Le traçage logiciel d'applications parallèles : conception et ajustement de qualité

Eric Maillet

Thèse

présentée par

Eric MAILLET

pour obtenir le grade de Docteur

de l'Institut National Polytechnique de Grenoble

(arrêté ministériel du 30 Mars 1992)

(Spécialité : **Informatique**)

Le traçage logiciel d'applications parallèles : conception et ajustement de qualité

Date de soutenance : 6 septembre 1996

Composition du jury

Président :

Michel Adiba

Examinateurs :

Claude Jard (Rapporteur)

Brigitte Plateau (Directeur de thèse)

Pierre Valiron

Jan Van Campenhout (Rapporteur)

Jean-Marc Vincent

Thèse préparée au sein du

Laboratoire de Modélisation et Calcul – IMAG

à Simone
à mes parents
et à tous mes amis

Remerciements

Je tiens à remercier Michel Adiba qui m'a fait l'honneur de présider ce jury de thèse.

Je remercie également Claude Jard et Jan Van Campenhout d'avoir accepté de rapporter sur ce travail de thèse.

Je remercie Pierre Valiron qui a accepté de participer à mon jury de thèse.

Un grand merci également à mes directeurs de thèse Brigitte Plateau et Jean-Marc Vincent. Les nombreuses discussions qu'on a eu ensemble, que ce soit dans le cadre de l'équipe ALPES ou en tête à tête, étaient pour moi d'une très grande valeur.

Merci à tous ceux qui ont lu le manuscrit de ma thèse, me faisant part de leurs commentaires et suggestions. Je pense bien évidemment aux rapporteurs, Pierre, Brigitte et Jean-Marc, mais aussi à Jacques Chassin, Florin Teodorescu, Pierre-Eric Bernard et João Paulo Kitajima.

Je tiens également à remercier mes collègues de bureau Fred Guinand et Pierre-Eric qui, bien qu'«imperturbables» devant leurs écrans (que ce soit au travail ou au jeu ;-), étaient toujours présents, prêts à discuter (et pas que d'informatique) créant ainsi une ambiance de travail particulièrement agréable que j'ai bien appréciée lors de la rédaction. Merci aussi à tous les membres de notre ancienne équipe du centre ville, au sein de laquelle j'ai commencé ma thèse : Brigitte, Jean-Marc, Jacques, Denis, Jean-Louis, Gilles, Joelle, Nathalie, João Paulo, Alain, Pascal, Fred, Cécile, Yves, Titou, Lolo, Phil, Xtof, Yannick, Michel C., Michel R., Alexandre, Luc, Evelyne et honte à moi si j'ai oublié quelqu'un ! Merci également à Alain, l'«autre Luxembourgeois» de Grenoble...

Je remercie également Laurent et Matthieu qui étaient particulièrement efficace dans leur contribution au codage de Tape/PVM. Merci aussi à tous les utilisateurs de Tape/PVM, en particulier à João Paulo, Wlodzimierz et Francesco, dont les remarques m'ont permis de corriger un grand nombre de «bugs».

Finalement, je tiens tout particulièrement à remercier mes parents qui qui m'ont toujours encouragé à faire des études et si aujourd'hui, j'en suis arrivé là, je le dois beaucoup à eux. Finalement j'exprime toute ma gratitude à Simone pour sa présence, ses encouragements et sa bonne humeur durant

ces trois années de thèse à Grenoble.

Résumé de la thèse

En raison de leur faible coût, les traceurs logiciels sont aujourd'hui largement utilisés pour mesurer les performances d'exécutions d'applications parallèles. A part leur caractère économique, ces traceurs sont de surcroît facilement portables, s'adaptent à un nombre variable de processeurs («scalabilité») et fournissent des mesures dont la sémantique est celle de l'application tracée. Malgré ces atouts, le traceur logiciel ne peut pas fournir la même *qualité de mesures* qu'un traceur matériel ou hybride. Ces derniers disposent en effet d'une horloge globale de temps physique et de dispositifs dédiés à la génération et à l'évacuation des informations tracées. D'une part, cela leur permet de dater les actions réparties de façon causalement cohérente et d'autre part, l'instrumentation peut être effectuée sans influencer le comportement de l'exécution observée.

Cette thèse se concentre sur la notion de *qualité représentative des traces* obtenues par voie logicielle sur des exécutions de programmes parallèles communiquant par messages. Nous proposons une série de modèles et de méthodes permettant de réajuster la qualité d'une telle trace afin d'approcher la qualité des mesures obtenues sur un système de trace matériel.

Dans la *première partie* de la thèse, nous commençons par une présentation générale de la mesure et de l'analyse des performances de programmes parallèles, et nous passons en revue quelques techniques et outils existants.

Dans la *deuxième partie*, nous étudions en détail le problème de datation physique dans un environnement d'exécution parallèle dépourvu d'une *horloge physique globale*, mais équipé d'un ensemble d'horloges distinctes, une par processeur. En général, il y a un écart non-négligeable entre les valeurs de ces horloges, écart qui, en plus, est variable dans le temps (dérive). C'est pourquoi, dans un tel environnement, la datation répartie des actions d'un calcul parallèle est difficile. Après avoir rappelé le principe des méthodes statistiques de calcul de temps global, nous proposons une technique qui permet de réduire considérablement le temps d'échantillonnage des horloges. Cette méthode, appelée SBA, a été largement validée par la pratique et a déjà été adoptée dans différents environnements de programmation parallèle,

dont ATHAPASCAN (projet APACHE à l'IMAG), POM (projet PAMPA à l'IRISA) ainsi que le projet européen SEPP (*Software Engineering for Parallel Processing*). Elle offre un accès suffisamment précis et confortable au temps global pour pouvoir rivaliser avec une solution matérielle.

Nous abordons ensuite le problème de l'*effet de sonde* qui résulte du partage des ressources du système entre l'outil d'instrumentation logiciel et l'application instrumentée. Cet effet se manifeste tout d'abord par une augmentation du temps d'exécution, allant jusqu'à 25% sur les systèmes qu'on a étudiés. Pour les applications non-déterministes, comme celles basées sur une décomposition dynamique du travail, l'effet de sonde peut en plus entraîner un changement de comportement. L'analyse des mesures fournies par l'instrumentation peut donc mener à des modifications inefficaces de l'application qui n'améliorent en rien ses performances d'exécution. Nous présentons différents modèles de *correction des perturbations*, permettant de compenser l'effet de sonde par un traitement post-mortem des traces dans le but de retrouver la dynamique originale d'une exécution non-instrumentée. Nous discutons les conditions d'applicabilité de ces modèles. Pour les applications à comportement déterministe, comme les nombreuses applications numériques basées sur une décomposition statique du travail, la correction permet de retrouver exactement la dynamique des exécutions non-instrumentées (aux perturbations indirectes près). Sur ce type d'application, nos expériences montrent que la correction enlève de 70% à 95% des perturbations du temps d'exécution. Nous traitons ensuite plus en détail le cas des applications non-déterministes, comme celles basées sur une décomposition dynamique du travail, et nous introduisons la notion d'approximation conservatrice : la trace dite approchée, calculée par le correcteur des perturbations, appartient à la même classe de comportement que la trace de départ. L'apport des techniques de réexécution déterministe est également discuté.

Dans la *troisième partie*, on présente l'outil de trace Tape/PVM, développé dans le cadre de cette thèse. Les méthodes de qualité de traces proposées dans la deuxième partie ont été implantées dans Tape/PVM : le traceur construit automatiquement une référence globale de temps pour la datation des événements et inclut un outil de correction d'intrusion. Par ailleurs, les traces obtenues peuvent être converties au format PICL ce qui permet leur exploitation avec l'outil de *visualisation* Paragraph. Tape/PVM propose également un outil de prédiction de performances et un outil de *test d'adéquation de modèles des temps de communication* sur des traces d'exécutions réelles.

Table des matières

Table des figures

Liste des tableaux

Première partie

Présentation

Chapitre 1

Introduction générale

Les demandes en puissance de calcul des applications scientifiques, sans cesse croissantes, ont contribué à l'évolution rapide du calcul parallèle et distribué durant la dernière décennie. De plus en plus d'industries s'orientent vers cette solution. Citons par exemple la simulation numérique en industrie aéronautique ou la simulation de dynamique moléculaire en industrie pharmaceutique. En effet, le calcul parallèle à haute performance (*High Performance Computing*, en anglais) joue un rôle crucial dans l'innovation technique ; il permet de réduire le coût et d'augmenter la qualité d'un grand nombre de procédés industriels. D'autres domaines, comme le calcul symbolique ou la programmation logique tirent également profit de l'apparition des systèmes de calcul parallèles. Ces systèmes permettent de traiter plus efficacement des problèmes de taille de plus en plus grande.

Dans le domaine du calcul parallèle haute performance, la tendance actuelle évolue vers les machines parallèles à mémoire distribuée. Contrairement aux machines à mémoire partagée, l'utilisation d'une mémoire locale à chaque processeur permet d'assembler un nombre de processeurs beaucoup plus élevé, pour s'approcher de ce qu'on appelle le parallélisme massif. En même temps, les progrès enregistrés dans les techniques de communication permettent l'échange efficace de données entre un nombre élevé de processeurs [76] ; la distance entre processeurs communiquants, variable qui jouait un rôle central dans les performances des communications sur les premières machines, devient de plus en plus marginale sur les systèmes récents comme le T3D de CRAY ou le SP2 d'IBM [89].

Toutefois, à cause de la complexité à caractériser les performances d'une machine parallèle, il est très difficile d'exploiter efficacement l'extraordinaire puissance de calcul qu'elle offre. C'est pour cette même raison que la prédiction *a priori* des performances d'une application donnée sur une machine donnée est une tâche très difficile – la dynamique d'une application ne peut

être connue précisément qu'*a posteriori*, après avoir expérimenté une exécution effective de cette application. La mise au point des performances d'une application sur une machine parallèle passe ainsi par une succession de révisions et d'observations expérimentales de l'application. Durant ce cycle, le programmeur ajuste petit à petit le grain de découpage du travail et tente au mieux de recouvrir les communications par le calcul. Il devient clair que l'optimisation d'une application parallèle revêt un caractère expérimental et empirique bien plus prononcé que pour une application séquentielle classique. Dans un environnement de programmation parallèle, la disponibilité d'outils de mesure des performances d'exécution est donc cruciale.

Les techniques classiques de comptage et d'échantillonnage (*profiling*) sont insuffisantes pour décrire le comportement fonctionnel et les performances d'une application parallèle. La prise de traces d'événements (*event-driven monitoring*) s'est révélée la technique la mieux adaptée à la compréhension et à l'analyse de la dynamique des exécutions d'une application parallèle sur un système parallèle ou distribué [51, 81]. La prise de traces, dont l'implantation est loin d'être triviale, peut se faire au niveau matériel, logiciel système ou applicatif.

Cette thèse se concentre sur les traceurs logiciels destinés aux développeurs d'applications parallèles communiquant par messages. Par traceur logiciel, on entend un mécanisme de trace qui ne requiert pas de ressources matérielles supplémentaires et qui partage les ressources du système avec l'application observée. En raison de leur faible coût, les traceurs logiciels sont aujourd'hui largement utilisés pour mesurer les performances d'exécutions d'applications parallèles. A part leur caractère économique, ces traceurs sont de surcroît facilement portables, s'adaptent à un nombre variable de processeurs («scalabilité») et fournissent des mesures dont la sémantique est celle de l'application tracée. Malgré ces atouts, les traceurs logiciels ne peuvent pas fournir la même *qualité de mesures* que les traceurs matériels ou hybrides. Ces derniers disposent en effet d'une horloge globale de temps physique et de dispositifs dédiés à la génération et à l'évacuation des informations tracées [16, 51]. D'une part, cela leur permet de dater les actions réparties de façon causalement cohérente et d'autre part, l'instrumentation peut être effectuée sans influencer le comportement de l'exécution observée.

Le but de ce travail de thèse est le développement d'une méthodologie de mise au point de la qualité des traces d'exécution obtenues par voie logicielle et la validation de cette méthodologie par l'expérience. Nous proposons une série d'algorithmes de correction de traces basés sur un modèle exécutif des applications et sur des modèles de divers éléments de l'architecture de la machine sous-jacente (horloges physiques, communications). Les outils de post-traitement de traces basés sur cette méthodologie permettent de réajuster

la qualité d'une trace afin d'approcher la qualité des mesures obtenues avec un système de trace matériel équipé d'une horloge physique globale et de sondes matérielles non-intrusives.

Organisation de ce document

Ce document est organisé en quatre parties. La première partie, «Présentation», comprend deux chapitres. Le chapitre 1, «Introduction générale», que vous êtes en train de lire, a rapidement présenté le contexte et la problématique du traçage logiciel de programmes parallèles. Le chapitre 2, «La mesure des performances de programmes parallèles», se concentre plus en détail sur la méthodologie et les techniques utilisées et présente quelques environnements de mesure et d'analyse de performance existants.

La deuxième partie, «La qualité représentative des traces», introduit notre méthodologie d'ajustement de la qualité des traces. Elle comprend deux chapitres qui peuvent être lus indépendamment.

Le chapitre 3, «La datation globale des événements», traite le problème de l'absence de référence globale de temps physique. Nous allons voir qu'il est possible d'obtenir une estimation suffisamment précise d'une référence globale de temps pour dater les événements de façon causalement cohérente. Après une analyse détaillée de la nature de l'erreur d'estimation, nous proposons une méthode de calcul du temps global par interpolation, qui permet d'obtenir une bonne précision, même avec des périodes d'échantillonnage courtes.

Le chapitre 4, «La correction de l'effet de sonde», propose un modèle des perturbations induites par le traceur logiciel. Nous montrons comment ces perturbations peuvent être caractérisées et enlevées des traces d'exécution d'applications parallèles. Après avoir décrit le problème du non-déterminisme de comportement de certaines applications, nous donnons une série de résultats expérimentaux sur des applications numériques.

La troisième partie de la thèse, «Le traceur Tape/PVM», décrit le traceur logiciel Tape/PVM développé dans le cadre de ce travail de thèse. Elle est organisée en deux chapitres.

Le chapitre 5, «La génération de traces d'exécution», commence par la description de l'environnement ALPES, dont fait partie Tape/PVM, avant de présenter l'architecture générale du traceur.

Le chapitre 6, «Le post-traitement des traces», décrit nos méthodes d'ajustement de qualité de traces, présentées dans la deuxième partie, telles qu'elles ont été implantées dans Tape/PVM. A part les outils de calcul d'un temps global physique et de correction des perturbations, Tape/PVM propose aussi

des outils de tests de modèles de communication et de prédiction de performances.

La dernière partie, «Conclusion», donne le bilan et les perspectives de ce travail de thèse.

Chapitre 2

La mesure des performances de programmes parallèles

Dans ce chapitre, nous proposons une introduction à la mesure des performances de programmes parallèles. Il s'agit de mettre en évidence les différences avec la programmation séquentielle classique, de montrer sous quelle forme la performance d'une exécution parallèle peut être présentée, de décrire les techniques d'instrumentation et la problématique associée. Nous finirons par la présentation de deux environnements de mesure et d'évaluation de performances récents et nous situerons Tape/PVM, outil de trace développé par l'auteur de la thèse, par rapport à ces environnements.

2.1 Introduction

La motivation principale pour le développement d'une application scientifique parallèle ou distribuée est la vitesse d'exécution : l'utilisation d'une architecture multi-processeur, que ce soit une machine parallèle ou un réseau de stations de travail[1], doit permettre non seulement d'augmenter la *vitesse* de traitement d'un travail mais aussi d'accroître la *complexité* de ce travail. A ce propos, on peut donner l'exemple de la météorologie, où le parallélisme permet d'effectuer des simulations basées sur des modèles plus complexes des phénomènes atmosphériques.

Une fois que les phases de vérification et de validation d'une application parallèle sont terminées, le programmeur se concentre sur l'amélioration des performances de cette application, i.e. sur la réduction de son temps d'exécution. En effet, les performances d'une première implémentation sont géné-

1. Dans le cas de l'utilisation d'un réseau de stations de travail, on parle souvent de *machine parallèle virtuelle* [29, 32] et de *calcul distribué*.

ralement décevantes au vu de ce que l'étude de complexité de l'algorithme utilisé et les données techniques de la machine parallèle cible pourraient laisser croire [64].

Les *performances applicatives*, comme la vitesse d'exécution ou le taux d'accélération (cf. section 2.2), sont en général *instables* sur une architecture parallèle ; de plus, on est incapable actuellement de *prédire* les performances d'une application donnée sur une architecture parallèle donnée [81]. Les *performances architecturales*, (données techniques) comme la performance de crête[2] (en MIPS, MFLOPS) ou la fréquence d'horloge des processeurs, ne sont que des indicateurs très grossiers des performances applicatives. Même dans le domaine des machines séquentielles, il devient de plus en plus hasardeux de se fier aux seules performances architecturales pour prédire quelle machine sera la plus rapide dans l'exécution d'un travail donné. Dans le domaine des machines parallèles, le nombre de facteurs qui jouent sur la performance est encore bien plus élevé : non seulement elles renferment de multiples copies des ressources classiques (CPU, registres, caches, système d'entrée-sortie) mais elles disposent aussi d'éléments spécifiques comme les canaux de communication inter-processeur, par exemple. L'existence de copies multiples de ressources implique des couches matérielles et logicielles supplémentaires implémentant les protocoles de cohérence de cache (système fortement couplé) ou de communication (système faiblement couplé), par exemple.

Les jeux d'essais (*benchmarks*, en anglais) dont le but est de fournir une évaluation quantitative de la puissance d'une machine évoluent vers de véritables collections d'applicatifs d'un grand nombre de domaines du calcul scientifique et de l'ingénierie [8, 35]. Ces jeux d'essais mettent à l'épreuve la puissance de calcul et de communication des architectures parallèles et fournissent des indicateurs de performances architecturales et applicatives pouvant servir à comparer différentes architectures.

L'instabilité et la difficulté de prédiction des performances applicatives résultent des interactions complexes et généralement non-déterministes entre le logiciel applicatif, le système d'exploitation (incluant la gestion des ressources de communication) et le matériel [81]. Afin d'exploiter le plus efficacement possible les ressources offertes par une machine parallèle, le programmeur doit recourir à des *outils de mesure de performance* (cf. section 2.2) qui lui permettent d'isoler expérimentalement des phénomènes indicateurs de problèmes de performance qu'il est incapable de prédire autrement.

La mise au point des performances d'un programme parallèle passe ainsi par une série de phases de mesures, d'évaluation et d'optimisation attribuant

2. *peak performance*

à la programmation parallèle un caractère expérimental plus prononcé que pour la programmation séquentielle classique. La répétition de ces phases permet

- d'adapter au mieux le code applicatif aux particularités de l'architecture multi-processeur utilisée ;
- d'avancer dans la compréhension des interactions complexes entre l'application et l'architecture multi-processeur, ce qui pourra guider nos choix *a priori* lors de développements ultérieurs (le grain de découpage, par exemple).

Du point de vue de l'ingénierie du logiciel, il est indispensable de partager au mieux ces expériences au sein des équipes de développement. La rédaction systématique de rapports d'expérience permet de gagner un temps précieux. Dans certains cas, l'adaptation de l'application aux caractéristiques de l'architecture utilisée va à l'encontre de la *portabilité* de cette application. Toutefois, le protocole expérimental sous-jacent, basée sur des phases successives de mesure et d'analyse, reste valide dans la plupart des environnements parallèles.

2.2 Les outils de mesure de performance

Le but d'un outil de mesure de performance est de présenter au développeur les éléments qui lui permettent de remonter jusqu'à l'origine du problème de performance qui pénalise l'exécution de son application. Nous distinguons deux catégories d'outils selon que le résultat est présenté sous forme d'une métrique de performance ou sous forme graphique.

2.2.1 Les métriques de performance

Les métriques les plus simples sont scalaires et sont basées sur le temps d'exécution de l'application. Tel est le cas du rapport entre le temps séquentiel et le temps parallèle, i.e. le *taux d'accélération* (*speedup*, en anglais). Il s'agit là d'une mesure de succès, car elle nous permet de nous situer par rapport au cas idéal [81], à savoir l'accélération linéaire en fonction du nombre de processeurs. Cette mesure ne fournit cependant aucune indication sur la localisation d'un problème de performance.

Les outils classiques comme *prof* ou *gprof* [31], bien connus du développeur de programmes séquentiels, mesurent les performances sous la forme d'un profil d'exécution (*execution profile*) indiquant la proportion du temps

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
41.5	0.17	0.17			dot
24.4	0.10	0.27	5	20.	_write
9.8	0.04	0.31	1	40.	initmat
7.3	0.03	0.34	40200	0.0007	unf
4.9	0.02	0.36			_mcount
4.9	0.02	0.38	40200	0.0005	random
4.9	0.02	0.40	41000	0.0005	Abs
2.4	0.01	0.41	1	10.	_profil

FIG. 2.1 – *prof: exemple d'un profil d'une exécution séquentielle d'une résolution d'un système linéaire par la méthode de Jacobi (matrice de dimension 200). La colonne «Name» contient le nom des fonctions du programme classées en fonction du pourcentage du temps d'exécution total consommé (colonne «%Time»). La colonne «msec/call» donne la durée moyenne d'un appel de procédure. Sur cet exemple, la fonction «dot», qui calcule un produit scalaire, consomme le plus de temps CPU. On note également la présence d'une fonction «_profil» représentant le code d'instrumentation inséré par l'éditeur des liens: sur cet exemple, le sur-coût induit par l'observation est de 2,4% du temps d'exécution. prof fonctionne par échantillonnage.*

d'exécution total par procédure (cf. figure 2.1). Ces outils permettent au programmeur d'isoler les procédures à optimiser. Pour un programme parallèle, les profils des différents processus peuvent être agrégées pour former un seul profil. Cette métrique ne fournit qu'un bilan statistique sur les performances d'exécution et ne permet pas de savoir à quels moments précis de l'exécution le problème de performance a lieu. En plus, en raison des *dépendances* entre les processus d'un programme parallèle, ce n'est pas nécessairement la procédure qui consomme le plus de temps CPU dont l'optimisation conduira en fin de compte à une diminution du temps total d'exécution [37].

La simple extension des métriques séquentielles est clairement insuffisante pour le processus d'optimisation d'un programme parallèle. C'est pourquoi des métriques de performance spécifiques à l'exécution d'un programme parallèle ont été proposées. Ces métriques sont basées sur l'*histoire de l'exécution d'un programme*, histoire qui peut être représentée par un ensemble d'événements significatifs datés (envois et réceptions de messages, appels et retours de procédures) muni de l'ordre partiel de dépendance causale (graphe d'activité du programme). Le *profil du chemin critique* a été proposé par Yang et Miller en 1988 [97] – les auteurs proposent de calculer un profil des pro-

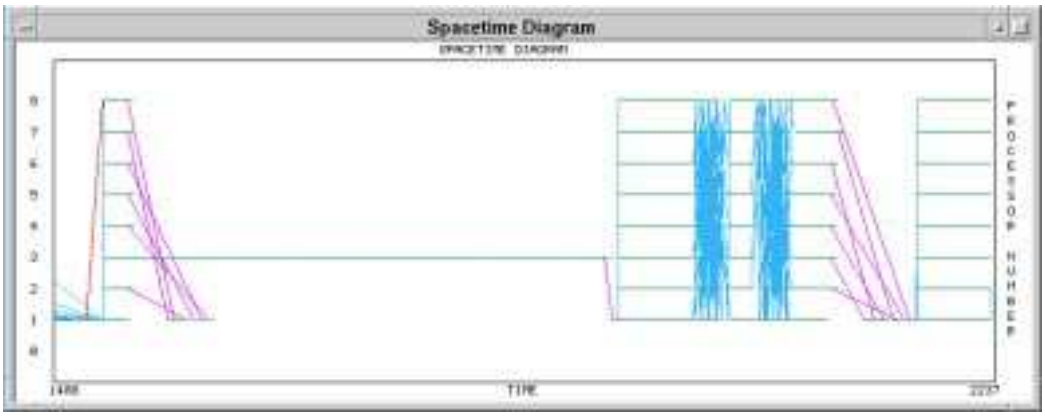


FIG. 2.2 – *Le diagramme espace-temps de Paragraph : illustration visuelle d'un problème de déséquilibrage de charge. L'unité de temps est de 100ms.*

cédures le long du chemin critique du graphe d'activité. D'autres métriques ont été proposées : le lecteur intéressé trouvera dans [38] un comparatif de plusieurs métriques et une analyse de leur qualité à conduire à une amélioration effective des programmes. Dans certains cas, ces métriques aboutissent à des résultats contradictoires et la question d'une métrique idéale reste un problème ouvert. Néanmoins, un environnement de mesure de performance se doit de proposer à l'utilisateur une ou plusieurs de ces métriques.

2.2.2 La visualisation

L'évaluation d'une métrique fournit des résultats numériques, en général une statistique par procédure, guidant l'utilisateur vers la partie de son programme susceptible de causer un problème de performance. Plutôt que de résumer la dynamique d'une exécution par de telles statistiques, la *visualisation* tente de représenter cette dynamique sous forme graphique. Comme pour les métriques, il est très difficile de choisir une visualisation idéale, qui soit simple à comprendre et qui permette d'isoler rapidement l'origine d'un problème de performance. Ainsi, l'outil de visualisation Paragraph [34] présente plus de 30 vues différentes et permet à l'utilisateur, moyennant un effort de programmation, de rajouter ses propres vues.

La pratique des systèmes faiblement couplés communiquant par messages a montré que les vues représentant l'historique d'une exécution, comme le *diagramme espace-temps* (chronogramme) et le *diagramme de Gantt* s'avèrent parmi les plus utiles pour comprendre une dynamique d'exécution. Le diagramme espace-temps montre les états d'activité de chaque processus ainsi que les communications entre processus en fonction du temps. Le diagramme

de Gantt ne montre pas les communications inter-processus, mais illustre les transitions de chaque processus entre trois états : actif, bloqué et système. L'inspection de ces diagrammes permet d'isoler facilement des problèmes de déséquilibrage de charge résultant de contraintes de dépendance entre les processus (blocage en attente de messages). La figure 2.2 montre le diagramme espace-temps d'une partie de l'exécution d'un programme parallèle de tri[3]. L'exécution totale prend 6 minutes ; l'extrait qui nous intéresse ici représente environ 1 minute. Lors de la première itération, le processeur 3 est anormalement lent, sans doute à cause d'activités concurrentes du système d'exploitation et il ne s'agit pas ici d'un problème de performance au niveau applicatif. A partir de la deuxième itération, le processeur 3 est en phase avec les autres.

Nous n'allons pas décrire ici les aspects et la problématique propres à la visualisation et nous renvoyons le lecteur intéressé à [34, 3] pour plus de détails. Il nous importe ici de remarquer que *les outils de visualisation nécessitent tout l'historique d'une exécution* (fichier de trace) dont la taille en termes d'espace de stockage est généralement très importante. L'enregistrement efficace et non-intrusif de ces informations est un des principaux problèmes de l'instrumentation. Il s'agit là d'une différence importante avec les métriques dont l'évaluation peut souvent se faire à la volée.

2.3 L'observation «in vivo»

L'évaluation de la performance d'une exécution, que ce soit graphiquement ou par le biais d'une métrique, implique l'enregistrement ou le calcul, *in vivo*, d'un certain nombre d'informations sur la dynamique de cette exécution. Le programme exécutable doit être *instrumenté* pour qu'il produise ou calcule ces informations lors de son exécution. L'instrumentation consiste dans l'insertion de points de sonde dans le code applicatif : les techniques d'instrumentation sont abordées en sous-section 2.4.1. Le terme d'*observation* est relatif aux aspects qualitatifs (nature et niveau d'abstraction) et quantitatifs (temps physique) des informations produites par les points de sonde.

2.3.1 Les niveaux d'observation

De façon générale, un système informatique comprend une hiérarchie de niveaux. On peut considérer 4 niveaux potentiels d'instrumentation : le maté-

3. Il s'agit du noyau IS du jeux d'essais du NAS [91] instrumenté avec le traceur Tape/PVM [63] sur un IBM-SP2.

riel, le logiciel système, l'interface de programmation [4] (API) et l'application. La nature et le volume des informations à enregistrer, de même que les techniques d'instrumentation associées, dépendent fortement du niveau auquel on se place.

Dans le cadre de cette thèse on s'intéresse à l'optimisation des performances de programmes parallèles écrits dans un langage de haut niveau (C, Fortran) utilisant une API (PVM [29], MPI [72], Athapascan-0 [17]) pour assurer la communication entre processus. Pour les programmes parallèles, les problèmes de performance les plus délicats résident au niveau de la dépendance entre processus : temps de blocage sur un verrou, sur une réception de message ou à une barrière de synchronisation. Vu que la communication entre processus est effectuée par le biais de l'API, les dépendances inter-processus, ainsi que les états de blocage qui en résultent, sont directement conditionnés par les appels des routines de l'API. Pour fixer les idées, considérons une API de programmation par échange de messages sur une architecture à mémoire distribuée et intéressons-nous aux primitives d'envoi et de réception de messages. Comme le montre la figure 2.3, on peut définir différents états pour un processus, suivant qu'il est *actif*, *bloqué* ou en train d'effectuer une *entrée/sortie* pour lire (réception) ou écrire (envoi) les données véhiculées par un message. Les transitions entre ces états sont conditionnées par les appels à l'API et les informations dynamiques sur ces appels (dates de *début* et de *fin* des appels) sont essentielles pour l'évaluation d'une métrique en termes de temps d'activité et de blocage (rapport calcul/communication, par exemple) ou pour représenter graphiquement, en fonction du temps, les états des processus d'un programme parallèle (cf. le diagramme espace-temps et le diagramme de Gantt de Paragraph [34]).

Sans entrer dans les détails, on peut remarquer qu'au *niveau applicatif* on ne peut mesurer que les dates de début et de fin des appels API (en les encadrant de points d'instrumentation dans le code source). En particulier, pour la réception d'un message, il n'est pas possible de distinguer l'état *bloqué* de l'état *entrée/sortie*, car la détection de la transition *arrivée message* requiert l'observation du niveau d'abstraction sous-jacent, à savoir celui de la couche API. L'instrumentation de cette couche n'est possible que si l'on dispose du code source de la couche de l'API. C'est le cas de la bibliothèque de communication PVM, par exemple.

