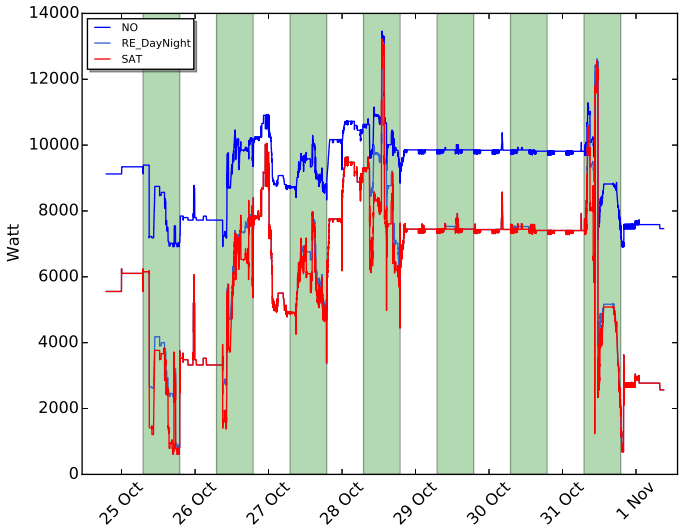


Figure 4.13: No-OnOff (NO), Renew-E (RE_DayNight, alternating green and brown energy) and Seq-Aw-T (SAT).

Combined actors	Total energy consumed (J)	# On→Off
Seq-Aw-T	4,015,736,064	964
Seq-Aw-T, Renew-E	4,132,487,936	440
Seq-Aw-T, Renew-E, Cool-Aw	5,162,120,192	342
Seq-Aw-T, Renew-E, Cool-Aw, Elec-SF (All)	5,386,375,168	315

Table 4.11: Progressively combined actors.

the chosen combination of actors does have an effect on energy consumption and the number of scheduled sequences.

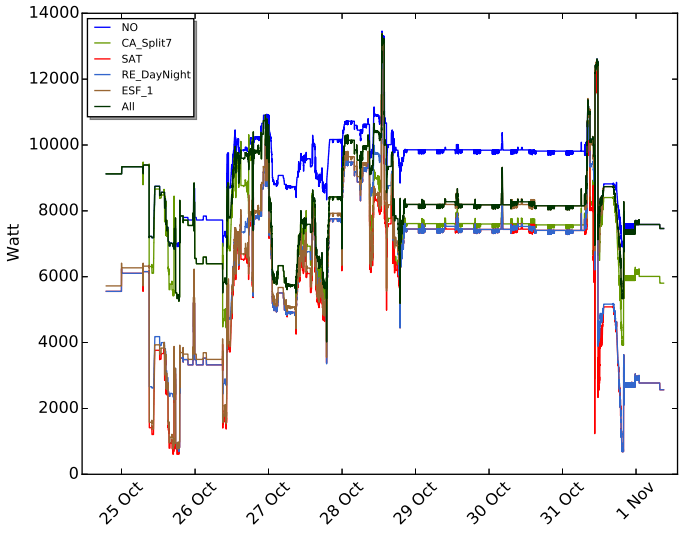


Figure 4.14: Independent actors and all combined actors (All).

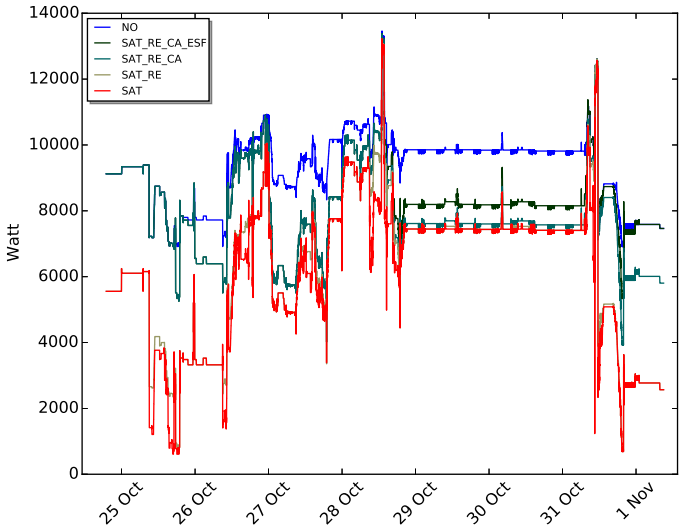


Figure 4.15: Progressively combined actors.

4.8 Conclusion

The energy efficiency of servers is increasing with Moore's law. Yet, due to an increased demand for Internet-based services, the energy consumption of large-scale systems keeps growing and it is becoming more and more a worrying concern. Although shutdown techniques are available to reduce the overall energy consumption during idle periods, they are rarely employed because of their supposed impact on hardware.

Simulation results combining real workload traces and energy calibration measurements with an energy focused actor, Seq-Aw-E, allows us to draw several conclusions. We showed that this energy-focused actor can save, even in production data centers, important amounts of energy otherwise wasted during idle periods. This conclusion remains true for envisioned future hardware with power-proportional processing units. This study also showed that reducing the con-

sumption while in Off state has a greater impact on energy savings than reducing the switching energy and time costs between On and Off states. For this reason, S3 (Suspend-to-RAM) and S4 (Suspend-to-Disk) states are currently not beneficial in terms of energy consumption.

Although Seq-Aw-E permits great energy savings, switching on and off large scale infrastructures can be a real challenge due to several other constraints: temperature, power capping, renewable energy provision, etc. While it is often assumed that nodes can be turned off at no cost, we explore realistic scenarios where several constraints (power capping, electricity, thermal) may prevent us from turning off a node at a given time step. We formally define actors targeting various scenarios. Furthermore, we explain how these actors can be combined together. A possible usage of these actors is illustrated through a set of simulations on a real workload trace, showing the gain in energy that can be achieved given the constraints on the platform, and providing clear guidelines about when a node can be turned off. Overall, the gain of the non-realistic actor where nodes are instantaneously turned off during an idle period is very small over the sequence-aware actor that turns off a node only if there is time to turn it on again before the next computation, and accounts for the power consumption during the Off→On and On→Off sequences. Other actors (*electricity-aware, power-capping-aware, cooling system-aware, renewable energy-aware*) further constrain the number of Off→On, hence leading to more energy consumed, but better matching real-life scenarios.

Applying and following our previously proposed methodology permits us to underline, model and evaluate the shutdown leverage. By evaluating, underling and modeling the costs of such a leverage, we were able to create a combinable and extensible set of actors that answer a wide range of concerns such as power capping, energy budget or renewable energy usages. Each actor is focused on one constraint. With this study, we showed that it is possible to use a leverage to minimize a metric, while ensuring and respecting fixed constraints through simple actors that could be combined at ease. Although this actor combination allows us to respect constraints like power capping and to reduce energy, it is only focused on one leverage, here shutdown. This is a first building block of a more general approach that could include other leverages, such as number of threads or number of process throttling.

Chapter 5

Combining multiple leverages

Contents

In the previous chapter, we proposed a complete set of models to answer multiple constraints that could happen during the usage of multiple nodes of a facility at the same time. These models can be composed and activated at ease. Even if they could be extended to other constraints, they are not made to answer the energy effective usage of multiple leverages. In this chapter, we propose a solution to combine and use multiple leverages at the same time and use the right set of combination to answer the chosen constraints while being energy efficient.

Energy consumption is one of the major challenges of modern data centers and supercomputers. By applying Green Programming techniques, developers have to iteratively implement

and test new versions of their software, thus evaluating the impact of each code version on their energy, power and performance objectives. This approach is manual and can be long, challenging and complicated, especially for High Performance Computing applications.

In this chapter, we formally introduce the definition of the Code Version Variability (CVV) leverage and we present a first approach to automate Green Programming (*i.e.,* CVV usage) by studying the specific use-case of an HPC stencil-based numerical code, used in production. This approach is based on the automatic generation of code versions implied by a Domain Specific Language (DSL) and on the automatic choice of code version through a set of actors. Moreover, a real case study is introduced and evaluated though a set of benchmarks to show that several trade-offs are introduced by CVV. Finally, different kinds of production scenarios are evaluated through simulation to illustrate possible benefits of applying various actors on top of the CVV automation.

5.1 Introduction

Taking into account energy issues while programming a software is often called *Green Programming* (GP). However, on one hand, by using such a technique, a developer has to write and handle multiple versions of a code, and he has to compare them manually to finally choose the one which suits the best his constraints and objectives (*e.g.,* energy, power, performance etc.). On the other hand, the growth of supercomputing capabilities increases both the energy consumption and the complexity of supercomputer usage, which makes difficult and very technical the development of applications on such machines. In such a complex context, it is even harder for a green programmer to deal with the generation, the comparison and the choice of the version of code while taking into account modular constraints. Moreover, these constraints are related to HPC systems administrators more than application developers such as, for example, constraints related to contracts with electrical providers.

In this chapter, we propose three main contributions:

- a complete process toward automated Green Programming for production numerical simulations;
- a real case-study of our automated process to show its applicability;
- and a set of evaluations of our case-study to show both the relevance of the CVV leverage for better trade-offs between metrics, and the percentage gain by using our Green Programming automation.

The remaining of this chapter is structured as follows. Section 5.2 presents the complete automated process to take advantage of the CVV leverage. A complete case-study is then detailed in Section 5.3. Sections 5.4 and 5.5 respectively detail the experimental setup and our set of evaluations onto our case-study. Then, the evaluation of the CVV leverage through our automatic solution is presented in Section 5.6, where we show that the automatic use of the CVV leverage on production HPC applications is relevant. Finally, Section 5.7 concludes this work.

5.2 Toward CVV Leverage Automation

In this section are presented the first two contributions of this chapter which are, first, the introduction of the Code Version Variability Leverage, and second, a complete process for its

usage automation onto production runs of a HPC application.

5.2.1 Code Version Variability Leverage

Considering that a given application could be implemented in various ways, we consider that having the choice between code versions is also a leverage. We call this leverage the Code Version Variability Leverage (CVV).

Definition 4 *Considering a given application $\mathcal{A}$, the Code Version Variability (CVV) leverage $L_{\mathcal{A}}$ is defined as $L_{\mathcal{A}}(S_{\mathcal{A}}, s_c, f_{\mathcal{A}})$, where $S_{\mathcal{A}} = \{v_0, \ldots, v_n\}$ is the set of available code versions of $\mathcal{A}$, $s_c = v_c$ is the current selected code version, and $f_{\mathcal{A}}$ is a way to change the current code version (*e.g., *executing a different binary).*

The CVV leverage is used in the specific case of HPC applications where the different code versions are in fact representing different parallel implementations.

We do not address the case where $f_{\mathcal{A}}$ is called during the execution of $\mathcal{A}$. This is left for future work. Instead, we consider that $f_{\mathcal{A}}$ can be called between two production runs of $\mathcal{A}$.

5.2.2 Green Programming automation: from generation to usage

Green programming (GP) consists in changing the way an application is implemented to improve its energy efficiency (energy consumption, but also power-related metrics etc.) Thus, automatic generation of several code versions (CVV) is the first necessary step to simplify GP.

However, in practice, particularly in the context of HPC applications, GP can be very difficult to apply. Actually, implementing a single version of a large scale parallelized HPC application is a long and difficult task, thus implementing multiple versions become almost infeasible. Moreover, when considering GP, the entire development process is left to the application developer.

For this reason, we propose in this chapter a complete automated process to take advantage of the CVV leverage. This process is depicted in Figure 5.1.

The CVV automation process is composed of three different phases. The first phase is responsible for the automatic generation of code versions. To do so, we propose to use Domain Specific Languages (DSLs). Among existing solutions to ease HPC programming, Domain Specific Languages target a specific domain, in opposition to general purpose (parallel) languages. By explicitly knowing the targeted domain, DSLs are able to automatically generates very efficient HPC codes [36, 86, 95, 102]. Most of the time, DSLs are used to generate the code that reaches the smallest execution time for a given application and a given hardware architecture. We use DSLs as a mechanism to generate multiple versions of a code instead of a single one, thus creating the set of states for the CVV leverage, represented by different squared colors in Figure 5.1.

The second phase of the CVV automation process is to use a given subset of production runs of an application to combine leverages, thus building a *knowledge*, which is complete at t_k. The number of runs needed to reach a complete knowledge t_k depends on the prediction degree handled in the knowledge building process: from "Null", where all leverages combinations has to be performed, to "High" where all of the knowledge is present from start (without the use of any previous run). One can note that the knowledge is built upon a given set of metrics.

The knowledge built in the second phase is then used within the third phase, for any new production run that happens after t_k, to take decision regarding the code version to use for this new production run according to the current constraints. The element which is responsible for

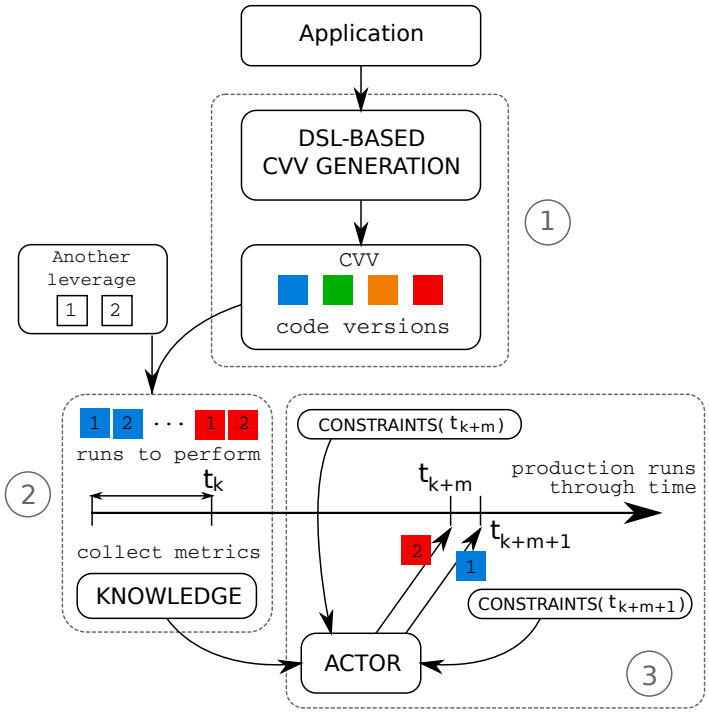


Figure 5.1: Automation process of the CVV leverage composed of three phases, one to generate, thanks to a DSL, code versions of the application, one to build the knowledge, and finally one to choose, thanks to an actor, the best trade-off for each next production run.

this decision is called an *actor*. An example of actor is the *OnDemand* linux governor which chooses the DVFS current state depending on the current system load[1]. We previously defined actors in the description of our proposed methodology in Chapter 3.

Details on the specific case-study considered will be given in the next section. However, one can note that the presented automation process is studied for *regular* production numerical simulations only. A regular application is a simulation where performed computations are always the same, whatever values of input and parameters are. This property is important to be able to re-use the knowledge built in the second phase.

Finally, as the Green Programming automation is targeted by the above process, four types of metrics (that will be detailed in the next section) are considered within the knowledge, one related to execution time, one related to energy consumption, and two related to power consumption. Considered constraints will also be related to these metrics.

5.3 Case study description

In this section is described the case-study addressed within this chapter. Regarding the automation process depicted in Figure 5.1, this section presents first the application use-case, second the DSL used to generate CVV states, third how the knowledge is built and used by actors, and finally which constraints are handled.

[1]https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

5.3.1 FullSWOF2D application

As already explained, the automation process presented in the previous section targets regular production numerical simulations. A numerical simulation simulates a physical phenomenon by approximating the exact solution of partial differential equations through a set of numerical schemes (computations). A numerical simulation discretizes the time through a time loop. At each time iteration, a set of numerical computations are applied onto the entire (or a subset) discretized space domain (namely a mesh). A numerical simulation is typically composed of (i) a number of iterations, (ii) a mesh size, (iii) a set of numerical parameters (single numerical values) and (iv) a set of input data sets representing physical quantities (*e.g.,* speed, pressure etc.). These physical quantities are mapped onto the mesh.

For a given domain size (*i.e.,* mesh size), a production numerical simulation is used many times by physicists, modifying input data sets and numerical parameters, to be as close as needed to the real phenomenon to understand it.

A numerical simulation can be regular or irregular. Here, regular simulations are handled. More particularly, stencil-based numerical simulations are considered. Thus, the same set of computations are performed whatever numerical values of input data sets and parameters are. As a result, by considering the same set of machines (*i.e.,* same cluster) and the same input size, performance behavior of stencil-based codes stays the same[2]. This makes possible to reuse the knowledge built within the automation process for many production runs.

As an example of production numerical simulation, we consider FullSWOF2D[3] [29] (denoted *FS2D*), developed at the MAPMO laboratory, University of Orléans, France. FS2D consists in solving the Shallow Water equations (two dimensional Navier-Stokes equations) using finite volumes methods especially chosen for hydrodynamic purposes (transitions between wet and dry areas, small water heights, etc.). FS2D is a complex numerical simulation composed of 32 stencil kernels and 66 local kernels [14, 30].