De même, on peut introduire un état supplémentaire en cas de faute de page (figure 2.3) : il y a alors une transition entre l'état *actif* et l'état *chargement de page*. Un défaut de page n'étant pas directement visible au niveau applicatif, une observation à ce niveau ne permet pas de distinguer le temps

4. *Application Programming Interface*

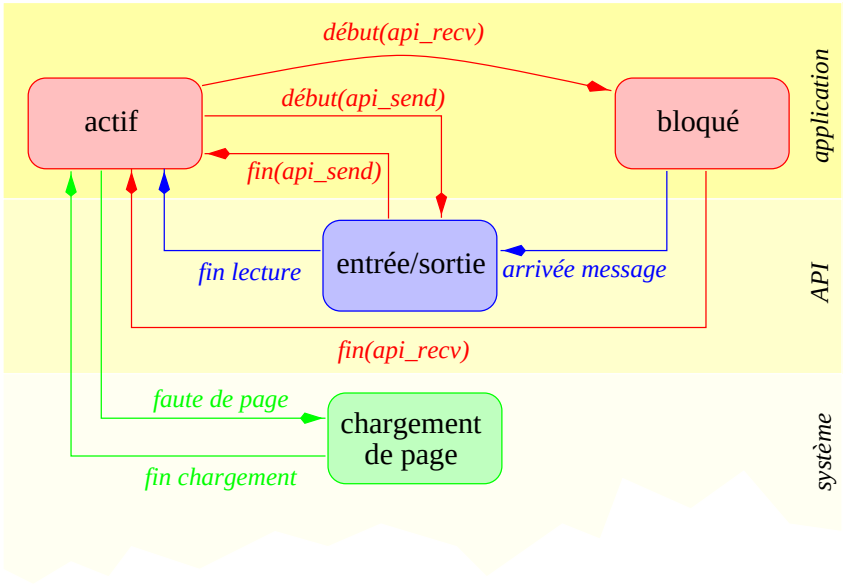


FIG. 2.3 – *Diagramme des états observables et non-observables d'un processus.*

---

de calcul du temps consacré par le système au remplacement de pages (gestion de la mémoire virtuelle). Bien que le système d'exploitation enregistre des statistiques sur le nombre total de fautes de pages par processus, ces informations ne permettent pas de savoir quelle partie du code applicatif a provoqué la faute de page (*où*), ni à quel moment cette faute de page s'est produite (*quand*). Même si le code source du système d'exploitation est disponible et que l'on parvienne à dater les fautes de pages, il est difficile en général de faire le lien entre un tel événement système et la partie concernée du code applicatif. Dans [39], Irvin et Miller proposent un modèle de caractérisation des performances qui permet de faire la correspondance entre des *informations de performance* de bas niveau avec des entités du niveau applicatif. Ce modèle appelé NV (*Noun-Verb*) est basé sur une modélisation de chaque niveau d'abstraction en termes de *noms* (éléments structuraux du programme) et de *verbes* (actions sur les noms).

La méthodologie de qualité représentative développée dans cette thèse concerne les mesures prises par voie logicielle et les niveaux d'instrumentation auxquels on s'intéresse sont donc naturellement ceux de plus haut niveau, i.e. les niveaux applicatif et API.

2.3.2 Les méthodes d'observation

Après avoir décrit les niveaux potentiels d'observation en sous-section 2.3.1, nous décrivons dans cette sous-section les méthodes d'observation couramment utilisées : chronométrage, échantillonnage et traçage événementiel.

Le chronométrage (*timing*)

Le chronométrage de l'exécution d'une application est sans doute une des premières étapes dans la mesure des performances. Toutefois, pour localiser plus précisément la source d'un problème de performance, il faut chronométrer des parties du code applicatif. Ainsi, l'on pourra mesurer le temps total passé dans une procédure en encadrant cette procédure de points d'instrumentation qui lisent l'horloge locale du processeur, effectuant la différence entre les deux estampilles et accumulant le délai résultant dans une variable prévue à cet effet. En général, le chronométrage implémente une observation au niveau applicatif.

A condition que la résolution de l'horloge physique soit suffisamment précise pour mesurer le temps d'exécution du composant fonctionnel étudié, cette approche permet de construire le profil *exact* d'une exécution. En plus, elle ne requiert qu'un espace de fonctionnement restreint, généralement un accumulateur par composant fonctionnel chronométré. Son désavantage est de provoquer l'exécution de deux points d'instrumentation (deux lectures d'horloge, une soustraction) à chaque exécution du composant fonctionnel étudié, ce qui peut induire des *perturbations* importantes de l'exécution.

L'échantillonnage

L'échantillonnage consiste à observer périodiquement l'état du système et à incrémenter un compteur associé à l'état observé [81]. Les outils de mesure de performance basés sur le calcul d'un *profil d'exécution*, comme *gprof* [31] par exemple, échantillonnent le compteur ordinal à intervalles de temps fixes et utilisent sa valeur comme pointeur vers une plage d'adresses pour incrémenter un compteur associé à cette plage. Après l'exécution du programme, la valeur de chaque compteur est proportionnelle au temps passé à exécuter du code dans la plage d'adresses associée. Des informations statiques connues au moment de la compilation permettent de faire le lien entre une plage d'adresses et les entitées du code source correspondantes : les procédures ou, à un grain plus fin, même les blocs élémentaires. En plus de ces observations du niveau applicatif, l'échantillonnage peut également être utilisé pour obtenir des informations du niveau système comme par exemple le

temps qu'un processus donné passe en mode utilisateur et en mode système[5].

Les implantations classiques des techniques d'échantillonnage utilisent une période d'échantillonnage entre 10 et 20 millisecondes. Les programmes dont le temps d'exécution est court peuvent donc causer des erreurs d'échantillonnage élevées. L'échantillonnage est une technique très efficace pour le calcul d'un profil d'exécution des procédures : elle ne construit qu'une *approximation du profil exact* calculé par le chronométrage mais a le grand avantage de manipuler un volume modeste d'informations et d'être peu intrusive (cf. aussi la figure 2.1).

Le traçage d'événements

Le traçage d'événements consiste à générer une séquence d'événements. Chaque *événement* correspond à une action significative, physique ou logique, qui modifie l'état du système, comme par exemple les appels et les retours de procédures ou le début et la fin de lecture de message. La connaissance de cette séquence d'événements, encore appelée *trace*, qu'elle soit enregistrée ou exploitée à la volée, permet de reconstruire les états du système, de les représenter graphiquement, ou de calculer une métrique de performance. Les métriques présentées dans [38], dont celle du chemin critique, peuvent être évaluées à partir d'une trace d'événements[6]. Puisqu'il permet d'observer toute l'histoire d'une exécution, le traçage apparaît donc comme bien plus *général* et *flexible* que l'échantillonnage. Le traçage consiste en général en une observation du niveau applicatif ou du niveau de l'API. Chaque enregistrement d'événement contient les attributs suivants [68, 81] :

- *quelle* action a eu lieu (i.e. un identificateur d'événement), dans *quel* processus et, le cas échéant, dans quel processus léger (*thread*, en anglais),
- la *date* d'occurrence de l'événement,
- la référence vers le code applicatif qui a donné lieu à l'occurrence de l'événement,
- des informations supplémentaires caractéristiques de l'événement (par exemple, l'identificateur du ou des processus destinataires dans le cas d'une action d'envoi de message).

5. cf. l'appel système UNIX *getrusage*

6. Notons toutefois que la disponibilité d'une trace dans son ensemble (enregistrée sur disque) n'est pas une condition nécessaire pour calculer le profil du chemin critique. Le lecteur intéressé peut se référer à [36] où un algorithme de calcul du profil *à la volée* est présenté.

La trace permet donc de savoir *quelle* action a lieu, *où* et *quand*. Non seulement elle permet de dériver le graphe dynamique d'appels de procédures (plutôt que le graphe d'appel fourni par *gprof* [31]) mais, sur un système parallèle, elle fournit également la suite des interactions entre processeurs (échanges de messages sur une architecture à mémoire distribuée ou utilisation de verrous pour accéder à la mémoire partagée). La connaissance de ces interactions, permet de savoir quand et où les processus sont bloqués (attente) et fournit donc au programmeur les informations nécessaires pour remonter à la source d'un déséquilibre de charge.

Grâce à son caractère général, le traçage événementiel est aujourd'hui à la base de la majorité des outils d'évaluation de performance de programmes parallèles [80, 34, 94, 96, 95, 52, 23]. Pourtant, cette approche a aussi ses désavantages. Nous pensons qu'ils sont principalement au nombre de trois :

1. la génération d'événements peut atteindre une fréquence très élevée ce qui pose le problème du *volume des données* à enregistrer et à analyser [77] ;

2. l'exécution du code d'instrumentation nécessaire à l'enregistrement peut consommer un temps CPU important ce qui conduit à la dégradation des performances (*perturbation*) du système observé ;

3. la *datation physique* des événements, causalement cohérente[7], pose des problèmes sur un système parallèle ou distribué dont les horloges physiques ne sont pas parfaitement accordées.

Le volume des données générées et la perturbation du programme observé sont des problèmes étroitement corrélés. Nombre de chercheurs proposent des extensions matérielles au système informatique offrant une horloge physique globale ainsi que des lignes de sortie dédiées à haut débit, permettant une évacuation des informations de trace sans perturber le système de façon significative [11, 16, 78, 90, 51]. Toutefois, en raison du coût généralement élevé de ces extensions, nombre d'environnements sont basées sur des solutions logicielles qui sont, de surcroît, plus portables et s'adaptent à un nombre variable de processeurs («scalabilité»). L'étude de solutions logicielles aux problèmes présentés ci-dessus est donc d'une grande importance.

2.3.3 Comparatif

Le chronométrage et le traçage sont des techniques assez proches dans la mesure où elles utilisent des points d'instrumentation semblables : alors que le

7. L'ordre total sur l'ensemble des événements induit par la datation physique doit être une extension linéaire de l'ordre partiel causal.

chronométrage se contente de mesurer la durée d'une action, le traçage stocke effectivement l'événement correspondant dans un historique (graphe d'activité). Les deux approches peuvent fortement perturber l'exécution d'une application. Le traçage pose en plus les problèmes du volume des données à manipuler et de la nécessité d'une référence globale de temps. L'environnement ANNAI [93, 23] permet de configurer les points de trace soit en mode chronométrage, pour l'évaluation d'une métrique, soit en mode traçage, pour l'enregistrement d'un historique complet (cf. sous-section 2.4.3).

Le chronométrage et l'échantillonnage permettent de calculer des profils d'exécution. L'échantillonnage calcule un profil approché alors que le chronométrage effectue un calcul exact au prix de perturbations supplémentaires. Les deux techniques sont basées sur les horloges physiques locales et manipulent un volume de données très faible par rapport au traçage.

Alors que le chronométrage et le traçage fournissent en général une observation du niveau applicatif ou du niveau de l'API, l'échantillonnage permet de remonter également des informations du niveau système.

2.4 Les outils de traçage existants

Cette section commence à présenter les méthodes d'instrumentation couramment utilisées par les outils de trace – pour chaque méthode nous donnons les références bibliographiques des principaux outils basées sur cette méthode. Nous décrivons ensuite deux environnements de mesure de performance pour des systèmes faiblement couplés communiquant par échange de messages. On a porté notre choix sur AIMS [95] et ANNAI [93, 23], car ces environnements sont récents, flexibles, ergonomiques et proposent une grande richesse de fonctionnalités. Par ailleurs, les problèmes de qualité représentative traités dans cette thèse sont également abordés dans ces environnements. Finalement, nous situons le traceur Tape/PVM, développé par l'auteur de la thèse, par rapport à ces environnements.

2.4.1 Les techniques d'instrumentation des programmes

L'instrumentation, i.e. l'insertion du code qui détecte et enregistre les événements applicatifs (observation) peut se faire, de façon non exclusive, aux différents stades du processus de construction du programme. Nous distinguons (cf. la figure 2.4)

1. l'instrumentation directe du code source du programme,

2. l'instrumentation à la compilation,

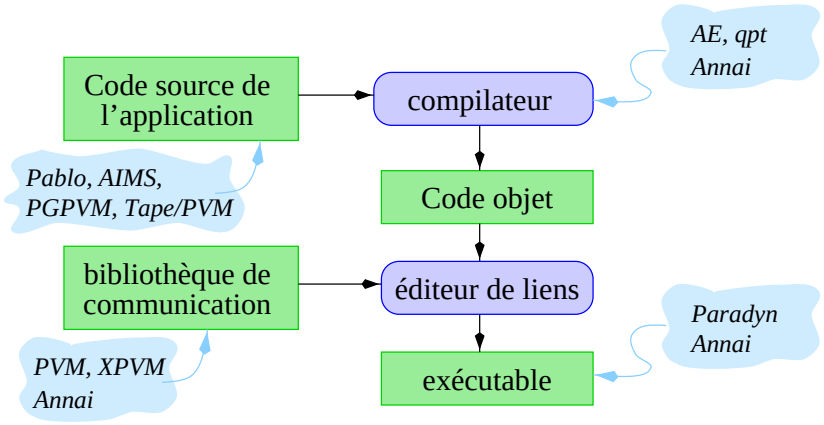


FIG. 2.4 – *Possibilités d'insertion du code d'instrumentation.*

3. l'instrumentation de la bibliothèque de communication (l'API), et
4. l'instrumentation de l'exécutable.

L'*instrumentation du code source* consiste à modifier le programme de l'utilisateur, en y insérant, avant la compilation, le code qui génère les événements. Par conséquent, cette approche requiert un préprocesseur (propre à chaque langage de programmation) qui effectue la traduction du code source vers un code équivalent instrumenté. Typiquement, un tel préprocesseur insère du code d'instrumentation à chaque appel de l'API de communication ainsi qu'aux entrées et sorties des blocs élémentaires comme les procédures et les boucles. Cette méthode à l'avantage d'être facile à implanter (elle ne nécessite pas de modification du compilateur) et de laisser à l'utilisateur le contrôle complet sur ce qui est instrumenté. Elle a le grand désavantage de nécessiter une recompilation complète du programme. Pablo [80], AIMS [94, 96, 95], PGPVM [88], Tape/PVM [63, 66] et POM [32] utilisent cette technique.

L'*instrumentation pendant la compilation* a l'avantage de donner accès aux informations construites par le compilateur, comme par exemple les dépendances au niveau des données et des boucles [37]. Par ailleurs, cette approche facilite l'intégration d'informations sémantiques dans les traces [56]. Elle a le désavantage de nécessiter l'accès au code source du compilateur. AE, qpt [56] et ANNAI [93, 23] utilisent cette technique.

L'*instrumentation de la bibliothèque de communication*, ou du «*runtime*», ne requiert aucune modification, ni même de recompilation du programme – ce que l'utilisateur appréciera d'autant plus que son programme est constitué de nombreux modules de code source dont la compilation peut durer longtemps. Il suffit en général de le relancer en activant le code de traçage compilé

en dur dans la bibliothèque et qui, par défaut, est dans un état latent. Cette approche est utilisée par PVM et XPVM [52], mais est également offerte par ANNAI. Toutefois, les événements spécifiques à l'application ne sont pas visibles au niveau de la bibliothèque. Ainsi, il est impossible de détecter un appel de procédure, et, *a fortiori*, d'évaluer une métrique de profil à partir de la trace.

L'*instrumentation directe de l'image exécutable a également de nombreux avantages*. Tout comme l'instrumentation de la bibliothèque de communication, elle est indépendante du langage de programmation et ne requiert pas de recompiler le programme. Par ailleurs, elle permet d'instrumenter à la fois le code applicatif et le code de la bibliothèque[8] tout en se passant du code source de l'application. Cette technique permet également d'effectuer l'instrumentation à la volée, i.e. pendant l'exécution du programme. L'utilisateur peut donc intervenir «dynamiquement» sur le grain et la localisation de l'instrumentation. L'instrumentation de l'image exécutable a le désavantage de ne pas être facilement portable entre des systèmes d'exploitation différents. En plus, comme l'instrumentation de la bibliothèque de communication, elle n'a pas accès aux informations sémantiques du code applicatif. PARADYN [73] et ANNAI offrent ce type d'instrumentation.

Toutes ces méthodes doivent fournir un *système de tamponnage des événements* et assurer le *vidage* de ces tampons, soit vers un composant d'analyse dynamique, soit vers un composant de sauvegarde sur fichier. Si la fréquence événementielle devient trop élevée, les processus se synchronisent sur la vitesse de ce composant ce qui conduit à l'écroulement des performances du système. Nous n'allons pas décrire ces techniques ici. Le lecteur intéressé pourra se référer à la description de Tape/PVM dans le chapitre 5 de la thèse. Par ailleurs, le manuel de l'utilisateur du traceur de Pablo contient une bonne présentation des aspects techniques du traçage [75].

2.4.2 L'environnement AIMS

AIMS [94, 96, 95] a été développé au centre de recherche NASA AMES[9]. L'environnement propose une boîte à outils permettant d'optimiser et de prédire les performances de programmes parallèles communiquant par échange de messages. Il comprend un ensemble d'outils logiciels permettant la mesure et l'analyse de performance.

L'instrumentation se fait au niveau du code source (C ou Fortran utilisant différents API de communication par messages dont PVM). Toutefois,

8. Pourvu que la bibliothèque soit liée statiquement à l'application.
9. `http://www.nas.nasa.gov/NAS/Tools/Projects/AIMS/`

l'outil d'insertion du code d'instrumentation (*xinstrument*) ne se limite pas à un simple préprocesseur comme dans le cas de PGPVM et Tape/PVM. En effet, il effectue une analyse syntaxique et construit le graphe d'appel de procédures et de boucles. Ce graphe est présenté graphiquement à l'utilisateur qui, à l'aide de la souris peut spécifier exactement ce qu'il souhaite instrumenter (procédures, boucles, appels à l'API, entrées/sorties sur système de fichier). De cette façon, AIMS combine les avantages de l'instrumentation du code source avec ceux de l'instrumentation à la compilation, tout en restant indépendant du compilateur. Le code instrumenté obtenu doit être recompilé et lié à une bibliothèque contenant les routines de formattage et de stockage des événements (*monitor*), qui elle seule dépend de l'architecture cible.

Aims inclut un système de *synchronisation des horloges physiques* et un mécanisme de *correction des perturbations* induites par la prise de trace. L'outil de correction prend en compte le surcoût associé aux appels de la bibliothèque de trace ainsi que le temps de vidage des tampons de trace. Il est basé sur les travaux de Sarukkai-Malony [83] sur lesquels on reviendra dans le chapitre 4.

Aims inclut quatre noyaux de post-traitement de traces que nous décrivons brièvement ci-dessous. Le lecteur intéressé par plus de détails peut consulter les références [94, 96, 95]. VK (*View Kernel*) est l'outil de visualisation de Aims. Cet outil fournit un diagramme espace-temps et un diagramme de Gantt. Mais, à la différence de Paragraph, les éléments de cette vue peuvent être pointés à la souris pour afficher l'extrait du code source correspondant (*source-code click-back*). SK (*Statistics Kernel*) évalue tout un ensemble de métriques de performance à partir de la trace. Par exemple, il calcule les temps d'exécution, de blocage et d'activité par procédure et par processeur. IK (*Index Kernel*) évalue plusieurs indices dont la valeur indique un problème de performance provoqué par un déséquilibre de charge, par la surcharge d'un lien de communication, par le temps passé à communiquer ou par une utilisation inefficace des liens de communication. Ce type d'information peut utilement compléter une visualisation, surtout dans le cas où le nombre de processeurs est élevé, causant par là un encombrement de la plupart des vues. MK (*Modeling Kernel*) permet d'étudier la scalabilité d'une application en fonction du nombre de processeurs et de la taille du problème.

2.4.3 L'environnement ANNAI

L'environnement Annai [93, 23] a été développé au *Centro Svizzero di Calcolo Scientifico* (CSCS) qui est affilié à l'ETH de Zurich[10]. Il comprend

10. `http://www.cscs.ch/`

trois outils logiciels partageant une interface graphique commune : un outil de support à la parallélisation (*PST*), un outil de déverminage parallèle (*PDT*) et un outil de mesure et d'analyse de performance (*PMA*). Ici nous nous intéressons surtout à l'outil ANNAI/PMA.

L'environnement supporte aussi bien les programmes de haut niveau écrits en HPF (*High Performance Fortran*) que les programmes parallèles explicites basés sur la bibliothèque de communication par messages MPI. L'outil PST joue le rôle de compilateur pour les deux types de programmes.

Alors que l'instrumentation de AIMS intervient exclusivement au niveau du code source de l'application, ANNAI/PMA supporte à la fois l'instrumentation de la bibliothèque de communication, du système de compilation et l'instrumentation dynamique.

Plutôt que d'instrumenter les communications en repérant les appels correspondants à l'API dans le code source, comme le fait AIMS, ANNAI/PMA utilise une bibliothèque de communication instrumentée. Non seulement, cette approche ne nécessite pas de recompiler le programme, mais elle permet aussi d'observer les communications à un niveau plus fin ; ainsi, ANNAI permet de distinguer l'état de blocage de l'état de lecture des données lors de la réception d'un message (cf. section 2.3.1 et le chapitre 4).

L'instrumentation du système de compilation permet d'observer le programme en termes des composants fonctionnels du code source : routines, boucles encapsulantes, blocs conditionnels et blocs d'instructions. La structure hiérarchique de ces composants est familière à l'utilisateur et peut être exploitée pour lui présenter les métriques de performance de façon structurée et facilement compréhensible (cf. la figure 2.5).

Le code d'instrumentation de la bibliothèque de communication et du système de compilation est, par défaut, dans un état latent. Au lancement du programme, l'utilisateur peut préconfigurer le code d'instrumentation pour qu'il génère, par exemple, un profil d'exécution des procédures (l'outil est alors en mode *chronométrage*). L'originalité d'ANNAI/PMA est qu'il offre la possibilité de changer cette configuration de façon dynamique dans le contexte de l'exécution du programme. Au cours de l'exécution, le profil peut attirer l'attention sur une procédure particulière[11] et l'utilisateur a la possibilité d'activer le code de traçage latent dans cette procédure. Comme pour les points d'arrêt (*breakpoints*) pour le déverminage, l'utilisateur a la possibilité de spécifier des *points d'action d'instrumentation* (*intrumentation*

11. L'accessibilité du profil pendant l'exécution du programme nécessite le vidage des tampons de trace et l'envoi des informations correspondantes au programme d'analyse, ce qui peut considérablement perturber le programme. C'est pourquoi ANNAI offre la possibilité d'expliciter des points de pause auxquels les informations sont communiquées à l'outil d'analyse.

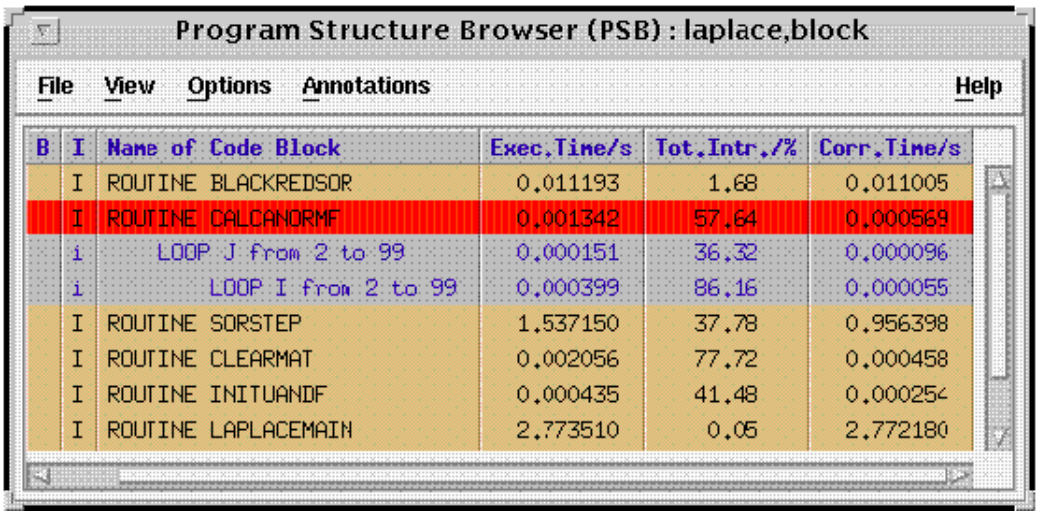


FIG. 2.5 – *Affichage de la structure fonctionnelle d'un programme avec* ANNAI*: temps d'exécution mesuré et estimé (extrait de [23]).*

action points) qui seront traités dès que le flot d'exécution les rencontre [23]. Dans une telle action d'instrumentation, l'utilisateur peut reconfigurer les paramètres d'instrumentation.

La traceur de ANNAI peut fonctionner en mode profil (chronométrage) ou en mode traçage. En mode profil, le temps séparant les événements d'entrée et de sortie des routines et des blocs est simplement accumulé. En mode traçage, les événements sont effectivement sauvegardés dans des tampons de trace, ce qui pose le problème du vidage de ces derniers. Pour l'évaluation des traces, ANNAI propose un outil de visualisation comportant les vues les plus utiles comme le diagramme espace-temps et le diagramme de Gantt.

Les concepteurs d'ANNAI/PMA ont porté un soin particulier à l'évaluation de la perturbation induite par l'enregistrement des informations de performance. Dans [23] une caractérisation des perturbations induites par les différents points d'instrumentation est présentée. Connaissant le nombre de fois qu'une procédure (ou bloc) a été appelée, on peut estimer le temps d'instrumentation correspondant en utilisant la caractérisation du point d'instrumentation associé. Ainsi, la figure 2.5 montre le temps mesuré, le temps estimé et le pourcentage de perturbation estimé pour différents composants fonctionnels d'un programme. Sur les vues graphiques, les perturbations sont représentées par un état spécial. Cette approche permet de localiser les endroits du programme qui sont les plus perturbés. Toutefois, elle ne calcule par d'approximation de la dynamique non-instrumentée du programme, comme le correcteur de perturbations de AIMS.

2.4.4 Le traceur Tape/PVM

Tape/PVM est un outil de traçage événementiel du niveau applicatif adapté aux applications parallèles basées sur la bibliothèque de communication par messages PVM [29]. Il comprend aussi une série d'outils de post-traitement de traces. Tape/PVM a été développé par l'auteur dans le cadre de ce travail de thèse. La structure de Tape/PVM est dérivée du traceur Tape [2] qui a été conçu pour une machine parallèle à base de Transputers, le Meganode [4]. Par ailleurs, l'organisation de Tape/PVM a été fortement inspirée de AIMS ; leur approche nous a semblé en effet la plus ouverte, la plus portable et la plus facile à mettre en oeuvre.

Bien que Tape/PVM soit publiquement disponible, cet outil n'est pas destiné comme alternative à AIMS, ANNAI ou tout autre environnement de traçage existant. Notre but était de construire un outil de trace pour les *besoins internes* de l'environnement ALPES [48] (basé alors sur PVM), outil sur lequel on puisse avoir le contrôle complet. Nous avons diffusé le code source de Tape/PVM dans l'espoir qu'il soit utile et puisse servir dans d'autres organismes de recherche ; actuellement l'outil est effectivement utilisé dans plusieurs laboratoires de recherche en Europe et au Brésil. Par ailleurs, Tape/PVM a servi de *cadre expérimental* pour la validation de la méthodologie d'ajustement de la qualité des traces proposée dans cette thèse. Ainsi, Tape/PVM fournit un algorithme original d'échantillonnage des horloges physiques permettant de construire une référence globale de temps physique sur des architectures multi-processeur dépourvues d'horloge physique globale (cf. le chapitre 3). Tape/PVM mesure automatiquement les perturbations qu'il inflige à l'application instrumentée et fournit un outil de correction post-mortem qui exploite ces mesures pour enlever les perturbations des traces. Comme celui de AIMS, notre algorithme de correction des perturbations est basée sur les travaux de Malony et Sarukkai [83] ; il est original dans la mesure où il permet de minimiser le nombre de recours à un modèle pour évaluer les temps de communication (cf. le chapitre 4).

Le contexte historique et l'architecture logicielle de Tape/PVM sont présentés plus en détail dans le chapitre 5 qui situe également Tape/PVM par rapport aux autres outils de trace destinés spécifiquement à PVM, comme XPVM [52] et PGPVM [88].

2.5 Conclusion

Dans ce chapitre nous avons montré comment les performances d'un système parallèle peuvent être quantifiées (métriques) et représentées (visuali-

sation) dans le but de remonter à l'origine d'un problème de performance. Nous avons également présenté les différents niveaux d'abstraction auxquels l'observation du système peut être effectué : on s'intéresse principalement au niveau applicatif et au niveau de l'API, des informations de plus bas niveau n'étant pas faciles à mettre en relation avec les événements du niveau applicatif. Actuellement, de nombreux environnements utilisent une instrumentation basée sur le traçage événementiel. Cette technique fournit toute l'histoire d'exécution d'un programme sous forme d'une trace à partir de laquelle toutes les métriques et visualisations de performance imaginables peuvent être dérivées.

Le traçage logiciel, plus portable et économique que les solutions matérielles, est confronté aux problèmes d'absence de temps physique global et de perturbation du programme observé. Ces problèmes, s'ils ne sont pas abordés avec soin peuvent gravement détériorer la qualité de la trace produite jusqu'au point où les informations obtenues ne sont plus représentatives d'une exécution non-instrumentée. Dans cette thèse, nous nous proposons de développer une méthodologie d'ajustement de la qualité des traces obtenues par voie logicielle pour approcher la qualité des mesures qu'on pourrait obtenir avec des extensions matérielles.

Deuxième partie

La qualité représentative des traces d'exécutions parallèles

Chapitre 3

La datation globale des événements

«Non seulement nous n'avons pas l'intuition directe de l'égalité de deux durées, mais nous n'avons même pas celle de la simultanéité de deux événements qui se produisent sur des théâtres différents.»
– Henri Poincaré, *La science et l'hypothèse.*

3.1 Introduction

Dans un système parallèle multi-processeur, chaque processeur, ou carte de processeurs est équipé d'un oscillateur local qui, périodiquement, interrompt le ou les processeurs pour générer ce que l'on appelle un tic d'horloge. La *référence de temps* d'un processeur, la valeur de son horloge physique, est le nombre de tics d'horloge qui ont eu lieu depuis l'initialisation de ce processeur. En général, les noeuds d'un système multi-processeur ne sont pas démarrés au même instant, ce qui cause un *décalage* initial entre les horloges physiques du système. Par ailleurs, les fréquences des différents oscillateurs ne sont pas exactement les mêmes. Ainsi, les horloges avancent avec des vitesses différentes les unes par rapport aux autres. Le décalage entre horloges physiques est donc variable dans le temps : on dit que les horloges *dérivent* les unes par rapport aux autres. Les fabricants d'oscillateurs évoquent des erreurs de fréquence typiquement de l'ordre d'une part par million (1μs/s)[1]. Suite à ces remarques d'ordre physique, il parait clair que dans ce type de système il n'existe pas de référentiel global pour le temps physique.

Nous appelons *événement* le début ou la fin d'une action, exécutée par un processeur, qui change l'état du noeud correspondant, comme par exemple les

registres du processeur, les canaux d'entrée/sortie ou encore la mémoire [53]. En l'absence d'une référence globale de temps, nous ne pouvons ni décider si un événement e a eu lieu avant ou après un événement e' d'un autre processeur, ni mesurer le temps physique qui sépare l'occurrence de ces deux événements. Si l'on s'intéresse à définir un ordre sur l'ensemble ou un sous-ensemble des événements d'une exécution d'un programme sur une machine parallèle, on peut utiliser le *temps logique* [54, 25, 71]. Par exemple, les horloges logiques vectorielles peuvent être utilisées pour capter l'ordre partiel causal entre les événements induit par l'échange de messages. Dans [41, 42], Jard *et al.* présentent une méthode pour construire l'ensemble des configurations possibles d'un système distribué par rapport à une exécution dont l'histoire (ordre partiel causal entre événements) est accessible sous forme de trace d'exécution basée sur le temps logique vectoriel. L'ensemble de ces configurations est alors étudié afin de vérifier certaines propriétés. Cependant, *l'évaluation des performances* d'une implantation d'un algorithme parallèle sur un système multi-processeur requiert la mesure du temps physique entre événements, ce à quoi les horloges logiques ne sont pas adaptées.

Afin de mesurer le temps écoulé entre deux événements de processeurs différents, nous devons dater ces événements en utilisant une référence globale de temps physique. Les méthodes de construction de temps global définissent pour chaque processeur P_i $(i = 1..p)$ une fonction $LC_i(t)$ qui représente la vision du temps global du processeur P_i à l'instant t. t est une référence de temps absolue qui est indépendante du système multi-processeur utilisé. Intuitivement, t peut-être considéré comme le temps affiché par notre montre ou encore comme le temps mondial (*wall-clock time* ou *world time*, en anglais). En pratique, les fonctions $LC_i(t)$ sont construites à partir des horloges physiques $C_j(t)$ $(j = 1..p)$ du système. Le temps global LC est défini comme étant le vecteur de $\mathbb{R}^p$ dont la i$^{\text{e}}$composante est $LC_i(t)$. Un événement e du processeur P_i peut ainsi être daté par $LC_i(t(e))$, où $t(e)$ représente la date en temps absolu t à laquelle e a eu lieu. Dans le cas idéal, toutes les composantes de LC sont identiques et avancent à la même vitesse que notre référence absolue t. Cependant, suite aux limites physiques des horloges système C_i, un tel cas idéal n'est pas accessible en pratique. Les propriétés de temps global suivantes sont alors introduites, afin de définir une approximation du temps global idéal [43, 79, 46]:

- $P1$ (croissance): $\forall t \in \mathbb{R}, \forall d > 0, \forall i \in \{1..p\}$, $LC_i(t+d) > LC_i(t)$;
- $P2$ (accord): $\exists \epsilon > 0, \forall i, j \in \{1..p\}, \forall t \in \mathbb{R}$, $|LC_i(t) - LC_j(t)| < \epsilon$, ce qui définit la *granularité* ϵ du temps physique global ; deux événements ne peuvent être ordonnancés avec certitude que quand leurs dates sont distantes d'au moins ϵ [79] ;

- $P3$ (justesse): $\forall i \in \{1..p\}$, $|\frac{dLC_i(t)}{dt} - 1| < \rho$, ce qui veut dire que chacune des fonctions LC_i appartient à une enveloppe linéaire du temps absolu ;

- $P4$ (respect de la causalité): pour tout événement e_i (sur le processeur P_i) et e_j (sur le processeur P_j) tels que e_i est une cause de e_j, on a $LC_i(t(e_i)) < LC_j(t(e_j))$. Dans des systèmes parallèles communiquant par échanges de messages, e_i peut correspondre à l'envoi d'un message de P_i à P_j [54] et e_j à la réception de ce message sur P_j. De façon similaire, sur des systèmes à mémoire partagée, on a la même relation pour e_i correspondant à l'action de libération d'un sémaphore S ($V(S)$) sur P_i et pour e_j correspondant à l'action de déblocage de P_j sur ce sémaphore (fin du $P(S)$).

Les méthodes présentées dans ce chapitre sont destinées à des systèmes à mémoire distribuée communiquant par l'intermédiaire de messages (comme les réseaux de Transputers par exemple). Pour de tels systèmes, la condition suffisante suivante permet de s'assurer du respect de la causalité (la preuve peut être trouvée dans [43], par exemple) :

THÉORÈME *Si le temps global LC vérifie les propriétés $P1$, $P2$ et $P3$, et si $\epsilon < \mu(1-\rho)$, alors LC vérifie aussi $P4$ (μ étant le temps minimal de transmission d'un message).*

La propriété $P4$ est très importante dans le cas où le temps global LC est utilisé pour générer des traces d'exécutions parallèles pour l'évaluation des performances. La plupart des outils d'analyse de trace supposent en effet que les traces soient causalement cohérentes. Tel est le cas, par exemple, de l'outil de visualisation PARAGRAPH dont le simulateur est interrompu en cas d'incohérence causale (les événements en cause sont appelés «tachyons») [34]. A part l'accord, la justesse et la cohérence causale de la datation globale, un outil logiciel de prise de traces est confronté au problème de l'intrusion induite par l'instrumentation. En effet, la lecture d'horloge, le stockage d'un événement dans des tampons et le vidage de ces tampons prennent un temps non négligeable, ce qui peut avoir un effet sensible sur le comportement de l'application observée. Ce comportement peut, le cas échéant, être différent de celui d'une exécution non-instrumentée. Malheureusement, un grand nombre des algorithmes classiques de construction de temps global sont intrusifs à leur tour et sont, par conséquent, mal adaptés pour la trace précise des événements.

Dans [54], L. Lamport propose un algorithme pour construire une référence globale de temps physique pour des systèmes communiquant par échanges de messages. Par construction, l'algorithme respecte la causalité.

Cependant, le temps global obtenu ne reste pas dans une enveloppe linéaire du temps absolu t (i.e. il n'est pas juste au sens de $P3$), et sa granularité ϵ dépend de la variabilité des temps de transmission de messages. En plus, à cause des vitesses différentes des horloges, un nombre minimal de messages doit être transmis entre processeurs. Ainsi, si l'application n'envoie pas assez de messages, il convient d'augmenter le nombre de messages de façon artificielle, perturbant ainsi cette application.

Une multitude d'autres algorithmes pour construire un temps global ont été proposés dans la littérature. Dans [18], une méthode est présentée où les différents sites d'un réseau interrogent périodiquement un serveur de temps centralisé et ajustent leurs «horloges» LC_i en fonction de la réponse de ce serveur. La méthode estime le temps qu'il faut pour obtenir la réponse du serveur afin d'accroître la précision de l'ajustement. Dans un autre type d'algorithme, les différents sites diffusent périodiquement la valeur de leur horloge à tous les autres sites du réseau. Le temps entre deux diffusions successives est appelé *intervalle de resynchronisation*. L'horloge LC_i d'un site donné est alors ajustée en prenant la moyenne des valeurs d'horloges LC_j reçues des autres sites du système. Dans [55], un tel algorithme de moyenne est présenté pour des topologies complètement connectées et qui gère d'éventuelles pannes (algorithme de convergence interactive, CNV). Ce type d'algorithme s'oriente vers les systèmes d'exploitation distribués et requiert des processus clients et serveurs sur les différents sites, augmentant la charge des processeurs et le trafic sur le réseau, ce qui perturbe l'exécution d'autres applications. Les implantations standard sur stations de travail UNIX, comme par exemple le *Network Time Protocol* (NTP) utilisent des démons de synchronisation de temps garantissant une granularité de 1-3ms [1]. Afin de minimiser les perturbations induites par l'algorithme CNV et d'affiner sa granularité, une extension matérielle de cet algorithme est présentée dans [78]. Cette extension permet d'atteindre une granularité de 100-200μs, granularité qui peut être rendue indépendante du temps de diffusion maximum d'un message sur le réseau.

Une toute autre classe d'algorithmes utilise des *méthodes statistiques* pour estimer la relation, supposée linéaire, entre les valeurs des différentes horloges (modèle linéaire) [21, 33, 43]. Ainsi, un événement peut être daté en utilisant l'horloge physique du processeur sur lequel il a lieu, puis, une transformation linéaire peut être appliquée sur cette date afin de l'exprimer dans un référentiel de temps commun – le temps local d'un processeur particulier du système choisi comme référence. La granularité du temps global obtenu est indépendante de la variabilité des temps de communication et l'estimation ne requiert pas d'échanges de messages supplémentaires pendant l'exécution de l'application qui utilise l'estimateur du temps global. Ainsi, les méthodes

statistiques ne perturbent pas les applications et sont, par conséquent, bien adaptées à la datation des événements lors de la prise de traces pour l'évaluation des performances. Le désavantage majeur des méthodes statistiques réside dans le fait qu'elles nécessitent un échantillonnage des horloges physiques du système : plus la taille des échantillons est grande et plus la phase d'échantillonnage (i.e. l'intervalle de temps pendant lequel les horloges sont échantillonnées) est longue, le mieux sera la qualité de l'estimation du temps global. Cependant, si la phase d'échantillonnage devient trop longue, le lancement d'une application peut être considérablement retardé, jusqu'au moment où l'estimateur peut être évalué et le temps global devient accessible. Dans ce chapitre, nous allons rappeler les concepts des méthodes statistiques et nous allons montrer comment ces méthodes peuvent être mises en oeuvre sous la contrainte de phases d'échantillonnage aussi courtes que possible. Cette contrainte est cruciale pour une *implantation efficace* du temps global statistique [67].

En section 3.2, nous allons présenter deux méthodes statistiques pour estimer un temps global. Ces méthodes ont été proposées dans [33] et [22] et sont présentées ici dans un cadre formel commun. Les relations entre ces méthodes sont mises en évidence. Les résultats de quelques expériences préliminaires sont présentées : la difficulté pour déterminer un bon équilibre entre longueur de la phase d'échantillonnage et la qualité de l'estimation du temps global est relevée. Ceci nous amène à une analyse expérimentale plus détaillée d'un échantillon, décrite dans la section 3.3. Le modèle linéaire, malgré le fait qu'il permet un bon ajustement sur les données de l'échantillon, cache en réalité un bruit, dont la forme se révélera être périodique, mais dont l'amplitude reste suffisamment faible pour permettre une estimation d'un temps global causalement cohérent (sous réserve que la phase d'échantillonnage soit suffisamment longue). En section 3.4, nous allons montrer qu'une implantation d'une méthode statistique, effectuant un échantillonnage de quelques minutes avant de lancer l'application cliente du temps global, peut provoquer des anomalies de datation suite à la nature du bruit observé en section 3.3 – l'erreur sur le temps global est non bornée, et le temps global ne vérifie pas la propriété d'accord. Nous proposons alors une méthodologie d'échantillonnage, dite par interpolation, que l'on croit la mieux adaptée : malgré des phases d'échantillonnage courtes, elle permet la datation cohérente sur des périodes arbitrairement longues, l'erreur d'estimation étant bornée. Nous avons eu l'occasion de tester cette méthodologie sur différentes configurations de systèmes. Nous faisons le bilan sur nos expériences à la fin de la section 3.4. Avant de conclure le chapitre en section 3.6, nous allons montrer, en présentant l'approche choisie par IBM pour ses systèmes SP, comment tirer profit des spécificités d'un système parallèle donné, afin d'améliorer les qualités du

temps global.

3.2 Les méthodes statistiques d'estimation de temps global

Cette section introduit le lecteur aux méthodes statistiques en présentant deux algorithmes d'estimation de temps global qui ont été introduits dans [33] et [22]. Les estimateurs sont présentés sous forme matricielle comme il est d'usage en théorie d'estimation des moindres carrés ordinaires[1] (*OLS* [45]).

3.2.1 Le modèle d'horloge physique et le temps global

Chaque processeur P_i $(i = 1..p)$ dispose de sa propre horloge physique C_i. Nous notons $lt_i(t)$ le temps lu sur C_i. Par exemple, la résolution de l'horloge des Transputers $T800$ est de $64\mu s$[2] ce qui implique que $lt_i(t)$ est un multiple de $64\mu s$. Les systèmes modernes, comme les RS/6000 d'IBM, sont équipés d'horloges dont la résolution est de l'ordre de la nanoseconde. Une analyse des propriétés physiques des oscillateurs quartz couramment utilisés pour les horloges des ordinateurs permet de modéliser $lt_i(t)$ comme suit :

$$lt_i(t) = \alpha_i + \beta_i t + \delta_i,\ i \in \{1..p\}, \tag{3.1}$$

où t est le temps absolu, la constante α_i le décalage au temps $t = 0$, la constante β_i (proche de 1) la dérive de l'horloge et la variable aléatoire δ_i modélise la résolution r de l'horloge (discrétisation) et autres perturbations aléatoires [43] ; sa valeur absolue maximale est ainsi proche de $r/2$. Nous supposons que δ_i est de moyenne nulle. Ce modèle de $lt_i(t)$ n'est correct que si l'on suppose que les paramètres physiques de l'environnement (salle machine), comme la température, la pression et le voltage restent stables, et que t est suffisamment petit pour négliger les effets de vieillissement du cristal. En effet, le désavantage majeur des oscillateurs à cristal est leur sensibilité à la température ($\simeq 10^{-6}/°C$) et le taux élevé de vieillissement ($\simeq 10^{-7}/jour$) [60]. Si les conditions précédentes ne sont pas vérifiées, les paramètres α_i et β_i pourraient ne plus être constants.

Éliminant t de l'équation 3.1 écrite pour i et j, nous trouvons la dépendance linéaire entre lt_i et lt_j :

1. Les résidus sont supposés non-corrélés et de même variance σ^2 (homoscédasticité).

2. Une horloge d'une résolution $1\mu s$ est accessible aux tâches s'exécutant en mode haute priorité.

$$lt_j(t) = \alpha_{i,j} + \beta_{i,j} lt_i(t) + \delta_{i,j},\ i,j \in \{1..p\}, \tag{3.2}$$

où $\beta_{i,j} = \beta_j/\beta_i,\ \alpha_{i,j} = \alpha_j - \beta_{i,j}\alpha_i$ et $\delta_{i,j} = \delta_j - \beta_{i,j}\delta_i$. $\delta_{i,j}$ est une variable aléatoire de moyenne nulle : convolution des deux variables aléatoires δ_i et δ_j, sa valeur absolue maximale est inférieure à r. Le but d'un algorithme d'estimation de temps global est l'évaluation, aussi précise que possible et pour tout $j \in \{1..p\}$, des paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$, où *ref* est l'indice d'un processeur de référence dont l'horloge physique est choisie comme base commune de temps. Toute date lue sur une horloge locale lt_j $(j \neq ref)$ peut alors être exprimée sur lt_{ref} par l'équation suivante :

$$LC_j(t) = \frac{lt_j(t) - \alpha_{ref,j}}{\beta_{ref,j}} \simeq lt_{ref}(t),\ j \in \{1..p\}. \tag{3.3}$$

$LC_j(t)$ est le temps global vu par le processeur P_j à l'instant t. Plus les différents $LC_j(t)$ sont proches de $lt_{ref}(t)$, plus le temps global est précis.

Finalement, pour garantir la propriété $P3$ (justesse), on admettra que $lt_{ref}(t)$ est synchronisé avec le temps absolu t. En pratique, on peut imaginer que le site de référence P_{ref} est équipé d'un récepteur de temps universel UTC (*Universal Coordinated Time* [86]).

3.2.2 Les échantillons

Pour estimer la valeur des paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$ $(j \in \{1..p\} \setminus \{ref\})$ avec une bonne précision, un certain nombre de lectures d'horloges doit être effectué sur un intervalle de temps suffisamment long que l'on appelle *phase d'échantillonnage*. Nous supposons que le processeur de référence P_{ref} peut communiquer avec tous les autres processeurs P_j – soit directement sur un réseau ou sur un canal physique, soit indirectement en utilisant un routeur de messages. A l'étape k $(k \in \{1..n\})$ de la phase d'échantillonnage, P_{ref} échange un message avec chacun des $p-1$ autres processeurs. Pour chaque échange avec P_j, les temps locaux suivants sont enregistrés (figure 3.1) : $lt_{ref}(S^k_{ref})$, $lt_j(R^k_j)$, $lt_{ref}(R^k_{ref})$ et $lt_j(S^k_j)$. Après la n$^{\text{e}}$ étape (i.e. après $(n-1)\Delta t$ unités de temps d'échantillonnage), les données relevées sont assemblées en $p-1$ échantillons de deux colonnes chacun, et de taille $2n$:

$$S_j = \{\, (lt_{ref}(S^k_{ref}), lt_j(R^k_j)),\ (lt_{ref}(R^k_{ref}), lt_j(S^k_j)) \,\}_{k=1}^{n},\ j \in \{1..p\} \setminus \{ref\}.$$

Les valeurs des paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$ seront estimées à partir de ces échantillons. La section suivante présente les estimateurs.

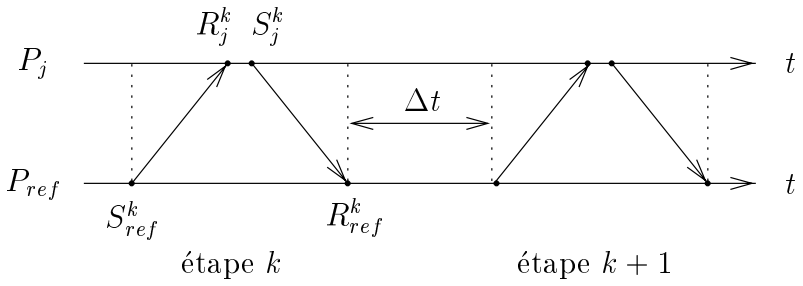


FIG. 3.1 – *La phase d'échantillonnage: à la réception d'un message, P_j le retourne instantanément à P_{ref}. P_{ref} attend alors Δt unités de temps avant de commencer l'étape suivante.*

3.2.3 L'estimation du temps global

Dans cette section nous présentons deux algorithmes simples et nous montrons comment ils sont reliés entre eux. Nous supposons que la résolution des horloges physiques est suffisamment fine pour mesurer le temps de transmission de messages vides (la granularité doit être inférieure à la latence d'envoi de messages). Dans des systèmes parallèles disposant de matériel de communication très performant, cette hypothèse peut ne pas être valide. Dans [43] et [44], le lecteur intéressé peut trouver une solution très originale à ce problème.

L'estimateur de Haddad *et al.*

Dans [21, 33], Y. Haddad *et al.* présentent deux estimateurs. Le premier utilise une analyse des moindres carrés pour estimer l'écart $\alpha_{ref,j}$ et la dérive $\beta_{ref,j}$ entre deux horloges lt_{ref} et lt_j. Le second utilise un algorithme géométrique. Dans cette section, nous ne présentons que le premier, car il est plus facile à comprendre et se prête bien à l'illustration des propos de ce chapitre.

Soit $\tau_{ref,j}$ (resp. $\tau_{j,ref}$) la variable aléatoire positive modélisant le temps de transmission d'un message de taille donnée de P_{ref} à P_j (resp. de P_j à P_{ref}). Nous supposons que $\tau_{ref,j}$ et $\tau_{j,ref}$ sont indépendantes du temps t (notons qu'en pratique, les variations de charge du médium de communication se reflètent dans la variance de ces variables aléatoires). Nous avons les relations suivantes (cf. figure 3.1 et les équations 3.1 et 3.2) :

$$lt_j(R_j^k) = lt_j(S_{ref}^k + \tau_{ref,j}) = \alpha_{ref,j} + \beta_{ref,j} lt_{ref}(S_{ref}^k) + u_{ref,j}^+, \qquad (3.4)$$

$$lt_j(S_j^k) = lt_j(R_{ref}^k - \tau_{j,ref}) = \alpha_{ref,j} + \beta_{ref,j} lt_{ref}(R_{ref}^k) + u_{ref,j}^-, \qquad (3.5)$$

où

$$u^{+}_{ref,j} = \delta_{ref,j} + \beta_{ref,j}\beta_{ref}\tau_{ref,j}, \tag{3.6}$$
$$u^{-}_{ref,j} = \delta_{ref,j} - \beta_{ref,j}\beta_{ref}\tau_{j,ref}. \tag{3.7}$$

Puisque nous avons supposé que la résolution des horloges physiques est très fine par rapport aux temps de transmission, il vient que $|\delta_{ref,j}| \ll \tau_{ref,j}$ et $|\delta_{ref,j}| \ll \tau_{j,ref}$, et par conséquent $u^{+}_{ref,j} > 0$ et $u^{-}_{ref,j} < 0$. Ce qui veut dire, en d'autres termes, que l'ensemble des points

$$U_j = \{\, (lt_{ref}(S^k_{ref}),\ lt_j(R^k_j))\,\}_{k=1}^{n}$$

est au-dessus de la droite $\alpha_{ref,j} + \beta_{ref,j} lt_{ref}$ et que

$$L_j = \{\, (lt_{ref}(R^k_{ref}),\ lt_j(S^k_j))\,\}_{k=1}^{n}$$

est en-dessous de cette droite. Le modèle linéaire suivant peut être utilisé pour l'échantillon S_j :

$$Y^{(j)} = X^{(j)}\beta^{(j)} + u^{(j)},$$

où

$$\begin{aligned}
Y^{(j)} &= \left[\ lt_j(R^1_j) \quad lt_j(S^1_j) \quad \cdots \quad lt_j(R^n_j) \quad lt_j(S^n_j)\ \right]', \\
X^{(j)} &= \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ lt_{ref}(S^1_{ref}) & lt_{ref}(R^1_{ref}) & \cdots & lt_{ref}(S^n_{ref}) & lt_{ref}(R^n_{ref}) \end{bmatrix}', \\
u^{(j)} &= \left[\ u^{+}_{ref,j} \quad u^{-}_{ref,j} \quad \cdots \quad u^{+}_{ref,j} \quad u^{-}_{ref,j}\ \right]', \\
\beta^{(j)} &= \left[\ \alpha_{ref,j} \quad \beta_{ref,j}\ \right]'
\end{aligned}$$

sont des vecteurs et matrices de $\mathbb{R}^{2n}$, $\mathbb{R}^{2n} \times \mathbb{R}^2$, $\mathbb{R}^{2n}$ et $\mathbb{R}^2$ respectivement. Notons que l'opérateur prime ($'$) représente la transposition de matrices. $u^{(j)}$ est le vecteur inconnu, aléatoire des résidus (variables aléatoires) et $\beta^{(j)}$ est le vecteur inconnu avec les paramètres du modèle linéaire. Sous les hypothèses standard des moindres carrés ordinaires $E(u) = 0$, $E(uu') = \sigma^2 I$ (homoscédasticité[3] et non-corrélation des résidus) et X matrice non-aléatoire, l'estimateur

$$\widehat{\beta^{(j)}} = (X'^{(j)}X^{(j)})^{-1}X'^{(j)}Y^{(j)} \tag{3.8}$$

est le meilleur estimateur linéaire sans biais de $\beta^{(j)}$ (théorème de Gauss-Markov; cf. [45] par exemple). Cependant, une des hypothèses de base de

3. On parle d'*homoscédasticité* dans le cas où tous les résidus ont la même variance σ^2 (par opposition à *hétéroscédasticité*).

OLS n'est pas vérifiée, à savoir $E(u) = 0$. En effet, si l'on suppose que les communications sont symétriques, i.e. $E(\tau_{ref,j}) = E(\tau_{j,ref}) = \bar{\tau}_j$, alors il vient des équations 3.6 et 3.7 que

$$E(u^{(j)}) = \beta_{ref,j}\beta_{ref}\bar{\tau}_j \left[\begin{array}{cccccccc} 1 & -1 & 1 & -1 & \cdots & 1 & -1 \end{array} \right]'.$$

Cependant, l'utilisation de OLS peut être justifiée, toujours dans le cas où les communications sont symétriques, par le fait que la variable aléatoire $\sum_{k=1}^{2n} u_k^{(j)}$ est de moyenne nulle. Notons aussi que la matrice $X^{(j)}$ est aléatoire, du moins en ce qui concerne les dates de retour de messages $lt_{ref}(R_{ref}^k)$, sur lesquelles on n'a pas le contrôle. Les résultats de l'analyse sont alors à interpréter comme étant conditionnels par rapport à une réalisation de $X^{(j)}$.

L'estimateur des temps de communication

Cette section présente l'estimateur introduit par Dunigan dans [22] et montre ses rapports avec l'estimateur de Haddad *et al.*

Si l'on suppose, comme à la fin de la section précédente, que les temps de transmission de messages sont symétriques, alors on peut estimer $lt_j(R_{ref}^k)$ par

$$\widehat{lt_j}(R_{ref}^k) = \frac{lt_j(R_j^k) + lt_j(S_j^k)}{2} + \frac{lt_{ref}(R_{ref}^k) - lt_{ref}(S_{ref}^k)}{2}. \qquad (3.9)$$

Pour l'interprétation de cet estimateur, reportons-nous à la figure 3.1 : le premier terme de la somme dans l'équation 3.9 estime le temps local sur P_j entre la réception du message et son renvoi à P_{ref}, tandis que le deuxième terme estime le demi temps d'aller-retour (*semi-round-trip time*, en anglais) [22]. Notons que dans l'équation 3.9, nous ajoutons un délai mesuré sur lt_{ref} (deuxième terme) à une date exprimée sur lt_j pour obtenir une date sur lt_j – ceci n'est valide que sous l'hypothèse que ces deux horloges avancent à la même vitesse, i.e. que la dérive entre lt_{ref} et lt_j est négligeable durant le demi temps d'aller-retour.

En utilisant l'équation 3.9, nous pouvons transformer l'échantillon S_j en l'échantillon

$$R_j = \{ (lt_{ref}(R_{ref}^k), \widehat{lt_j}(R_{ref}^k)) \}_{k=1}^n$$

qui est de taille n et qui représente directement lt_j en fonction de lt_{ref}. Pour avoir une idée de la qualité de l'estimation en 3.9, nous étudions la variable aléatoire :

$$e_{ref,j}^k = lt_j(R_{ref}^k) - \widehat{lt_j}(R_{ref}^k). \qquad (3.10)$$

A partir des équations 3.2, 3.4 et 3.5, il vient

$$e_{ref,j}^k = (\beta_{ref,j} - 1)\,\frac{lt_{ref}(R_{ref}^k) - lt_{ref}(S_{ref}^k)}{2} - \beta_{ref,j}\,\beta_{ref}\,\frac{\tau_{ref,j} - \tau_{j,ref}}{2}.$$

D'où, en supposant que les $lt_{ref}(R^k_{ref})$ et $lt_{ref}(S^k_{ref})$ sont non-stochastiques et que les communications sont symétriques,

$$E(e^k_{ref,j}) = (\beta_{ref,j} - 1)\,\frac{lt_{ref}(R^k_{ref}) - lt_{ref}(S^k_{ref})}{2},$$

ce qui représente la variation de l'écart entre les horloges lt_{ref} et lt_j pendant le demi temps d'aller-retour.

Partant des équations 3.2 et 3.10, il vient

$$\widehat{lt_j}(R^k_{ref}) = \alpha_{ref,j} + \beta_{ref,j} lt_{ref}(R^k_{ref}) + r^k_{ref,j},$$

où $r^k_{ref,j} = \delta_{ref,j} - e^k_{ref,j}$ est une variable aléatoire et $E(r^k_{ref,j}) = -E(e^k_{ref,j})$.

Dans l'introduction à ce chapitre, nous avons déjà vu que l'erreur de fréquence reportée par les fabricants d'oscillateurs est de l'ordre d'un sur un million. Cette erreur[4], nous la retrouvons dans les équations précédentes sous la forme $|\beta_{ref,j} - 1|$, sa valeur étant de l'ordre de 10^{-6}. La valeur de $lt_{ref}(R^k_{ref}) - lt_{ref}(S^k_{ref})$, quant à elle, représente le temps d'aller-retour et sa valeur dépend de la bande passante du médium de communication, de la distance entre P_{ref} et P_j et de la taille du message. Par exemple, sur le Meganode, machine à base de Transputers utilisant le logiciel de routage VCR [49], nous avons observé des temps d'aller-retour inférieurs à 4ms pour des messages de 1 octet (la distance maximale entre deux processeurs est de 127 liens). Ainsi, sur ce système, $E(r^k_{ref,j})$ à une valeur de quelques nanosecondes, ce qui échappe complètement à la résolution de l'horloge physique, dont la résolution maximale est de $1\mu s$. En pratique, il est donc raisonnable de supposer que $E(r^k_{ref,j}) = 0$ et que *a fortiori* l'estimateur $\widehat{lt_j}(R^k_{ref})$ est sans biais. Ainsi, les n points de l'échantillon R_j se trouvent *sur* la droite $\alpha_{ref,j} + \beta_{ref,j} lt_{ref}$, alors que les $2n$ points de l'échantillon S_j sont positionnés au-dessus (ensemble U_j) et en-dessous (ensemble L_j) de cette droite. La transformation de S_j à R_j est illustrée graphiquement sur la figure 3.2. En pratique, la symétrie de l'échange du message à la k$^{\text{e}}$ itération d'échantillonnage, i.e. $\tau^k_{ref,j} = \tau^k_{j,ref}$, n'est vérifiée qu'en moyenne, pour k de 1 à n. Nous allons en effet observer un certain nombre de mesures aberrantes (cf. section 3.4.4). En effet, on a les deux cas suivants :

1. $\tau^k_{ref,j} > \tau^k_{j,ref} \Longrightarrow e^k_{ref,j} < 0 \Longrightarrow r^k_{ref,j} > 0$
   $\Longrightarrow$ le point r^k est au-dessus de la droite ;

2. $\tau^k_{ref,j} < \tau^k_{j,ref} \Longrightarrow e^k_{ref,j} > 0 \Longrightarrow r^k_{ref,j} < 0$
   $\Longrightarrow$ le point r^k est en-dessous de la droite.

4. d'estimation sont donnés en section 3.2.4 de ce chapitre et dans [43] (2×10^{-5} pour le iPSC/2; 5×10^{-5} pour le FPS-T40).

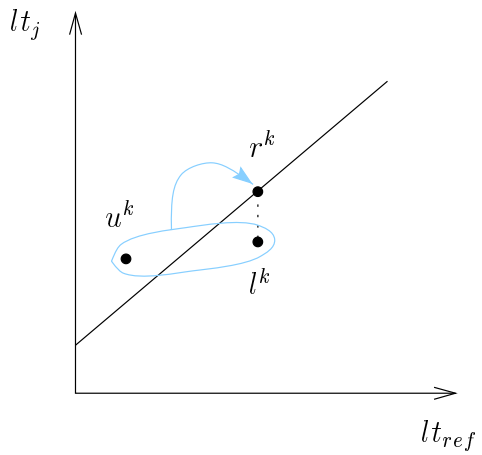


FIG. 3.2 – *Transformation de l'échantillon S_j en R_j. Chaque couple de points $(\, u^k = (lt_{ref}(S^k_{ref}),\, lt_j(R^k_j)), l^k = (lt_{ref}(R^k_{ref}),\; lt_j(S^k_j))\,)$ est transformé en un seul point $r^k = (\, lt_{ref}(R^k_{ref}),\; \widehat{lt_j}(R^k_{ref}))$ qui devrait être sur la droite.*

Les conséquences pratiques de telles mesures aberrantes sont discutées en section 3.4.4.

L'estimation de $\beta^{(j)} \in \mathbb{R}^2$ à partir de l'échantillon R_j est alors effectuée comme en section 3.2.3 en utilisant l'estimateur des moindres carrés ordinaires de l'équation 3.8, où

$$\begin{aligned} Y^{(j)} &= \left[\begin{array}{ccc} \widehat{lt_j}(R^1_{ref}) & \cdots & \widehat{lt_j}(R^n_{ref}) \end{array}\right]', \\ X^{(j)} &= \left[\begin{array}{ccc} 1 & \cdots & 1 \\ lt_{ref}(R^1_{ref}) & \cdots & lt_{ref}(R^n_{ref}) \end{array}\right]', \\ u^{(j)} &= \left[\begin{array}{ccc} r^1_{ref,j} & \cdots & r^n_{ref,j} \end{array}\right]' \end{aligned}$$

sont des vecteurs et matrices dans $\mathbb{R}^n$, $\mathbb{R}^n \times \mathbb{R}^2$ et $\mathbb{R}^n$, respectivement. Vu les caractéristiques physiques des horloges présentées plus haut, $E(u^{(j)}) = 0$. Comme en section 3.2.3 on fera l'approximation que $X^{(j)}$ est une matrice non-stochastique. Les hypothèses d'homoscédasticité et de non-corrélation du résidu $u^{(j)}$ sont discutées en section 3.3 lors de l'étude détaillée de l'erreur d'estimation.