As an illustration, in production, FS2D will be run many times with the same input size. Actually inputs of FS2D are 8 numerical parameters (*e.g.,* hydraulic conductivity, water viscosity, pressure etc.), and 6 input data sets (*e.g.,* rain, speed in each dimension etc.). Each parameter and data set can be initialized in very different manners to study different physical cases (already flooded grounds, dry grounds etc.). When considering simply 2 possible values for each parameter and 2 possible input data sets, the number of possible runs is the Cartesian product $2^8 \times 2^6 = 2^{14} = 16,384$. This illustrates that a production numerical simulation can be used many times using the same input size.

FS2D will be the considered application for the rest of this chapter.

5.3.2 The Multi-Stencil Language

The domain specific Multi-Stencil Language (MSL) [14, 30] enables to automatically generate multiple HPC code versions of a multi-stencil numerical simulation from a lightweight data-driven description of a numerical application and a set of sequential kernel codes. The semantic and performances of MSL have been shown in [30]. MSL is used to generate four HPC code versions of FS2D, thus producing the set of states $S_{\mathcal{A}}$ of the CVV leverage.

These four versions are based on two different parallelization techniques. The first technique, namely data parallelization, divides the studied domain (data) into equally balanced

[2] `http://www.agner.org/optimize/instruction_tables.pdf`
[3] `http://www.univ-orleans.fr/mapmo/soft/FullSWOF/`

sub-domains. Each sub-domain is computed by one computational resource (typically a core) and communications between resources are added to perform correct computations. The second technique, namely task parallelization, divides a program into sub-tasks. Each task is computed by one computational resource and task dependencies are introduced to respect computation order. The scheduling of task dependencies can be statically computed before the execution, or can be dynamically decided at runtime. In MSL these techniques are implemented by using the Message Passing Interface (MPI) and the OpenMP Application Programming Interface. The four code versions produced by MSL are: (1) *MpiBase*, where data parallelization is applied by domain decomposition and by using MPI; (2) *MpiOmpFor*, where data parallelization is introduced at two different levels, first, by domain decomposition with MPI, and second, by using parallel loops of OpenMP; (3) *MpiOmpForkJoin*, where both data and task parallelization techniques are combined. The adopted task parallelization technique is a static fork/join scheduling implemented using OpenMP; and finally (4) *MpiOmpDyn*, where both data and task parallelization techniques are also combined, but where the adopted task parallelization technique is the dynamic scheduling of tasks introduced in OpenMP 4.5[4].

One can note that these four code versions represent different approaches to parallelize the code. Many other code versions could be studied such as versions using various cache optimizations, different types of data, etc. These four versions, though, are difficult to write by hand, thus being a pertinent case-study for GP automation.

5.3.3 Knowledge, actors and constraints

To entirely explain the case-study addressed within this chapter, it is needed to describe how the knowledge is built and used by the automation process.

First, as depicted in Figure 5.1 the knowledge is built by using a certain number of production runs until t_k is reached, which means that the knowledge is complete. The number of runs to perform before reaching t_k depends on the number of possible combinations when exploring a set of leverages. In this chapter three different leverages are considered. The first one is the CVV leverage described in Section 5.2, the last two ones are the number of MPI processes and OpenMP threads for a given parallel application on a given subset of nodes. These leverages have already been used in [6, 70, 75]. Because the literature (previously developed in section 2.4) often tightly couples these two leverages, we will treat it as a unique leverage. As an example, our complete knowledge (combinations of CVV and #Processes/#Threads) when 12 cores are available per machine (Table 5.1 of Section 5.4.1) contains 55 production runs. As illustrated before, a very light use of FS2D in production already leads to 16,384 runs. This shows that our methodology is realistic and feasible in our case-study.

Of course, when increasing the number of leverages, the size of the knowledge to build also increases. For this reason, actors could be more or less intelligent and could need a smaller knowledge to take a good decision (*e.g.,* machine learning techniques). This type of actors will be simulated during our evaluations in Section 5.5.

For each of the production runs used to build the knowledge (before t_k), four metrics are collected. The first metric is the *Execution Time*, denoted *time*. It measures the entire execution time of one job, including initialization time.

The three remaining metrics are energy-related metrics. To define these metrics, we first need to introduce notations:

[4] `http://www.openmp.org/mp-documents/openmp-4.5.pdf`

- N is the number of computing nodes used by a job;
- $T = \{t_0, \ldots, t_{n-1}\}$ is the set of n timestamps of energy consumption measurements of a job; t_0 and t_{n-1} represent the starting and ending timestamps, respectively;
- p_j^i, where $i \in [0, N-1]$, and $j \in [0, n-1]$, represents the power consumption (in Watts), of a node i for the timestamp t_j;
- $P_j = \sum_{i \in [0,N-1]} p_j^i$ represents the cumulated power measurements of all nodes for a given timestamp $t_j \in [0, n-1]$.

The second metric is the *Maximum Cumulated Watt* and is denoted $maxCWatt$. It represents the cumulated maximum power witnessed during the run of the application $\mathcal{A}$ for the set of current selected states s_c of considered leverages. It reflects how much the application, when considering the current combination of leverage states, stresses the computing nodes on which it is executed. It is defined as:

$$maxCWatt = \max_{j \in [0,n-1]} P_j \tag{5.1}$$

The third metric is the *Average Cumulated Watt* and is denoted $avrgCWatt$. It represents the cumulated average power consumption of the application $\mathcal{A}$ for the set of current states. It is defined as follows:

$$avrgCWatt = \frac{\sum_{j \in [0,n-1]} P_j}{n} \tag{5.2}$$

Finally, the fourth metric is the *Cumulated Joules* and is denoted $CJoules$. It represents the cumulated energy consumption of the run for the current leverages combination. It is the energy consumption of all nodes used between t_0 and t_n for the execution of $\mathcal{A}$. It is defined as follows:

$$CJoules = \sum_{j \in [0,n-2]} (t_{j+1} - t_j) * P_j \tag{5.3}$$

A power capping constraint indicates a maximum power consumption value to not overpass during a certain period of time. It is used in the literature as a possibility to constrain a complete computing facility's power consumption within a given power budget [16]. In the rest of this chapter we consider this type of constraints. Power capping represents the type of constraints imposed by electrical providers within their contracts or through a scheduler imposing various power capping to every user [53]. One can note that this constraint can evolve through time. In addition to this constraint, two functions have to be minimized: the execution time of each run; and the energy consumption of each run.

In this section has been presented our complete considered case-study. This case-study illustrates that our automation process of Green Programming is feasible. In the rest of this chapter are detailed the experiments conducted on this case-study.

5.4 Experimental setup

In this section is detailed the experimental setup used for evaluations. First, the hardware is described, then the chosen configurations to build knowledges are given.

Table 5.1: Hardware configuration of Grid'5000 Taurus nodes and TGCC Curie thin nodes

	Taurus Grid'5000	**Curie Thin Nodes**
CPU model	Intel Xeon E5-2630	SandyBridge
Number of CPU	2	2
Cores per CPU	6	8
Total Memory (GB)	32	64
Compiler [-O3]	gcc 4.9.1	gcc 4.9.1
MPI	OpenMPI	Bullxmpi
Network	10 Gigabit Ethernet	fat-tree Infiniband

5.4.1 Hardware and energy monitoring

To conduct our evaluation, we use the *Grid'5000* experimental platform and the *Curie* supercomputer. Presented experiments have been conducted on the cluster named *Taurus* of the site of Lyon. The hardware configuration of this cluster is given in Table 5.1. The main advantage of using Grid'5000 is that computing nodes are well equipped in terms of energy monitoring. Each node is monitored by a wattmeter with a precision of 0.125 Watt (W) and that reports the average of 36,000 measurements each second. Finally, Grid'5000 provides an API which returns the power used for every node every second [88] which can be requested during execution (online) or after the end of the execution (offline). Such a hardware setup allows us to have a fine grain knowledge on energy consumption during a run. We used up to four of those nodes.

The TGCC Curie[5] is a French petascale supercomputer ranked as the 93^{th} supercomputer of the Top500 list of November 2017[6]. It is composed of three different types of nodes, each with a specific hardware configuration. Experiments have been performed on the *Thin Nodes* (Table 5.1) of the TGCC Curie supercomputer. Measurements on Curie thin nodes are done at the electrical cabinet with dedicated wattmeters and are updated approximately every 5 minutes. Energy monitoring can only be performed offline. Thus, such a configuration is less accurate than Grid'5000 for energy consumption measurements. However, it allows us to perform larger scale experiments, up to 2048 cores. To tackle the accuracy issue, we have performed much longer jobs on Curie.

5.4.2 Knowledge configurations and representation

By using code versions (CVV) generated by MSL on FS2D for Phase 1 of Figure 5.1, we run a set of benchmarks to build *Knowledges*. A *Knowledge* is built upon application production runs that combine leverages for a given configuration (domain size and number of iterations). Moreover, each run collects a set of metrics, as detailed in Sections 5.2.2 and 5.3.3. Two *Knowledges* have been built in our evaluations and are summarized in Table 5.2 for Grid'5000 and Curie experiments.

As already explained, one *Knowledge* uses a subset of production runs to collect metrics regarding the combination of two application leverages. First, one of the available code versions of FS2D generated by MSL (CVV) and second the number of processes and threads (#Processes/#Threads) chosen to run FS2D. For each run, the four metrics detailed in Section 5.3.3 are measured (execution time, maxCWatt, avrgCWatt, and CJoules).

[5] `http://www-hpc.cea.fr/fr/complexe/tgcc-curie.htm`
[6] `http://www.top500.org/lists/2017/11/`

Table 5.2: Knowledge configurations on FS2D.

Knowledge	Cluster	#nodes	Domain size	#Iterations
A	Taurus Grid'5000	4	4000×4000	100
B	Taurus Grid'5000	4	1000×1000	100
C	Thin Curie	64/128	20000×20000	10000

To analyze multiple metrics at the same time, we have chosen to use a pareto representation and its associated pareto frontier (or pareto-front) which has been defined and used many times and formally defined for energy concerns in [6]. Figure 5.2 gives an example of a 2D pareto frontier, where each axis is a metric and each point represents measures registered for a given job of a *knowledge*.

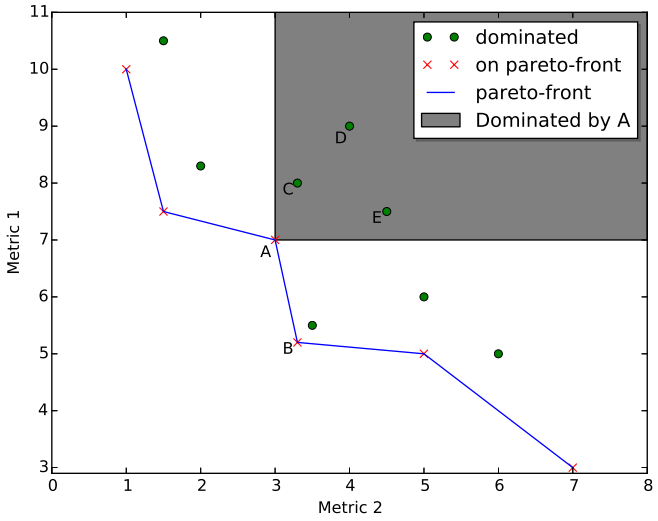


Figure 5.2: Pareto frontier example

Points on the pareto frontier represent the set of best solutions (relative to the remaining points), for a trade off between the two chosen metrics. Thus, they represent choices where no improvement for a metric can be made without deteriorating the second one. Points on the pareto-front are called *dominant points* while others are called *dominated points*. For example, in Figure 5.2, choosing B over A decreases the first metric but increases the second one. Points C, D, and E are dominated by A, which means that both metrics increase compared to A.

There is a wide panel of possible trade-offs between two chosen metrics. The trade-off could be between two energy metrics or between an energy metric and the execution time.

Our benchmark framework (Section 5.4.2) executes a set of jobs which are the combination of two application leverages. First, the set of available code versions of FS2D (CVV), and second, the #Processes/#Threads configuration chosen to run FS2D. From the results produced by one *knowledge*, a pareto can be built, where each point represents one job. To build a pareto, two metrics among the described ones in Section 5.3.3 have to be chosen.

As already given as an example, on the Taurus cluster of Grid'5000 where $cores = 12$ the total number of runs to perform to get the complete Knowledge A (pareto) is 55. On Curie, for which $cores = 16$, 84 runs are needed to build Knowledge B. One can note that these numbers of runs are small compared to the number of times the physicist (or its scripts) should possibly

run the FS2D simulation ($2^{14} = 16,384$).

5.5 Case study evaluation

In this section, we present an experimental setup in order to show that the CVV leverage introduces new variability of choice. To underline this variability, we combine it with another leverage: the number of processes and threads noted #Processes/#Threads leverage. The results presented in this section use the same hardware and metrics than the previous section.

5.5.1 Evaluation of the CVV leverage

First, we would like to show in our evaluations that choosing one code version or another while measuring time, maxCWatt, avrgCWatt and CJoules, leads to a non trivial trade-off. Table 5.3 reports measurements of the four metrics when executing the same knowledge configuration A, with the four code versions generated by MSL, on a single Taurus node. The Taurus node is used with its full capacity, thus using its 12 cores.

Table 5.3: Time, maxCWatt, avrgCWatt and CJoules for the four different code versions generated by MSL on FS2D (CVV).

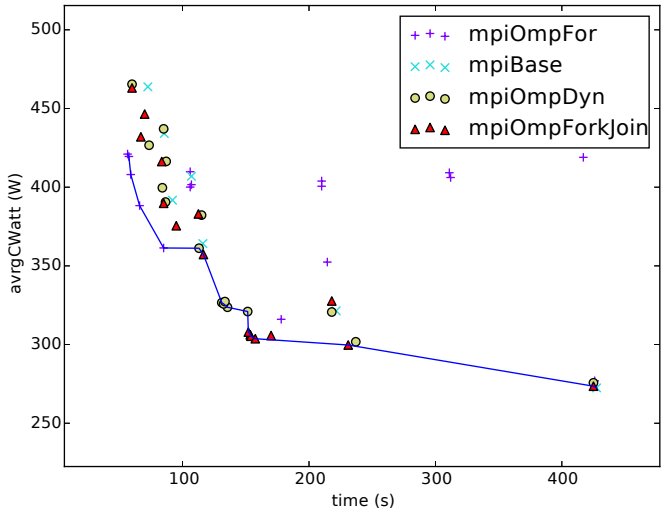
CVV leverage state	*time* (s)	*maxCWatt*	*avrgCWatt*	*CJoules*
MpiOmpDyn	133.37	**253.25**	**237.97**	31916.5
MpiOmpFor	**128.25**	257.87	239.80	**30854.12**
MpiOmpForkJoin	**130.75**	257.0	239.29	**31515.25**
MpiBase	142.5	**254.87**	**235.22**	33733.87

From Table 5.3, we can observe that "MpiOmpFor" and "MpiOmpForkJoin" are minimizing *time* and *CJoules*, respectively. However, these code versions also have the highest values for *maxCWatt* and *avrgCWatt*. As a result, and as expected, a correlation exists between the execution time and the energy consumption (CJoules). However, minimizing these metrics leads to high power consumption that could be problematic in the case of power capping constraints either for a cluster administrators or a green scheduler translating energy budget to a power capping. Moreover, Table 5.3 shows that for every state of the CVV leverage (code version), non negligible variability can be observed in the four metrics.

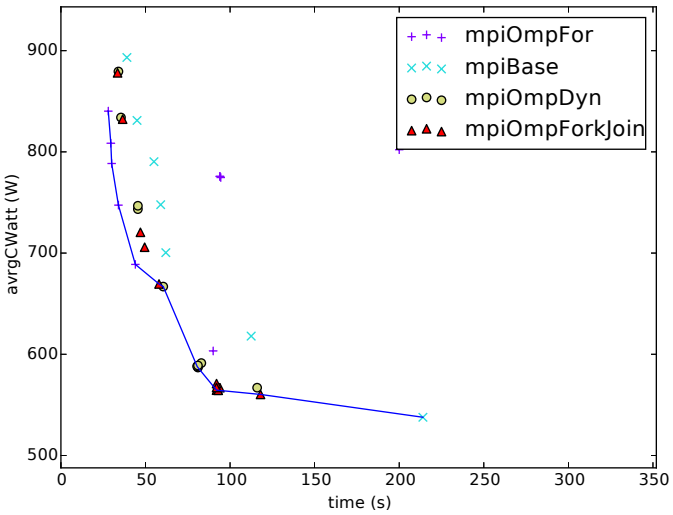
5.5.2 Grid'5000 experiments: fine grain energy monitoring

For each code version, many different configurations of #Processes/#Threads are possible, each point for one symbol (or color) represents one configuration. For example, the code version MpiOmpForkJoin can be run be using 4 MPI processes and 12 OpenMP threads per MPI process, or can be run by using 8 MPI processes and 6 OpenMP threads per MPI process. In this case cores of the four nodes are fully used (12 per node), but the same benchmark can be executed by using only 4 MPI processes and 2 OpenMP threads, etc.

For example, for Figure 5.3b, if a power constraint is set to $600W$, the chosen state for the CVV leverage would be "mpiOmpDyn". In fact, it is the first point on the pareto-front to answer the fixed constraint. Thus by definition, it is the point that minimizes execution time while satisfying the power constraint.



(a) Pareto representing benchmarks (2 nodes)

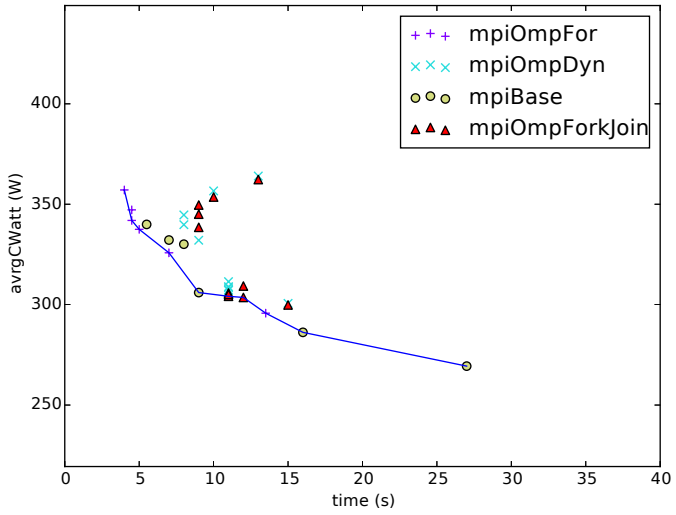


(b) Pareto representing benchmarks (4 nodes)

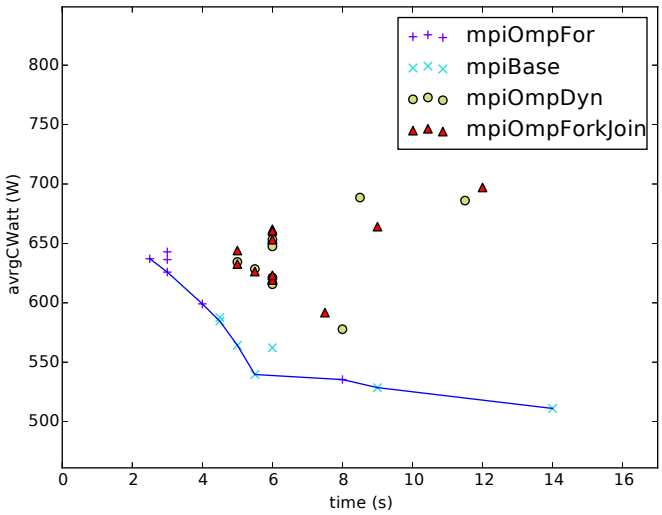
Figure 5.3: Paretos with metrics *time* and *avrgCWatt*, for knowledge A of Table 5.2.

Figures 5.3b and 5.3b both presents a pareto on the metrics *time* and *avrgCWatt*, where all runs of the knowledge A are represented (55 different runs). Each run has been performed 8 times and a median is computed. The pareto frontier is represented in blue. One can note a variability of code versions on the pareto frontier. This means that among the set of best choices for a trade-off between time and avrgCWatt, multiple code versions are represented. As a result, the CVV leverage improves the trade-off that #Processes/#Threads leverage alone could reach. Thus, it shows that the CVV leverage has an important impact on pareto frontiers and that a trade-off could be needed to take into account multiple constraints and/or objectives (here represented as metrics).

Figure 5.4 represents the same paretos that Figure 5.3, but for Knowledge B. First, the pareto frontiers behave differently when only changing the domain size of the application. This



(a) Pareto representing benchmarks (2 nodes)



(b) Pareto representing benchmarks (4 nodes)

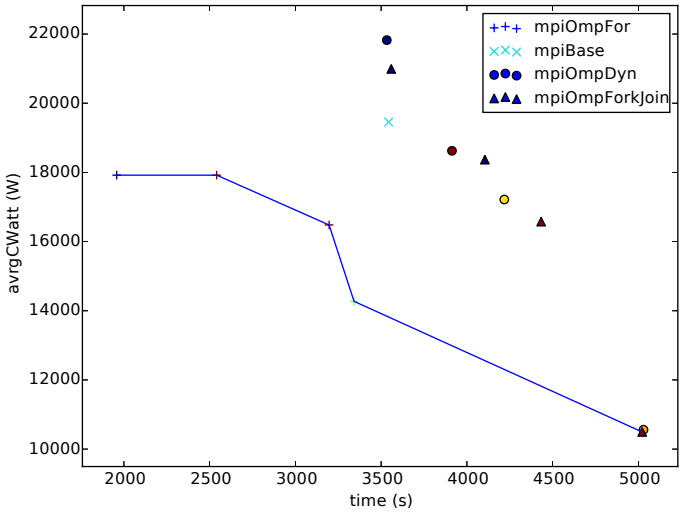
Figure 5.4: Paretos with metrics *time* and $avrgCWatt$, for knowledge B of Table 5.2.

is explained by different performance behaviors when changing this parameter, as shown in [14]. Second, and as previously observed, more than one code version is on pareto frontiers which means, again, that the CVV leverage can improve quality of choices.

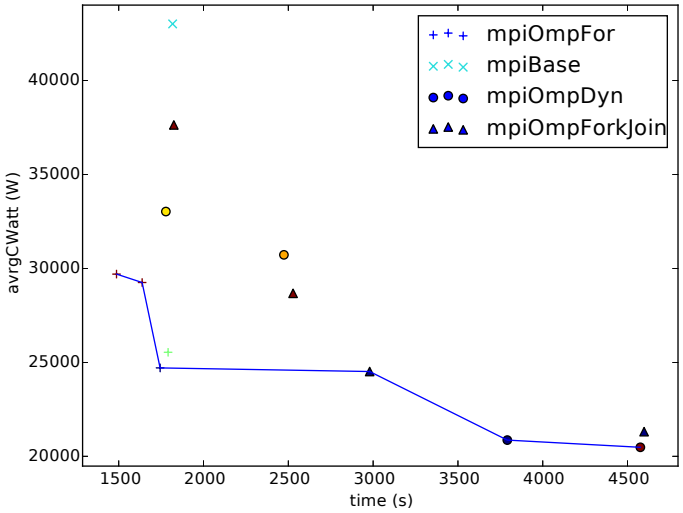
5.5.3 TGCC Curie experiments: large scale

To conduct large scale experiments, TGCC Curie is used. The lack of precision for energy monitoring of Curie has forced us to run very heavy configurations of the application, as described in Table 5.2: execution time is between one and two hours. Moreover, available monitoring on Curie restricts possible measurements to *time* and $avrgCWatt$.

Figures 5.5a and 5.5b show the results of Knowledge C from Table 5.2. More precisely,



(a) Pareto representing benchmarks using exclusively 1024 cores (or 64 nodes)



(b) Pareto representing benchmarks using exclusively 2048 cores (or 128 nodes)

Figure 5.5: Paretos with metrics *time* and $avrgCWatt$, for knowledge C of Table 5.2.

Figure 5.5a illustrates the use of 1024 cores (*i.e.,* 64 nodes), and Figure 5.5b the use of 2048 cores (*i.e.,* 128 nodes). Because of our limited access to thin nodes of TGCC Curie, less jobs have been performed than on Grid'5000 resulting in less points onto the pareto. However, the same conclusions can be drawn as the results show that the CVV leverage is also relevant at larger scale.

To conclude this section, the usage of CVV leverage implies high variability. In fact, in all figures, if only one CVV leverage state (i.e one symbol) was available, many relevant choices onto the pareto and more critically onto the pareto-front would not have been possible. Thus, the usage of pareto-front helps to answer and automate CVV and #Processes/#Threads leverages

position choice for a trade-off between two chosen metrics.

5.6 Simulation of Production Scenarios

In previous evaluations it has been shown that the CVV leverage, in our case-study, improves possible choices when looking for a trade-off between different metrics, possibly not correlated. However, when improving possible choices, the size of the knowledge to build also increases. For this reason, this section evaluates the entire automated process presented in this chapter by simulating production scenarios on our case-study. This section shows that the automation of Green Programming through the usage of the CVV leverage combined to the #Processes/#Threads leverage is relevant and can lead to substantial gains. To have a complete control over the applied scenarios, we have chosen to simulate different production and power constraints scenarios as if using a real production platform with other users.

As a reminder, Figure 5.1 represents the automation process of the CVV usage. This automation process is composed of three phases. Phases 2 and 3 are the ones concerned by production runs and are the ones simulated within this section. The second phase builds the knowledge needed in the third phase by using t_k first production runs. The third phase uses the knowledge as well as information on constraints to make adequate choices for each new production run. For a given scenario our simulation computes the energy gain and the percentage of violation of input constraints. Even if this section simulates production scenarios, results collected during our real-case experiments are used. Thus, the knowledge presented in Figure 5.3b is used within our simulation.

Three different elements are simulated within a given scenario: (1) the production scenario; (2) the energy and power constraints considered during the production scenario; and (3) the set of actors considered.

5.6.1 Production scenarios

The first production scenario is called *soft*. A total number of $2^{10} = 1,024$ runs are performed within this entire production scenario which is much less than the example given throughout this chapter ($16,384$ production runs). Thus, this scenario is not in favor of our process. This production scenario has a low frequency usage with four runs a day (two of them during the night, and two of them during the daytime). This scenario represents a soft arrival of production runs during 256 days.

In the second production scenario, namely *hard*, the same total number of runs are performed. However a high arrival frequency is simulated. Actually, twenty runs are performed per day which leads to a hard use of production resources for 52 days (51 full days, plus 4 extra runs during 52th day). To make these scenarios more realistic we also introduce vacancies or maintenance periods where runs are not performed.

5.6.2 Constraints

For the *power constraints*, we have chosen to simulate two types of power capping constraints. On one hand, the first constraint, namely *Fixed*, represents a power capping value (*i.e.,* maximum value to not overpass) constant through time. To choose a real case power capping for knowledge A, we refer to results displayed in Figure 5.3b, where we have chosen the rounded

value equidistant to the minimum and maximum reached *avrgCWatt* on the pareto-front. Thus, $650W$ has been chosen as *Fixed* constraint.

The second power constraint is denoted *day-night*. In this constraint, the maximum power value is low during daytime and high during night. For knowledge A, $600W$ and $800W$ have been chosen for day and night power constraints, respectively.

5.6.3 Actors

The two first actors considered in our simulation do not base their choice on any knowledge. The first actor of this family is called *Usual*. This actor illustrates what usually happens in production, *i.e.,* a single code version and a single number of threads and number of processes are used for all runs. The second one is denoted *Random*. This actor randomly chooses one code version and one #Processes/#Threads leverage state for each production run. One can note that both *Usual* and *Random* can perform choices that do not respect input constraints. However, the power capping constraints have been chosen such that *Usual* never violates it. One can note that this choice is not in favor of our process once again.

The third actor is the one we advocate in here. It is called *BuildKlg*. This actor makes choices by using a full knowledge (*i.e.,* complete paretos).

The last considered actor is called *Ideal*. This actor uses advanced machine learning strategies to be able to make choices with a partial knowledge of previous runs. Thus, this actor reduces the number of runs needed to reach t_k. As this chapter does not focus on the proposal of new actors, we have made the hypothesis that this *Ideal* actor is able to accurately discover the complete knowledge without any previous run, which would be the perfect actor, even if not feasible. Thus this actor represents the theoretical best case of our simulation.

Both *BuildKlg* and *Ideal* aims at first respecting power capping constraints and second minimizing execution time and energy consumption.

5.6.4 Simulation results

This section analyzes the results of simulation for every proposed actor on any production scenario and for any considered power constraint.

Two metrics are considered in results. First, the *Violation* metric represents the amount of joules consumed over the fixed power limit (the bigger the value, the worst the actor is). We could imagine that every Joule consumed over the limit represent an extra cost. However, as the input cost per Joule highly depends on the infrastructure or electrical provider policies, we represents the *percentage of violation* metric rather than the cost. Second, the total energy consumption is represented.

Table 5.4 displays the results of these simulations. The total energy consumed and the violation of constraints are represented for every scenario. Percentage of saved energy and constraint violation are given using the *Usual* actor as a reference. Actually, for the *Usual* actor, the CVV leverage position is set to *mpiOmpDyn*. While the #Prcocess/#Threads leverage is set to 4/10 (4 MPI processes and 10 threads per MPI process). Moreover, we have chosen this configuration because it always answers power capping constraints. Thus the simulated overall consumption of *Usual* actor will always be the same, given that the chosen run is always the same.

Regarding the violation rate, *Random* is the worst actor. One can note that *BuildKlg* has very low percentage of violation (3.88% in the worst case, 0.41% in the best case). Moreover, *BuildKlg* is very close to *Ideal* which is the best possible actor for this metric. The differences

Table 5.4: Simulation results based on knowledge A in terms of energy consumption, violation of constraints, and associated percentages compared to the *Usual* actor.

Actor	Energy (J)	Violation (J)	% gain	% Violation
Soft, Fixed				
Usual	54192768,00	0,00	0,00	0,00
Ideal	31659648,00	0,00	41,58	0,00
Random	55849891,19	7465509,76	-3,06	13,37
BuildKlg	32619458,62	133449,82	39,81	0,41
Soft, Day-night				
Usual	54192768,00	0,00	0,00	0,00
Ideal	43075597,13	1440879,81	20,51	3,35
Random	55867662,19	7074282,00	-3,09	12,66
BuildKlg	43567042,87	1686303,71	19,61	3,87
Hard, Fixed				
Usual	54192768,00	0,00	0,00	0,00
Ideal	31659648,00	0,00	41,58	0,00
Random	55179811,31	3242795,56	-1,82	5,88
BuildKlg	32619458,62	133449,81	39,81	0,41
Hard, Day-night				
Usual	54192768,00	0,00	0,00	0,00
Ideal	47837186,63	288175,96	11,73	0,60
Random	55849891,19	7465509,76	-3,06	13,37
BuildKlg	48165244,49	573153,42	11,12	1,19

between these two are the discovery part. In fact, during the pareto construction (discovering all the CVV and #Processes/#Threads states combinations) the *BuildKlg* actor violates the constraints. Even *Ideal* has penalties on *Day-Night* scenario. This is due to the fact that we only consider knowledge of constraints at the start of a run. Thus, such penalties are due to a change of the constraint value during the run (*e.g.,* for job starting during the night and finishing during the day).

If we only focus on the percentage of gain compared to *Usual*, the tendencies are the same for every scenario. *Random* is always worst than *Usual* (negative percentage of gain), showing that the current state of both the CVV and the #Prcocess/#Threads leverages is not to be chosen randomly. For *BuildKlg*, we can see that for each case, energy savings are not negligible (around 11% in the worst case and up to 39% in the best case). *Ideal* reaches the best energy savings but is very close to *BuildKlg* (a difference of 1.77% in the worst case), implying that such a clever actor may not be needed, in our case study.

In our evaluations we have shown that our automated process of Green Programming is applicable on a real case-study and can almost reach ideal results for both the total energy consumption and the rate of power capping violations. Thus, this work leads to energy and money savings.

5.7 Conclusion

Energy consumption is a growing concern for modern datacenters and supercomputers. Their energy consumption increases with their size and has become a physical and financial limitation. Green Programming (GP) is one of the available leverages to control energy consumption of applications by adapting their behavior or versions of code. However, in GP a programmer has to write multiple versions of a code, has to compare them manually and has to choose the version that satisfies his constraints and objectives. When considering HPC applications, such a process becomes almost infeasible as implementing one parallel version of a code is already a complicated and long process.

Four contributions have been presented toward automated GP. First, we have introduced a formal definition of the Code Version Variability (CVV) leverage. During evaluations we have underlined that the CVV alone, as well as combined to another leverage, offers more variability of choices, thus better trade-offs between execution time, energy consumption and power metrics.

Second, we have presented and detailed a first approach toward Green Programming (GP) automation in the specific case of production applications that are regulars. This automation process has three main steps: (i) the generation of code versions by using a DSL; (ii) the construction of the knowledge by using a subset of production runs; and finally (iii) the automatic choice, for each additional production run, of a combination of leverages states, by using an actor and by considering current constraints.

Third, our automation process of GP has been applied to a real case-study where a real-case numerical simulation has been selected, where a real-case DSL [30] has been used to produce different code versions and where real-case constraints have been considered. This case-study has shown the feasibility of our automation.

Finally, we have shown in our evaluations that our automated GP, applied onto our case-study, gets significant energy savings as well as very low constraint violations compared to a usual production case (no leverages considered), compared to a random case (by randomly choosing states of leverages), and compared to a theoretical Ideal case (where the knowledge is known from the start). Moreover, these results have been validated by simulation on two different production scenarios and by considering different kinds of constraints through time. In addition to this, results have shown that using very sophisticated actors, for example by using machine learning techniques, is not relevant in our case-study.

Through this combination of three leverages, we are able to provide an automation process of green programming for HPC applications. The chosen leverages were the CVV, the number of processes and finally the number of threads. This first step was, like in the previous chapter, to underline the fact that the proposed leverages have great influence on the energy related metrics, thus are energy and power leverages. This study of energy leverages was made for the CVV leverage alone, as $\#Processes/\#Threads$ has been studied as an energy leverage in the literature. We showed that through the given process to combine and use combined leverages, we were able to minimize the energy consumption while always fulfilling the chosen constraints. Thus, this method permits automatic usage of combined leverages for better energy efficiency while answering fixed energy and power constraints. However, it does not permit to extract understandable knowledge, hints or ranking about the combined leverages, usable by users such as developers or site administrators.