Les intervalles de confiance

Sous les hypothèses des moindres carrés ordinaires, $E(u) = 0$, $E(uu') = \sigma^2\, I$ et X matrice non-stochastique, l'estimateur $\widehat{\beta^{(j)}}$ de l'équation 3.8 est,

parmi tous les estimateurs linéaires sans biais, celui de variance minimale (théorème de Gauss-Markov)[5]. On peut montrer que [45]

$$V(\widehat{\beta}) = \sigma^2 \, (X'X)^{-1}. \tag{3.11}$$

C'est une matrice de $\mathbb{R}^2 \times \mathbb{R}^2$ dont les termes de la diagonale représentent la variance empirique de $\alpha_{ref,j}$ et $\beta_{ref,j}$ et dont les termes hors-diagonale représentent les covariances empiriques. On peut montrer par ailleurs que

$$\widehat{\sigma}^2 = \frac{\| \ Y - Y^* \ \|^2}{N-2} \tag{3.12}$$

est un estimateur sans biais de σ^2, où $Y^* = X\widehat{\beta}$ est l'estimation de Y, et où N vaut $2n$ ou n pour les estimateurs de Haddad et de Dunigan, respectivement.

Si l'on rajoute l'hypothèse que le vecteur u des résidus est gaussien[6], i.e.

$$u \sim LG_N(0, \sigma^2 I)$$

alors $\widehat{\beta}$ devient une transformée linéaire d'un vecteur gaussien et est donc lui-même gaussien :

$$\widehat{\beta} \sim LG_2(\beta, (X'X)^{-1}\sigma^2).$$

On peut alors former les variables de Student

$$t = \frac{\widehat{\beta}_i - \beta_i}{\widehat{\sigma}\sqrt{a_{ii}}} \sim t(N-2), \ \text{pour i} = 1, 2,$$

où a_{ii} désigne le i$^{\text{e}}$élément de la diagonale de $(X'X)^{-1}$. Dans le cas de l'estimation du temps global, nous allons voir expérimentalement que u n'est pas gaussien (cf. section 3.3). Néanmoins, cette hypothèse permet de dériver des intervalles de confiance pour les composantes de β, i.e. $\alpha_{ref,j}$ et $\beta_{ref,j}$, ce qui nous donne le moyen, dans la pratique, de calculer un indicateur de la précision de l'estimation.

3.2.4 Quelques expériences préliminaires

Cette section présente les résultats obtenus par l'application des deux méthodes statistiques décrites précédemment aux échantillons relevés sur un système réel. Nous allons voir qu'une analyse plus détaillée des échantillons est nécessaire pour expliquer certaines des observations sur ces expériences préliminaires.

5. Pour alléger la notation, on omettra (j) en exposant de β, $\widehat{\beta}$, u, X et Y.
6. La loi normale est encore appelée loi de Laplace-Gauss et nous la notons LG.

Les deux méthodes ont été évaluées sur des échantillons de grande taille[7] relevés sur le réseau de Transputers de la machine Meganode. Dans tous les cas, nous avons observé que la deuxième méthode, utilisant l'échantillon R_j, estime les paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$ $(j = 1..p)$ avec une meilleure précision que la première, utilisant l'échantillon S_j. Le tableau 3.1 montre les erreurs relatives (écart-type sur valeur estimée) sur $\alpha_{ref,j}$ et $\beta_{ref,j} - 1$ pour un échantillon S_j de taille 1000 (R_j de taille 500) obtenu par des échanges de messages sur 45 minutes. Les valeurs du tableau 3.1 sont en fait les erreurs relatives moyennes sur les 16 horloges physiques du Meganode. Le coefficient de corrélation r est toujours égal à (ou très proche de) 1 (pour S_j et R_j), ce qui indique une très bonne qualité d'ajustement par le modèle linéaire.

échantillon	écart	dérive
S_j (Haddad *et al.*)	0,26 ppm	79 ppm
R_j (temps de communication)	0,15 ppm	4 ppm

TAB. 3.1 – *Erreurs relatives sur écart et dérive d'horloges obtenus par l'estimateur de Haddad et al. et par l'estimateur des temps de communication. Les erreurs sur les dérives sont relatives à* $\beta_{ref,j} - 1$.

Nous pensons que l'algorithme basé sur l'estimation des temps de communication donne une meilleure précision dû au fait que l'échantillon R_j vérifie mieux les hypothèses des moindres carrés ordinaires. On peut noter que dans le cas de l'estimateur de Haddad *et al.* les résidus sont alternativement positifs ($u^+_{ref,j}$) et négatifs ($u^-_{ref,j}$), ce qui implique une autocorrélation négative. Par ailleurs, l'estimateur de Haddad *et al.* a été conçu à l'origine pour calculer le temps global à partir des événements de communication datés, générés par l'application elle-même [21, 33], et non pas à partir d'un échantillon relevé spécifiquement pour l'estimation du temps global.

Une autre observation importante est que les valeurs des paramètres $\beta_{ref,j}$, représentant la vitesse des horloges lt_j par rapport à une horloge de référence lt_{ref}, sont instables. En effet, des échantillons de grande taille, relevés, par exemple, à plusieurs *jours* d'intervalle, donnent des différences significatives dans l'estimation de la valeur d'un même paramètre de dérive. Dans beaucoup de cas, il n'y a même pas de recouvrement des intervalles de confiance associés. L'instabilité est donc due à la nature du processus physique sous-jacent et non pas à l'erreur d'estimation. Nous pensons que c'est le taux de

7. Ici, et dans la suite du chapitre, nous entendons par *échantillon de grande taille*, un *grand* nombre de points (de l'ordre de mille) collectés uniformément sur une période suffisamment *longue* (de l'ordre d'une heure).

vieillissement du cristal (en $jours^{-1}$) qui est en cause ici. Ainsi, la valeur des paramètres de dérive ne peut malheureusement pas être précalculée en vue d'une utilisation sur plusieurs jours.

Finalement, pour pallier au problème de la longueur de la période d'échantillonnage, nous avons essayé de trouver un bon équilibre entre la durée d'échantillonnage et la précision obtenue sur l'estimation. Ceci est un aspect important pour l'implémentation *efficace* des méthodes statistiques [67]. C'est pourquoi nous avons lancé l'estimation sur les premières parties d'un échantillon de grande taille, par exemple le quart et la moitié de tous les points présents dans l'échantillon, ce qui correspond à des durées de plusieurs heures. Dans les deux cas, les méthodes statistiques donnent un très bon ajustement (coefficient de corrélation r proche de 1). Cependant, les valeurs estimées sont différentes, sans intersection des intervalles de confiance correspondants. Pour expliquer cette observation, qui lève nos doutes sur le caractère linéaire des échantillons, nous devons faire une analyse plus détaillée des échantillons.

3.3 Étude de l'erreur d'estimation

3.3.1 Analyse expérimentale détaillée d'échantillons

Afin de mieux comprendre les observations faites lors des expériences préliminaires décrites en section 3.2.4, nous allons procéder à une analyse expérimentale détaillée de la relation entre les horloges système lt_j ($j = 1..p$) et l'horloge de référence lt_{ref}. Le but est de montrer le degré d'adéquation des échantillons au modèle linéaire. Cette étude nous permettra de déterminer un équilibre optimal entre la durée de la phase d'échantillonnage (i.e. le surcoût imposé au système) et la précision sur l'estimation du temps global.

Puisque l'on s'intéresse à la relation directe entre lt_j et lt_{ref}, nous étudions un échantillon R_j : la première colonne de R_j représente le temps local lt_{ref} sur P_{ref}, tandis que la deuxième colonne représente le temps local estimé lt_j sur P_j au même instant (cf. section 3.2.3).

Notre analyse sera illustrée sur des échantillons particuliers, mais les mêmes résultats ont été obtenus sur les autres échantillons que nous avons relevés sur nos systèmes. Dans une première sous-section, nous étudions un échantillon relevé sur le réseau de Transputers de la machine Meganode. Dans une deuxième sous-section nous montrons que les mêmes observations sont faites sur des échantillons relevés sur un IBM-SP2.

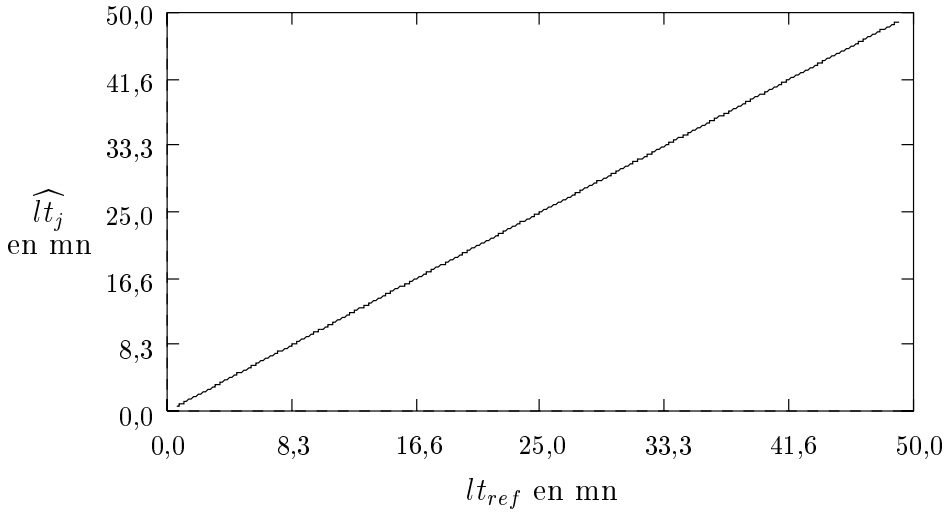


FIG. 3.3 – *Tracé de l'échantillon R_j représentant* 50 *minutes de temps de référence.*

Meganode

L'échantillon que nous étudions a été obtenu par $n = 500$ échanges de messages entre P_{ref} et P_j, attendant $\Delta t = 6$ secondes entre deux échanges successifs ($\simeq 50$ minutes pour le relevé de l'échantillon). La figure 3.3 montre le tracé des points de l'échantillon R_j. Le coefficient de corrélation r indique une très bonne adéquation du modèle linéaire ($r = 1$). L'écart est

$$\alpha_{ref,j} = 3\,795\,451 \pm 4\mu s$$

et la dérive est

$$\beta_{ref,j} = 1,000\,006\,020 \pm 0,000\,000\,002.$$

Les erreurs d'estimation montrées ici sont simplement les écarts-type empiriques, tels qu'ils sont obtenus par la formule 3.11, et sont donc, en conséquence, très pessimistes. Notons que $\beta_{ref,j}$ est bien de la forme $1 \pm \kappa$ où κ est de l'ordre de 10^{-6}. Ici, $\kappa > 0$ signifie que l'horloge lt_j est plus rapide que la référence lt_{ref}.

Pour étudier l'erreur d'approximation de $\widehat{lt_j}(R^k_{ref})$ par la droite $\alpha_{ref,j} + \beta_{ref,j} lt_{ref}(R^k_{ref})$, nous étudions l'échantillon

$$E_j = \{\ (lt_{ref}(R^k_{ref}),\ \widehat{lt_j}(R^k_{ref}) - \alpha_{ref,j} - \beta_{ref,j} lt_{ref}(R^k_{ref}))\ \}^n_{k=1},$$

qui peut être calculé aisément à partir de R_j. La deuxième colonne de E_j est en fait une réalisation du vecteur aléatoire des résidus $r^k_{ref,j}$ de l'estimateur

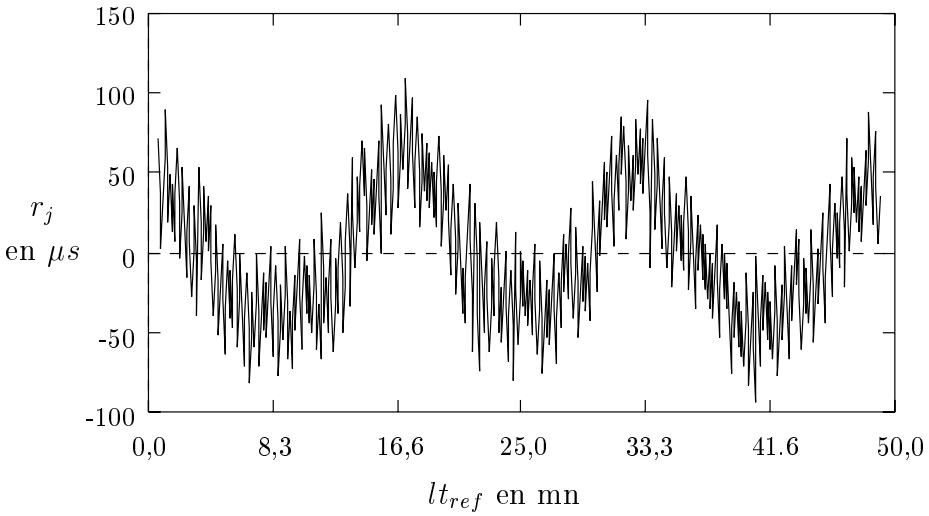


FIG. 3.4 – *Tracé de l'échantillon E_j qui montre l'erreur commise en approchant $\widehat{lt_j}$ par une fonction linéaire de lt_{ref} (droite des moindres carrés).*

---

des temps de communications (cf. section 3.2.3). Ici, nous le notons simplement r_j. La figure 3.4 montre le tracé de l'erreur dont la forme se révèle être périodique.

Comme nous l'avons déjà remarqué en section 3.2.1, la vitesse relative des horloges dépend des paramètres physiques de l'environnement. Dans tous les échantillons que nous avons analysés, la période de l'erreur, qui est d'environ 17 minutes, est la même que celle du mécanisme de régulation de température dans la salle machine[8]. Cette forme particulière, plutôt inattendue, de l'erreur d'estimation explique pourquoi nous avons obtenu des valeurs différentes pour l'estimation d'un même paramètre de dérive β_j en utilisant différentes parties contiguës d'un même échantillon en entrée des estimateurs (cf. section 3.2.4).

Sur l'échantillon étudié ici, l'erreur, ou le bruit, qui se superpose à la tendance linéaire principale est compris entre -90 et +109 μs. Notons que ce bruit est amplifié de façon artificielle à cause de la résolution limitée à $64\mu s$ de l'horloge du Transputer qui se reflète dans les «zigzag» sur le signal r_j et forme une véritable «bande» traduisant l'erreur de discrétisation ($\delta_{ref,j}$). Les valeurs réelles du signal sont centrées dans cette bande, et sa vraie amplitude avoisine les $50\mu s$. D'autres échantillons, obtenus par des expériences entre

8. Pour des raisons matérielles, nous n'avons pas cherché à corréler statistiquement le signal d'erreur avec un relevé de la température de la salle machine. Cependant, nous encourageons ce type d'expérience.

P_{ref} et d'autres processeurs ou par des expériences beaucoup plus longues[9] (jusqu'à 4 heures), révèlent un bruit dont la valeur absolue est inférieure à 192 μs (y compris l'erreur de discrétisation). Ainsi, le temps global $LC \in \mathbb{R}^p$ défini par l'équation 3.3 vérifie la propriété d'accord (cf. section 3.1) avec $\epsilon = 384\mu s$. Ceci est une borne très pessimiste car nous avons simplement pris le double de l'amplitude A maximale observée comme valeur pour ϵ. Notons que nous devons considérer le double de l'amplitude comme granularité ϵ, car $|LC_j - lt_{ref}| < A$ et, par conséquent, $|LC_j - LC_i| < 2A$ $(i \neq j)$. Par ailleurs, nos expériences montrent que $\mu = 512\mu s$ est une borne inférieure raisonnable pour le temps de communication minimal (message vide sur 1 lien). Puisque $\epsilon < \mu$, le temps global LC respecte aussi la causalité (cf. le théorème en section 3.1, où $\rho \simeq 10^{-6}$ peut être négligé). Dans des cas où l'instabilité de l'environnement est telle que ϵ est plus grand que μ, un modèle linéaire ne peut plus garantir la datation cohérente, même si la tendance linéaire principale est déterminée avec précision.

IBM-SP2

Le IBM-SP2 dispose, comme le Meganode d'un ensemble d'horloges physiques indépendantes. La machine dispose aussi d'une horloge globale physique qui est implantée au niveau de son *Switch* haute performance (*HPS*). En section 3.5 nous allons discuter une méthode de synchronisation des horloges système à partir du temps global du *Switch* qui est proposée par IBM. Notre étude s'intéresse à la validité du modèle linéaire pour un ensemble d'oscillateurs en général. Pour le moment, nous ne prenons donc pas en compte l'existence de l'horloge du *Switch*.

Sur le SP2 nous avons relevé un échantillon R_j par un programme utilisant la bibliothèque de communication MPI-F [26] (implantation IBM de MPI optimisée pour le *Switch*). Cet échantillon contient 922 points relevés sur une période d'échantillonnage de plus de deux heures ($\Delta t = 8$ secondes). La partie inférieure de la figure 3.5 montre l'évolution du décalage $lt_j - lt_{ref}$ en fonction de lt_{ref}. On trouve

$$\alpha_{ref,j} = 59\,726 \pm 1\mu s$$

et

$$\beta_{ref,j} = 1,000\,000\,2052 \pm 0,000\,000\,0002$$

où les erreurs d'estimation sont les écarts-type. Remarquons que la dérive relative $\beta_{ref,j}$ des horloges étudiées ici est très faible (de l'ordre de 10^{-7}). La

9. différents signaux r_j (qui ont été relevés en parallèle) : les amplitudes sont différentes, mais les signaux sont tous en phase.

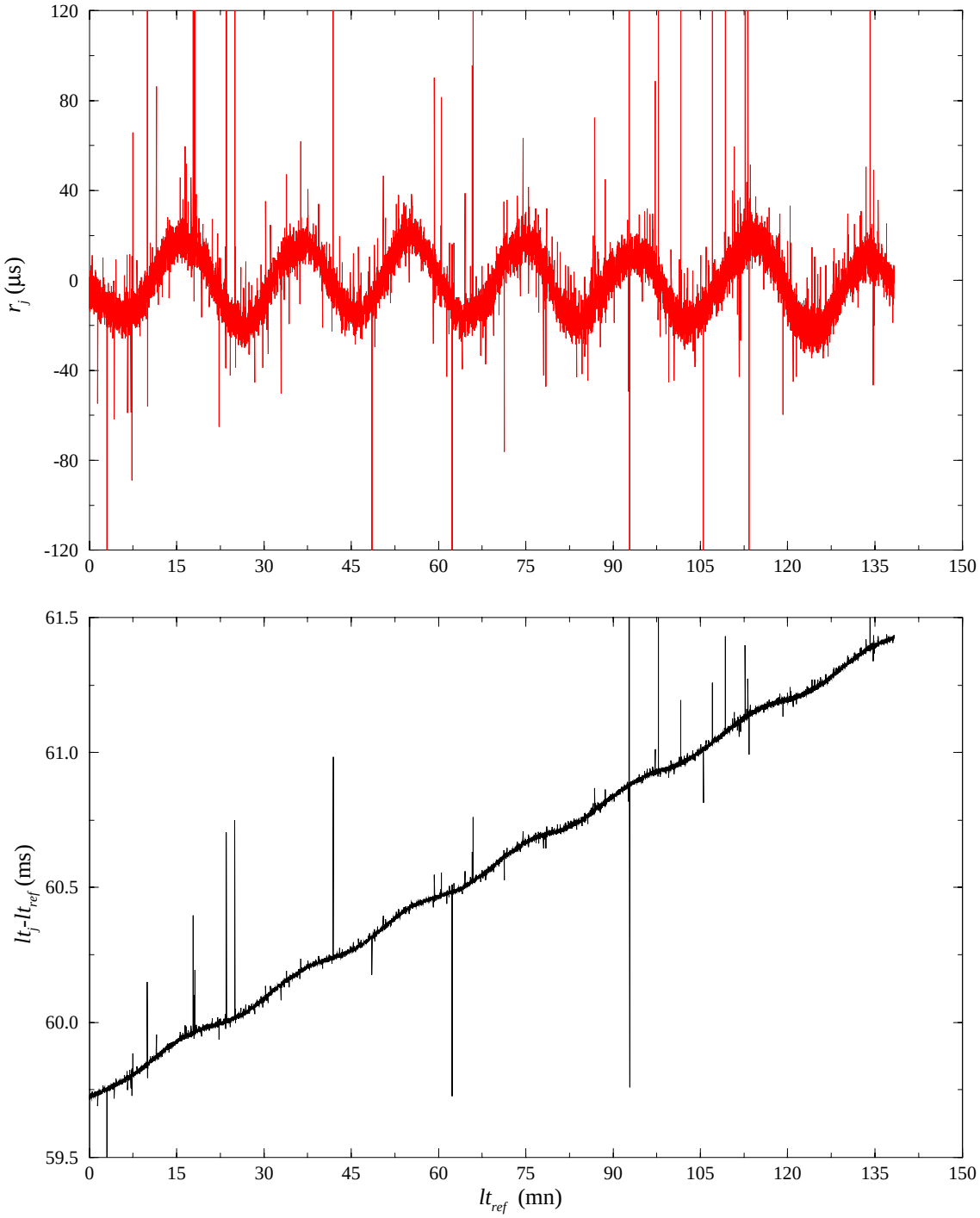


FIG. 3.5 – *Échantillon d'horloges du IBM-SP2. La partie inférieure représente* $lt_j - lt_{ref} = (\beta_{ref,j} - 1)\, lt_{ref} + \alpha_{ref,j}$. *La partie supérieure montre l'erreur d'approximation par la droite des moindres carrés.*

partie supérieure de la figure montre le tracé de l'échantillon E_j qui représente le bruit autour de la droite des moindres carrés. Comme dans le cas du Meganode, l'apparition d'oscillations est très nette et le signal possède aussi une structure en «bande». Cette fois, cette bande est nettement plus étroite (environ $12\mu s$) et n'est pas due à la résolution de l'horloge (de l'ordre de la nanoseconde sur un RS/6000[10]), mais à la variance des temps de communication. On observe aussi un certain nombre de mesures aberrantes. Une telle mesure est obtenue lors d'un échange de message dissymétrique (cf. section 3.2.3). La période des oscillations est aussi d'environ 17 minutes, ce qui confirme la thèse de l'influence de la température ambiante – le Meganode et le SP2 se trouvent en effet dans le même salle machine. Sur le SP2, l'amplitude du bruit est inférieure à $20\mu s$. En utilisant la même borne que dans la section précédente, il faut donc s'attendre, avec une méthode statistique, à une erreur maximale de $\epsilon = 40\mu s$. Sur le SP2 il existe différentes bibliothèques de communication par messages dont la latence des communications est différente :

- PVM Oak Ridge [29] sur TCP/IP : malgré l'implantation de TCP/IP optimisée par IBM pour tirer profit du *Switch*, la latence des communications reste très élevée ; elle est proche de 3ms ;
- MPL, MPICH et MPI-F [26] : MPICH est une version domaine public de MPI qui est basée sur MPL, la bibliothèque propriétaire de IBM pour les systèmes SP. La latence de MPICH est supérieure à $55\mu s$. MPI-F est basée sur une extension de MPL – les latences des deux bibliothèques sont comparables et sont supérieures à $40\mu s$.[11]

Dans le cas de PVM, la latence est donc de deux ordres de grandeur supérieure à l'erreur d'estimation – le respect de l'ordre causal est ainsi garanti. Pour MPL et MPI-F, la latence minimale approche, voire égale, l'erreur maximale d'estimation. L'approche par le modèle linéaire est donc moins satisfaisante que dans le cas du Meganode. On peut alors se poser la question si la modélisation de l'erreur pourrait nous apporter quelque chose. C'est l'objectif de la section suivante.

3.3.2 Modélisation de l'erreur

Les expériences de la section précédente ont permis de dégager la nature oscillatoire des résidus du modèle linéaire. Il est clair qu'une telle forme

10. Source: comp.unix.aix

11. Ce sont les chiffres donnés par IBM. La latence effectivement mesurée avec les primitives d'émission bloquantes de MPI-F est supérieure à $200\mu s$ [40].

d'erreur est hétéroscédastique et qu'il y a corrélation. Les hypothèses des moindres carrés ordinaires ne sont donc pas vérifiées. L'objectif de cette section est d'étudier l'apport d'une modélisation de la nature oscillatoire du bruit. Celle-ci ferait donc partie du modèle plutôt que du résidu. Cette approche a trois avantages :

1. les résidus correspondants vérifient les propriétés d'homoscédasticité et de non-corrélation, ce qui permet d'obtenir une bonne estimation de la variance des résidus et de la variance des paramètres du modèle (nous revenons sur ce point en section 3.3.3) ;

2. elle fournit un estimateur précis à la fois de l'amplitude et de la période du bruit (par rapport à l'estimation grossière qu'on a faite dans la section 3.3.1 depuis les tracés) ; la comparaison de la valeur estimée de l'amplitude avec la latence des communications nous permettra soit d'accepter soit de rejeter le modèle linéaire ;

3. elle fournit un estimateur de la tendance linéaire principale de l'échantillon (écart à l'origine et dérive) ; la comparaison des valeurs estimées avec celles obtenues par le modèle linéaire simple nous permettra de conclure sur la pertinence du modèle linéaire.

Le modèle oscillatoire non-linéaire

Dans le cas de données comprenant une composante périodique régulière, on peut inclure un terme sinusoïdal dans le modèle. Les résidus r_j du modèle linéaire sont modélisés comme suit :

$$r^k_{ref,j} = A_{ref,j} sin(\theta_{ref,j}\, lt_{ref}(R^k_{ref}) + \theta^0_{ref,j}) + v^k_{ref,j}, \quad k = 1, .., n,$$

où $A_{ref,j}$ est l'amplitude du bruit, $\theta_{ref,j}$ est la période du bruit, $\theta^0_{ref,j}$ est la phase à l'origine et $v^k_{ref,j}$ est le résidu (variable aléatoire) pour la k$^{\text{e}}$ observation. Ces trois nouveaux paramètres s'ajoutent aux deux paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$ du modèle linéaire. Le vecteur inconnu des paramètres $\beta^{(j)} \in \mathbb{R}^5$ devient alors :

$$\beta^{(j)} = \left[\begin{array}{ccccc} \alpha_{ref,j} & \beta_{ref,j} & A_{ref,j} & \theta_{ref,j} & \theta^0_{ref,j} \end{array} \right]'.$$

Considérons l'opérateur vectoriel $\mathbf{f}(\beta^{(j)}, X^{(j)})$ de $\mathbb{R}^5 \times \mathbb{R}^n$ dans $\mathbb{R}^n$ dont les composantes f_k de $\mathbb{R}^5 \times \mathbb{R}^n$ dans $\mathbb{R}$ sont définies par

$$f_k(\beta^{(j)}, X^{(j)}) = \alpha_{ref,j} + \beta_{ref,j}\, lt_{ref}(R^k_{ref}) + A_{ref,j}\, sin(\theta_{ref,j}\, lt_{ref}(R^k_{ref}) + \theta^0_{ref,j})$$

où

$$X^{(j)} = \left[\begin{array}{ccc} lt_{ref}(R^1_{ref}) & \cdots & lt_{ref}(R^n_{ref}) \end{array} \right]'.$$

On peut alors écrire l'équation de régression non-linéaire

$$Y^{(j)} = \mathbf{f}(\beta^{(j)}, X^{(j)}) + v^{(j)}$$

où

$$Y^{(j)} = \left[\begin{array}{ccc} \widehat{lt_j}(R^1_{ref}) & \cdots & \widehat{lt_j}(R^n_{ref}) \end{array} \right]'$$

et

$$v^{(j)} = \left[\begin{array}{ccc} v^1_{ref,j} & \cdots & v^n_{ref,j} \end{array} \right]'.$$

Comme dans le cas des moindres carrés ordinaires (OLS), nous supposons que $E(v^{(j)}) = 0$ et que $E(vv') = \sigma^2 I$ (homoscédasticité et non-corrélation). La méthode des moindres carrés consiste alors à trouver $\widehat{\beta^{(j)}}$ qui minimise

$$Q(\beta^{(j)}) = \| Y^{(j)} - \mathbf{f}(\beta^{(j)}, X^{(j)}) \|^2 .$$

Nombre d'algorithmes ont été proposés pour calculer $\widehat{\beta^{(j)}}$ de façon itérative. Le plus classique est sans doute l'algorithme de Gauss-Newton dont nous rappelons ici le principe [12]. Soit $\beta_i^{(j)}$ l'estimation de $\beta^{(j)}$ à la i$^{\text{e}}$ itération. Notons $\mathbf{F_i} \in \mathbb{R}^n \times \mathbb{R}^5$ la matrice jacobienne de l'opérateur $\mathbf{f}$ évaluée en $\beta_i^{(j)}$:

$$\mathbf{F_i} = \left. \frac{\delta \mathbf{f}(\beta^{(j)}, X^{(j)})}{\delta \beta^{(j)}} \right|_{\beta^{(j)} = \beta_i^{(j)}} .$$

Selon les implantations de la méthode, cette matrice est soit fournie analytiquement, soit approchée numériquement. On effectue alors les itérations

$$\beta_{i+1}^{(j)} = \beta_i^{(j)} + (\mathbf{F_i}'\mathbf{F_i})^{-1}\mathbf{F_i}' \left[Y^{(j)} - \mathbf{f}(\beta^{(j)}, X^{(j)}) \right]$$

jusqu'à convergence. Notons $Y^* \in \mathbb{R}^n$ le résultat de l'application de l'opérateur $\mathbf{f}$ à la valeur estimée de $\beta^{(j)}$:

$$Y^* = \mathbf{f}(\widehat{\beta^{(j)}}, X^{(j)}).$$

L'estimateur de σ^2 de la variance des résidus s'écrit alors

$$\widehat{\sigma}^2 = \frac{\| Y^{(j)} - Y^* \|^2}{n - 5}$$

d'où l'on déduit la matrice de variance-covariance asymptotique de $\widehat{\beta^{(j)}}$:

$$\widehat{V}(\widehat{\beta^{(j)}}) = \widehat{\sigma}^2 \, (\widehat{\mathbf{F}}'\widehat{\mathbf{F}})^{-1}$$

où $\widehat{\mathbf{F}}$ est l'évaluation de la matrice jacobienne en $\beta^{(j)} = \widehat{\beta^{(j)}}$. Notons que les estimateurs de σ^2 et $\widehat{\beta^{(j)}}$ ne dépendent pas de la méthode utilisée pour le calcul de $\beta^{(j)}$.

12. En pratique, on utilisera plutôt une méthode plus complexe basée sur la combinaison des méthodes Gauss-Newton et Levenberg-Marquardt [20], comme le font la plupart des logiciels de régression non-linéaire.