Chapter 6

Automatic discovery, combination, benchmark and knowledge extraction for combined leverages

Contents

Large-scale distributed systems and supercomputers consume huge amounts of energy. As described in chapter 2, a large set of capabilities and techniques that we call leverages exist to modify power and energy consumption in large-scale systems. This includes hardware-related leverages (such as Dynamic Voltage and Frequency Scaling), middleware (such as scheduling policies) and application (such as the precision of computation) energy leverages. Discovering such leverages, benchmarking and orchestrating them, remains a real challenge for most of the users. In this chapter, we propose a solution to automatically build the table of leverages, a score table that permits to evaluate combined leverages, associated with a large set of computing resources. We also propose algorithms and predicates that ease the reading of the table of leverages and extract knowledge from it. This is evaluated on several platforms and algorithms. We show that the construction of the table can be parallelized at very large scale in order to reduce its execution time while maintaining precision of observed knowledge.

6.1 Introduction

While Chapter 2 underline the fact that studies have been conducted on each one of the most commonly used leverages, only few works consider combining them. For instance, in [70], and [75], the authors combine the number of OpenMP threads and DVFS, and in [13], the authors combine shutdown and DVFS leverages. In the case of shutdown, this leverage has obvious impacts on other leverages: in the off state, no other leverage can be employed at the application level, for instance. Indeed, the utilization of a given energy leverage can impact both the utilization and the efficiency of another leverage. Moreover, the variety of leverages and the complexity of modern hardware architectures, in terms of size and heterogeneity, makes the energy efficiency more complex to reach for users.

In this chapter, we propose a first approach toward a completely automated process to characterize the energy leverages available on computing nodes. The key idea of our contribution consists in building a score table with a value for each leverage combination and each studied metric. These scores are obtained through the execution of a representative benchmark. Based on this score table, we can provide hints to users about the most suitable solution for their application.

This chapter makes the following contributions:

1. We propose a generic framework formalizing the combination of leverages through the definition of a table of energy leverages;
2. The table of energy leverages: a tool to help a user, a developer or an administrator to choose which leverage or leverage combination suits the best his energy or power objectives;
3. We present a comprehensive experimental method based on benchmarks and a detailed overview of its concrete implementation to build the table of energy leverages;
4. Algorithms to extract knowledge about the interaction of leverages and their influence on energy consumption;
5. We analyze experimental results on several servers demonstrating how to parallelize the building of the table.

The remaining of this chapter is structured as follows. Section 6.2 shows our process to build the table of leverages, and Section 6.3 explains how this formalism is implemented. Section 6.4

presents the experimental setup and a first full example of table of leverages. Section 6.5 then presents how to exploit the raw data of the table of leverages and extract useful knowledge, while Section 6.6 focuses on the genericity of our process. Section 6.7 demonstrates the parallelization of the creation process of the table of leverages. Finally, Section 6.8 concludes this work.

6.2 Formalism of table of leverages

In this section, we describe the methodology applied to build a table of energy leverages, which relies on metrics and benchmarks to characterize the performance and energy impact of each leverage combination on a given node. For each metric and each benchmark, a score is attributed to a given leverage combination. First, we describe the basic concepts used to build the table: the metrics and benchmarks. Then, we present the formal definition of the table of leverages, and finally, the methodology for building it.

6.2.1 Metrics

Leverages may influence the quality of service or performance of an application. For instance, shutdown techniques may induce latency in waking up the required nodes. Consequently, for these leverages, users need to determine their acceptable trade-off between energy-related metrics and performance metrics.

Here, three different metrics that represent both energy and performance constraints are explored. These metrics are measured for a given period of time corresponding to the time spent during the execution of the benchmark.

The two first metrics are energy and power related metrics. To define them, we introduce the following notations:

- $T = \{t_0, \ldots, t_N\}$ is the set of timestamps of energy consumption measurements of a given run; t_0 and t_N represent the starting and ending timestamps, respectively;
- p_j, $j \in [0, N]$, represents the power consumption (in Watt), of the considered node for the timestamp t_j.

Average Watt denoted $avrgWatt$, it represents the average power consumption of a chosen run. It is defined as follows:

$$avrgWatt = \frac{\sum_{j \in [0,N]} p_j}{N+1}. \tag{6.1}$$

Joules denoted $Joules$, it represents the energy consumption of the run. It contains the energy consumption of the complete node used between t_0 and t_N. It is defined as follows:

$$Joules = \sum_{j \in [0,N-2]} (t_{j+1} - t_j) \times p_j. \tag{6.2}$$

Execution time Finally, the execution time, denoted $Time$, is the whole execution time of a run, including initialization time.

6.2.2 Benchmarks

A benchmark corresponds to a self-contained application that is representative of typical applications or portions of applications. The benchmark is compiled before the run. Once launched, the metrics previously defined are collected during its execution.

Here, for the sake of clarity, we evaluate only one benchmark for a set of embedded leverages. We chose to focus on a well-known CPU intensive code: the line per line matrix multiplication (LpL MM) of dense random matrices. The choice of such a benchmark was also supported by a recent survey about energy consumption in data-centers [32], where authors provide a breakdown of a computing node's energy consumption. They underline the fact that for a supercomputer type of node, thus composed with a multi-core CPU, here an Intel Xeon, and large amount of memory, here DDR3 type of memory, the CPU still holds the larger part of the energy consumed, here 60% of the energy consumed by the node.

The same algorithm is implemented for the various leverage combinations. The considered leverages in this application are multi-thread, computation precision and vectorization (previously exposed in Section 2.4). As a reminder, multi-thread is the leverage exploiting intra-node parallelism through the spawn of threads. The computation precision is the leverages used to modulate the precision of computation and finally, vectorization is the leverage using vectorial capabilities of current computing components. For the last two leverages, a different state means a different version of code, here generated by hand. Automatic generation of code version is possible had has been used in Chapter 5 but it is not the focus of this chapter.

6.2.3 Formalization of the table of leverages

Format of the table of leverages

Here, we describe how to compute the score associated to each metric for each leverage. Let X, Y, Z be the sets of available states of three leverages χ, ψ, ω: $X = \{x_0, \ldots, x_{n_x}\}$, $Y = \{y_0, \ldots, y_{n_y}\}$, and $Z = \{z_0, \ldots, z_{n_z}\}$.

Let $g_1, \ldots, g_m$ be the measured metric functions, as for instance $avrgWatt$, $Joules$, and $Time$. For all u $(1 \leq u \leq m)$, $g_u(x_i, y_j, z_k)$ is the value of metric g_u for the states x_i, y_j, z_k respectively for the leverages χ, ψ, ω.

In the table of leverages, each line corresponds to a combination of states for each leverage and the columns correspond to the measured metrics. In order to ease the comparison, we normalize each value on the minimum value for each metric. These normalized values constitute the scores indicated in the table of leverages. Let $h_1, \ldots, h_m$ be the normalized versions of $g_1, \ldots, g_m$. So, we have, for $1 \leq u \leq m$:

$$h_u(x_i, y_j, z_k) = \frac{g_u(x_i, y_j, z_k)}{\min\limits_{x_{i'} \in X, y_{j'} \in Y, z_{k'} \in Z} g_u(x_{i'}, y_{j'}, z_{k'})},$$

with $h_u(x_i, y_j, z_k)$ being the value in the table of leverages in column of metric u and corresponding to the line for the states x_i, y_j, z_k respectively for the leverages χ, ψ, ω.

Methodology to build the table

Building the table of leverages requires to run the benchmark in its adequate version for each leverage combination. Hereafter, we describe our methodology for running all the required executions.

Algorithm 1: Building the table of leverages: benchmark execution for each leverage combination for a given set of metrics.

Input: LeverageTree: leverages to benchmark
Input: SelectedStates: name of states of leverages being currently benchmarked

```
1  mM: metric measurements;
2  for s_c in root(LeverageTree).S do
3      if root(LeverageTree) is leaf then
4          Add s_c to SelectedStates;
5          mM.start();
6          Benchmark(SelectedStates).exec();
7          mM.end();
8          tableOfLeverages[SelectedStates] ← mM;
9      else
10         Add s_c to SelectedStates;
11         Algorithm1(unseen_children(LeverageTree), SelectedStates);
12     end
13 end
```

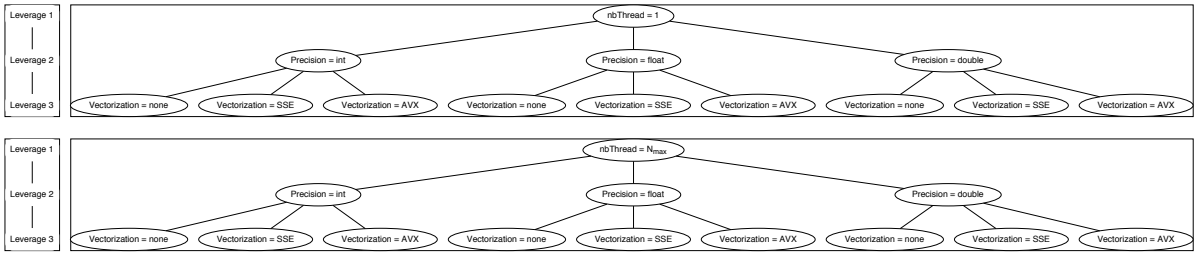


Figure 6.1: Example of $LeverageTree$ input for Algorithm 1.

Algorithm 1 shows the generic pseudo-code to execute the adequate benchmark version on the correct leverage combination for a given set of metrics. This algorithm has two inputs: $LeverageTree$ is a tree representing the set of selected states on the studied leverages, and $SelectedStates$ keeps trace of every current state of leverage involved so far. The functions $root(X)$ and $unseen\_children(X)$ return respectively the root of tree X, and the first unseen child node of tree X. mM corresponds to an entity gathering metric values (as defined before in our case: $avrgWatt$, $Joules$, and $Time$). $Benchmark$ corresponds to the entity that matches the current state of every leverage $SelectedStates$ and the corresponding binary file to execute, $exec$ corresponds to the execution of the benchmark. Thus, for all the considered leverages ($LeverageTree$), the algorithm is executed recursively over their respective states (S) and collects the metrics (mM) before moving to the next leverage combination. The metrics gathered during the executions are saved in the $TableOfLeverages$ entity.

Figure 6.1 shows an example of input used for Algorithm 1 in the following table of leverages of this work. Rounded bullets represent states of the three considered leverages. The benchmark chooses the corresponding binary, for leverages having different binaries in set of states S, here *Precision* and *Vectorization*. Leverage *nbThreads* changes its state through environment variable.

When the execution of Algorithm 1 with Figure 6.1 as input is finished, the table of leverages

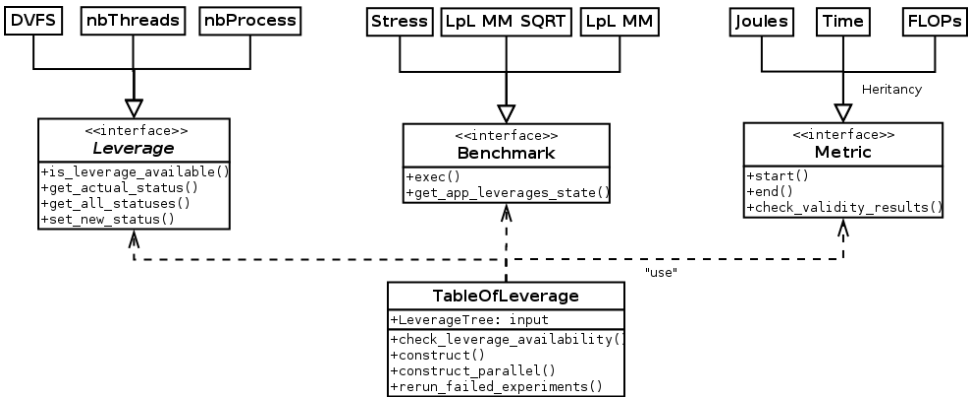


Figure 6.2: Framework UML diagram

is complete for the considered benchmark.

6.3 Implementation of table of leverage

In order to be able to build the table of leverages, we created a framework to identify available known leverages on a given hardware, benchmark the leverages combinations and collect the associated metrics. This tool can run on a single node or on an entire cluster. It is designed to be as flexible as possible on three basic concepts: leverage, metric and benchmark. This framework fits the needs of a wide type of users, going from basic users without specific knowledge to more experts ones capable of implementing new leverages, benchmarks and metrics collection methods.

6.3.1 Leverages

As shown in Figure 6.2, the framework provides multiple interfaces to fully describe our basic concepts and to define a contract between a developer that wants to extend the explored leverages, studied metrics or used benchmark. These contracts are implemented through a fully abstract class, forcing a class inheriting from it to implement the needed functions. Thus, every leverage class must be able to detect its availability (*is_leverage_available()*), to retrieve its current status (*get_actual_status()*), to retrieve its list of available statuses (*get_all_statuses()*) and to change its actual status with a valid one (*set_new_status()*). For example, the availability of the DVFS can be validated if the file */sys/devices/system/cpu/cpu0/cpufreq* exists. The actual status of the DVFS leverage for a specific core, here 0, can be found in a configuration file */sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq*. Reading the *scaling_available_frequencies*[1] file permits to extract all statuses. Finally, the current state can be changed using the command *cpufreq-set*.

6.3.2 Metrics

The framework also provides an interface for the metrics. It imposes to be able to start and end the monitoring of a metric. It also imposes a metric to check the validity of obtained results

[1]Path file is */sys/devices/system/cpu/cpu0/cpufreq/*

(*check_ validity_ results()*). The implementations of the metric contract is mandatory for a user using the framework with other focuses than energy and power. Thus, such contract allow various focus for a given study that is diretly linked to the focus of the user.

Grid'5000 provides an api, Kwapi, to get the collected data from the wattmeters in a given period of time. Once a benchmark has been executed on a node, the contract asks kwapi for the consumption of the node during this period of time.

Another possibility given by Grid'5000 is to use the live metric webpage. This webpage returns the consumption metrics of the nodes of Grid'5000 every second. We created a script that collects them every second. The framework then gets the metrics from the script by giving the starting and ending timestamps of the benchmark execution.

Such method could also be exported to platforms without wattmeters. For example, we implemented the contract for captors such as RAPL or IPMI, also to get energy related metrics.

The framework also implements a contract to retrieve FLOPs (FLoating point OPeration per seconds). In order to retrieve the FLOPs, a script that collects the flops metrics during a benchmark execution using the PAPI framework is deployed on used nodes. This method is not specific to Grid'5000 and could be used on a different architecture.

6.3.3 Benchmark

The final contract is relative to the benchmark execution. The first function executes the given binary. The second gives the current states of application leverages. A family of leverage is relative to the application. Thus, the state of application leverages changes for every binary (*get_ app_ leverages_ state()*).

The framework copies and executes the given binary on chosen nodes. We assume that compiled and ready to used binary files are passed to the framework.

6.3.4 Construction of the table of leverages

The table of leverage class uses previously presented contracts to implement Algorithm 1. Various modes of construction are provided.

Default method *construct()* method runs the same experiment on every node. Thus, every node will make the same leverage exploration. This method can be time consuming. For example, using our testbed, combining the first and last status of *nbThreads* leverage, *Precision* (int, float and double) and *Vectorization* (SSE3 and None) leverages, takes approximately 1 hour and 30 minutes.

Automatic node spread work method In the *construct_ parallel()* method, the framework runs one scenario by dividing and assigning automatically work on nodes. The execution time of the framework would be divided by the number of used nodes. If we take the previous example combining the *nbThreads*, *Precision* and *Vectorization*, with 5 nodes, the execution time of the framework for this scenario is around 18 minutes. However, the user will need to have a good knowledge of the nodes and asked metrics in order to ensure coherent results.

6.4 Experimental setup and first table of leverages

In this section, we present the table of leverages built on a node from Grid'5000.

6.4.1 Experimental setup

To evaluate our methodology in various computing environments, Grid'5000 is used as a testbed [7].

Table 6.1: Server Node characteristics

Features	Taurus	Nova
Server model	Dell PowerEdge R720	Dell PowerEdge R430
CPU model	Intel Xeon E5-2630	CPU E5-2620 v4
# of CPU	2	2
Cores per CPU	6	8
Memory (GB)	32	32
Storage (GB)	2 x 300 (HDD)	2 x 300 (HDD)
Date of arrival	11.2012	03.2017

As our focus is on energy and performance related metrics, we used the Grid'5000 Lyon site that provides the energy consumption of every node from all available clusters is monitored through dedicated wattmeters, exposing one power measurement per second with a 0.125 Watts accuracy. Employed servers' characteristic are shown in Table 6.1.

6.4.2 Table of leverages for three leverages

We apply our previous methodology for the three chosen leverages to the CPU intensive benchmark. This allows us to explore all possible states of chosen leverages and to build a complete table of leverages. It has the following format: the first three columns present the states of the $\#Threads$, $Precision$, and $Vectorization$ leverages respectively, while the last three columns show the normalized results of the three metrics $avrgWatt$, $Joules$, and $Time$, respectively, for every combination of leverage.

As described in Table 6.2, a line of the table of leverage represents results of all gathered metrics for the execution of a representative load for a chosen combination of leverages. The results are normalized as shown in Section 6.2.3. The table of leverages gathers the knowledge of a node, here Nova (Table 6.1), for a given workload done for multiple states of leverages combined.

Explanation of the table: A lot of unexpected results, at first sight, are detected in Table of leverage 6.2, like the combination with *int* being better than *float* and *double* when *1* and *none* are the chosen state for the *nbThreads* and *Vectorization* leverages, with this trend being reversed when *nbThreads* is set to *32*.

From the set of combination with *1* as the chosen state for leverage *nbThreads*, it is logic to see that *int* is quicker than *float* then *double* from a cache usage perspective. Indeed, more data can be brought into the cache to compute without the need to fetch new data compared to float or double representations that need more space for the same amount of elements. As for the *SSE3* and *AVX2* combinations, we have tremendous gain while using it compared to *None*, as it uses vectorial capabilities of the used core. Using a leverage usually comes with a cost. This statement is also true for the *Vectorization* leverage. An operation on vectors has costs, even if it is low. For example it is known that loading and saving vectors has a non null cost.

With only one active thread, the current architecture, Broadwell here, allows turbo boost, a technology that permits to reach a much higher frequency than the available ones (here it can reach 3.0 GHz, when average frequency is 2.1 GHz). Also, when the OS detects too much load

Table 6.2: Table of energy leverage states for LpL MM benchmark on a Nova node

Leverage states			avrgWatt(W)	Joules(J)	Time(sec)
#Threads	Prec.	Vect.			
1	int	none	1.05	65.09	61.89
1	int	SSE3	1.06	28.26	26.56
1	int	AVX2	1.06	29.32	27.67
1	float	none	1.05	72.97	69.67
1	float	SSE3	1.06	33.8	31.89
1	float	AVX2	1.05	36.8	34.89
1	double	none	1.06	81.59	76.89
1	double	SSE3	1.07	58.52	54.89
1	double	AVX2	1.06	57.72	54.22
32	int	none	1.43	13.48	9.44
32	int	SSE3	1.4	4.68	3.33
32	int	AVX2	1.0	1.0	1.0
32	float	none	1.45	7.4	5.11
32	float	SSE3	1.41	3.76	2.67
32	float	AVX2	1.56	3.11	2.0
32	double	none	1.53	8.34	5.44
32	double	SSE3	1.53	8.52	5.56
32	double	AVX2	1.54	7.0	4.56

on a core, it context switches the running process and runs it on another core. By doing so, the kernel saves the states (stack, registers) of the current process and loads it on another core, implying a storing and loading cost of the given process. This phenomenon can happen several times during a second. Thus, saving and charging states can create a lot of cache misses, which could be dramatic with usage of vectorization, where loading and saving vectors is not free. As *AVX2* has longer vectors, its operations costs on vectors can be longer than *SSE3*. Thus, it starts to be beneficial only when comparing *double* combinations for such a *Vectorization* leverage.

When threads are up to 32, data is more likely to be shared between caches of various used cores. Without the previous struggles from caches for one core and because it is also well known that floating points operations (*float* and *doubles* here) are well optimized on current architectures and perform better than integers, {32, float, none} and {32, double, none} perform better than {32, int, none}. All threads are sharing data on separated cache, *SSE3* and *AVX2* outperforms the none configuration, with *AVX2* always outperforming *SSE3* for a fixed combination. Due to this data repartition between caches implied by the chosen configuration of the *nbThreads* leverage, there is enough computation to overcome costs of larger vector operations, here *AVX2* for all combinations.

Note that the best combination for all metrics used here is always the {32, int, AVX2} combination. This result is the best combination to choose only if we have no constraints about leverage choices. It is expected to see variation, as leverages highly modulate the usage of nodes, either from intensity of usage for example of caches, core usage, availability of specific leverages (like seen with turboboost with one thread).

Results of metrics from combination of leverages is thus complicated to fully explain without a detailed knowledge of the architecture, the underlying used leverages and their influences on

a given context. We propose predicates that helps a user underlining relevant tendencies from the table of leverages. For example, this table could help a user to choose a combination taking into account a fixed leverage state. Or to answer the following question: is there a leverage or a state of leverage that is always better for a given metric?

6.5 Exploiting the table of leverages

In this section, we describe a methodology to exploit the table of leverages and to extract useful knowledge, such as the influence and impact of one or multiple leverages on a given metric or set of metrics. We propose two ways of extracting a score for each leverage. The first one corresponds to the actual table: it normalizes the results of a given metric for every explored configuration. The second one computes a ratio of contribution for each leverage in order to expose the most relevant leverage (the one with the largest contribution to the considered metric).

We define below four exploitation scenario that can be supported by analyzing the table of leverages. We illustrate how to answer these questions on the selected table (Table 6.2). These questions target a single metric, h_u.

Question 1: Is a selected combination of energy leverages states the best one for metric h_u? If a given combination is always the best, it means it should always be applied, if possible, if one wants to optimize h_u. Consider a combination of states x_a, y_b, z_c of leverages χ, ψ, ω for metric h_u. We need to check whether for all $i \in [0, \ldots, n_x]\backslash\{a\}, j \in [0, \ldots, n_y]\backslash\{b\}$, and $k \in [0, \ldots, n_z]\backslash\{c\}$, we have:

$$h_u(x_a, y_b, z_c) \leq h_u(x_i, y_j, z_k).$$

On Nova nodes and for the three leverages (Table 6.2), the best combination for all three studied metrics is {32, int, AVX2}.

Question 2: When I fix a state, do I always improve metric h_u? Consider state x_a of leverage χ. We want to check whether for all $i \in [0, \ldots, n_x]\backslash\{a\}$, for all $l, j \in [0, \ldots, n_y]$, and for all $m, k \in [0, \ldots, n_z]$, we have:

$$h_u(x_a, y_l, z_m) \leq h_u(x_i, y_j, z_k).$$

On the example of Table 6.2, for the $Joules$ and $Time$ metric, only the n_{max} (here, 32) state of *nbThreads* leverage answers this predicate, meaning that using this state will always be beneficial. No specific results can be obtained with this question for the $avrgWatt$ metric, meaning that no leverage state is always better for this metric when used.

Question 3: If mulitple states are fixed for a subset of leverages, is a given state for the remaining leverages the best choice to optimize h_u? Consider that the state of leverages ψ, ω is fixed to y_b, z_c. We are asking whether state x_a of leverage χ is the best choice for metric h_u. Therefore, we need to check whether for all $i \in [0, \ldots, n_x]\backslash\{a\}$, we have:

$$h_u(x_a, y_b, z_c) \leq h_u(x_i, y_b, z_c),$$

which tells for instance that for the fixed combination {32, SSE3}, the best state for the *Precision* leverage is *float*, when considering the $Joules$ or $Time$ metric (Table 6.2). Although, when focusing on $avrgWatt$ as the studied metric, for the {32, SSE3} fixed combination, the best state for the *Precision* metric is *int*.

If only state z_c for leverage ω is fixed, and we consider states x_a and y_b of leverages χ and ψ respectively, we check whether for all $i \in [0, \ldots, n_x]$ and for all $j \in [0, \ldots, n_y]$, we have:

$$h_u(x_a, y_b, z_c) \leq h_u(x_i, y_j, z_c).$$

Concerning the *Joules* metric (Table 6.2) for the fixed state *float* of the *Precision* leverage, the best combination for the *nbThreads* and *Vectorization* leverages is {32, AVX2}. Although, for the *avrgWatt* metric, fixing the state *float* of the *Precision* leverage, the best combination for the *nbThreads* and *Vectorization* leverages is {32, SSE3}, respectively.

Applying this predicate allows to extract relevant results. Concerning the *Joules* and *Time* metrics, for the *Precision* and *Vectorization* leverages, no state emerges as the best one. In fact, it highly depends on the chosen state of other leverages. One could for instance expect *int* to always be the best state, but when comparing the {32, double, none} with {32, int, none}, we see that the *double* combination is more effective than the *int* combination. Similar conclusions can be drawn when the *Vectorization* leverage is used. *AVX2* has larger vectors than *SSE3*, thus we would expect it to be always more efficient. We note however that when *nbThreads* state is equal to 1, {1, float, SSE3} is more effective than {1, float, AVX2}, leading to a different best choice when combined to the n_{max} state (here, 32), where {32, float, AVX2} is more effective than {32, float, SSE3}. Note that this combination emerges as the best one when *SSE3* is fixed.

Concerning the *avrgWatt* metric, we also get relevant knowledge. In opposition to the *Joules* and *Time* metrics, no state emerges as the best one for the studied leverages. As *AVX2* has larger vectors than *SSE3*, we would expect it to always stress more the CPU, thus always having higher values for this metric. It is the case with the {32, float} and {32, double} combinations. However, it is not observed with other combinations. When *nbThreads* is set to 1, *int* is always the best choice to minimize this metric, whatever the chosen state for *Precision* and *Vectorization* leverages. Moreover, when *Vectorization* and *nbThreads* are set to any studied states, *int* is also always the best choice to minimize the *avrgWatt* metric.

Question 4: Given a combination for all the leverages, how can we rank the states in terms of contribution for metric h_u? To answer this question, we consider a set of states x_a, y_b, z_c of leverages χ, ψ, ω. Then, for each state $w \in \{x_a, y_b, z_c\}$, we compute the contribution score $mc(w)$ for this state on metric h_u as follows. For state x_a of leverage χ:

$$mc(x_a) = \frac{h_u(x_a, y_b, z_c)}{\max\limits_{i \in [0,\ldots,n_x]} h_u(x_i, y_b, z_c)}.$$

For state y_b of leverage ψ:

$$mc(y_b) = \frac{h_u(x_a, y_b, z_c)}{\max\limits_{j \in [0,\ldots,n_y]} h_u(x_a, y_j, z_c)}.$$

For state z_c of leverage ω:

$$mc(z_c) = \frac{h_u(x_a, y_b, z_c)}{\max\limits_{k \in [0,\ldots,n_z]} h_u(x_a, y_b, z_k)}.$$

Then, we rank the contribution scores $mc(x_a)$, $mc(y_b)$, $mc(z_c)$ in ascending order to answer the question.

Table 6.3 presents the scoring related to the table of leverages previously presented in Table 6.2. For the best combination {32, int, AVX2}, the ranking goes as follows for the *Joules*

Table 6.3: Ranked impacts of energy leverage states for LpL MM benchmark on a Nova node

Leverage states			Ranked impact for		
#Threads(T)	Prec.(P)	Vect.(V)	avrgWatt	Joules	Time
1	int	none	P,T,V	P,T,V	P,T,V
1	int	SSE3	P,V,T	V,P,T	V,P,T
1	int	AVX2	P,V,T	V,P,T	V,P,T
1	float	none	P,V,T	P,T,V	P,T,V
1	float	SSE3	V,P,T	V,P,T	V,P,T
1	float	AVX2	P,V,T	V,P,T	V,P,T
1	double	none	P,T,V	P,T,V	P,T,V
1	double	SSE3	V,P,T	V,P,T	V,P,T
1	double	AVX2	P,V,T	V,P,T	V,P,T
32	int	none	P,T,V	T,P,V	T,P,V
32	int	SSE3	P,V,T	T,V,P	T,V,P
32	int	AVX2	P,V,T	T,V,P	T,V,P
32	float	none	P,T,V	T,P,V	T,P,V
32	float	SSE3	V,P,T	T,P,V	T,P,V
32	float	AVX2	P,V,T	T,V,P	T,V,P
32	double	none	P,T,V	T,P,V	T,P,V
32	double	SSE3	V,T,P	T,P,V	T,P,V
32	double	AVX2	P,T,V	T,V,P	T,V,P

metric: "T,V,P" or "*nbThreads, Vectorization, Precision*", meaning that the chosen state for *nbThreads* here is the most contributing state in this combination, followed by the *Vectorization* and then *Precision* states. Thus, for this combination, the precision leverage with the *int* position has the lowest contribution.

This ranking points out relevant results for the *Joules* metric. We notice a switch between two positions of a given leverage for the fixed combination of other leverage states: {32, double}. In fact, when comparing the scoring of {32, double, SSE3} with {32, double, AVX2}, we get respectively "*nbThreads, Precision, Vectorization*" and "*nbThreads, Vectorization, Precision*". In the first case, *double* and *SSE3* have the same worst possible score, 1.0, meaning that it is the worst state of this leverage for this combination. In the second case, *AVX2* scores better than *SSE3* and thus, it is above *double*.

When *nbThreads* is set to 1, we note that combinations including *SSE3* and *AVX2* states always have the *Vectorization* leverage state as the most contributing one, which leads to the conclusion that it is always better to use *SSE3* and *AVX2* states for the *Vectorization* leverage when *nbThreads* is set to 1.

For the {32, float, SSE3} combination, we get the scoring "*nbThreads, Precision, Vectorization*". *float* gets a better score and thus a better position than *SSE3* because it is the best leverage state for the {32, SSE3} combination, leading to the conclusion that choosing *float* instead of other *Precision* leverage states contributes more than choosing *SSE3* instead of other *Vectorization* leverage states for this combination.

For the $avrgWatt$ metric, scoring underlines the fact that when choosing *int* as a state of *Precision* leverage and for a fixed state of the *Vectorization* leverage, the sorting is always the same. In fact, {32, int, none}, {32, int, SSE3} and {32, int, AVX2} get the exact same sorting

of contribution that {1, int, none}, {1, int, SSE3} and {1, int, AVX2}, respectively. Moreover, *int* is always the most contributing leverage state, which shows that *int* is always a good choice to improve this metric.

This scoring also underlines the fact that in order to minimize the $avrgWatt$ metric, a user should better focus on *Precision* and *Vectorization* leverages, as *nbThreads* is never the most contributing one.

This scoring highlights results that would have been difficult to notice just by looking at the table. It allows a user to quantify how much a leverage position used in a combination contributes to the overall performance for a given metric.

6.6 Genericity of tables of leverages

In this section, we analyze the possible generalization of the table of energy leverages, i.e., the generalization of the extracted knowledge while varying the hardware node and the benchmark. Matrix multiplication is known to appear in a lot of CPU intensive codes and thus is known to be a representative code for CPU intensive kernels [4]. First, we analyze correlation between tables of leverages for the three selected energy leverages on different hardwares devices (i.e., heterogeneous nodes) running the same benchmark. Then, we build the table of leverages for another CPU-intensive only benchmark in order to illustrate the possible dependence between the table of leverages extracted knowledge and the selected benchmark.

6.6.1 Hardware architecture dependence

Choosing a different node with a different hardware architecture implies hardware-level leverage variability. First, we study the impact of the chosen node on the table of leverage. The same leverage exploration and table exploration that Table 6.2 is executed on a different family of nodes, the "Taurus" nodes, as detailed in Table 6.1.

Table 6.4: Table of energy leverage states for LpL MM benchmark on a Taurus Node

Leverage states			avrgWatt(W)	Joules(J)	Time(sec)
#Threads	Prec.	Vect.			
1	int	none	1.0	15.56	22.77
1	int	SSE3	1.02	6.03	8.68
1	float	none	1.0	16.86	24.65
1	float	SSE3	1.02	7.19	10.32
1	double	none	1.01	18.61	26.87
1	double	SSE3	1.03	12.52	17.87
24	int	none	1.49	3.78	3.71
24	int	SSE3	1.46	1.28	1.29
24	float	none	1.52	2.27	2.19
24	float	SSE3	1.46	1.0	1.0
24	double	none	1.58	2.55	2.35
24	double	SSE3	1.56	2.45	2.29

Table 6.4 presents the obtained table of leverages on a "Taurus" node with the previously studied benchmark (LpL MM). For the Taurus node, processors are older than for the Nova

node and *AVX2* is not a possible state for the *Vectorization* leverage. Thus, no combination with *AVX2* will appear in this table of leverages.

The answer of Question 1 (Section 6.5) for Table 6.4 concerning the *Joules* metric is {24, float, SSE3}, in other words, it is the best combination of leverages for this metric for a Taurus node. This combination is also the best choice that does not have *AVX2* as a chosen state for *Vectorization* leverage on Table 6.2 for the Nova node.

For the Question 3, for Nova node (Table 6.2), we fix {n_{max}, SSE3} and underline the fact that *float* is the best state for the *Precision* leverage, which is also true for Taurus node (Table 6.4).

This question also permits to underline a switch between {1, float, SSE}, {1, int, SSE3} and the usage of the n_{max} state (here, 24) for the Taurus node (Table 6.4). In fact, {int, SSE3} and {24, float, SSE3} are better than {float, SSE3} and {24, int, SSE3}, respectively. The phenomenon also happens with the Nova node (Table 6.2).