Application numérique

On a appliqué le modèle oscillatoire à l'échantillon d'horloges relevé sur le SP2. Les valeurs estimées des 5 paramètres sont les suivantes [13] :

$$\alpha_{ref,j} = 59\,726,800 \pm 0,800 \ \mu s,$$

$$\beta_{ref,j} = 1,000\,000\,2046 \pm 0,000\,000\,0002,$$

$$A_{ref,j} = 17,5 \pm 0,6 \ \mu s,$$

$$\theta_{ref,j} = 0,320\,800 \pm 0,000\,800 \ \frac{\text{rad}}{\text{mn}}$$

et

$$\theta^0_{ref,j} = 2,72 \pm 0,06 \text{ rad.}$$

La partie inférieure de la figure 3.6 montre l'estimation de la nature oscillatoire du bruit alors que la partie supérieure montre le résidu correspondant. Cette fois, l'axe des résidus nuls ($v_j = 0$) est bien centré dans la bande. Par rapport aux résidus r_j de la figure 3.5, *le caractère hétéroscédastique disparaît.* Par ailleurs, le test d'autocorrélation de Durbin-Watson reporte une valeur de 1.784, ce qui indique une *autocorrélation positive très faible* (une valeur de 2 permet de conclure à l'absence d'autocorrélation [45]). Ce même test appliqué au modèle linéaire simple, donne une valeur de 0.892, indiquant une forte autocorrélation positive. Contrairement à r_j, les résidus v_j du modèle oscillatoire vérifient l'hypothèse d'homoscédasticité et de non-corrélation – les écarts-type empiriques des 5 paramètres sont par conséquent de bonnes approximations des écarts-type théoriques (cf. la section 3.3.3).

3.3.3 La pertinence du modèle linéaire

Les échantillons que nous avons analysés en section 3.3.1 nous ont permis de bien dégager la nature de l'erreur d'estimation du modèle linéaire et de déterminer avec précision la valeur des paramètres de dérive et d'écart d'horloges. L'utilisation du modèle oscillatoire dans la section 3.3.2 nous a permis d'estimer la période et l'amplitude du bruit. Le fait que le double de l'amplitude soit inférieur à la latence minimale de communication prouve que *le modèle linéaire est suffisant pour garantir la datation cohérente d'événements distribués* (sous condition d'avoir un échantillon de grande taille afin de pouvoir capter la tendance linéaire principale). Dans le modèle linéaire, le bruit oscillatoire est considéré comme faisant partie des résidus. Ces derniers sont donc de nature hétéroscédastique, sont corrélés positivement (cf.

13. Après 7 itérations il y a convergence du vecteur des paramètres.

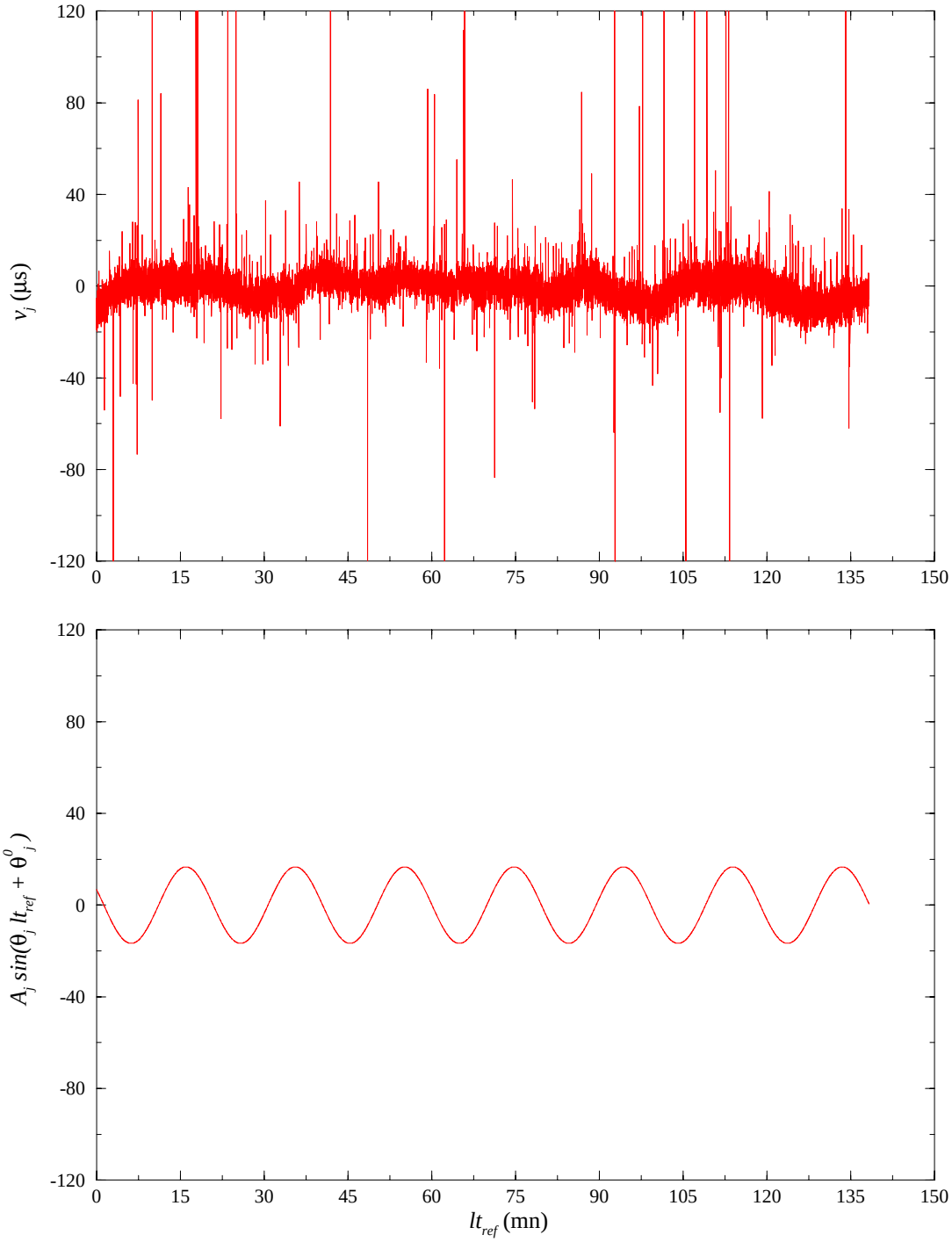


FIG. 3.6 – *Le modèle oscillatoire pour l'échantillon d'horloges du SP2 (partie inférieure) et les résidus correspondants (partie supérieure).*

test de Durbin-Watson) et ne vérifient pas les hypothèses de la méthode des moindres carrés ordinaires, ce qui peut entraîner les conséquences suivantes [45] :

1. $\widehat{\beta^{(j)}}$ n'a plus la propriété d'estimateur de variance minimale (toutefois, il reste sans biais) ;
2. l'estimateur $\widehat{\sigma}^2$ de l'équation 3.12 peut fortement sous-estimer la variance des résidus ;
3. l'écart-type empirique des composantes de $\widehat{\beta^{(j)}}$ calculé à partir de la formule 3.11 (racine carrée des valeurs de la diagonale) peut sous-estimer l'écart type théorique.

Malgré tout, $\widehat{\beta^{(j)}}$ reste un estimateur sans biais de $\beta^{(j)}$ [14]. Cependant, les résultats d'inférence d'intervalles de confiance sous-estiment l'erreur d'estimation et sont donc à interpréter avec précaution. Leur utilisation en pratique va quand même s'avérer intéressante : ce sont en effet de bons indicateurs du «bruit système» (cf. section 3.4.4). Si l'on compare, finalement, les valeurs et écarts-type de $\alpha_{ref,j}$ et $\beta_{ref,j}$ estimés par le modèle linéaire avec ceux estimés par le modèle oscillatoire, on constate que les différences sont minimes, ce qui confirme encore une fois l'applicabilité du modèle linéaire malgré la nature des résidus.

La détermination précise de la valeur des paramètres du modèle décrite dans cette section n'est possible que si l'on dispose d'un échantillon de grande taille montrant la relation entre les horloges lt_{ref} et lt_j sur plusieurs périodes oscillatoires c.-à-d. sur une ou plusieurs heures. Cependant, pour l'implantation d'algorithmes de construction de temps global, nous ne pouvons pas utiliser des périodes d'échantillonnage arbitrairement longues afin d'obtenir la précision souhaitée. La section prochaine discute ce type de contraintes dans le cadre d'une implantation *efficace* des estimateurs.

3.4 Implantation

Dans la section précédente, nous avons montré que le temps global calculé à partir d'échantillons de grande taille est suffisamment précis pour permettre la datation cohérente d'événements répartis. Cependant, en pratique, il n'est pas possible de relever un échantillon de taille importante chaque fois

14. A condition que $E(u) = 0$, ce qui n'est pas vérifié dans le modèle linéaire. Ici, l'approximation $E(u) = 0$ peut être justifiée par le fait que les résidus sont équilibrés de part et d'autre de l'axe $u = 0$, sous réserve de prendre un échantillon suffisamment long.

qu'une application requiert le calcul d'une référence globale de temps. Dans cette section, nous décrivons les contraintes d'implantation que nous devons prendre en compte sur les *systèmes de traitement par lots*, et nous présentons plusieurs stratégies d'implantation avec leurs avantages et désavantages. Ces stratégies sont évaluées par rapport à la forme du bruit observée en section 3.3.1. Notre but est de calculer un temps global pour une application, sans retarder son lancement de façon significative.

3.4.1 Contraintes dues au système

Nous supposons que le système parallèle pour lequel on calcule le temps global fonctionne en mode de traitement par lots : les applications sont soumises au système les unes après les autres moyennant un système de gestion de file d'attente ou de réservation.

Le système *Meganode* du LMC est équipé de 128 Transputers T800[15] qui sont interconnectés par un réseau reconfigurable [4]. Les processeurs sont redémarrés entre deux lancements successifs d'applications, ce qui initialise les valeurs des horloges physiques. Ce processus d'initialisation dépend de la taille du code des tâches qui sont chargées sur les noeuds [9]. En conséquence, chaque application doit échantillonner les horloges pour permettre le calcul du temps global.

Bien que ce ne soit pas une machine qui effectue du traitement par lots (au sens classique du terme), le IBM-SP2 est souvent utilisé comme telle. Pour faire des mesures de performance précises, on peut en effet réserver la machine sur une période précise, durant laquelle on en détient l'utilisation exclusive. Contrairement au Meganode, le chargement d'une application sur un ensemble de noeuds n'a aucun effet sur les horloges de ceux-ci. Il est donc *a priori* concevable d'utiliser les mesures d'horloges faites par une exécution d'application pour les exécutions d'applications (éventuellement différentes) suivantes. Pour le moment, dans un souci de généralité, *nous admettons que chaque application est responsable d'estimer son propre temps global.*

Plus la taille des échantillons est grande et plus la période d'échantillonnage est longue, meilleure sera la précision du temps global et plus les délais subis par l'application seront importants. Ainsi, une stratégie d'implantation de temps global doit trouver un bon équilibre entre précision et délai d'exécution dû à l'échantillonnage.

15. Les T800 sont regroupés par 8 sur 16 cartes. Chaque carte dispose d'un oscillateur local. La machine ne comporte donc que 16 "horloges".

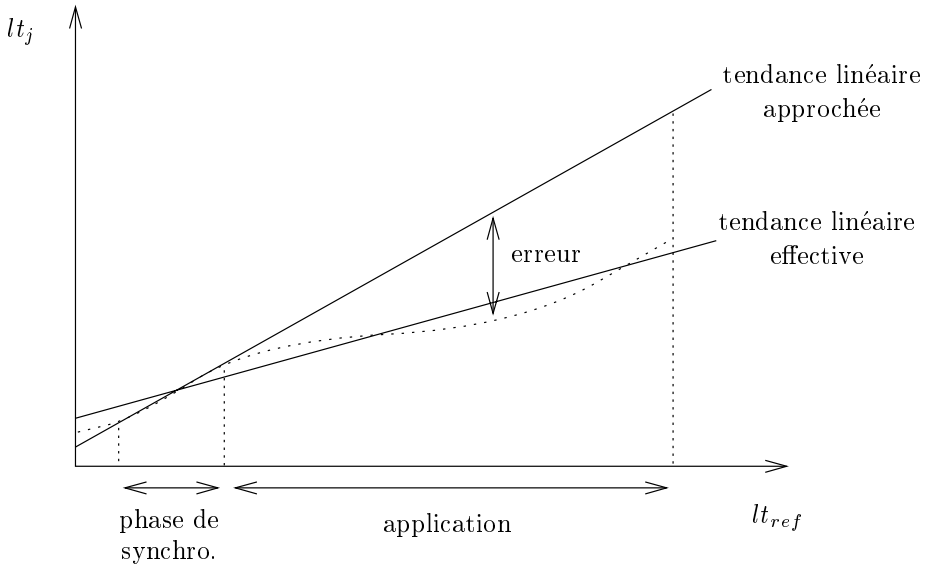


FIG. 3.7 – *L'extrapolation du temps global depuis une phase de synchronisation courte peut entraîner d'importantes erreurs d'approximation.*

3.4.2 Stratégie SB : extrapolation du temps global

Quand toutes les tâches sont chargées, une phase de synchronisation est effectuée lors de laquelle le processeur de référence échange un certain nombre de messages avec les autres processeurs P_j du système, afin de relever les échantillons S_j ($j \in \{1..p\} \backslash \{ref\}$) à partir desquels le temps global est estimé par les algorithmes présentés dans la section 3.2. Puis, l'application est lancée (stratégie *Sample Before*). Pour dater les événements tracés, l'application (ou l'outil de trace) utilise les horloges locales. Après l'exécution, les dates du fichier de trace sont transformées en dates globales par l'équation 3.3 utilisant les valeurs estimées des paramètres $\alpha_{ref,j}$ et $\beta_{ref,j}$ sauvegardées dans un fichier par la phase de synchronisation. Notons qu'il est également possible de corriger les dates à la volée, à condition que chaque processeur connaisse les valeurs des deux paramètres du modèle linéaire.

Afin de limiter le délai subi par l'application, la phase de synchronisation ne doit pas excéder quelques minutes, ce qui réduit la précision de l'estimation des paramètres. La figure 3.7 montre l'erreur d'estimation du temps global résultant d'une phase de synchronisation trop courte. A cause du bruit qui se superpose à la tendance linéaire principale, la phase de synchronisation ne peut pas capter cette tendance. Les indicateurs statistiques (le coefficient de corrélation, par exemple) reportent toutefois une bonne adéquation du

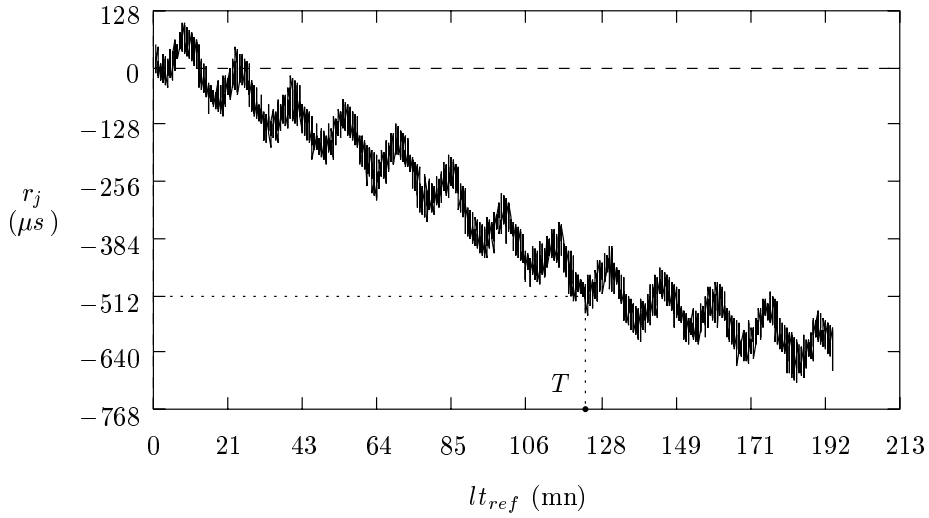


FIG. 3.8 – *Erreur d'approximation du temps global par une stratégie SB de 5 minutes (échantillon Meganode). Après un temps T d'environ 2 heures, des incohérences de datation apparaissent entre P_{ref} et P_i.*

modèle linéaire, car, en effet, l'approximation sur la phase de synchronisation elle-même est très bonne. Cependant, si cette approximation est *extrapolée* au-delà de la phase de synchronisation, pour estimer le temps global dans l'application elle-même, l'erreur d'approximation croît rapidement, comme le montre la figure 3.7, et la datation cohérente ne peut plus être garantie. La figure 3.7 explique aussi pourquoi les estimations de la valeur d'un même paramètre de dérive, estimée à partir de différentes parties contiguës d'un même échantillon sont différentes (cf. section 3.2.4).

Pour illustrer ce phénomène sur un échantillon réel, nous traçons un échantillon E_j (Meganode), comme en section 3.3.1, sauf que les paramètres du modèle sont calculés à partir de la première partie de l'échantillon R_j, correspondant aux 5 premières minutes de la phase d'échantillonage, plutôt qu'à partir de l'intégralité de l'échantillon R_j. La figure 3.8 montre le tracé obtenu. On constate qu'au-delà des premières 5 minutes, qui correspondent à la phase de synchronisation, l'erreur d'estimation du temps global augmente rapidement. Après environ 2 heures, l'erreur est suffisamment grande pour causer des incohérences de datation entre P_{ref} et P_j (nous utilisons toujours $\mu = 512\mu s$ comme latence minimale sur le Meganode)[16]. A ce moment là, une autre phase de synchronisation pourrait être effectuée pour calculer une nou-

16. Suite à des effets de compensation d'erreur, les incohérences de datation entre P_i et P_j, autres que P_{ref}, peuvent apparaître plus tard.

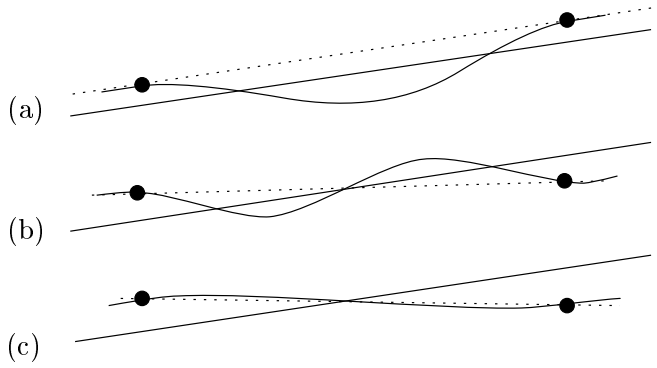


FIG. 3.9 – *L'erreur d'approximation de la stratégie SBA dépend de la position relative des phases de synchronisation (considérées comme ponctuelles sur la figure).*

---

velle estimation des valeurs des paramètres. Malheureusement, comme nous l'avons déjà souligné dans l'introduction à ce chapitre, de telles resynchronisations perturbent l'application en cours d'exécution. Afin d'augmenter la précision obtenue par la stratégie SB, tout en évitant les resynchronisations, la phase de synchronisation initiale doit être allongée. Une phase de synchronisation dont la longueur est proche de la période du bruit donnerait des résultats bien meilleurs, puisqu'elle approcherait de près la tendance linéaire effective. Sur nos systèmes, la période du bruit est de 17 minutes, ce qui est bien trop long. Par conséquent, la stratégie SB ne peut être utilisée que pour des applications dont le temps d'exécution n'est pas trop long.

3.4.3 Stratégie SBA : interpolation du temps global

L'idée consiste à effectuer une phase d'échantillonnage *avant et après* l'exécution de l'application qui a besoin du temps global (*Sample Before-After*). Les échantillons obtenus par ces deux phases d'échantillonnage sont concaténés, et les paramètres du modèle d'horloge sont calculés à partir de l'échantillon qui en résulte. La figure 3.9a montre que l'erreur d'approximation maximale (distance entre la droite en pointillés et la courbe) est bornée par le double de l'amplitude A du bruit. Sur la figure 3.9c l'erreur est la plus petite. La figure 3.9b montre un cas intermédiaire. Notons que la droite (non-pointillée) des figures 3.9 représente la tendance linéaire principale qu'on obtiendrait à partir d'un échantillon de grande taille, comme en section 3.3.1. L'erreur d'approximation de la stratégie SBA étant inférieure au double de l'amplitude du bruit, i.e. $|LC_j - lt_{ref}| < 2A$, il en résulte que

$|LC_j - LC_i| < 4A$ $(i \neq j)$. La granularité ϵ du temps global est donc égale à $4A$ (par opposition à $\epsilon = 2A$ si l'on utilise un échantillon de grande taille). Si l'on se réfère aux temps de latence donnés en section 3.3.1, on voit qu'il y a possibilité d'incohérence de datation avec la bibliothèque de communication MPI sur le IBM-SP2 ($\epsilon = 68\mu s, \mu = 40\mu s$)[17]. Remarquons qu'il est possible de remédier à ce problème en augmentant la longueur des deux phases d'échantillonnage pour mieux s'approcher de la tendance linéaire effective. La borne $\epsilon = 4A$ est une borne au pire, car on considère que les phases de synchronisation sont ponctuelles sur la figure 3.9.

Toutefois, le gain en précision de la stratégie SBA par rapport à la stratégie SB est considérable. Il faut cependant noter qu'elle ne permet pas l'accès au temps global pendant l'application, les paramètres n'étant évalués qu'après la deuxième phase d'échantillonnage. Ceci n'est pas un désavantage dans le cas où le temps global est utilisé pour dater les événements lors de prises de traces destinées à une analyse post-mortem.

Pour montrer l'erreur d'approximation de la stratégie SBA en pratique, nous procédons comme lors de l'analyse de la stratégie SB en traçant un échantillon E_j (Meganode). Cette fois, les paramètres utilisés pour le calcul de E_j sont estimés à partir des points de l'échantillon R_j correspondant aux premières 150 secondes et aux dernières 150 secondes, ce qui correspond à un délai total de 5 minutes, comme lors de l'analyse de la stratégie SB. La figure 3.10 montre le tracé (pour faciliter la comparaison, nous utilisons la même échelle que sur la figure 3.8). L'erreur reste suffisamment petite pour éviter les incohérences de datation.

La stratégie SBA a deux avantages significatifs sur la stratégie SB :

1. elle permet la *datation cohérente sur des durées d'exécution arbitrairement longues*, sans perturber l'application, tant que la relation entre lt_{ref} et lt_j reste linéaire et que le bruit reste suffisamment petit par rapport à la latence des communications ; dans [44], la linéarité à été vérifiée sur une durée de 10 heures – pour des périodes encore plus longues, le modèle linéaire n'est plus valide à cause du taux de vieillissement des cristaux d'horloges (cf. section 3.2.1) ; dans notre cas, où l'on utilise le temps global pour dater les événements lors de prises de traces, les temps d'exécution sont généralement bien plus courts et ne dépassent guère une ou plusieurs heures (problème du volume des traces) ;

2. elle permet la *mesure cohérente d'intervalles de temps (à bornes répar-*

17. Si l'on prend $A = 50\mu s$ sur le Meganode, $\epsilon = 200\mu s$ et il n'y a donc pas d'incohérences de datation($\mu = 512\mu s$).

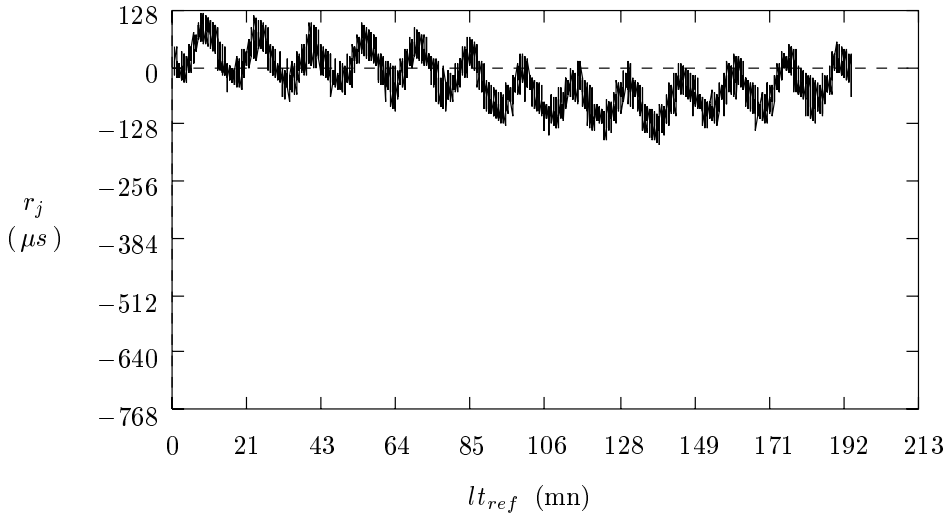


FIG. 3.10 – *Erreur d'approximation du temps global par une stratégie SBA de 5 minutes (échantillon Meganode). Il n'y a pas d'incohérences de datation, même pour les applications dont le temps d'exécution est très grand.*

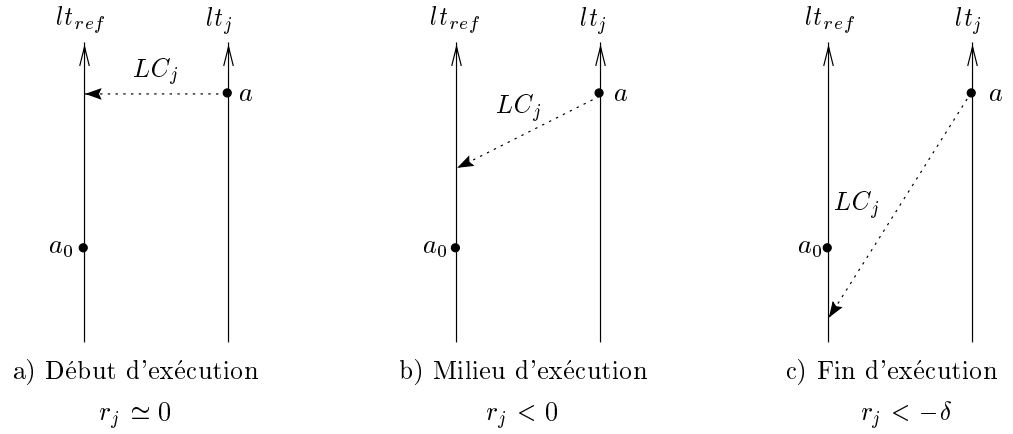


FIG. 3.11 – *Stratégie SB : mesure de la durée δ d'un intervalle $[a_0, a]$ à bornes réparties (a_0 sur P_{ref}). Le temps évolue de bas en haut. L'erreur r_j grandit en valeur absolue de la figure a) à la figure c). La figure c) montre la mesure d'un délai «apparemment» négatif (cette évolution de l'erreur correspond à celle de la figure 3.7).*

ties) ; son erreur d'approximation est bornée par le double de l'amplitude du bruit, alors que l'erreur induite par la stratégie SB augmente (en valeur absolue) avec le temps – le temps global SB ne vérifie pas la propriété d'accord, ce qui veut dire qu'un même intervalle de temps physique peut paraître plus ou moins long selon qu'il est mesuré au début ou à la fin de l'application (cf. figure 3.11); dans certains cas la mesure d'une durée peut même être négative (*délai pseudonégatif*).

Notons le *lien entre datation cohérente et mesure cohérente d'intervalles de temps*. En effet, si a_0 correspond à l'envoi d'un message à partir de P_{ref}, et a à la réception de ce même message sur P_j, l'intervalle $\delta = t(a) - t(a_0)$ représente le temps de communication et il y a incohérence de datation si $r_j < -\delta$ (cf. figure 3.11c). Le premier point ci-dessus est donc un corollaire du deuxième. Dans le cas d'une erreur d'approximation r_j croissante et positive de la stratégie SB (cas contraire de celui de la figure 3.11), la mesure d'un même intervalle de temps devient de plus en plus grande, ce qui masque le «dépistage causal» de la violation de la propriété d'accord. Enfin, dans les deux cas, erreur SB croissante par valeurs positives ou négatives, les statistiques sur la mesure des temps de communication basée sur le temps SB seront erronnées.

3.4.4 SB et SBA par la pratique

Nous avons eu la possibilité de tester les stratégies d'échantillonnage sur différentes configurations de systèmes. Cette section fait le bilan sur nos expériences avec le temps global statistique et propose quelques extensions et conseils pour garantir une utilisation optimale.

Coexistence avec Network Time Protocol (NTP)

Le protocole NTP organise un ensemble de stations de travail en une structure d'arbre. La station qui est à la racine de l'arbre représente le temps universel : son horloge est souvent synchronisée avec une référence temporelle diffusée par ondes radio ou elle est équipée d'une horloge atomique. Plus on remonte les niveaux de l'arborescence, appelés «strates» (*strata*, en anglais), plus on se rapproche du temps de référence universel, et plus la notion de temps des machines correspondantes devient *juste* (au sens de la propriété $P3$ introduite dans la section 3.1 de ce chapitre). Au sein d'un réseau local, les machines recalent périodiquement leurs horloges sur celle du serveur NTP de ce réseau, qui elle-même recale son horloge sur un serveur NTP d'un autre réseau local qui est son prédecesseur dans l'arborescence. Ces *resynchronisations périodiques* assurent l'*accord* du temps (au sens de la propriété

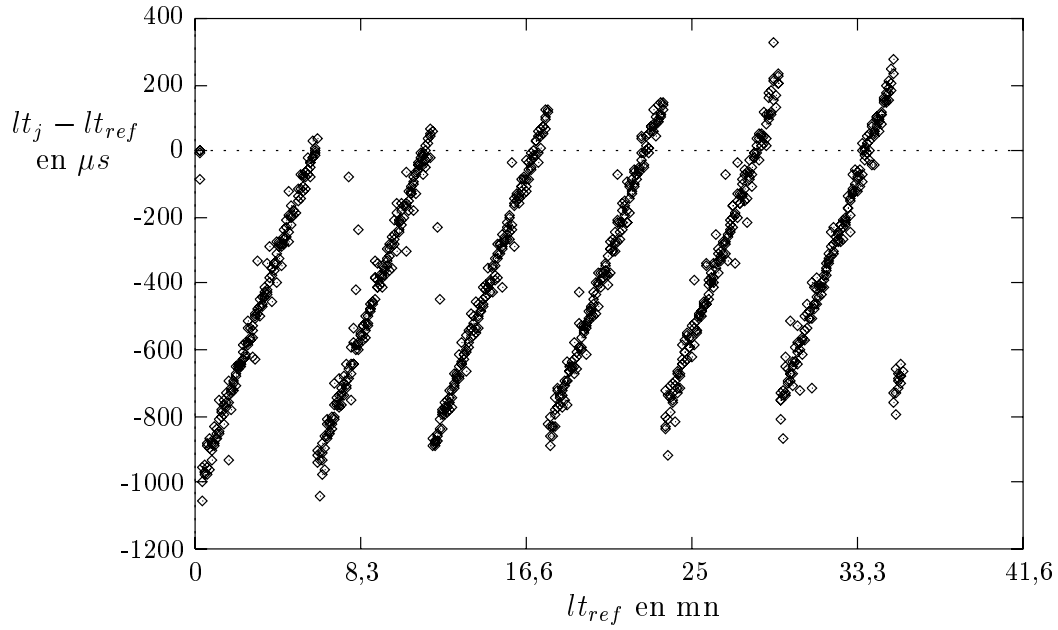


FIG. 3.12 – *Effet du Network Time Protocol (NTP) sur l'écart entre deux horloges (échantillon IBM SP2). Ce tracé est à comparer avec la partie inférieure de la figure 3.5, page 47, où NTP est désactivé.*

---

*P*2) au sein du réseau local. Généralement, la granularité est de l'ordre de la milliseconde, ce qui peut paraître suffisant pour un certain nombre d'applications. L'exemple classique est le programme UNIX *make* qui compile de nouveaux fichiers binaires s'il constate que les fichiers source correspondants sont plus récents. Dans un système distribué, ce mécanisme peut ne pas marcher correctement dans le cas où les binaires se trouvent sur une autre machine que le code source et que les machines ne sont pas en accord sur le temps global. Alors que le temps NTP permet de résoudre le problème *make*, sa granularité n'est pas assez fine pour distinguer une émission de message de la réception correspondante (délai de l'ordre de la dizaine ou de la centaine de microsecondes, selon les systèmes ; cf section 3.3.1).