Although, it also permits to underline the fact that, when using the *SSE3* state, it is always better than the *none* state for *Vectorization* leverage on Taurus node (Table 6.4), which is also the case for Nova node (Table 6.2), except for {32, double, SSE3}, which is worse than {32, double, none} for the *Joules* metric.

6.6.2 Benchmark dependence

Choosing a different benchmark implies various application-level leverage availability. We study a benchmark very close to the initial line per line matrix multiplication (LpL MM). We introduce a "Sqrt" operation for every multiplication operation, in the most inner loop. Thus, this new benchmark keeps the same complexity, either in computation or memory. We also choose to introduce this operation because it modifies the usage of vectorization (SSE3 and AVX2 have an sqrt operation for vectors), thus representing a different CPU-intensive only benchmark.

Table 6.5: Table of energy leverage states for LpL MM SQRT benchmark on a Nova node

Leverage states			avrgWatt(W)	Joules(J)	Time(sec)
#Threads	Prec.	Vect.			
1	float	none	1.01	125.02	167.79
1	float	SSE3	1.02	10.37	13.74
1	float	AVX2	1.02	10.77	14.26
1	double	none	1.0	112.0	151.29
1	double	SSE3	1.01	33.54	44.71
1	double	AVX2	1.01	33.78	45.03
32	float	none	1.47	8.57	7.87
32	float	SSE3	1.35	1.0	1.0
32	float	AVX2	1.36	1.03	1.03
32	double	none	1.39	5.21	5.05
32	double	SSE3	1.4	3.22	3.11
32	double	AVX2	1.39	3.17	3.08

Table 6.5 illustrates the table of energy leverage states for the modified benchmark (multiplication of matrix with square root operations). All combinations with *int* are missing because sqrt operation result would have been stored in an *int*, thus loosing consequent information and therefore, the obtained application results would have been very different for this version. We

now compare this table with Table 6.2 which concerns the same Nova node for the unmodified benchmark (LpL MM).

The answer of Question 1 (Section 6.5) for the *Joules* metric cannot be {32, int, AVX2} because *int* is not a valid state for leverage *Precision*. Here, {32, float, SSE3} is the best combination of leverages, which is also the first non-*AVX2* result in Table 6.2 for the same metric.

For Question 3, for Table 6.2, we fix {32, SSE3} and underline the fact that *float* is the best state for the *Precision* leverage, which is also true for Table 6.5. On the other hand, when we fix only *float*, {32, AVX2} emerges as the best combination for other possible leverages for Table 6.2, while {32, SSE3} is the best one for Table 6.5.

This question also permits to underline a switch between {float, SSE3}, {float, AVX2} and the usage of the 32 state for the *nbThreads* leverage for Table 6.2. In fact, {float, SSE3} and {32, float, SSE3} are better than {float, AVX2} and {32, float, AVX2}, respectively. This phenomenon is not observed in Table 6.5.

6.6.3 Conclusions about genericity

For Question 2 (Section 6.5), as expected, all tables of energy leverages of this chapter show that using *nbThreads* at the maximum value will always be beneficial for the *Joules* metric.

Question 3 also permits to notice that when the number of threads is fixed, for a given fixed *Precision* leverage state, using *Vectorization* leverage in *SSE3* or *AVX2* state will always be beneficial for the *Joules* metric. This knowledge is true for every table of leverages in this chapter except for one combination implying the *SSE3* state. Indeed, on Table 6.2, the predicate exposes that the {32, double, SSE3} combination is not better than the {32, double, none} combination for this metric.

Furthermore, using the *AVX2* state for a fixed *Precision* leverage state of *double* is always better compared to other states for the *Vectorization* leverage.

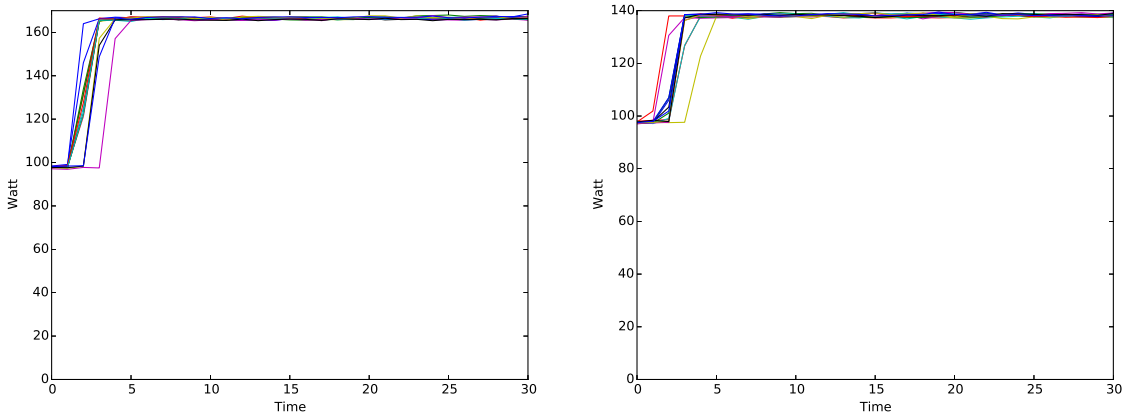
Our predicates also underline the fact that when *SSE3* is the chosen state for *Vectorization*, the chosen state to minimize the *Joules* metric is always the *float* state for the *Precision* leverage, whatever the chosen state for *nbThreads* is. For *AVX2* state, the chosen state to minimize the *Joules* metric is *int*, whatever the chosen state for *nbThreads*.

As we show with the previous analysis, between different architectures and benchmarks, minor differences have been found on the extracted knowledge. In fact, the comparison of different tables of leverages in different contexts (benchmarks or machines) underlines the fact that high level and finer grain knowledge could be generalized.

We believe that it is possible to have a finer analysis or a more generic table. For example, the proposed solution does not take into account the fact that the usage of *SSE3* and *AVX2* differs from one table to another. Another possible solution could be the scope of an assembly operation instead of studying an entire benchmark. In such a context, a table of leverages could be useful but hard to build. In fact, it would be extremely difficult to know with external wattmeters as ours if the measured results are noise or relevant information. Furthermore, it would be complex to determine what is a representative code for a specific operation, thus to determine a relevant benchmark for this operation.

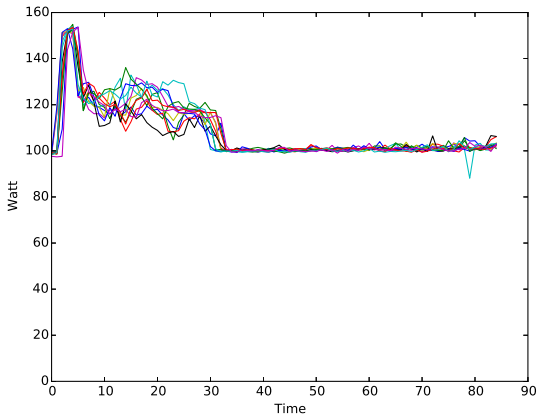
6.7 Parallelization of the table construction

The construction of the table of leverages may take a long time if many different leverage combinations are considered. In this section, we explain how the construction could be parallelized, so that the time to generate a complete table of leverages could be significantly reduced.

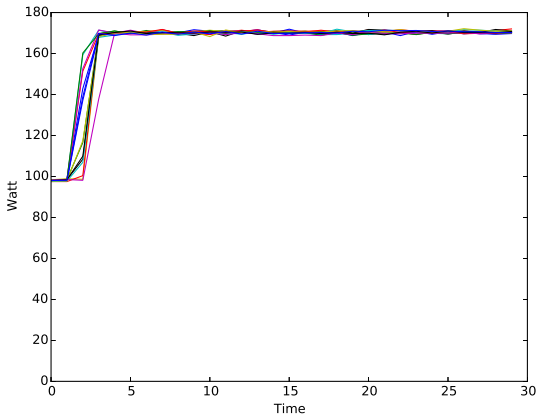


(a) CPU stress

(b) IO stress



(c) HDD stress



(d) RAM stress

Figure 6.3: Nova-1, 30 runs of various stresses

6.7.1 Re-usability of energy and performance metric, one node

The first hypothesis that we have to evaluate here is the fact that a node, exposed to the same load, gives the same metric results, with very low variation.

We run various intensive workloads using stress[2], a tool that applies a specific kind of stress to the used node. We execute 30 times every stress on a unique "Nova" node. Figure 6.3 shows results of such a protocol. For HDD stress (hard drive usage), three types of phases could be recognized. The first one from $t = 0s$ to $t = 10s$ and last one from $t = 35s$ to $t = 80s$ are the

[2] `https://people.seas.harvard.edu/~apw/stress/`

same for every run. The variation occurs between $t = 10s$ and $t = 35s$. When the disk is active and writing in various regions, the energy cost could differ depending on the chosen region for writing. Moreover, unlink() unix function assures that the link is removed but the file may not be deleted. We observe that for CPU, IO and RAM stress, all runs have approximately the same behavior, meaning that one run on the same node exhibit similar energy consumption, for these kinds of stress.

6.7.2 Re-usability of energy and performance metric, one family hardware

The second hypothesis that we have to evaluate here is the fact that extracted metrics could also be used by different nodes with the same hardware.

We stress several nodes of the same hardware (Taurus or Nova clusters, 10 Taurus nodes were used, while 5 Nova nodes were used) to observe how the standard variation concerning various energy-efficiency and performance-related metrics evolve. We chose these two families of hardware to evaluate a newly acquired node family ("Nova", 2017) and an old one ("Taurus", 2012). This evaluation is done with various intensive workloads that stress differently the energy consumption of a node (CPU, IO, hard drive usage, RAM).

Table 6.6: Average (Av.) and standard deviation (StD.) of various workloads for various energy and performance related metrics for various hardware architectures

Hardware family	Joules (J) Av. - StD.	AvrgWatt(W) Av. - StD.	Time(t) Av. - StD.
CPU			
Taurus	6807.0 - 68.8	205.84 - 1.37	32.81 - 0.39
Nova	4998.86 - 49.3	154.91 - 1.09	32.06 - 0.43
HDD			
Taurus	5055.98 - 365.33	140.58 - 2.98	35.85 - 2.4
Nova	9381.94 - 251.5	107.8 - 0.57	87.01 - 2.47
IO			
Taurus	3957.52 - 34.98	123.46 - 0.21	32.0 - 0.3
Nova	4194.53 - 68.06	130.3 - 0.67	32.04 - 0.66
RAM			
Taurus	5097.83 - 55.81	222.14 - 2.2	32.5 - 0.52
Nova	7282.26 - 115.89	158.53 - 0.8	31.93 - 0.44

Metrics and standard deviation averages of families of nodes are exposed in Table 6.6 for CPU, HDD, IO, and RAM workloads. For this experiment, every node is doing the same work at the same time. We get metrics results for every run (10 run averages on every node). We then average values for families of nodes. Note that the standard variation for the three chosen metrics are negligible for all stress benchmarks, except HDD (as already explained in the previous subsection). Even for the *Joules* metric, we note that the variation is under the second of idle consumption of both nodes for every benchmark, meaning that differences between metric results from the same family of hardware are negligible.

These experiments show that for the same workload, same energy and performance behavior could be witnessed for various nodes having the same configuration.

6.7.3 Table of leverages, variability between nodes

In this section, we evaluate the same hypothesis for our previously defined table of leverage (Table 6.2) with 5 nova nodes. For every leverage combination, we evaluate the standard deviation of obtained metrics on all nodes. For every leverage combination, we extract the average and the standard deviation of all used nodes. To explain these numbers easily, we also extract the percentage represented by the standard deviation to the average.

Table 6.7: Table of energy leverage states for LpL MM benchmark on a 5 Nova nodes, average, standard deviation and percentage of standard deviation for average

Leverage states			avrgWatt (W)			Joules (J)			Time (sec)		
#T.	Prec.	Vect.	Av.	StD.	%	Av.	StD.	%	Av.	StD.	%
1	int	none	118.99	3.53	2.96	66041.42	2016.32	3.05	555.0	1.1	0.2
1	int	SSE3	120.48	3.44	2.86	28844.54	865.88	3.0	239.4	0.49	0.2
1	int	AVX2	120.02	3.38	2.81	29862.48	881.6	2.95	248.8	0.4	0.16
1	float	none	118.55	3.32	2.8	74258.92	2085.38	2.81	626.4	0.49	0.08
1	float	SSE3	120.08	3.35	2.79	34416.58	1005.18	2.92	286.6	0.49	0.17
1	float	AVX2	119.48	3.44	2.88	37445.13	1079.41	2.88	313.4	0.49	0.16
1	double	none	120.06	3.43	2.86	83060.58	2379.14	2.86	691.8	0.4	0.06
1	double	SSE3	120.75	3.48	2.88	59578.0	1735.06	2.91	493.4	0.49	0.1
1	double	AVX2	120.45	3.44	2.86	58661.37	1740.92	2.97	487.0	0.89	0.18
32	int	none	161.0	4.41	2.74	13684.79	374.68	2.74	85.0	0.0	0.0
32	int	SSE3	158.72	4.67	2.94	4761.68	140.14	2.94	30.0	0.0	0.0
32	int	AVX2	113.39	3.62	3.19	1044.21	73.82	7.07	9.2	0.4	4.35
32	float	none	163.04	4.46	2.73	7499.91	205.06	2.73	46.0	0.0	0.0
32	float	SSE3	158.58	4.13	2.6	3805.9	99.07	2.6	24.0	0.0	0.0
32	float	AVX2	174.68	5.7	3.26	3211.55	48.17	1.5	18.4	0.49	2.66
32	double	none	171.79	5.14	2.99	8450.54	198.45	2.35	49.2	0.4	0.81
32	double	SSE3	172.62	4.97	2.88	8594.98	205.08	2.39	49.8	0.4	0.8
32	double	AVX2	173.6	5.27	3.03	6978.92	231.53	3.32	40.2	0.4	1.0

Table 6.7 presents the same exploration that the previously presented Table of leverages 6.2 for 5 nova nodes with average, standard variation and percentage represented by the standard deviation to the average, respectively, for every extracted metrics. While Table 6.8 presents the same exploration but for Taurus nodes. The same matrix dimension is used for both exploration (8192). Note that Taurus nodes does not have *AVX* as available state for the *vectorization* leverage.

Table 6.7 underlines the fact that the most stable metric is undeniably time, with only three combinations of leverage above or equal to 1% and only one combination, {1, int, none}, with a standard deviation above 1 second. Same goes for Table 6.8, where only two combinations of leverages are above 1%.

For Joules metric, every percentage is under 3.33% for Table 6.7, except for the less consuming combination of leverages, {32, int, AVX} at 7.07%, which is still reasonable. This high value for this combination could be explained by the fact that our wattmeters are giving a power value every second, thus if a run is a bit longer than the 9 seconds, it will get an extra power value that others will not have. Because there are not a lot of values (one per second), an extra value on such a short run has high repercussions on the standard deviation. Because runs are longer

Table 6.8: Table of energy leverage states for LpL MM benchmark on a 5 Taurus nodes, average, standard deviation and percentage of standard deviation for average

Leverage states			avrgWatt (W)			Joules (J)			Time (sec)		
#T.	Prec.	Vect.	Av.	StD.	%	Av.	StD.	%	Av.	StD.	%
1	int	none	143.3	3.34	2.33	101226.43	2371.0	2.34	706.4	0.49	0.07
1	int	SSE3	146.49	2.57	1.76	39406.17	693.39	1.76	269.0	0.0	0.0
1	float	none	143.74	2.63	1.83	109872.73	2050.37	1.87	764.4	0.49	0.06
1	float	SSE3	146.56	2.7	1.84	46926.95	842.17	1.79	320.2	0.4	0.12
1	double	none	145.88	2.89	1.98	121376.55	2457.85	2.02	832.0	0.63	0.08
1	double	SSE3	147.59	2.71	1.84	81676.02	1498.01	1.83	553.4	0.49	0.09
24	int	none	215.39	2.9	1.35	24684.46	420.55	1.7	114.6	0.49	0.43
24	int	SSE3	211.95	2.99	1.41	8308.44	146.41	1.76	39.2	0.4	1.02
24	float	none	218.57	3.3	1.51	14906.06	197.2	1.32	68.2	0.4	0.59
24	float	SSE3	210.8	3.15	1.5	6619.05	147.45	2.23	31.4	0.49	1.56
24	double	none	228.68	3.38	1.48	16739.08	252.47	1.51	73.2	0.4	0.55
24	double	SSE3	225.99	3.55	1.57	16044.3	221.76	1.38	71.0	0.63	0.89

on Table 6.8, percentage are more stable, between 2.34% and 1.32%. Finally, for the *AvrWatt* metric, every percentage is under 3.27% for Table 6.7 and under 3.55 for Table 6.8.

These previous results (Table 6.6, Table 6.7 and Table 6.8) analysis underlines the fact that for the same workload, same energy and performance behavior could be witnessed for various nodes having the same configuration, under a low percentage of difference. Thus, for large scale computing systems with large amounts of computing nodes with the same configuration, the table of leverage could be done on only one node, or derived from a segmented construction on multiple nodes and used as knowledge for other nodes.

6.8 Conclusion

There are a wide range of techniques, that we formally define as leverages, that permits to modulate the computing capabilities and/or the energy/power used by a device. We propose a generic solution to create a score table about multiple metrics for a given set of leverages, called the table of leverages. Then, we extract fine grain knowledge and hints from it, thanks to the defined predicates. Our solution underlines new knowledge about leverages alone and about combinations of leverages. Thus, it allows us to extract influences of leverages on each other and understandable knowledge by the user.

Generic knowledge could be extracted from tables using different CPU-intensive workloads or different hardware devices. For example, our solution underlines the fact that if *Precision* is set to the *double* state, it is always better to use it with *AVX2* state for the *Vectorization* leverage to minimize the *Joules* metric. Also, for *Vectorization* fixed to the *SSE3* state, our solution tells us that *float* is the best state to minimize the *Joules* metric. We also underline the fact that unexpected behavior can be seen when combining leverages. For example, we underline the fact that changing *float* or *int* to *double* for *Precision*, and keeping the *SSE3* state activated for *Vectorization* leverage, turns out to be counterproductive for the *Joules* metric.

Chapter 7

Conclusion

Contents

7.1 Conclusion

Energy consumption is one of the major problems of the current generation. With everyone being connected through multiple devices at the same time, asking requests to large-scale facilities all the time, energy consumption of IT systems has drastically increased through these last two decades. In 2017, the energy footprint of data centers and supercomputers around the world is estimated around 7% of global electricity. It is also responsible of 2% of global carbon emission [26]. This worrying consumption and carbon footprint has direct financial and environmental consequences on data center managers, such as Cloud providers and supercomputer operators. Reducing the energy consumption of large-scale computing systems is a mandatory step to address in order to build a sustainable digital society.

This thesis has investigated the reduction and the evaluation of energy consumption of large-scale computing systems through the usage of multi-level leverages present on such infrastructures. These leverages are usually easy to use but their influences on the energy consumption could be difficult to analyze, as their usage alone or combined could be counter productive if not wisely used. Thus, through our various contributions, we proposed methodologies, implementations and frameworks to evaluate, model and combine available leverages, alone or combined.

Chapter 2 presents our classification of available leverages in a data center or a supercomputer. It also gives a set of examples for each family with the most used and significant leverages available on such infrastructure. We conclude this chapter by pointing out the fact that no methodology is available in the literature to generically study leverages.

Based on this analysis, Chapter 3 presents our methodology to detect, evaluate, study and estimate a given leverage as an energy and power leverage, from the evaluation of the possible states of the studied leverage to the possible creation of actors answering multiple constraints at the same time.

In Chapter 4, this previously exposed methodology is then used to finely evaluate and model the shutdown leverage and to create generic actors that answer the dynamic usage of a facility with multiple constraints at the same time. Each of these actors has a unique focus on a given constraint to respect, such as power capping or the provenance of energy and are built to be easily composed.

A first methodology for the combination of leverages is then proposed in Chapter 5. This methodology is used on a production application executed on real infrastructure as Curie Supercomputer. Through this technique, green programming can be easily achieved and automated, even for HPC applications. We proposed a set of actors able to reduce the energy consumption and respect fixed constraints.

In Chapter 6, we present a framework to build the table of power and energy leverages. This highly extensible framework permits to discover, combine and benchmark a given set of leverages. It also permits to extract understandable knowledge from the given table.

7.2 Perspectives

While this thesis presented solutions and proposed methodologies to underline, model and evaluate various leverages alone and combined, a lot of perspectives remain open for large-scale energy consumption problematics.

7.2.1 Building the table of leverage for every phase

In Chapter 6, we showed that generic knowledge could be extracted from CPU-intensive phases with a representative benchmark. We focused on CPU-intensive phases because HPC applications are known to be CPU intensive most of the time.

It will be relevant to perform the same study on other phases, such as IO-intensive phases. This study would require to find the most relevant leverages for such a study to evaluate their influences. These findings can be done with the proposed framework, a chosen set of leverages to study and a representative IO benchmark.

7.2.2 Reducing the search state

Focusing on micro kernels that are representative of a possible phase of a real HPC application, as done in Chapter 6, is a first step to the reduction of the possible search state. Also, the parallelization of the creation of the table of leverages greatly reduces its time completion. If we consider more combined leverages, reduction of time to complete the table of leverage could be necessary. To achieve such a goal, prediction approaches could be used. Indeed, avoiding unnecessary or too long combinations to be executed could be beneficial for the execution time.

Such an approach could be risky too. In fact, if the focus of the user is to exploit the given predicate of the framework to better understand the interaction or the contribution of a leverage in a combination, he will not have full information, because such an approach will avoid or partly execute many combinations.

7.2.3 Supporting sub-application leverage

An approach that could be even more precise than the table per phase approach is the instruction level table approach, where all measures would be done on an instruction level. Thus, all application leverages would be replaced by all possible or most consuming instructions.

Yet, this solution raises two main issues: the generation of the benchmark and the monitoring. First, it can be hard to produce a benchmark or a section of code that uses only one instruction, and benchmarks it for a power and energy table of leverage. Such benchmark has to be produced in the nearest language to the machine, thus assembly, to be sure about which instruction is used. Even with a good assembly developer, it is complex to produce assembly code with only one major set of instructions. Second, the monitoring of such a fine grain benchmarking could require finer grain monitoring that what is commonly available on our actual infrastructures (i.e., external wattmeters and internal partial sensors such as RAPL). Thus, it could be hard to produce such an instruction table of leverage.

Bibliography

[1] Hayri Acar, Gülfem I ALPTEKIN, Jean-Patrick Gelas, and Parisa Ghodous. "Towards a Green and Sustainable Software." In: *ary Lif System* (2015), p. 471.

[2] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. "A Case Study of Energy Aware Scheduling on SuperMUC"." In: *International Conference on Supercomputing (ISC)*. 2014, pp. 394–409.

[3] Woongki Baek and Trishul M Chilimbi. "Green: a framework for supporting energy-conscious programming using controlled approximation." In: *ACM Sigplan Notices*. Vol. 45. 6. ACM. 2010, pp. 198–209.

[4] D.H. Bailey et al. "The Nas Parallel Benchmarks." In: *International Journal of Supercomputing Applications* 5.3 (1991), pp. 63–73.

[5] Peter E Bailey, David K Lowenthal, Vignesh Ravi, Barry Rountree, Martin Schulz, and Bronis R De Supinski. "Adaptive configuration selection for power-constrained heterogeneous systems." In: *2014 43rd International Conference on Parallel Processing*. IEEE. 2014, pp. 371–380.

[6] Prasanna Balaprakash, Ananta Tiwari, and Stefan M Wild. "Multi objective optimization of HPC kernels for performance, power, and energy." In: *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer. 2013, pp. 239–260.

[7] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. "Adding Virtualization Capabilities to the Grid'5000 Testbed." In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten Sinderen, Frank Leymann, and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[8] Kenneth C. Barr and Krste Asanović. "Energy-aware Lossless Data Compression." In: *ACM Trans. Comput. Syst.* 24.3 (Aug. 2006), pp. 250–291. ISSN: 0734-2071. DOI: 10.1145/1151690.1151692. URL: http://doi.acm.org/10.1145/1151690.1151692.

[9] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. "A taxonomy and survey of energy-efficient data centers and cloud computing systems." In: *Advances in computers*. Vol. 82. Elsevier, 2011, pp. 47–111.

[10] Gary L Bennett, James J Lombardo, RJ Hemler, and JR Peterson. "The General-Purpose Heat Source Radioisotope Thermoelectric Generator: Power for the Galileo and Ulysses Missions." In: *Proceedings of the 21st intersociety energy conversion engineering conference*. 1986.

[11] Gary L Bennett, James J Lombardo, Richard J Hemler, Gil Silverman, CW Whitmore, Wayne R Amos, EW Johnson, Alfred Schock, Roy W Zocher, Thomas K Keenan, et al. "Mission of daring: the general-purpose heat source radioisotope thermoelectric generator." In: *4th International Energy Conversion Engineering Conference and Exhibit (IECEC), San Diego, California, AIAA*. Vol. 4096. 2006, p. 2006.

[12] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. "Exascale computing study: Technology challenges in achieving exascale systems." In: *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep* 15 (2008).

[13] Nicolas Berthier, Eric Rutten, Noël De Palma, and Soguy Mak-Karé Gueye. "Designing autonomic management systems by using reactive control techniques." In: *IEEE Transactions on Software Engineering* 42.7 (2016), pp. 640–657.

[14] Julien Bigot, Hélène Coullon, and Christian Pérez. "From DSL to HPC Component-Based Runtime: A Multi-Stencil DSL Case Study." In: *WOLFHPC 2015 (Fifth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing)*. co-located with SC'15, Austin, Texas, United States, Nov. 2015, p. 10. DOI: 10.1145/2830018.2830020. URL: https://hal.inria.fr/hal-01215992.

[15] Kanishka Biswas, Jiaqing He, Ivan D Blum, Chun-I Wu, Timothy P Hogan, David N Seidman, Vinayak P Dravid, and Mercouri G Kanatzidis. "High-performance bulk thermoelectrics with all-scale hierarchical architectures." In: *Nature* 489.7416 (2012), pp. 414–418.

[16] Andrea Borghesi, Francesca Collina, Michele Lombardi, Michela Milano, and Luca Benini. "Power capping in high performance computing systems." In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2015, pp. 524–540.

[17] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. "Cloud computing." In: *IBM white paper* 321 (2007), pp. 224–231.

[18] François Broquedis, Thierry Gautier, and Vincent Danjean. "LIBKOMP, an Efficient openMP Runtime System for Both Fork-join and Data Flow Paradigms." In: *Proceedings of the 8th International Conference on OpenMP in a Heterogeneous World*. IWOMP'12. Rome, Italy: Springer-Verlag, 2012, pp. 102–115. ISBN: 978-3-642-30960-1.

[19] Paolo Cappelletti. "Non volatile memory evolution and revolution." In: *Electron Devices Meeting (IEDM), 2015 IEEE International*. IEEE. 2015, pp. 10–1.

[20] Juan M Cebrian, Lasse Natvig, and Jan Christian Meyer. "Improving energy efficiency through parallelization and vectorization on intel core i5 and i7 processors." In: *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* IEEE. 2012, pp. 675–684.

[21] Juan M Cebrián, Lasse Natvig, and Jan Christian Meyer. "Performance and energy impact of parallelization and vectorization techniques in modern microprocessors." In: *Computing* 96.12 (2014), pp. 1179–1193.

[22] Sravanthi Chalasani and James M Conrad. "A survey of energy harvesting sources for embedded systems." In: *Southeastcon, 2008. IEEE*. IEEE. 2008, pp. 442–447.

[23] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. "Managing Energy and Server Resources in Hosting Centers." In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2001, pp. 103–116.

[24] Sao-Jie Chen, Guang-Huei Lin, Pao-Ann Hsiung, and Yu-Hen Hu. *Hardware software co-design of a multimedia SOC platform*. Springer Science & Business Media, 2009.

[25] B Clark, WG Graves, JE Lopez-de-Cardenas, ME Gurfinkel, and AW Peats. "Working Document of the NPC Global Oil and Gas Study." In: *Topic Paper* 22 (2007).

[26] Gary Cook, J Lee, T Tsai, A Kong, J Deans, B Johnson, and E Jardim. "Clicking Clean: Who is winning the race to build a Green Internet?" In: *Greenpeace International, Amsterdam, The Netherlands* (2017).

[27] International Business Machines Corp. *IBM System/360 Model 91 Functional Characteristics, White Plains*. Tech. rep. (GA22-6907). 1966.

[28] Control Data Corporation. *Control Data 6400/6500/6600 Computer Systems...: Reference Manual ſs*. 1966.

[29] H. Coullon and S. Limet. "The SIPSim implicit parallelism model and the SkelGIS library." In: *Concurrency and Computation: Practice and Experience* (2015). ISSN: 1532-0634. DOI: 10.1002/cpe.3494. URL: http://dx.doi.org/10.1002/cpe.3494.

[30] Hélène Coullon, Julien Bigot, and Christian Perez. "Extensibility and Composability of a Multi-Stencil Domain Specific Framework." In: *International Journal of Parallel Programming* (Nov. 21, 2017). ISSN: 1573-7640. DOI: 10.1007/s10766-017-0539-5. URL: https://doi.org/10.1007/s10766-017-0539-5.

[31] Leonardo Dagum and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." In: *IEEE computational science and engineering* (1998), pp. 46–55.

[32] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. "Data center energy consumption modeling: A survey." In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 732–794.

[33] Michel Daydé, Benjamin Depardon, Alain Franc, Jean-François Gibrat, Romaric Guilllier, Yasaman Karami, Frédéric Suter, Bruck Taddese, Marie Chabbert, and Sylvie Thérond. "E-Biothon: an experimental platform for BioInformatics." In: *International Conference on Computer Science and Information Technologies (CSIT)*. 2015, pp. 1–4.

[34] Mark A Delucchi and Mark Z Jacobson. "Providing all global energy with wind, water, and solar power, Part II: Reliability, system and transmission costs, and policies." In: *Energy Policy* 39.3 (2011), pp. 1170–1190.

[35] Erik D. Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. "Scheduling to Minimize Gaps and Power Consumption." In: *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 2007, pp. 46–54.

[36] Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, Eric Darve, Juan Alonso, and Pat Hanrahan. "Liszt: A Domain Specific Language for Building Portable Mesh-based PDE Solvers." In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '11. Seattle, Washington: ACM, 2011, 9:1–9:12. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063396. URL: http://doi.acm.org/10.1145/2063384.2063396.

[37] Sheng Di and Franck Cappello. "Optimization of Error-Bounded Lossy Compression for Hard-to-Compress HPC Data." In: *IEEE Transactions on Parallel and Distributed Systems* 29.1 (2018), pp. 129–143.

[38] *DOE Shifts Exascale Plans into High Gear, Asks Supercomputing Vendors to Submit Proposals Top 500*. https://www.top500.org/news/doe-shifts-exascale-plans-into-high-gear-asks-supercomputing-vendors-to-submit-proposals/. 2018.

[39] Jack Dongarra et al. "The international exascale software project roadmap." In: *International Journal of High Performance Computing Applications* (2011), p. 1094342010391989.

[40] Jack Dongarra. "Visit to the national university for defense technology changsha, china." In: *Oak Ridge National Laboratory, Tech. Rep., June* (2013).

[41] Thomas H Dunigan. *Early experiences and performance of the Intel Paragon*. Tech. rep. Oak Ridge National Lab., TN (United States), 1994.

[42] EDF. *Fessenheim, en bref*. https://www.edf.fr/sites/default/files/documents/2015//fessenheim_en_bref.pdf. 2015.

[43] Vadim Elisseev, John Baker, Neil Morgan, Luigi Brochard, and Terry Hewitt. "Energy Aware Scheduling Study on BlueWonder." In: *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing, E2SC 2016, Salt Lake, Utah, USA, November 20, 2016*. 2016.

[44] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. "Power Limitations and Dark Silicon Challenge the Future of Multicore." In: *ACM Transactions on Computer Systems (TOCS)* 30.3 (Aug. 2012), 11:1–11:27.

[45] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. "Parallel job scheduling for power constrained HPC systems." In: *Parallel Computing* (2012).

[46] Wu-chun Feng and Kirk Cameron. "The green500 list: Encouraging sustainable supercomputing." In: *Computer* 40.12 (2007).

[47] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[48] Ian Foster, Mark Ainsworth, Bryce Allen, Julie Bessac, Franck Cappello, Jong Youl Choi, Emil Constantinescu, Philip E Davis, Sheng Di, Wendy Di, et al. "Computing just what you need: online data analysis and reduction at extreme scales." In: *European Conference on Parallel Processing*. Springer. 2017, pp. 3–19.

[49] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. "The Implementation of the Cilk-5 Multithreaded Language." In: *SIGPLAN Not.* 33.5 (May 1998), pp. 212–223. ISSN: 0362-1340.

[50] Bill Gallas and Vandana Verma. "Embedded Pentium (R) processor system design for Windows CE." In: *Wescon/98*. IEEE. 1998, pp. 114–123.

[51] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A. Kozuch. "Optimality analysis of energy-performance trade-off for server farm management." In: *Performance Evaluation* 67.11 (2010), pp. 1155–1171.

[52] Thierry Gautier, Joao V. F. Lima, Nicolas Maillard, and Bruno Raffin. "XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures." In: *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1299–1308. ISBN: 978-0-7695-4971-2.

[53] Yiannis Georgiou, David Glesser, Krzysztof Rzadca, and Denis Trystram. "A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC." In: *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE. 2015, pp. 617–626.

[54] Neha Gholkar, Frank Mueller, and Barry Rountree. "Power Tuning HPC Jobs on Power-Constrained Systems." In: *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*. 2016, pp. 179–191.

[55] H Julian Goldsmid. "Bismuth telluride and its alloys as materials for thermoelectric generation." In: *Materials* 7.4 (2014), pp. 2577–2592.

[56] Steve Greenberg, Evan Mills, Bill Tschudi, Peter Rumsey, and Bruce Myatt. "Best practices for data centers: Lessons learned from benchmarking 22 data centers." In: *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings in Asilomar, CA. ACEEE, August* 3 (2006), pp. 76–87.