Après cette description succinte de NTP, nous allons nous intéresser plus particulièrement à l'effet des resynchronisations périodiques sur la forme des échantillons d'horloge. La figure 3.12 montre la différence entre deux valeurs d'horloges du IBM-SP2, lt_j et lt_{ref}, en fonction de lt_{ref}. Comme l'on pouvait s'y attendre, les resynchronisations provoquent une forme en «dents de scie». Par ailleurs, l'amplitude des dents de scie reflète très nettement la granularité du temps global visée par NTP qui est d'une milliseconde dans la configuration étudiée au prix d'une resynchronisation toutes les 5 minutes environ.

Il est évident qu'une telle forme d'échantillon rend impossible toute utilisation de méthode statistique basée sur le modèle linéaire. Dans la pratique, on observe que la présence d'un mécanisme de resynchronisation périodique se manifeste par des intervalles de confiance anormalement larges des paramètres de décalage $\alpha_{ref,j}$ (entre 5% et 60%).

Sur le système IBM-SP2 du LMC nous avons donc décidé de réduire la *fréquence des resynchronisations* à *une fois par semaine*. L'écart maximal entre deux horloges est alors d'environ 6 secondes (en considérant une valeur pessimiste de 10^{-5} pour la dérive), ce qui reste acceptable à l'échelle humaine. Cette approche à les deux avantages suivants :

1. elle permet *l'utilisation des méthodes statistiques* (échantillonnage SB ou SBA) pour la datation précise d'événements répartis, tout en gardant une granularité du temps réseau acceptable à l'échelle humaine ;
2. en éliminant les resynchronisations périodiques, elle *réduit le bruit réseau* et, par la même, la variance des mesures de performances sur les applications parallèles ou réparties.

Le «bruit système»

La mesure des performances d'une application parallèle ou répartie nécessite la datation répartie d'événements. L'utilisation d'une méthode statistique permet de faire une telle datation de façon cohérente. L'application analysée doit relever des échantillons d'horloges avant (et après, selon la stratégie utilisée) son exécution – le temps d'exécution total se retrouve donc rallongé. La *précision de l'estimation* du temps global dépend du *nombre de points mesurés* et de la *longueur des phases d'échantillonnage*. Pour une longueur de phase d'échantillonnage donnée, le nombre de points à mesurer dépend du nombre de mesures aberrantes, c.-à-d. du nombre de disymétries dans les échanges de messages (la symétrie des échanges de messages est une hypothèse de base des méthodes statistiques), qui lui, dépend de la charge du réseau de communication. Cette charge peut résulter des applications d'autres utilisateurs et/ou des activités du système d'exploitation distribué : citons par exemple les démons NFS ou NTP. L'effet combiné de ces activités du système est souvent appelé «bruit système». Tout comme les dents de scie induites par NTP, ce bruit se reflète dans les écarts-type et les intervalles de confiance des paramètres d'horloge.

Les ressources du système *Meganode* du LMC sont entièrement dédiées à l'application en cours d'exécution. Le système d'exploitation se limite à une couche de routage de messages (*Virtual Channel Router*, VCR). En conséquence, les temps de communication sont extrèmement stables et il suffit de

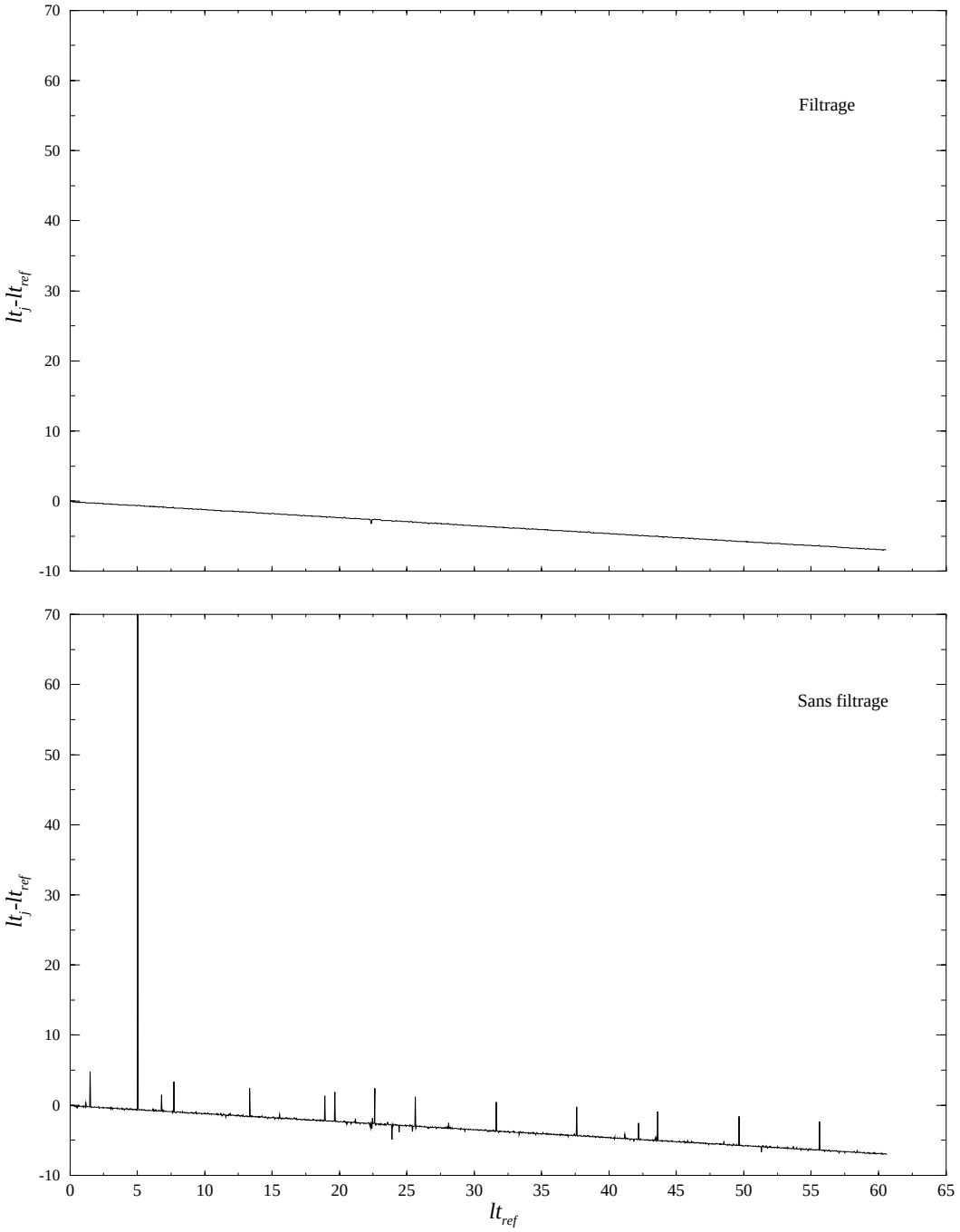


FIG. 3.13 – «*Bruit système*» *sur le IBM-SP2 (NTP désactivé) : les pics de la partie inférieure de la figure dus aux disymétries dans les échanges de messages disparaissent après application de l'algorithme de filtrage de la médiane mobile (partie supérieure).*

très peu de mesures pour obtenir une bonne estimation. En pratique, avec la stratégie SBA, 10 points de mesure sur 10 secondes par phase d'échantillonnage se sont révélés suffisants.

Sur le IBM-SP2, le «bruit système» est non-négligeable. La partie inférieure de la figure 3.13 montre l'estimation de la différence $\widehat{lt_j} - lt_{ref}$ entre deux horloges du système (les communications sont effectuées sur le cable Ethernet plutôt que par le *Switch* de la machine). Ces mesures ont été réalisées avec le système NTP désactivé. Les disymétries de communication sont mises en évidence par l'apparition de «pics». Le fait que la grande majorité de ces pics pointent vers le haut indique un retard dans la communication de P_{ref} à P_j par rapport à celle de P_j à P_{ref} (cf. sous-section 3.2.3). On note également l'apparition de perturbations périodiques, caractéristiques des activités du système d'exploitation. Pour enlever les mesures aberrantes, on utilise la technique de filtrage de la *médiane mobile*, classique en statistique[18]. La partie supérieure de la figure 3.13 montre la forme de l'échantillon après application du filtre. Pour une longueur de phase d'échantillonnage donnée, cette méthode permet de réduire considérablement l'écart-type sur les paramètres du modèle linéaire. En pratique, lorsque la machine est dédiée, avec la stratégie SBA et filtrage par médiane mobile, 15 à 20 points de mesure à raison d'un point par seconde se sont révélés suffisants.

Exécutions répétées d'une application

L'utilisation d'une technique de filtrage nous permet de réduire, voire d'éliminer, l'effet du bruit système sur la précision de l'estimation. Cependant, ce bruit a également un effet non-négligeable sur l'exécution des applications elles-mêmes. En effet, la nature non déterministe du bruit du système se reflète dans le comportement de celles-ci. Il peut simplement se manifester dans la variabilité du temps d'exécution d'une application, ou encore, de façon plus complexe, dans le changement du comportement logique de celle-ci (l'ordre partiel sur l'ensemble des événements de l'exécution, ainsi que cet ensemble lui-même, peuvent changer d'exécution en exécution). L'analyste des performances doit alors augmenter le nombre d'observations de l'application qu'il étudie afin d'obtenir des résultats représentatifs. Par observation, on entend ici l'enregistrement d'une trace d'exécution dont les événements sont datés avec le temps global statistique. Pour déterminer le nombre d'ob-

18. La méthode, implantée dans la plupart des logiciels statistiques, consiste à déplacer une fenêtre sur l'échantillon de points selon l'axe des abscisses et à prendre, pour chaque position de la fenêtre, le couple de points dont l'ordonnée est la valeur médiane des points de la fenêtre. Si la taille de l'échantillon et celle de la fenêtre sont respectivement n et k alors celle de l'échantillon filtré obtenu sera $n - k + 1$.

servations N nécessaires pour avoir une bonne estimation de la moyenne du temps d'exécution ou d'un indice de performances déduit de la trace, il aura recours à la loi des grands nombres. Typiquement, les exécutions successives de l'application tracée sont effectuées par un *script shell* de la forme:

```
répéter N fois
   exécuter le programme tracé avec la stratégie SBA
   exploiter le fichier de trace
   effacer (ou archiver) le fichier trace
fin répéter.
```

A chaque itération, le fichier de traces obtenu est exploité (calcul des indices de performance), puis effacé ou archivé, pour éviter l'encombrement du disque (les fichiers de trace ont souvent une taille importante, de l'ordre du mega-octet pour le simple traçage des appels à la bibliothèque de communication).

Sachant que l'application utilise une méthode statistique pour la datation répartie des événements tracés, *le temps total des exécutions peut augmenter de façon considérable suite à la répétition des phases d'échantillonnage.* Ainsi, dans le cas particulier d'exécutions répétées un grand nombre de fois, on préfère utiliser, pour la première exécution, une stratégie SB avec une période d'échantillonnage suffisamment longue pour capter la tendance linéaire principale des échantillons d'horloges (la durée d'échantillonnage serait de 17 minutes pour les systèmes étudiés dans ce chapitre). Les exécutions suivantes utiliseront alors les valeurs estimées des paramètres lors de la première exécution. Rappelons que le taux de vieillissement des cristaux d'horloge réduit la validité de ces coefficients à une durée d'un jour au maximum.

3.5 Discussion de l'approche de IBM

Les méthodes de construction de temps global statistique présentées dans ce chapitre sont tout à fait générales et ne font aucune hypothèse particulière quant à l'architecture du système parallèle. Toutefois, certains de ces systèmes, dont les IBM-SP, ont des particularités au niveau matériel qu'il est intéressant d'exploiter pour le développement d'une méthode de construction de temps global spécifique à ces systèmes. L'approche choisie par IBM est basée sur une méthode statistique combinée avec un mécanisme de resynchronisation périodique. Dans cette section, nous décrivons cette méthode et établissons quelques liens avec nos observations des sections précédentes.

Le IBM-SP2 dispose d'un réseau de communication haute performance

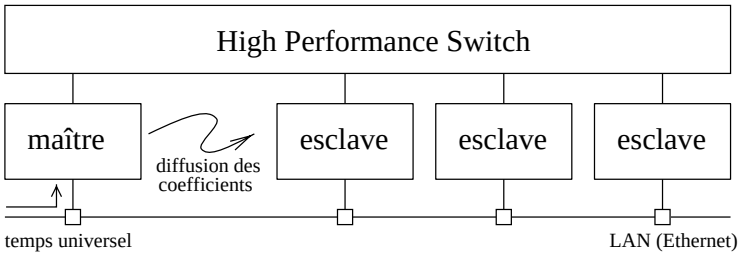


FIG. 3.14 – *Le maître diffuse les constantes de l'équation de temps* $lt_{ref} = a\,t_s + b$. *Les esclaves déterminent le temps global en lisant le temps du Switch,* t_s, *et en calculant* $lt_{ref} = a\,t_s + b$. *Le maître peut aussi recevoir un temps universel externe.*

(*High Performance Switch*) muni d'un ensemble de compteurs synchronisés à 200 nanosecondes près, sur l'ensemble du réseau[19]. Ces compteurs, appelés ATC, pour *Absolute Time Counter*, sont accessibles sur tous les processeurs et sont pilotés par un seul oscillateur : ils ne dérivent donc pas. Leur fonction primordiale est de permettre au *Switch* de basculer de façon synchrone entre ses deux modes d'opération principaux (modes *run* et *service*). Les ATC peuvent aussi être utilisés pour simplifier la construction d'une référence globale de temps pour les différents noeuds (RISC System/6000) du SP2. Bien entendu, ces noeuds disposent aussi, chacun, d'un oscillateur local indépendant, qui sert de référence temporelle au système d'exploitation et aux utilitaires système et utilisateur. Comme celles du Meganode, ces horloges sont décalées et dérivent mutuellement. Dans un travail récent [1], Abali et Stunkel proposent un mécanisme de synchronisation périodique de ces horloges système, basé sur les ATC. Ils atteignent ainsi une granularité des horloges de $5\mu s$, ce qui améliore considérablement la précision par rapport aux algorithmes de resynchronisation classiques $(1 - 3ms)$.

La méthode est basée sur le concept maître/esclave illustré sur la figure 3.14. L'idéé est d'estimer la relation, supposée linéaire

$$lt_{ref} = a\,t_s + b \tag{3.13}$$

qui lie le temps du *Switch*, t_s, au temps de l'horloge système, lt_{ref}, du processeur de référence P_{ref}.

Pour déterminer les constantes a et b, le démon maître lit périodiquement les estampilles (t_s^k, lt_{ref}^k) et calcule la droite des moindres carrés passant par les M dernières estampilles. A chaque nouvelle estampille, le démon maître

19. La machine est également doté d'un cable Ethernet qui est indépendant du *Switch*.

recalcule a et b et les diffuse aux démons esclaves. Ceux-ci, ayant accès au temps du *Switch*, t_s, peuvent alors estimer le temps de l'horloge de référence en utilisant l'équation 3.13 et ajuster leur horloge locale en conséquence, ce qu'ils font périodiquement, toutes les secondes. Dans l'implantation actuelle, la période d'échantillonnage du démon maître (et donc aussi celle de diffusion des paramètres) est de 10 secondes. D'après les auteurs, les diffusions périodiques sont nécessaires puisque l'horloge de référence peut changer de fréquence à cause des changements de température et de voltage, ce qui cause de «lents changements» des valeurs des paramètres a et b. Les démons esclaves vont donc même rester en phase avec le bruit sur l'horloge du processeur de référence, ce qui permet d'obtenir une précision de $5\mu s$, inférieure à l'amplitude de ce bruit qui est de $17\mu s$ (cf. section 3.3.1).

La diffusion des constantes est implantée sur le protocole UDP/IP (*User Datagram Protocol*) et est acheminée par le bus Ethernet du IBM-SP2. *A priori*, elle ne devrait donc pas interférer avec les communications des applications sur le *Switch*. Cependant, le réveil périodique des processus lourds, que sont les démons, risque d'engendrer un bruit non négligeable sur les différents noeuds de la machine, ce qui peut rendre la mesure de performances d'applications plus difficile. La variance de ces mesures (le temps d'exécution, par exemple) en sera augmentée et l'on devra accroître leur nombre afin d'obtenir des résultats représentatifs, i.e. des intervalles de confiance acceptables. C'est là le désavantage majeur des méthodes système basées sur la resynchronisation périodique, désavantage que nous avons déjà évoqué dans l'introduction à ce chapitre (section 3.1). Ceci dit, la solution d'IBM n'est pas encore diffusée actuellement et on n'a pas encore eu l'occasion de mesurer le bruit qu'elle engendre. Il se peut que ce bruit soit infime, bien inférieur à celui de NTP, ce qui ferait de l'approche IBM une excellente solution pour le SP2.

Pour conclure nos remarques sur l'approche d'IBM, on peut se demander s'il est vraiment nécessaire que les horloges des noeuds esclaves reflètent les oscillations du bruit autour de la tendance linéaire de l'horloge du maître. En effet, si l'on se contentait de capter cette tendance linéaire, ce qui revient à considérer a et b comme des constantes, on obtiendrait une précision de $17\mu s$ au lieu de $5\mu s$, ce qui est encore largement suffisant pour garantir la datation cohérente, et on éliminerait les resynchronisations périodiques.

3.6 Résumé et conclusions

Ce chapitre a montré que les méthodes statistiques sont bien adaptées à la construction d'une référence globale de temps utilisée pour la mesure des

performances d'exécutions réparties. Contrairement aux algorithmes de type système distribué, basés sur des resynchronisations périodiques, les méthodes statistiques n'utilisent pas les ressources du système pendant l'exécution des applications observées. Elles n'interfèrent donc pas avec ces dernières et n'entravent pas le caractère représentatif des exécutions observées.

Après la présentation de deux méthodes statistiques sous la forme matricielle des moindres carrés ordinaires (OLS), nous décrivons le problème de la durée des phases d'échantillonnage nécessaires au calcul des estimateurs. Réduire la durée de ces phases et augmenter la précision des estimations sont des démarches contraires.

L'analyse détaillée d'échantillons d'horloges, issus des machines Meganode et IBM-SP2 du LMC, montre qu'à la tendance linéaire principale se superpose un bruit périodique qui est sensible aux changements de température. Si la durée des phases d'échantillonnage est suffisamment longue, la tendance linéaire principale peut être captée, malgré le bruit. L'amplitude du bruit, et donc aussi le résidu du modèle linéaire, est d'environ $50\mu s$ sur le Meganode et de $17\mu s$ sur le IBM-SP2. Ces valeurs sont inférieures à la latence des communications : le modèle linéaire est donc suffisant pour garantir le respect causal du temps global. La comparaison des estimations du modèle linéaire avec celles d'un modèle oscillatoire qui prend en compte la périodicité du bruit et dont les résidus vérifient les hypothèses de régression (homoscédasticité et non-corrélation) permet de conclure à la pertinence du modèle linéaire.

En pratique, on ne peut pas se permettre d'utiliser des phases d'échantillonnage arbitrairement longues, afin de capter la tendance linéaire principale. Cela infligerait des délais considérables au lancement de toute application qui a besoin du temps global. Nous présentons deux stratégies d'échantillonnage sous la contrainte de phases d'échantillonnage courtes. Pour une même longueur de phase d'échantillonnage, la stratégie par interpolation (*SBA*) calcule un temps global bien plus précis que la stratégie par extrapolation (*SB*). L'erreur d'estimation du temps global SBA est bornée par le double de l'amplitude du bruit. La datation cohérente est possible sur des périodes dont la durée est seulement limitée par le vieillissement des cristaux d'horloge. L'erreur sur le temps SB est non bornée : la mesure d'intervalles de temps (à bornes réparties) n'est pas cohérente, ce qui induit des incohérences causales de datation.

D'après notre expérience, une méthode statistique d'estimation d'une référence globale de temps physique, combinée avec la stratégie d'interpolation SBA, offre un accès suffisamment précis et confortable «au temps global» pour pouvoir rivaliser avec une solution matérielle. Implantée en logiciel, elle est de surcroît plus portable et extensible («scalabilité»). Nous pensons

qu'une horloge globale physique n'est indispensable que dans un système parallèle utilisé pour l'exécution d'applications devant répondre à des contraintes de temps réel très fines.

Tout au long de ce chapitre, la construction d'une référence globale de temps est discutée dans la perspective de datation d'événements pour le traçage d'exécutions parallèles. La trace d'événements est une technique puissante pour l'amélioration des performances et la compréhension de la dynamique des applications. Pour obtenir l'accord, la justesse et le respect causal de la datation des événements répartis, tout en assurant le caractère représentatif des exécutions, un temps global précis et non-intrusif est requis. Cependant, la datation à elle seule n'est pas suffisante. Une fois datés, les événements doivent être stockés dans les tampons mémoire des différents processeurs. Ces tampons doivent être vidés sur disque ou par le réseau. Cette gestion des événements, si elle est logicielle, perturbe inévitablement l'application analysée. L'exploitation des traces résultantes peut alors mener l'analyste des performances sur des «fausses pistes». Un peu dans le même esprit que dans ce chapitre, où l'on a modélisé l'écart et la dérive entre horloges, on peut essayer de construire un modèle qui tient compte de l'intrusion induite par la prise de traces. On peut alors dériver un algorithme de correction de traces, basé sur le modèle d'intrusion, qui permet d'enlever les perturbations logicielles des traces. La faisabilité d'une telle approche a été démontrée par Malony *et al.* [69]. Dans le prochain chapitre, nous montrons comment nous avons adapté et étendu les techniques de correction de traces de Malony au modèles de programmation par messages de type PVM ou MPI.

Chapitre 4

Modélisation et correction de l'effet de sonde

> «*De même que les nombres transcendants, infiniment plus nombreux que les nombres algébriques, ne pourront jamais être écrits dans un quelconque système numéral, de même la réalité ne sera jamais atteinte, mais seulement approchée.* »
> – Paul Couteau, *Le grand escalier*.

4.1 Introduction

La trace d'événements logiciels est une technique bien établie dans le domaine de l'évaluation des performances et du déverminage d'applications parallèles ou distribuées. Avec le développement et la croissance du calcul distribué et parallèle, la trace d'événements logiciels est devenue la technique dominante pour l'évaluation des performances de ces systèmes [68]. En général, un système de traçage doit aborder 4 problèmes :

1. l'enregistrement des événements,
2. l'estampillage (logique ou/et physique),
3. la gestion des tampons de trace,
4. l'évacuation des tampons de trace par les sous-systèmes d'entrée/sortie.

L'enregistrement d'un événement implique des mécanismes pour produire un événement dans la trace, ce qui inclut l'instrumentation logicielle pour générer l'événement, ainsi que le stockage dans un tampon au moment de

l'exécution. La taille des tampons d'événements doit être adaptée à la bande passante du système de sortie.

Afin de réduire le surcoût induit par la prise logicielle de traces, dû principalement à l'exécution d'instructions supplémentaires, plusieurs chercheurs proposent des extensions matérielles dédiées au traçage. Ces extensions comprennent généralement une horloge globale matérielle très précise (100ns pour [51]), ainsi qu'un système de sortie très efficace pour l'évacuation des informations de trace (une bande passante de 120Mo/s pour [51]). Les informations correspondant aux événements étant, quant à elles, construites de façon logicielle par des instructions insérées dans le code de l'application de l'utilisateur (*instrumentation*), les traceurs utilisant des extensions matérielles sont qualifiés d'*hybrides*, par opposition aux traceurs purement *logiciels*. La sémantique des traces produites par les traceurs logiciels et hybrides est très proche de celle du code de l'application instrumentée – l'utilisateur peut ainsi localiser la source d'un problème de performance.

Notons qu'il existe également des systèmes de trace purement *matériels*. Bien que non-intrusifs, les «signaux» captés par ces systèmes sont difficiles, sinon impossibles, à corréler aux applications (écrites dans un langage de haut niveau) : on parle alors de «gradient sémantique» (*semantic gap*, en anglais) entre l'information obtenue (trace) et l'objet analysé (application).

L'utilisation d'un traceur logiciel ou hybride induit nécessairement une *perturbation de l'application instrumentée*. Cette altération, qui peut affecter les performances aussi bien que le comportement logique de l'application, est également appelée *effet de sonde* [27]. Bien qu'elle permette de réduire l'amplitude des perturbations, l'utilisation d'un traceur hybride n'élimine pas ce problème. Ainsi, les auteurs du traceur hybride ZM4 recommandent de «réduire intelligemment le nombre d'événements tracés pour n'en garder que l'essentiel» [51]. L'utilisateur d'un traceur, qu'il soit logiciel ou hybride, doit donc établir un équilibre délicat entre le volume d'informations qu'il souhaite obtenir et la précision de ces informations. Par analogie au principe d'incertitude de Heisenberg, célèbre en physique des particules, Malony propose *le principe d'incertitude de l'instrumentation* [68]:

- l'instrumentation perturbe l'état du système,
- les phénomènes d'exécution et l'instrumentation sont couplés logiquement,
- volume et précision sont antithétiques.

L'effet de l'instrumentation sur les performances et le comportement des programmes parallèles étant, en général, assez mal compris, la mesure détaillée

des performances est souvent rejetée par peur d'obtenir des données altérées. Dans ce cas, une quantité réduite d'informations plus précises est préférée, quitte à devoir inférer le comportement du système observé à partir d'un volume insuffisant de données [68]. Dans d'autres cas, une attitude orthogonale, moins prudente, est adoptée : constatant que les décisions déduites des traces d'exécutions d'applications instrumentées permettent effectivement d'améliorer les performances d'exécutions non-instrumentées, on peut en conclure que le surcoût de la prise de traces est suffisamment petit pour ne pas changer fondamentalement la dynamique des exécutions. Tel est la remarque faite à propos de la bibliothèque de communication tracée PICL [34]. Toujours est-il qu'on n'est pas certain que la décision d'amélioration des performances déduite d'une trace altérée soit la meilleure. Par ailleurs, est-ce que la même attitude reste valide dans le cas où l'on souhaite augmenter le volume des traces, par exemple, en traçant les appels de procédures en plus des seules primitives de communication ?

Comme d'autres chercheurs [70, 69, 57, 61, 28], nous pensons que les perturbations induites par la prise de traces peuvent être *modélisées* et *corrigées* par un traitement post-mortem des traces. Une telle trace corrigée est une approximation de la dynamique non-instrumentée de l'application et reflète donc les performances effectives de cette dernière. Ainsi, trois objets sont impliqués dans le processus de correction de traces :

1. la trace T, reflétant une dynamique d'exécution de l'application potentiellement perturbée ;
2. la trace idéale T_0, qu'on obtiendrait par une instrumentation parfaite, non intrusive ;
3. la trace approchée T_a, obtenue par l'application d'un modèle de correction des perturbations à la trace T.

En général, en l'absence d'un traceur matériel non intrusif, le temps total d'exécution est le seul indice de performance accessible d'une exécution T_0. La qualité de l'approximation T_a par rapport à T_0 peut alors être évaluée en comparant les temps d'exécution correspondants. La disponibilité d'un outil de correction d'intrusions permet non seulement à l'analyste de performances d'augmenter le nombre de points de traces, mais peut aussi changer la philosophie de développement d'outils de trace. Ainsi, le concepteur de l'outil peut envisager d'investir des cycles de processeur dans la réduction du volume des informations de trace, en utilisant, par exemple, une technique de compression à l'exécution des événements. Du moment que le surcoût correspondant est mesurable, il peut être pris en compte par le processus de correction des perturbations.

La contribution fondamentale à la modélisation et à la correction des perturbations est le travail de Malony [68] et de Malony-Reed [70, 69]. Ces auteurs proposent un modèle d'analyse des perturbations pour des applications parallèles de type «*fork/join*» où les flots d'exécution concurrents sont considérés comme indépendants. Ce modèle, dit *orienté temps* (*time based*, en anglais), prend en compte l'accumulation des perturbations le long du chemin critique ainsi qu'un éventuel changement de chemin critique, suite à l'instrumentation. Une extension de ce modèle, dit *orienté événement* (*event based*, en anglais), tolère la synchronisation entre flots d'exécution concurrents par barrières, sémaphores et séquenceurs/compteurs. Les modèles de Malony et Reed sont destinés à une machine parallèle à mémoire partagée (comme le Alliant FX/80 qu'ils utilisent pour valider leurs modèles). Dans un travail plus récent, Sarukkai et Malony étendent encore le domaine de l'analyse des perturbations en proposant un modèle de correction pour les communications par messages dans les applications parallèles de type SPMD tournant sur des machines à mémoire distribuée (le Intel iPSC/860 est utilisé pour valider ce modèle).

Alors que Malony *et al.* proposent des modèles adaptés à des schémas de synchronisation particuliers, Gannon, Lumpp *et al.* introduisent une méthode de correction de perturbations (*perturbation tracking*, en anglais) pour des systèmes dynamiques modélisés par des réseaux de Pétri temporisés (RPT) [61, 62, 28]. Les auteurs montrent, sur des exemples simples, comment leur modèle s'applique à la correction des intrusions induites par la prise de traces d'exécutions parallèles sur une classe très large de systèmes (mémoire partagée ou distribuée). En pratique, la méthode se heurte au problème de la construction d'un réseau de Pétri temporisé à partir du code source d'une application parallèle. Pour une application communiquant par messages, par exemple, il faut détecter les correspondances possibles entre primitives d'émission et de réception [62]. A notre connaissance, la méthode n'a pas encore pu être validée sur des programmes réels.

Ce chapitre introduit le lecteur aux modèles de correction des perturbations des traces d'exécution parallèles sur des machines à mémoire distribuée communiquant par messages. Après avoir détaillé les notations et hypothèses de base en section 4.2, nous présentons, en section 4.3, le modèle de correction pour des traces d'exécution de flots séquentiels indépendants. Il s'agit là, essentiellement, d'une reformulation du modèle *orienté temps* de Malony [68]. Ensuite, en section 4.4, nous montrons comment nous avons adapté le modèle de perturbation de la barrière de Malony au cas d'un ensemble de flots communiquant par rendez-vous. En section 4.5, nous présentons alors le cas, plus complexe, où ces flots communiquent par échanges de messages en utilisant des primitives d'émission asynchrones bloquantes et de réception

synchrones bloquantes et nous montrons que, pour ce type de communications, le modèle de correction doit avoir recours à la modélisation des temps de communication. A ce propos, nous présentons l'approche de Sarukkai et Malony (sous-section 4.5.4). Nous détaillons alors notre propre approche, qui, par rapport à celle de Sarukkai et de Malony, permet de réduire considérablement le nombre de modélisations des temps de communication, ceci dans le but de garder un maximum d'informations de la trace source (sous-section 4.5.5). La section 4.6 discute de la correction des traces d'applications non-déterministes, dont non seulement les performances, mais aussi le comportement logique peuvent être affectés par l'effet de sonde. Avant de conclure, en section 4.8, nous présentons nos résultats expérimentaux en section 4.7.

4.2 Notations et hypothèses

4.2.1 La trace

La sortie d'un outil d'instrumentation est un fichier de *trace* T qui consiste en une série d'événements estampillés e, triés dans l'ordre chronologique, et qui décrit la dynamique d'un ensemble de flots d'exécution parallèles d'une application communiquant par messages. Nous supposons que chaque flot est exécuté sur un processeur différent : il y a correspondance bi-univoque entre l'ensemble des flots et l'ensemble des processeurs (*monoprogrammation*).

Nous considérons seulement des événements de haut niveau, du même niveau d'abstraction que le code source de l'application. Un *événement* est le début (ou la fin) d'une action qui est exécutée par un flot et qui change l'état de ce flot (variables, tampons d'entrées/sorties). Nous distinguons deux types d'événements :

1. les événements locaux,
2. les événements de synchronisation.

Les premiers sont relatifs aux actions impliquant exclusivement le flot qui a donné lieu à l'événement en question. Tels sont, par exemple, les événements d'entrée et de sortie de procédures. Les deuxièmes sont relatifs aux actions induisant une dépendance causale entre deux flots. Tels sont, par exemple, les événements d'émission et de réception de messages. Dans notre analyse, nous ne considérons que les communications de type rendez-vous (synchrones), ainsi que celles offertes par les bibliothèques de communication PVM ou MPI où l'émission est asynchrone. Nous supposons que l'outil de trace enregistre la totalité des événements de synchronisation avec les identificateurs des flots correspondants ; en plus des simples caractéristiques

temporelles de l'exécution, la trace contient donc toutes les informations pour reconstruire l'ordre partiel causal sur l'ensemble des événements.

A tout événement e, on associe une liste d'*attributs*, qui contient des informations spécifiques à l'action correspondante. Cette liste comprend au moins l'identification du flot générateur de l'événement, la date physique d'occurrence $t(e)$, ainsi que le type de l'événement $type(e)$. $t(e)$ est une date exprimée dans un référentiel global de temps physique, commun à tous les flots de contrôle. La granularité du temps global, traitée en détail dans le chapitre 3, doit être suffisamment fine pour garantir une datation des événements qui est causalement cohérente. Les événements de synchronisation contiennent, en dehors de ces trois attributs de base, l'identificateur explicite du flot partenaire (émetteur ou récepteur)[1], ainsi que le temps d'exécution de l'action de synchronisation :

- pour l'émission asynchrone, le délai de préparation du tampon d'émission (nous noterons $t(S)$ la date du début de l'émission) ;
- pour l'émission synchrone, le délai de blocage en attente du récepteur (nous noterons $t(SS)$ et $t(ES)$ les dates de début et de fin d'exécution de la primitive[2]) ;
- pour la réception (synchrone), le délai de blocage en attente du message (nous noterons $t(SR)$ et $t(ER)$ les dates de début et de fin d'exécution de la primitive[3]).

Si A est une action qui a une durée mesurable, comme, par exemple, une réception synchrone, le temps d'exécution de A peut faire partie des attributs de A, ou deux événements peuvent être enregistrés, l'un au début de A, l'autre à la fin. La technique utilisée dépend de l'outil de trace. Dans ce chapitre, nous supposons que les délais de blocage sur des actions de communication font partie des attributs des événements correspondants : cette technique est plus efficace que celle qui utilise deux événements, car elle ne nécessite qu'un seul stockage d'entête (attributs de base).

L'algorithme de correction des perturbations va affecter à chaque *date mesurée* $t(e)$ une estimation $t_a(e)$, appelée *date approchée*, de la *date effective* $t_0(e)$ à laquelle l'événement aurait eu lieu si l'application sous-jacente

1. Dans le cas d'une *réception anonyme*, acceptant un message en provenance de tout émetteur potentiel, nous admettons que l'outil de trace fournit l'identité de l'émetteur effectif dont le message est reçu au moment de l'exécution instrumentée. La section 4.6 discutera plus en détail du non-déterminisme des applications.

2. SS est pour *Start of Send*, ES est pour *End of Send*.

3. SR est pour *Start of Receive*, ER est pour *End of Receive*.

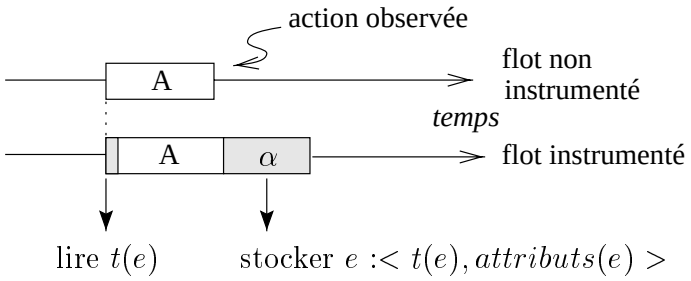


FIG. 4.1 – *Le modèle d'une perturbation directe*

n'était pas instrumentée. L'algorithme doit détecter les événements de synchronisation afin de prendre en compte la propagation de l'accumulation des perturbations locales (coût de génération d'événements locaux) sur le flot partenaire, par le biais de l'opération de synchronisation sous-jacente. Les types des événements locaux ne sont pas pertinents pour la correction des perturbations : les perturbations locales induites par ces événements sont simplement accumulées, quel que soit leur type.

4.2.2 Les perturbations directes et indirectes

Une *perturbation directe* α est le coût en temps pour enregistrer un événement. Ce coût est localisé autour du point d'instrumentation et résulte directement de l'exécution des instructions supplémentaires d'instrumentation. Il est donc *mesurable* au niveau applicatif. Afin de faciliter notre exposé, nous supposons que le coût α induit par une perturbation directe est le même pour chaque événement. En pratique, α dépendra de la taille de l'événement (longueur de sa liste d'attributs) et de la bande passante accessible sur le système de sortie (mémoire, disque ou réseau, selon l'outil de trace utilisé). La figure 4.1 montre notre modèle de perturbation directe. Pour une action observée A, la perturbation directe de l'enregistrement de l'événement correspondant comprend :

- le temps de lecture de l'horloge physique locale pour obtenir la date $t(e)$ du début de l'action ;
- le temps de stockage des attributs de l'événement après la fin de l'action A (exécution d'instructions de stockage en mémoire, exécution d'une primitive de sortie sur disque ou sur réseau si besoin est).

A elle seule, la perturbation directe ne prend pas en compte toutes les perturbations potentielles induites par l'enregistrement d'un événement. Un tel

enregistrement peut en effet provoquer un ensemble d'effets de bord, appelés *perturbations indirectes*, localisés en dehors des points d'instrumentation et qui peuvent affecter l'exécution de sections de code non-instrumentées. On peut évoquer les situations suivantes :

- *Inhibition des optimisations à la compilation* : la présence d'instructions supplémentaires d'instrumentation peut inhiber les optimisations à la compilation et changer l'allocation des registres. Le code résultant est donc moins efficace. Malony montre que l'effet d'inhibition des optimisations est plus important pour l'instrumentation d'un code Fortran que pour un code C, un compilateur C effectuant moins d'optimisations, en général [68].

- *Interférence au niveau des accès aux caches* : la génération d'un événement peut occuper des lignes de cache et vider des granules au dépend de l'application. Quand celle-ci reprend le contrôle, il peut y avoir une série de défauts de cache dont l'effet cumulé est comparable à un changement de contexte [68]. Les mêmes remarques sont valables pour la mémoire virtuelle – l'utilisation de tampons d'événements peut provoquer le vidage d'une page applicative.

- *Augmentation du nombre de changements de contexte* : elle résulte de l'augmentation du temps d'exécution de l'application suite à l'accumulation des perturbations directes.

- *Changement des performances apparentes des communications* : l'instrumentation d'une suite de primitives de communication sur un même flot répartit l'exécution de ces primitives sur un intervalle de temps plus large. La fréquence des accès au système de communication étant ainsi diminuée, la bande passante accessible sur ce système peut paraître plus élevée au vu de l'application. Notons qu'il s'agit là d'un effet bénéfique de l'instrumentation. L'effet contraire est également envisageable – c'est le cas lorsque plusieurs actions de communication de flots différents se retrouvent rapprochées dans le temps suite à l'instrumentation, provoquant ainsi une contention au niveau du système de communication qui n'aurait pas eu lieu en l'absence d'instrumentation.

- *Réordonnancement des événements* : l'instrumentation peut changer l'ordre d'arrivée des messages au niveau d'un flot récepteur. Ce changement d'ordre peut altérer le comportement logique du flot récepteur et donc ses performances. Dans un programme parallèle communiquant par messages, une réception anonyme (recevant un message de

tout émetteur potentiel) peut être source d'indéterminisme. Le non-déterminisme est discuté plus en détail dans la section 4.6.

Il est concevable de mesurer une partie des perturbations indirectes. Par exemple, les processeurs DEC 21164 de Digital et Power2 d'IBM intègrent des mécanismes matériels de mesure de performances qui permettent aux utilisateurs d'analyser les interactions matérielles et logicielles sur des applications exécutées par ces processeurs [84]. Ces mécanismes comprennent des compteurs du nombre de défauts de cache et, dans le cas du Power2, des détails sur les performances des accès mémoire et des entrées/sorties. En principe, le modèle de correction des perturbations peut comptabiliser toute perturbation directe ou indirecte, dès que les délais qu'elle induit au niveau de l'application instrumentée est mesurable (durée du délai et date où il est infligé à l'application). En général, même si des dispositifs de mesures des perturbations indirectes sont accessibles, il est difficile, en raison du bas niveau des informations qu'ils fournissent, d'établir le lien entre cause et effet. En raison de cette difficulté, et dans un souci de généralité, *nous supposons que nous ne disposons que des seules informations sur les perturbations qui sont mesurables au niveau applicatif et que ces perturbations mesurables se limitent aux perturbations directes.* En section 4.7, nous allons voir expérimentalement que les modèles de correction, malgré le fait qu'ils soient basés entièrement sur des mesures du niveau applicatif, permettent d'enlever la plus grande partie des perturbations.

Le succès dans l'utilisation d'une méthode de correction des perturbations dépend donc directement du rapport entre perturbations directes et indirectes. *Dans une perspective d'application d'un algorithme de correction des perturbations, les développeurs d'outils de trace tenteront ainsi de réduire les perturbations indirectes au minimum, plutôt que de réduire les perturbations dans leur ensemble.* Nous revenons ici sur le fait, déjà évoqué en section 4.1, que la disponibilité d'outils de correction de perturbations change la philosophie d'implantation d'outils de trace. On évitera, par exemple, d'envoyer un événement par le réseau à une tâche collectrice dès que cet événement est enregistré. Une telle collecte d'événements *in vivo* consomme une partie non négligeable de la bande passante du médium de communication, pénalisant ainsi les communications de l'application instrumentée (perturbation indirecte) [63]. C'est pourtant ainsi que fonctionne le mécanisme de trace intégré dans la bibliothèque de communication PVM. C'est l'une des raisons qui nous ont conduits à développer Tape/PVM, un nouvel outil de trace pour PVM (cf. la troisième partie de la thèse).

4.3 Le modèle des flots séquentiels indépendants

4.3.1 Le modèle de correction des perturbations

Des traces de flots séquentiels indépendants ne contiennent que des événements locaux. L'occurrence de chacun de ces événements est retardé à cause des délais d'enregistrement des événements (perturbations directes) qui le précèdent dans le même flot. Pour estimer la date effective du k^{e} événement e_k sur un flot i, il suffit alors de retrancher la somme de tous ces délais de sa date mesurée :

$$t_a(e) = t(e) - acc^i, \quad \text{où} \quad acc^i = (k-1)\alpha \tag{4.1}$$

est l'accumulation des $k-1$ délais des enregistrements précédant e_k. Ici intervient l'hypothèse de monoprogrammation des flots. Dans le cas où plusieurs flots d'exécution sont multiprogrammés sur un même processeur, le modèle de correction doit être complété par un modèle de l'ordonnanceur des flots afin de prendre en compte l'exclusion mutuelle des flots vis-à-vis du processeur. Dans cette thèse nous limitons notre étude des modèles de correction de perturbations au cas d'un ensemble de flots d'exécution monoprogrammés.

Anticipant le cas où les flots sont synchronisés, nous définissons, pour chaque flot i un événement de base e_0^i par rapport auquel les perturbations sont accumulées et dont on connaît les dates mesurée et approchée :

$$tb^i = t(e_0^i) \text{ et } tb_a^i = t_a(e_0^i).$$

L'événement de base devra être refixé sur le flot i chaque fois que l'algorithme de correction aura traité une réception bloquante. On illustrera ceci à l'aide d'un exemple dans la sous-section suivante. Dans le cas d'un ensemble de flots indépendants, on peut prendre $tb^i = tb_a^i = 0$.

La figure 4.2 illustre l'équation de correction des perturbations 4.1 en y introduisant la notion de base de temps. $t(e) - tb^i$ est le délai mesuré entre e et e_0^i, tandis que $[t(e) - tb^i] - acc^i$ est le délai approché entre e et e_0^i si le flot i n'était pas instrumenté (sur la figure 4.2 acc^i vaut 3α au niveau de e). La date approchée de e, $t_a(e)$, est alors calculée en additionnant ce délai approché au temps de base approché tb_a^i. Nous spécifions cette opération sous forme de procédure :

Procedure *CorrEvt* (**in**: $t(e), tb^i, tb_a^i, acc^i$; **out**: $t_a(e)$)

L'algorithme de correction s'écrit alors comme suit :

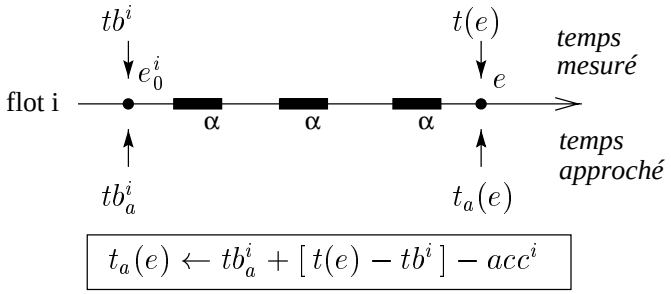
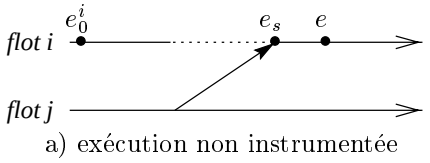
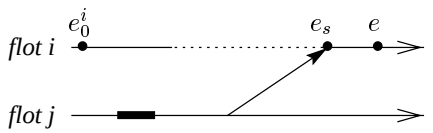


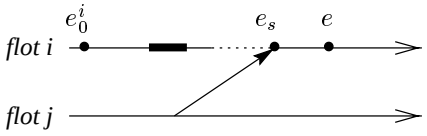
FIG. 4.2 – *Correction des perturbations sur un flot d'exécution séquentiel indépendant.*



a) exécution non instrumentée



b) perturbation du seul émetteur



c) perturbation du seul récepteur

FIG. 4.3 – *Synchronisation par réception bloquante : l'importance de la base de temps.*

$tb^i \leftarrow 0;\ tb_a^i \leftarrow 0;\ acc^i \leftarrow 0; (\forall i)$
tant que $T \neq \{\}$
  $e \leftarrow suiv(T);$
  $i \leftarrow flot(e);$
  $CorrEvt\ (\ t(e), tb^i, tb_a^i, acc^i, t_a(e)\ );$
  $acc^i \leftarrow acc^i + \alpha;$
continuer

où $suiv(T)$ extrait et retourne le prochain événement de la trace T, et où $flot(e)$ retourne l'identificateur du flot sur lequel l'événement e a eu lieu. Notons que la première ligne met les bases de temps à zéro et que ces mêmes bases sont utilisées dans tout l'algorithme.

4.3.2 Réflexions sur la base de temps

Pour un flot d'exécution indépendant, la perturbation au niveau d'un événement donné e est l'accumulation de toutes les perturbations directes entre l'événement de base e_0^i de ce flot et e. Ceci n'est plus valide si e est précédé d'une ou plusieurs actions de communication synchrones, auquel cas les perturbations au niveau de e dépendent également de celles du ou des flots partenaires. Prenons l'exemple d'un événement de réception, e_s, qui précède e sur un même flot d'exécution i (figure 4.3a), et considérons les deux cas suivants :

1. seul, le flot émetteur j est affecté de perturbations (figure 4.3b) ;

2. seul, le flot récepteur i est affecté de perturbations (figure 4.3c).

Dans le premier cas, les perturbations retardent l'émission du message, augmentant ainsi le délai d'attente (ligne pointillée) à la réception. Si l'on prend toujours e_0^i comme événement de base lors de l'estimation du temps effectif de e par l'équation de la figure 4.2, les perturbations au niveau de e sont considérées comme nulles et la valeur approchée $t_a(e)$ sera trop élevée. Dans le deuxième cas, les perturbations retardent la requête de réception, ce qui réduit, voire élimine, le délai d'attente à la réception. Si l'on prend toujours e_0^i comme événement de base, les perturbations localisées avant la requête de réception du flot i, entièrement ou partiellement absorbées par la diminution du délai d'attente, seront comptabilisées dans leur totalité au niveau de e, et la date approchée $t_a(e)$ sera inférieure à la valeur réelle $t_0(e)$.

Les modèles de correction des perturbations *orientés événement* des sections 4.4 et 4.5 prennent en compte les changements dans les délais de blocage des actions de synchronisation induits par l'instrumentation et fournissent des estimateurs des dates effectives (début et fin) des actions de synchronisation. Les événements locaux, suivant un événement de synchronisation corrigé, mettons e_s, peuvent alors être corrigés par la procédure *CorrEvt*, après avoir fixé e_s comme événement de base. Ainsi, dans le contexte des flots synchronisés, l'événement de base e_0^i est toujours le dernier des événements de synchronisation corrigés sur le flot i (de type **ER** ou **ES**). Tout l'art des modèles de correction d'intrusion est de dériver des estimateurs de la date effective des événements de base à partir des seules informations de la trace.

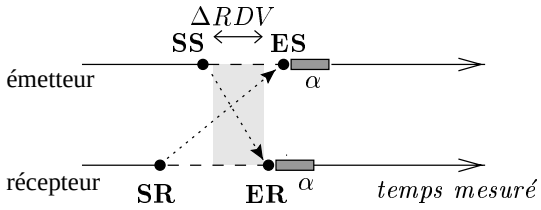


FIG. 4.4 – *Modèle de perturbation de l'échange de message par rendez-vous.*

4.4 Le modèle du rendez-vous

Dans une communication par rendez-vous les primitives de réception et d'émission sont synchrones. L'échange effectif du message n'a lieu que si les deux partenaires sont au rendez-vous. Ce type de communication est utilisé sur les Transputers. A cause de son caractère synchrone, un échange de message par rendez-vous peut être réalisé sans avoir recours à des tampons. Rappelons aussi que des programmes parallèles basés sur des communications par rendez-vous sont plus sensibles aux intrusions que ceux basés sur des communications à émission tamponnée, asynchrone (ce modèle est étudié en section 4.5). En effet, un rendez-vous crée une dépendance causale bi-directionnelle entre la paire de flots impliqués dans la synchronisation (*schéma «papillon»*). Ainsi, une perturbation du récepteur affecte l'émetteur, alors que pour une communication à émission asynchrone, l'émetteur n'est pas affecté par les perturbations du récepteur[4].

Le modèle de correction des perturbations pour le rendez-vous, que nous présentons ci-dessous, est une adaptation du modèle de la barrière de Malony [68]. La figure 4.4 montre les 4 événements pertinents pour le rendez-vous : **SS**, **ES**, **SR** et **ER**. Les lignes croisées en pointillés montrent le schéma papillon de dépendance causale entre ces événements. Les perturbations directes, dues à l'exécution des instructions d'enregistrement des événements, sont représentés par des rectangles ombrés étiquetés par α[5]. Le rectangle ombré entre les deux flots de contrôle représente le délai ΔRDV durant lequel le transfert des données du message est effectué.

La première étape de correction des intrusions est l'estimation du délai ΔRDV :

4. Ici, la combinaison des mots «effet», «perturbation» et «papillon» est purement fortuite et ne sous-entend en rien un quelconque chaos.

5. Rappelons, qu'au sens «enregistrement», il n'y a que deux événements dans la trace, un pour l'émission et un pour la réception, comprenant le délai de blocage parmi leurs attributs. Dans cette section, on distingue pourtant quatre événements qui doivent être compris dans le sens «instant dans le temps».

$$\Delta RDV = Min(t(ER), t(ES)) - Max(t(SR), t(SS)).$$

Nous supposons que le délai du transfert de message ΔRDV est le même dans une exécution instrumentée et non-instrumentée (absence de perturbation indirecte au niveau de la communication). Nous revenons sur la validité de cette hypothèse lors de la présentation des résultats expérimentaux en section 4.7.

Soient,

$$\begin{aligned} e_f = ER,\ e_l = ES \quad si \quad t(ER) < t(ES), \\ e_f = ES,\ e_l = ER \quad si \quad t(ER) \geq t(ES), \end{aligned}$$

le premier (e_f) et le dernier (e_l) des événements de sortie. Supposant ensuite que $t_a(SR)$ et $t_a(SS)$ ont déjà été calculés par une analyse antérieure, nous calculons $t_a(e_f)$ et $t_a(e_l)$ comme suit :

$$\begin{aligned} t_a(e_f) &= Max(t_a(SR), t_a(SS)) + \Delta RDV, \\ t_a(e_l) &= t_a(e_f) + t(e_l) - t(e_f). \end{aligned} \tag{4.2}$$

Notons que dans un échange par rendez-vous idéal, $t_a(ER)$ et $t_a(ES)$ sont égaux (tout comme $t(ER)$ et $t(ES)$). Cependant, en pratique, les dates de sortie du rendez-vous sont souvent légèrement différentes, suite aux limites de l'implantation du rendez-vous et à la précision limitée du temps global (cf. chapitre 3). Nous choisissons alors, comme dans le modèle de correction de la barrière de Malony, de placer les dates approchées de sortie dans le même ordre que leurs équivalents mesurés.

Le processus de correction de la trace consiste alors en un parcours séquentiel de la trace, appliquant le modèle de correction décrit ci-dessus à toutes les communications rencontrées. Après que les dates approchées $t_a(ER)$ et $t_a(ES)$ d'une communication donnée ont été calculées sur les flots i et j, respectivement, les bases de temps doivent être mises à jour comme suit :

$$\begin{aligned} tb^i \leftarrow t(ER);\ tb^i_a \leftarrow t_a(ER);\ acc^i \leftarrow \alpha, \\ tb^j \leftarrow t(ES);\ tb^j_a \leftarrow t_a(ES);\ acc^j \leftarrow \alpha, \end{aligned}$$

ce qui revient à prendre les événements de sortie du rendez-vous comme événements de base e^i_0 et e^j_0 (cf. aussi la section 4.3.2). Les événements locaux suivant **ER** et **ES** sur les flots i et j, sont ensuite corrigés par la procédure *CorrEvt*, jusqu'à ce que le prochain événement **SR** où **SS** est rencontré, annonçant par là la correction d'un autre rendez-vous.

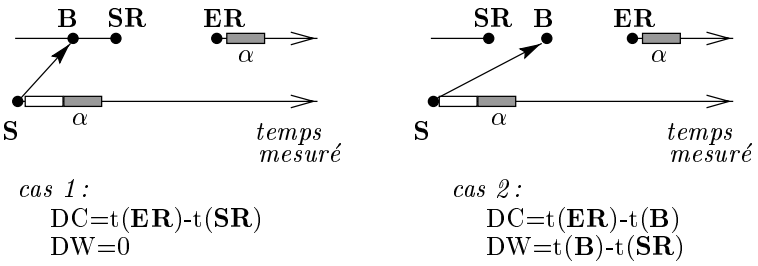


FIG. 4.5 – *Émission non bloquante : les deux schémas de communication possibles*

4.5 Le modèle de l'envoi asynchrone

4.5.1 Les deux schémas de communication

Contrairement au rendez-vous, un *envoi asynchrone* est indépendant de l'état d'avancement du récepteur et la primitive d'émission peut se terminer avant qu'une requête de réception correspondante n'ait été postée[6]. Ce type d'asynchronisme implique l'utilisation d'un système de tampons de messages. Le spectre des implantations possibles d'un tel système est très large : le message peut être tamponné au niveau de l'émetteur ou au niveau du destinataire, par exemple. En général, ces mécanismes ne sont pas visibles au niveau de l'interface de programmation de la bibliothèque de communication. Notre modèle de l'envoi asynchrone devra donc faire abstraction de ces mécanismes de bas niveau, tout en approchant au mieux leur effet sur le comportement de l'application en termes d'événements observables au niveau du code source.

Notre modèle distingue deux schémas de communication, selon que le message est accessible au niveau du destinataire avant que celui-ci ne poste sa requête de réception ou après. Pour faire cette distinction nous introduisons un événement, désigné par **B**, qui correspond à l'instant précis d'*accessibilité du message* au niveau du destinataire. La sémantique de cette notion d'accessibilité dépend du système de tamponnage de messages sous-jacent. Plus précisément, considérons les deux choix de tamponnage suivants :

1. *le message est tamponné sur le destinataire* : **B** correspond à la fin de

6. La primitive d'envoi asynchrone n'est terminée qu'à partir du moment où l'enveloppe et les données du message ont été recopiées vers la couche de communication, afin que l'émetteur puisse réutiliser le tampon d'émission. C'est pourquoi on parle de primitive d'émission asynchrone *bloquante* [72]. On ne traite pas les primitives de communication non bloquantes.

l'action de constitution du tampon sur le récepteur ;

2. *le message est tamponné sur l'émetteur* : **B** correspond à l'arrivée au niveau du destinataire du message «*request-to-send*» signalant que l'émetteur est prêt à expédier son message à toute action de réception intéressée exécutée par le flot destinataire.

La figure 4.5 montre les deux schémas de communication rencontrés dans les traces d'exécution. Les perturbations directes sont étiquetées par α et marquées par des rectangles ombrés. Côté émetteur, la perturbation est précédée par le coût de copie du message de l'espace utilisateur dans l'espace système. Ce délai de blocage, représenté par un rectangle blanc, dépend de la latence de la communication, qui, elle, est fonction de l'efficacité de l'implantation du système de tampons. Côté récepteur, les informations sur **B**,**SR**, et **ER** sont stockées dans un seul enregistrement (événement) une fois que la primitive de réception est terminée[7].

Dans le *cas 1* de la figure 4.5, le temps d'exécution de la primitive de réception consiste seulement dans le délai DC de transfert des données du message dans l'espace utilisateur. Ce transfert est une simple copie locale de tampon dans le cas où les messages sont tamponnés sur le récepteur, et un téléchargement depuis l'émetteur (*remote-load*, en anglais) dans le cas où les messages sont tamponnés à l'émission.

Dans le *cas 2* de la figure 4.5, le temps de transfert des données est précédé par une période DW durant laquelle le flot récepteur est inactif en attente de l'arrivée du message.

4.5.2 Un premier modèle de correction des perturbations

Ce premier modèle suppose que la date $t(B)$ de l'événement «arrivée de message» **B** est mesurable, ce qui implique généralement une instrumentation au niveau de l'exécutif de la bibliothèque de communication. Le premier pas dans la correction des perturbations consiste à calculer la date approchée $t_a(B)$ de l'événement **B**. Supposons que $t_a(S)$ ait déjà été calculé précédemment. $t_a(B)$ est alors calculé en additionnant le temps de communication mesuré à $t_a(S)$, ou, de façon équivalente, en propageant l'accumulation des perturbations en **S** par la communication :

7. On parlera de trois événements au sens «instant dans le temps» alors qu'il n'y a qu'un seul événement au sens «enregistrement».

$$t_a(B) = \underbrace{t(B) - t(S)}_{\textit{temps de comm. mesuré}} + t_a(S). \tag{4.3}$$

Notons que cette estimation n'est correcte que si l'on suppose que l'instrumentation n'a pas d'effet sur le temps de la communication (un tel effet est considéré comme perturbation indirecte, cf. section 4.2.2).

Le deuxième pas consiste à calculer $t_a(ER)$. Nous supposons que $t_a(SR)$ a déjà été calculé par une analyse précédente. $t_a(ER)$ est alors calculé en distinguant les deux schémas de communication de la figure 4.5 mais, cette fois, sur l'échelle du temps approché. Le tableau suivant montre comment $t_a(ER)$ est calculé (on note que DC est fonction du schéma de communication sur l'échelle mesuré) :

$t_a(B) \leq t_a(SR)$	$t_a(SR) < t_a(B)$
$t_a(ER) = t_a(SR) + DC$	$t_a(ER) = t_a(B) + DC$
$DC_a = DC$ $DW_a = 0$	$DC_a = DC$ $DW_a = t_a(B) - t_a(SR)$

Tout comme le temps de la communication, nous supposons que le délai DC de transfert des données, que ce soit une copie de tampon ou un téléchargement, est le même dans l'exécution instrumentée et non-instrumentée.

Le processus de correction consiste dans le parcours séquentiel du fichier de trace, effectuant ces calculs pour toutes les communications rencontrées. Après que la date $t_a(ER)$ ait été calculée sur le flot i, les dates de base et l'accumulateur des perturbations directes doivent être mis à jour sur le flot i :

$$tb^i \leftarrow t(ER);\ tb^i_a \leftarrow t_a(ER);\ acc^i \leftarrow \alpha.$$

Ainsi, **ER** devient le nouvel événement de base e^i_0 par rapport auquel vont être accumulées les perturbations directes suivant la réception sur le flot i. Les dates des événements locaux ainsi que les événements **S** suivant **ER** sur le flot i sont alors corrigées par la procédure *CorrEvt* jusqu'au prochain événement **SR** qui annonce la correction d'une autre synchronisation.