[57] Gerd Griepentrog, Simon Hüttinger, and Guilhem Vidiella. "21 EU HeatReCar Project: Waste Heat Recovery by Thermoelectric Power Conversion in Light-Duty Trucks." In: *Thermoelektrik* 94 (2009), p. 267.

[58] William Gropp and Ewing Lusk. "The MPI communication library: its design and a portable implementation." In: *Scalable Parallel Libraries Conference, 1993., Proceedings of the*. IEEE. 1993, pp. 160–165.

[59] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision." In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[60] Jie Han and Michael Orshansky. "Approximate computing: An emerging paradigm for energy-efficient design." In: *Test Symposium (ETS), 2013 18th IEEE European*. IEEE. 2013, pp. 1–6.

[61] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011, p. 22.

[62] Intel Hewlett-Packard. *Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification*. 2004.

[63] Chung-Hsing Hsu and Ulrich Kremer. "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction." In: *ACM SIGPLAN Notices*. Vol. 38. 5. ACM. 2003, pp. 38–48.

[64] Rafael J. Wysocki Intel Corp. *System Power Management Sleep States*. `https://www.kernel.org/doc/html/latest/admin-guide/pm/sleep-states.html`. 2014.

[65] Brian Jeff. "Big.LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration." In: *Annual Design Automation Conference (DAC)*. 2012, pp. 1143–1146.

[66] Bhavesh Khemka et al. "Utility Driven Dynamic Resource Management in an Oversubscribed Energy-Constrained Heterogeneous System." In: *International Parallel and Distributed Processing Symposium Workshop (IPDPS) Workshops*. 2014.

[67] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. "System level analysis of fast, per-core DVFS using on-chip switching regulators." In: *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. IEEE. 2008, pp. 123–134.

[68] Sangpil Lee, Keunsoo Kim, Gunjae Koo, Hyeran Jeon, Murali Annavaram, and Won Woo Ro. "Improving energy efficiency of GPUs through data compression and compressed execution." In: *IEEE Transactions on Computers* 66.5 (2017), pp. 834–847.

[69] Elie Lefeuvre, Adrien Badel, Claude Richard, Lionel Petit, and Daniel Guyomar. "A comparison between several vibration-powered piezoelectric generators for standalone systems." In: *Sensors and Actuators A: Physical* 126.2 (2006), pp. 405–416.

[70] Dong Li, Bronis R de Supinski, Martin Schulz, Kirk W Cameron, and Dimitrios S Nikolopoulos. "Hybrid MPI/OpenMP power-aware computing." In: *IPDPS*. Vol. 10. 2010, pp. 1–12.

[71] Gang Li, Vishal Shrotriya, Jinsong Huang, Yan Yao, Tom Moriarty, Keith Emery, and Yang Yang. "High-efficiency solution processable polymer photovoltaic cells by self-organization of polymer blends." In: *Nature materials* 4.11 (2005), pp. 864–868.

[72] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. "Understanding the energy efficiency of simultaneous multithreading." In: *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM. 2004, pp. 44–49.

[73] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. "Dynamic Right-sizing for Power-proportional Data Centers." In: *IEEE/ACM Transactions on Networking (TON)* 21.5 (Oct. 2013), pp. 1378–1391. ISSN: 1063-6692.

[74] Chris Lomont. "Introduction to intel advanced vector extensions." In: *Intel White Paper* (2011), pp. 1–21.

[75] Aniruddha Marathe, Peter E Bailey, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R de Supinski. "A run-time system for power-constrained HPC applications." In: *International Conference on High Performance Computing*. Springer. 2015, pp. 394–408.

[76] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst D Simon. "Top500 supercomputer sites." In: *Proceedings of SC* (2001), pp. 10–16.

[77] Sparsh Mittal. "A survey of techniques for improving energy efficiency in embedded computing systems." In: *International Journal of Computer Aided Engineering and Technology* 6.4 (2014), pp. 440–459.

[78] P. Murali and S. Vadhiyar. "Metascheduling of HPC Jobs in Day-Ahead Electricity Markets." In: *High Performance Computing (HiPC)*. 2015.

[79] Andrey Nikolaev, Ilya Burylov, and Sania Salahuddin. "Intel® version of STAC-A2 benchmark: toward better performance with less effort." In: *Proceedings of the 6th Workshop on High Performance Computational Finance*. ACM. 2013, p. 7.

[80] Anne-Cécile Orgerie, Laurent Lefèvre, and Jean-Patrick Gelas. "Save Watts in Your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems." In: *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. Melbourne, Australia, Dec. 2008, pp. 171–178. DOI: 10.1109/ICPADS.2008.97.

[81] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. "Practical Resource Management in Power-Constrained, High Performance Computing." In: *High-Performance Parallel and Distributed Computing (HPDC)*. 2015.

[82] Alex Peleg and Uri Weiser. "MMX technology extension to the Intel architecture." In: *IEEE micro* 16.4 (1996), pp. 42–50.

[83] Steven Pelley, David Meisner, Thomas F Wenisch, and James W VanGilder. "Understanding and abstracting total data center power." In: *Workshop on Energy-Efficient Design*. Vol. 11. 2009.

[84] Gregory F Pfister. *In search of clusters: the coming battle in lowly parallel computing*. Prentice-Hall, Inc., 1995.

[85] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems." In: *Workshop on Compilers and Operating Systems for Low Power*. 2001, pp. 182–195.

[86] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines." In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. Seattle, Washington, USA: ACM, 2013, pp. 519–530. ISBN: 978-1-4503-2014-6. DOI: 10.1145/2491956.2462176. URL: http://doi.acm.org/10.1145/2491956.2462176.

[87] SKT Ravindran, T Huesgen, M Kroener, and P Woias. "A self-sustaining micro thermomechanic-pyroelectric generator." In: *Applied Physics Letters* 99.10 (2011), p. 104102.

[88] François Rossigneux, Laurent Lefevre, Jean-Patrick Gelas, and Marcos Dias De Assuncao. "A generic and extensible framework for monitoring energy consumption of openstack clouds." In: *The 4th IEEE International Conference on Sustainable Computing and Communications (Sustaincom 2014)*. IEEE. 2014, pp. 696–702.

[89] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz. "Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound." In: *IEEE International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*. May 2012, pp. 947–953.

[90] Edward S Rubin, Anand B Rao, and Chao Chen. "Comparative assessments of fossil fuel power plants with CO2 capture and storage." In: *Proceedings of the 7th International Conference on Greenhouse Gas Control Technologies*. 2005.

[91] Mohsin Saleemi, Muhammet S Toprak, Shanghua Li, Mats Johnsson, and Mamoun Muhammed. "Synthesis, processing, and thermoelectric properties of bulk nanostructured bismuth telluride (Bi 2 Te 3)." In: *Journal of Materials Chemistry* 22.2 (2012), pp. 725–730.

[92] David Salerno. "Ultralow voltage energy harvester uses thermoelectric generator for battery-free wireless sensors." In: *LT Journal* 2010 (2010), pp. 1–11.

[93] Nadathur Satish, Changkyu Kim, Jatin Chhugani, and Pradeep Dubey. "Large-scale energy-efficient graph traversal: a path to efficient data-intensive supercomputing." In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE. 2012, pp. 1–11.

[94] Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. "Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi." In: *Parallel Processing and Applied Mathematics*. Springer, 2013, pp. 559–570.

[95] C. Schmitt, S. Kuckuk, F. Hannig, H. Köstler, and J. Teich. "ExaSlang: A Domain-specific Language for Highly Scalable Multigrid Solvers." In: *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. WOLFHPC '14. New Orleans, Louisiana: IEEE Press, 2014, pp. 42–51. ISBN: 978-1-4799-7020-9. DOI: 10.1109/WOLFHPC.2014.11. URL: http://dx.doi.org/10.1109/WOLFHPC.2014.11.

[96] Seagate. *Desktop HDD specification sheet*. http://www.seagate.com/staticfiles/docs/pdf/datasheet/disc/desktop-hdd-data-sheet-ds1770-1-1212us.pdf. 2012.

[97] Seagate. *NAS HDD specification sheet*. http://www.seagate.com/www-content/product-content/nas-fam/nas-hdd/_shared/docs/nas-hdd-8tb-ds1789-5-1510DS1789-5-1510US-en_US.pdf. 2015.

[98] John Shalf, Sudip Dosanjh, and John Morrison. "Exascale computing technology challenges." In: *High Performance Computing for Computational Science–VECPAR 2010*. Springer, 2010, pp. 1–25.

[99] Daniel L Slotnick. "The Conception and Development of Parallel Processors: A Personal Memoir." In: *Annals of the History of Computing* 4.1 (1982), pp. 20–30.

[100] G Jeffrey Snyder and Eric S Toberer. "Complex thermoelectric materials." In: *Nature materials* 7.2 (2008), pp. 105–114.

[101] D Suleiman, M Ibrahim, and I Hamarash. "Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction." In: *4th International Conference on Electrical and Electronics Engineering*. 2005.

[102] Y. Tang, R.A. Chowdhury, B.C. Kuszmaul, C-K Luk, and C.E. Leiserson. "The pochoir stencil compiler." In: *SPAA*. Ed. by Lance Fortnow and Salil P. Vadhan. ACM, 2011, pp. 117–128. ISBN: 978-1-4503-0743-7. URL: http://dblp.uni-trier.de/db/conf/spaa/spaa2011.html#TangCKLL11.

[103] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. "Exploration of pattern-matching techniques for lossy compression on cosmology simulation data sets." In: *International Conference on High Performance Computing*. Springer. 2017, pp. 43–54.

[104] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization." In: *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE. 2017, pp. 1129–1139.

[105] Marc Tchiboukdjian, Nicolas Gast, and Denis Trystram. "Decentralized list scheduling." In: *Annals of Operations Research* 207.1 (2013), pp. 237–259.

[106] Ananta Tiwari, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. "Auto-tuning for Energy Usage in Scientific Applications." In: *Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 – September 2, 2011, Revised Selected Papers, Part II*. Ed. by Michael Alexander, Pasqua D'Ambra, Adam Belloum, George Bosilca, Mario Cannataro, Marco Danelutto, Beniamino Di Martino, Michael Gerndt, Emmanuel Jeannot, Raymond Namyst, Jean Roman, Stephen L. Scott, Jesper Larsson Traff, Geoffroy Vallée, and Josef Weidendorfer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 178–187. ISBN: 978-3-642-29740-3. DOI: 10.1007/978-3-642-29740-3_21. URL: https://doi.org/10.1007/978-3-642-29740-3_21.

[107] Brian TSAY. "The Tianhe-2 Supercomputer: Less than Meets the Eye?" In: *SITC Bulletin Analysis* (2013).

[108] K Uchida, S Takahashi, K Harii, J Ieda, W Koshibae, K Ando, S Maekawa, and E Saitoh. "Observation of the spin Seebeck effect." In: *Nature* 455.7214 (2008), pp. 778–781.

[109] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson, and Patricia Stolf. ""Big, Medium, Little": Reaching Energy Proportionality with Heterogeneous Computing Scheduler." In: *Parallel Processing Letters* 25.3 (2015).

[110] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson, and Patricia Stolf. "Towards Generalizing "Big Little" for Energy Proportional HPC and Cloud Infrastructures." In: *IEEE International Conference on Big Data and Cloud Computing (BDCloud)*. 2014, pp. 703–710.

[111] Violaine Villebonnet, Georges Da Costa, Laurent Lefevre, Jean-Marc Pierson, and Patricia Stolf. ""Big, Medium, Little": Reaching Energy Proportionality with Heterogeneous Computing Scheduler." In: *Parallel Processing Letters* 25.03 (2015), p. 1541006.

[112] Philippe Virouleau, François Broquedis, Thierry Gautier, and Fabrice Rastello. "Using Data Dependencies to Improve Task-Based Scheduling Strategies on NUMA Architectures." In: *Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 531–544. ISBN: 978-3-319-43658-6. DOI: 10.1007/978-3-319-43659-3_39.

[113] David W Walker and Jack J Dongarra. "MPI: A standard message passing interface." In: *Supercomputer* 12 (1996), pp. 56–68.

[114] Benjamin Welton, Dries Kimpe, Jason Cope, Christina M Patrick, Kamil Iskra, and Robert Ross. "Improving i/o forwarding throughput with data compression." In: *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE. 2011, pp. 438–445.

[115] Qiang Wu, Margaret Martonosi, Douglas W Clark, Vijay Janapa Reddi, Dan Connors, Youfeng Wu, Jin Lee, and David Brooks. "A dynamic compilation framework for controlling microprocessor energy and performance." In: *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2005, pp. 271–282.

[116] Rong Xu, Zhiyuan Li, Cheng Wang, and Peifeng Ni. "Impact of data compression on energy consumption of wireless-networked handheld devices." In: *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*. IEEE. 2003, pp. 302–311.

[117] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E. Papka. "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems." In: *SuperComputing (SC)*. 2013.

[118] Xin Zhan and S. Reda. "Techniques for energy-efficient power budgeting in data centers." In: *ACM/EDAC/IEEE Design Automation Conference (DAC)*. May 2013, pp. 1–7.

[119] W. Zhang, Y. Wen, Y. Wah Wong, K. Chuan Toh, and C. H. Chen. "Towards Joint Optimization Over ICT and Cooling Systems in Data Centre: A Survey." In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 1596–1616.

[120] Yu Zhou, Somnath Paul, and Swarup Bhunia. "Harvesting wasted heat in a microprocessor using thermoelectric generators: modeling, analysis and measurement." In: *Proceedings of the conference on Design, automation and test in Europe*. ACM. 2008, pp. 98–103.

List of publications

Articles in International Journals

[121] Anne Benoit, Laurent Lefèvre, Anne-Cécile Orgerie, and Issam Raïs. "Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints." In: *International Journal of High Performance Computing Applications* 32.1 (Jan. 2018), pp. 176–188. DOI: 10.1177/1094342017714530. URL: https://hal.inria.fr/hal-01557025.

[122] João Vicente Ferreira Lima, Issam Raïs, Laurent Lefèvre, and Thierry Gautier. "Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms." In: *The International Journal of High Performance Computing Applications* (), p. 1094342018792079. DOI: 10.1177/1094342018792079. eprint: https://doi.org/10.1177/1094342018792079. URL: https://doi.org/10.1177/1094342018792079.

[123] Issam Raïs, Orgerie Anne-Cécile, Quinson Martin, and Lefèvre Laurent. "Quantifying the impact of shutdown techniques for energy-efficient data centers." In: *Concurrency and Computation: Practice and Experience* (). e4471 cpe.4471, e4471. DOI: 10.1002/cpe.4471. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4471. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4471.

Articles in International Conferences

[124] Anne Benoit, Laurent Lefèvre, Anne-Cécile Orgerie, and Issam Raïs. "Shutdown Policies with Power Capping for Large Scale Computing Systems." In: *Euro-Par 2017: Parallel Processing*. Ed. by Francisco F. Rivera, Tomás F. Pena, and José C. Cabaleiro. Cham: Springer International Publishing, 2017, pp. 134–146. ISBN: 978-3-319-64203-1.

[125] P. F. Dutot, Y. Georgiou, D. Glesser, L. Lefevre, M. Poquet, and I. Rais. "Towards Energy Budget Control in HPC." In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. May 2017, pp. 381–390. DOI: 10.1109/CCGRID.2017.16.

[126] Issam Raïs, Mathilde Boutigny, Laurent Lefèvre, Anne-Cécile Orgerie, and Anne Benoit. "Building the Table of Energy and Power Leverages for Energy Efficient Large Scale Systems." In: *HPCS2018 : IEEE The 16th Annual Meeting of International Conference on High Performance Computing & Simulation*. Orleans, France, 2018.

[127] Issam Raïs, Laurent Lefevre, Anne-Cécile Orgerie, and Anne Benoit. "Exploiting the Table of Energy and Power Leverages." In: *ICA3PP 2018 : 18th International Conference on Algorithms and Architectures for Parallel Processing*. Nov. 2018.

[128] Issam Raïs, Anne-Cécile Orgerie, and Martin Quinson. "Impact of Shutdown Techniques for Energy-Efficient Cloud Data Centers." In: *Algorithms and Architectures for Parallel Processing*. Ed. by Jesus Carretero, Javier Garcia-Blas, Ryan K.L. Ko, Peter Mueller, and Koji Nakano. Cham: Springer International Publishing, 2016, pp. 203–210. ISBN: 978-3-319-49583-5.

Articles in International Workshops

[129] J. V. F. Lima, I. Raïs, L. Lefevre, and T. Gautier. "Performance and Energy Analysis of OpenMP Runtime Systems with Dense Linear Algebra Algorithms." In: *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*. Oct. 2017, pp. 7–12. DOI: 10.1109/SBAC-PADW.2017.10.

[130] I. Rais, L. Lefevre, A. C. Orgerie, and A. Benoit. "An analysis of the feasibility of energy harvesting with thermoelectric generators on petascale and exascale systems." In: *2016 International Conference on High Performance Computing Simulation (HPCS)*. July 2016, pp. 808–813. DOI: 10.1109/HPCSim.2016.7568417.