4.5.3 Le problème de l'événement «arrivée de message»

Le modèle de correction présenté en section 4.5.2 suppose connue la date de l'événement **B** qui représente le premier instant à partir duquel les données du message peuvent être transférées dans l'espace d'adressage du flot

récepteur. Malheureusement, les bibliothèques de communication par messages fournissent seulement des informations partielles sur l'événement **B**. En PVM ou MPI par exemple, il n'y a aucun moyen, au niveau applicatif de mesurer la date d'occurrence $t(B)$ de **B**. En général, il existe une fonction sonde non bloquante, qui permet à l'appelant de savoir si un message donné, spécifié par sa source et son type (*tag*, en anglais), est arrivé ou pas. Cette fonction s'appelle `pvm_probe` en PVM [29] et `mpi_iprobe` en MPI [72]. Si cette fonction nous permet de distinguer les deux schémas de communication de la figure 4.5, elle ne permet pas, par contre, de calculer $t(B)$. Il n'est donc pas possible de calculer $t_a(B)$ (équation 4.3) et de savoir quel est le schéma de communication sur l'axe du temps approché.

Une solution possible au problème de la mesure de $t(B)$ est d'*évaluer le temps de communication par un modèle*. Différents modèles pour les temps de communication ont été proposés dans la littérature, dont le plus simple est le modèle linéaire qui suppose que le temps d'une communication entre processeurs voisins est linéaire en fonction de la taille du message [89]. Ce modèle peut être affiné pour tenir compte des communications avec routage (commutation de messages ou de paquets). Lors du processus de correction des perturbations, la modélisation des temps de communication permet de calculer $t_a(B)$ par la formule 4.3 et, *a fortiori*, d'en déduire $t_a(ER)$ (cf. le tableau de la section 4.5.2). Deux stratégies peuvent être envisagées pour la modélisation :

1. *modélisation systématique du temps de communication* : pour chaque communication détectée dans la trace, le modèle de communication est appliqué pour le calcul de $t_a(B)$;

2. *modélisation du temps de communication qu'en cas de nécessité* : le modèle de communication n'est appliqué qu'aux communications pour lesquelles le temps de communication n'est pas déductible de la trace.

La première stratégie est celle utilisée par les méthodes de *prédiction de performances*. L'utilisation d'un modèle de communication pour une architecture de communication autre que celle de la machine sur laquelle la trace a été enregistrée, permet même de prédire les performances de l'exécution sur ce type d'architecture. Elle a le désavantage de ne pas prendre en compte tous les indices de performance déductibles de la trace. Dans le *cas 2* de la figure 4.5, par exemple, il est possible d'estimer $t(B)$ par $t(ER) - \widehat{DC}$, où $\widehat{DC}$ est une estimation du temps de transfert des données du message de l'espace système dans l'espace du flot récepteur[8]. Lors de la correction des

8. L'estimation de DC peut être calculée à partir de schémas de communication du type *cas 1* de la figure 4.5.

perturbations on essaiera de garder le maximum d'informations de la trace et on utilisera donc la deuxième stratégie. La question qui se pose alors est la suivante : comment calculer $t_a(ER)$, sans connaître $t(B)$, ayant recours à la modélisation des temps de communications qu'en cas de nécessité ?

Une première réponse à cette question est proposée par Malony et Sarukkai dans [83]. On présente leur approche en section 4.5.4. En section 4.5.5, on propose alors notre propre réponse qui, par rapport à l'approche de Malony et Sarukkai, permet de diminuer sensiblement le nombre de modélisations des temps de communication.

4.5.4 L'approche de Sarukkai-Malony

Le modèle de correction proposé par Sarukkai-Malony [83][9] utilise le résultat suivant :

$$t_a(ER) = Max(t_a(SR), t_a(S) + \Delta_{com}) + DC \tag{4.4}$$

où Δ_{com} est le temps de la communication, DC est le temps de copie des données du message de l'espace système dans l'espace utilisateur. DC est supposé constant, de valeur connue. On suppose que les dates approchées $t_a(SR)$ et $t_a(S)$ ont déjà été calculées par une analyse antérieure. L'équation 4.4 découle directement du tableau donné en section 4.5.2 et de l'équation 4.3.

Le calcul du temps de communication est basé uniquement sur la position relative de $t(SR)$ et de $t(S)$ et ne nécessite aucune information sur l'événement **B**. Le cas $t(SR) < t(S)$ implique qu'on est dans le *cas 2* sur l'échelle de temps mesuré (cf. figure 4.5) – le temps de la communication peut alors être calculé directement à partir de la trace :

$$\widehat{\Delta}_{com} = t(ER) - t(S) - DC.$$

Dans le cas où $t(SR) > t(S)$, il n'est pas possible de déduire le temps de communication de la trace et l'on doit recourir à un modèle des temps de communication.

Le tableau 4.1 donne le nombre d'évaluations à partir de modèle requises pour la correction des traces de quelques applications numériques (PVM3.3) exécutées sur 4 processeurs d'un IBM-SP2. Ces mesures ont été obtenues par comptage du nombre de communications de *cas 1* dans des traces d'exécution générées par le traceur Tape/PVM. L'outil de correction des perturbations

9. Notre description du modèle de Sarukkai-Malony utilise les mêmes notations qu'en section 4.5.2, différentes de celles du papier original.

	nb. comm.	éval. modèle	éval. modèle %
FFT-2D	36	34	94
NAS-CG	7124	2493	35
JACOBI	306	67	22

TAB. 4.1 – *Algorithme de Sarukkai-Malony: nombre de recours au modèle de communication.*

de Tape/PVM effectue ce comptage automatiquement (cf. chapitre 6). L'application FFT-2D calcule une transformée de Fourier bi-dimensionnelle ; elle est décrite dans [15]. Les applications NAS-CG et JACOBI seront discutées en section 4.7.

4.5.5 Notre approche

Alors que l'approche de Sarukkai-Malony est basée sur la position temporelle relative des événements **S** et **SR** (de flots différents), notre approche est basée sur la position relative des événements **B** et **SR** (sur un même flot). Comme on l'a déjà indiqué dans la sous-section 4.5.3, cette position peut être déterminée avec une fonction de sonde comme `pvm_probe` ou `mpi_iprobe`. Dans la trace, l'événement de réception comportera un attribut supplémentaire de type booléen indiquant la présence du message au moment de l'appel de la primitive de réception (cet attribut contient donc la valeur du prédicat $t(B) \leq t(SR)$).

Les résultats suivants sont des conditions suffisantes pour que le schéma de communication approché soit le même que le schéma mesuré.

Proposition 1: Si $t(B) \leq t(SR)$ (*cas-1m*), et que les perturbations en **SR** sont inférieures à celles en **S** alors $t_a(B) \leq t_a(SR)$ (*cas-1a*).

Proposition 2: Si $t(SR) < t(B)$ (*cas-2m*), et que les perturbations en **SR** sont supérieures à celles en **S** alors $t_a(SR) < t_a(B)$ (*cas-2a*).

Preuve : (nous montrons seulement la proposition 1 ; la proposition 2 se déduit par dualité)

Par hypothèse, nous avons

$$\underbrace{t(SR) - t_a(SR)}_{\text{perturb. en } \mathbf{SR}} - \underbrace{[t(S) - t_a(S)]}_{\text{perturb. en } \mathbf{S}} \leq 0, \quad t(B) \leq t(SR).$$

En minorant $t(SR)$ par $t(B)$ dans la première de ces inégalités et en réarran-

FIG. 4.6 – *Arbre de décision pour la correction du schéma de communication à émission asynchrone (méthode qui minimise le nombre de recours au modèle de communication).*

	nb. comm.	éval. modèle	éval. modèle %
FFT-2D	36	9	25
NAS-CG	7124	1415	20
JACOBI	306	33	11

TAB. 4.2 – *Nombre de recours au modèle de communication avec notre approche.*

geant les termes, il vient

$$t(B) - t(S) + t_a(S) - t_a(SR) \leq 0.$$

Utilisant l'équation 4.3 on en déduit

$$t_a(B) - t_a(SR) \leq 0 \quad \text{(qed)}.$$

Ainsi, dans le cas où l'une de ces conditions suffisantes est vérifiée, nous n'avons pas besoin de la valeur de $t(B)$ pour décider si l'on est dans le *cas 1* ou dans le *cas 2* sur l'échelle de temps *approchée*. L'arbre de décision de la figure 4.6 montre comment nous calculons $t_a(ER)$ en utilisant les propositions ci-dessus. La modélisation du temps de communication n'est requise que dans le *cas-1m*, mais pas nécessairement dans le cas *cas-1a*.

Le tableau 4.2 donne le nombre d'évaluations à partir de modèle requises avec notre méthode. La comparaison avec le tableau 4.1 montre que, par rapport à la méthode de Sarukkai-Malony, le nombre de d'évaluations est réduit de façon significative, ce qui permet de garder un maximum d'informations de la trace originale.

4.6 Comportement non-déterministe

Cette section commence par introduire, en sous-section 4.6.1, la notion de non-déterminisme de comportement et d'*approximation conservatrice*. La sous-section 4.6.2 montre, sur deux classes d'applications, quelle est la *qualité* que l'on peut espérer d'une telle approximation . La sous-section 4.6.3 montre comment le non-déterminisme peut être masqué en exploitant la propriété de commutativité d'un opérateur collectif de plus haut niveau que celui des communications point-à-point – on propose un algorithme de correction basé sur un modèle simple de cet opérateur. Finalement, en sous-section 4.6.4 nous discutons de l'apport des techniques de réexécution déterministe pour pallier au problème du changement de comportement induit par l'instrumentation.

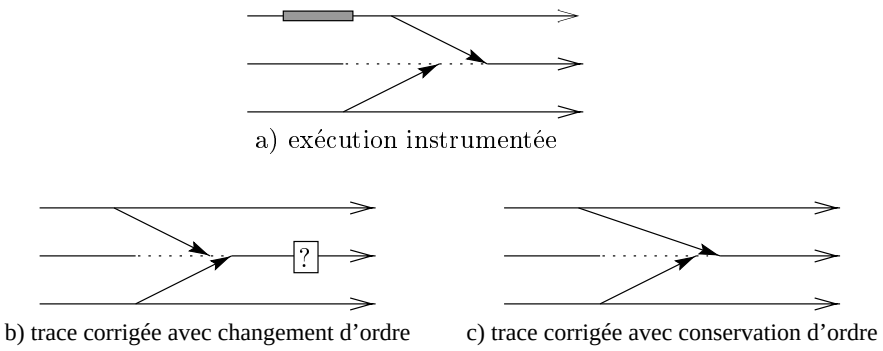


FIG. 4.7 – *Non-déterminisme : l'instrumentation peut induire un changement dans l'ordre de réception des messages.*

4.6.1 Non-déterminisme et approximation conservatrice

De nombreuses applications parallèles communiquant par messages utilisent des primitives de communication à caractère non-déterministe. En PVM, par exemple, l'appel `pvm_recv(-1,-1)` permet à un flot d'exécution de poster une requête de réception de message, quelque soit le type ou la provenance de celui-ci. De façon similaire, sur une machine à base de Transputers, la primitive `ALT` permet à un flot de recevoir un message quelque soit le canal par lequel il arrive. Sur une architecture multi-usagers, la charge variable des processeurs et du réseau d'interconnexion est à l'origine de l'instabilité de l'ordre dans lequel les messages sont lus par un flot récepteur utilisant des primitives de réception non-déterministes. Selon la logique de contrôle qu'il emploie, le comportement ultérieur du récepteur peut être très variable : nous parlerons de *changement de comportement logique* par opposition à la simple altération des performances. Plus formellement, nous disons que deux exécutions sont *équivalentes* si et seulement si elles donnent lieu à des ensembles identiques d'événements munis de la même relation d'ordre partiel causal (ordre de Lamport [54]). On peut montrer que si deux exécutions d'une application sont équivalentes, chaque flot d'exécution se comporte de la même façon dans les deux exécutions i.e. chaque flot exécute les mêmes instructions sur les mêmes données [59] (c'est une condition suffisante mais pas nécessaire). L'ensemble des exécutions possibles d'une application est ainsi partitionné en *classes d'équivalences* au sein desquelles toutes les exécutions ont le même comportement. Chaque exécution étant caractérisée par une trace d'exécution et une seule nous parlerons indistinctement d'exécution, de trace ou de trace d'exécution.

L'introduction de délais supplémentaires dans l'exécution d'une applica-

tion parallèle peut elle aussi provoquer un changement d'ordre à la réception des messages. Ce cas est illustré sur la figure 4.7a, où un flot d'exécution central reçoit des messages de deux flots voisins moyennant des primitives de réception non-déterministes. Nous supposons ici que le comportement ultérieur du récepteur dépende de l'ordre de réception. Le flot émetteur supérieur est fortement perturbé par l'instrumentation (représentée par un rectangle ombré sur la figure 4.7a). Il est clair qu'au cours d'une exécution non-instrumentée, l'ordre de réception des messages aurait été inversé, comme sur la figure 4.7b. Dans le cas d'un changement d'ordre induit par l'instrumentation, pourvu qu'il puisse le détecter, le processus de correction peut adopter les deux stratégies suivantes (indépendamment du modèle de communication) :

1. *Rétablir l'ordre de l'exécution non-instrumentée* (figure 4.7b) : vu que le changement d'ordre induit un changement du comportement du flot récepteur, la trace ne contient pas d'information sur le comportement non-instrumenté et le processus de correction doit être arrêté. La trace approchée T_a est une approximation de la dynamique non-instrumentée de l'application parallèle jusqu'au premier changement d'ordre. Cette stratégie doit se baser sur un *modèle de l'ordre de réception des messages* qui permet

   (a) d'*estimer l'ordre de l'exécution non-instrumentée*,

   (b) de *détecter un changement d'ordre*[10].

   L'exemple du modèle de correction d'un opérateur collectif présenté en sous section 4.6.3, montre l'emploi d'un modèle d'ordre pour la réception des messages.

2. *Conserver l'ordre de l'exécution instrumentée* (figure 4.7c) : cette stratégie, qui permet de corriger toute la trace, a le désavantage de calculer un comportement peu probable, voire impossible de l'application. L'ordre de lecture des messages étant conservé sur chaque flot d'exécution, la trace résultante T_a est munie de la même relation d'ordre partiel causal que la trace originale T. Nous appelons une telle exécution approchée T_a une *approximation conservatrice*. La stratégie de correction correspondante est dite *correction par approximation conservatrice*[11].

10. Pour détecter un changement d'ordre il est nécessaire (mais pas suffisant) que la trace d'une primitive de réception contienne le type et l'émetteur du message effectivement reçu ainsi que le masque de sélection pour le type et l'émetteur.

11. C'est la traduction que nous proposons pour l'expression *conservative approximation* utilisée par Malony [68].

L'*approximation conservatrice* reflète une exécution de l'*application inférée* A_I, dérivée de l'application analysée A et de la trace T. A_I est identique à A, sauf que toute primitive de réception anonyme de A est remplacée par une réception explicite du même message (source, type) que dans T[12]. En d'autres termes, A_I est l'application génératrice de toutes les exécutions de la classe d'équivalence à laquelle appartient l'exécution instrumentée qui a produit T – la stratégie de correction par approximation conservatrice représente la «meilleure» approximation de T_0 dans la classe d'équivalence de T.

Dans la pratique, l'algorithme de correction peut également utiliser une combinaison des deux stratégies. On pourrait utiliser la première stratégie en s'appuyant, en cas de conflit à la réception, sur la décision de l'utilisateur afin de savoir si l'ordre de réception conditionne le comportement ultérieur du récepteur et si la correction peut continuer, ce qui permettrait à l'algorithme de correction de transgresser la classe d'équivalence de l'exécution instrumentée. Ainsi, dans [83], Sarukkai et Malony supposent (implicitement) que l'ordre de lecture des messages sur les flots récepteurs ne conditionne pas le comportement de ces flots, ce qui leur permet de changer librement cet ordre lors de la correction : en dehors de l'approximation conservatrice ils proposent une stratégie qui lit les messages dans l'ordre LIFO (utilisation d'une pile d'émissions) et FIFO (utilisation d'une file d'émissions).

On pourrait également utiliser la deuxième stratégie, avertissant l'utilisateur à chaque fois qu'une conservation d'ordre est imposée, signalant par là que la correction a atteint la frontière de la classe d'équivalence de l'exécution instrumentée. Nous nous rapprochons ici des méthodes de détection de conditions de conflit (*race conditions*, en anglais) utilisées par les techniques de déverminage de programmes parallèles [74] et l'on tend quelque peu à s'éloigner de notre objectif initial de correction de l'effet de sonde au sens évaluation de performances.

Dans cette thèse, on s'intéresse surtout à la stratégie de l'approximation conservatrice. Cette stratégie a le double avantage suivant :

1. *elle est la plus facile à implanter*, car, construisant une exécution dans la même classe d'équivalence que l'exécution instrumentée, elle n'a pas besoin de détecter des changements d'ordre dans l'arrivée des messages (la détection de conditions de conflit est un problème non trivial [74]) ;

12. Pour le calcul de l'approximation conservatrice il suffit que la trace d'une primitive de réception contienne le message (source, type) effectivement reçu lors de l'exécution instrumentée, même s'il a été reçu avec un masque non-explicite. L'absence de ce masque de la trace obtenue T efface toute trace de non-déterminisme. Ainsi T représente directement la trace de l'application inférée.

2. *elle fournit une borne inférieure de la qualité d'approximation qui peut être obtenue par une méthode de correction des intrusions* (des stratégies plus sophistiquées prenant en compte le reséquencement des messages ne font qu'augmenter la qualité de l'approximation).

L'approximation conservatrice, telle que nous l'avons définie ci-dessus, retrouve la dynamique exacte de l'exécution non-instrumentée d'une application, pourvu que l'instrumentation n'induise pas de changement de comportement : c'est le cas, en particulier, des applications utilisant des schémas de communication déterministes où toute primitive de réception explicite la source et le type du message à recevoir.

4.6.2 La qualité de l'approximation conservatrice

Nous nous intéressons ici à la qualité de l'approximation conservatrice sur des traces d'exécution d'applications. Nous distinguons deux classes d'applications suivant l'instant auquel la décomposition du travail est décidée.

Décomposition statique du travail

L'ensemble du travail à effectuer est partitionné de façon prédéterminée (au moment de la compilation) et est indépendant de toute exécution. Un grand nombre d'applications et en particulier, les applications numériques, utilisent ce type de décomposition. Le partitionnement est effectué selon un modèle *a priori* de complexité des calculs. Chaque flot d'exécution effectue une suite de *phases de calcul* séparées par des *phases de communication*. Citons par exemple l'algorithme de Jacobi pour la résolution de systèmes linéaires, que nous utiliserons dans la section 4.7.1 pour évaluer la correction de l'intrusion en pratique. Bertsekas et Tsitsiklis [13] donnent un grand nombre d'autres algorithmes de ce type qu'ils qualifient de *synchrones*. Ces algorithmes sont déterministes «par nature» [58] et constituent un vaste champ d'application pour la correction des traces par approximation conservatrice.

Dans certains cas, la phase de calcul peut utiliser des primitives de réception non-déterministes pour améliorer les performances, sans pour autant que le comportement logique de l'application en dépende. C'est le cas de la réduction par un opérateur commutatif que nous traitons en détail dans la sous-section 4.6.3.

Décomposition dynamique du travail

La décomposition statique du travail ne tient pas compte du *comportement dynamique de l'architecture cible*. Sur un système multi-usager, comme

un réseau de stations de travail, les ressources de calcul et de communication sont partagées. Les modèles de complexité utilisés pour la décomposition statique du travail ne sont plus valables et conduisent à une utilisation inefficace des ressources. Dans un tel environnement d'exécution variable, les applications doivent avoir un comportement adaptatif.

Par ailleurs, la décomposition statique du travail ne peut pas être appliquée aux *modèles dynamiques de programmation* où le volume du travail et des communications à effectuer ne peut pas être connu a priori (problèmes irréguliers) [85]. Ces modèles sont utilisés pour l'implantation de langages fonctionnels, de langages à objets, des algorithmes construits suivant l'approche «diviser pour paralléliser» (*branch and bound*, A^*,...). D'autres exemples sont la programmation logique et le calcul formel.

Dans ces deux cas, système multi-usager et problèmes irréguliers, la décomposition dynamique (au moment de l'exécution) du travail s'impose – on parle de régulation dynamique de la charge. Vu que le spectre des algorithmes de régulation dynamique est très large (une classification générale peut être trouvée dans [85]), nous limitons notre discussion au cas simple, mais fréquent, où un seul flot d'exécution (le maître) distribue du travail aux autres flots d'exécution (esclaves) en fonction du rythme avec lequel ces derniers lui retournent les résultats. Le comportement logique du maître, et *a fortiori*, de l'application entière, dépend donc directement de l'ordre dans lequel les messages des esclaves lui parviennent. Cette méthodologie est couramment utilisée pour la résolution de problèmes d'optimisation combinatoire, comme le problème du voyageur de commerce par exemple [19]. Dans ce type de problèmes, le maître gère une file de priorité avec des sous-travaux qu'il distribue aux esclaves au fur et à mesure que ces derniers deviennent libres. De nouveaux sous-travaux sont générés dynamiquement et insérés dans la file de priorité gérée par le maître. Non-déterministe «par nature», le comportement de ce type d'applications est très sensible à toute intrusion, qu'elle soit causée par la multiprogrammation ou par un outil de prise de traces. Le mécanisme de régulation compense l'effet des intrusions en réduisant la quantité de travail distribuée aux flots esclaves ralentis. Ainsi, la trace d'exécution reflète toujours une distribution efficace du travail (pourvu que le régulateur soit efficace), même si l'application a été fortement instrumentée. Cependant, en retrouvant le temps de traitement effectif du travail distribué, l'approximation conservatrice introduit des temps d'inactivité dans la dynamique d'exécution et reflète une distribution de charge inefficace. La stratégie de correction par approximation conservatrice n'est donc pas *directement* applicable sur des traces d'exécution d'applications irrégulières.

```
réel a[m], b[m];                                  { morceaux locaux des vecteurs }
réel c;                                           { résultat sur flot zéro }
réel somme;                                       { résultat partiel }
MPI_Comm groupe;                                  { groupe de flots }

{ Calcul de la somme partielle locale }
somme = 0;
pour i de 1 à m
    somme = somme + a[i] * b[i]
finpour

{ Calcul de la somme globale }
groupe=MPI_COMM_WORLD;
mpi_reduce(somme, &c, 1, MPI_REAL, MPI_SUM, 0, groupe...);
...
```

FIG. 4.8 – *Le produit scalaire: exemple d'utilisation de l'opération de réduction globale de MPI.*

4.6.3 Extensions de l'approximation conservatrice

Nous avons indiqué en section 4.6.1 qu'il est possible d'étendre la stratégie de correction par approximation conservatrice pour détecter un changement dans l'ordre de lecture des messages. Cette idée est illustrée dans la présente section sur l'exemple d'un opérateur collectif, dont les propriétés permettent de continuer la correction, même après un changement d'ordre.

L'opération de réduction globale

L'instrumentation peut changer l'ordre d'arrivée des messages au niveau d'un flot récepteur. Dans certains cas, l'ordre d'arrivée de ces messages n'est pas déterminant pour le comportement de ce flot : tel est le cas pour les opérateurs de réduction *associatifs et commutatifs* (somme, produit, min, max) très répandus dans les applications numériques. Les communications sous-jacentes sont qualifiées de *collectives* [72]. De tels opérateurs spécifient un *groupe* de n flots de contrôle qui participent à la réduction et dont un flot, appelé *racine*, reçoit le résultat de l'opération collective. Dans la plupart des bibliothèques de communication, le programmeur peut expliciter une telle opération collective en utilisant une primitive *ad hoc*, comme par exemple, `pvm_reduce` en PVM [29], ou encore `mpi_reduce` en MPI [72]. Prenons l'exemple du calcul du produit scalaire de deux vecteurs a et b distribués

sur un groupe de n flots d'exécution. Le résultat est retourné au flot zéro. Le code MPI de l'algorithme, exécuté par chaque flot, est illustré sur la figure 4.8. La primitive `mpi_reduce` prend en argument le résultat partiel (*somme*) du flot appelant, la variable (c) qui doit contenir le résultat (sur la racine uniquement), l'ensemble E auquel appartiennent les opérandes, ici $E = \mathbb{R}^1$ (*1, MPI_REAL*), l'opérateur de réduction $\otimes : E \times E \to E$ (*MPI_SUM*), l'identificateur de la racine (le flot 0) et le groupe de flots participant à la réduction.

La sémantique d'un tel opérateur de réduction, de plus haut niveau que celle des communications point-à-point utilisées pour l'implanter[13], masque le non-déterminisme des communications sous-jacentes[14] et la trace du niveau applicatif ne contiendra qu'un seul événement de type «réduction» par flot impliqué dans l'opération collective. Il s'agit là d'une situation bénéfique au processus de correction qui, au niveau applicatif, aura à traiter un phénomène globalement déterministe. Pour illustrer le processus de correction sur un opérateur collectif, nous nous proposons ci-dessous un modèle simple du fonctionnement d'une opération de réduction et nous montrons comment on peut corriger les perturbations induites par l'instrumentation au niveau de l'exécution de cet opérateur.

Un modèle simple pour la réduction

La figure 4.9 montre notre *modèle pour le fonctionnement* de l'opérateur de réduction. Chaque flot d'exécution, excepté la racine, envoie son résultat partiel à la racine. La racine, exploitant la propriété de commutativité de l'opérateur de réduction $\otimes$, reçoit ces $n-1$ résultats partiels dans l'ordre «premier arrivé, premier servi» (FIFO) et construit le résultat final au rythme d'arrivée des résultats partiels[15].

La *trace d'exécution* de la réduction fournit $t(SRED_i)$ et $t(ERED_i)$ ($i = 1..n-1$), les dates d'émission des résultats partiels pour les flots différents de la racine, et $t(SRED_0)$ et $t(ERED_0)$, les dates de début et de fin de la

13. Notons qu'un constructeur peut fournir des routines collectives optimisées pour son architecture. Dans notre étude, nous supposons que ces routines sont intégralement basées sur les communications point-à-point.

14. Si, toutefois, non-déterminisme il y a. On peut imaginer une implantation de la communication collective utilisant un anneau virtuel, par exemple, ce qui donnerait un schéma de communication déterministe.

15. Cette centralisation des calculs conduit à une répartition inefficace du travail. Le degré de surcharge en calcul de la racine dépend de la complexité d'évaluation de l'opérateur $\otimes$ (fonction de la dimension de E) et du nombre de flots participant à la réduction. C'est pourtant ainsi que fonctionne la réduction dans la version actuelle de PVM (3.3). Le lecteur intéressé peut se reporter à [12], où une implantation efficace des communications collectives, basée sur les arbres α, est décrite.

$rp \in E$; { *résultat partiel local sur tous les flots* }
$rg \in E$; { *résultat global sur la racine seulement* }

$flot_i \neq racine$:
$[SRED_i]$
envoyer rp **à** *racine*;
$[ERED_i]$

$flot_i = racine$:
$[SRED_0]$
$rg{=}rp$; { *rés. partiel de la racine* }
pour j **de** 1 **à** $n-1$
($[SR_j]$)
recevoir &rp **de** -1;
($[ER_j]$)
$rg = rg \otimes rp$
finpour
$[ERED_0]$
retourner rg ;

FIG. 4.9 – *Modèle de fonctionnement de l'opération de réduction globale (`mpi_reduce`). Les expressions entre crochets indiquent les événements pertinents pour le processus de correction ($flot_i$ désigne l'identificateur du flot appelant).*

boucle de réception sur la racine (cf. figure 4.9). Comme dans les modèles de correction des communications point-à-point, nous supposons que les dates approchées $t_a(SRED_i)$ $(i = 1..n-1)$ et $t_a(SRED_0)$ ont déjà été calculées par une analyse antérieure. En revanche, les dates $t(SR_j)$ et $t(ER_j)$ $(j = 1..n-1)$ des primitives de réception exécutées par la racine ne sont pas connues, car on trace seulement la primitive de réduction au niveau applicatif. Le modèle de fonctionnement de la réduction de la figure 4.9 implique cependant les relations suivantes entre les dates de réception :

$$t(SR_{j+1}) = t(ER_j) + \Delta_{calc} \quad j = 1..n-2 \tag{4.5}$$

$$t(SR_1) = t(SRED_0) \tag{4.6}$$

$$t(ERED_0) = t(ER_{n-1}) + \Delta_{calc} \tag{4.7}$$

où Δ_{calc} est le temps d'exécution de l'opérateur $\otimes$ que nous supposons indépendant de j. Le *principe du processus de correction* est d'exploiter ces relations pour calculer les dates approchées $t_a(ER_i)$ $(i = 1..n-1)$ pour en déduire $t_a(ERED_0)$.

Le processus de correction de la réduction nécessite un *modèle de l'ordre* selon lequel les résultats partiels sont lus par les primitives de réception sur

le flot racine. Pour cela, nous considérons la bijection

$$\mu : \{1..n-1\} \to \{1..n-1\}$$

qui, à la primitive de réception de rang j (événements SR_j et ER_j), associe l'identité de l'émetteur $\mu(j)$ dont le message est lu par cette primitive (événements $SRED_{\mu(j)}$ et $ERED_{\mu(j)}$) lors de l'exécution instrumentée. De façon analogue, nous considérons la bijection μ_a couplant émetteur et récepteur dans l'exécution approchée – μ_a est une approximation de l'ordre μ_0 qui aurait eu lieu si l'application n'était pas instrumentée. Alors qu'un changement d'ordre ($\mu \neq \mu_a$) ne change pas le résultat de la réduction et n'altère donc pas, *a fortiori*, le comportement logique du flot racine, il peut changer les performances de la racine, ce dont le processus de correction doit tenir compte. Pour calculer μ_a, indispensable à l'algorithme de correction, on peut se baser sur un modèle des temps de communication, comme en section 4.5.2, et utiliser le critère suivant :

$$i < j \iff t_a(SRED_{\mu_a(i)}) + \widehat{\Delta}_{com} < t_a(SRED_{\mu_a(j)}) + \widehat{\Delta}_{com}, \ \ \forall i, j \in \{1..n-1\},$$

où $\widehat{\Delta}_{com}$ est l'estimation du temps de communication (qui dépend de la taille du message et de la distance de l'émetteur à la racine). En d'autres termes, le message issu de $SRED_{\mu_a(i)}$ est lu avant celui de $SRED_{\mu_a(j)}$ si et seulement si sa date d'arrivée asynchrone est plus petite (événement **B**). Cela traduit une lecture des messages dans l'ordre FIFO.

Algorithme de correction de la réduction

Nous supposons que les communications point-à-point sous-jacentes sont de type *rendez-vous* et que l'ordre approché μ_a des rendez-vous avec la racine a été déterminé. La durée du rendez-vous (transfert des données) $\Delta RDV_{\mu_a(j)}$ avec l'émetteur $\mu_a(j)$ ($j = 1..n-1$) est supposée connue ou calculable par un modèle. Supposons de même que le temps d'exécution Δ_{calc} de l'opérateur $\otimes$ soit donné. L'algorithme de correction s'écrit alors comme suit :

$t_a(SR_1) = t_a(SRED_0)$; {*éq.* 4.6}
pour j **de** 1 **à** $n-1$
  $t_a(ER_j) = Max(t_a(SR_j), t_a(SRED_{\mu_a(j)})) + \Delta RDV_{\mu_a(j)}$ {*éq.* 4.2}
  $t_a(ERED_{\mu_a(j)}) = t_a(ER_j)$ {*propr. du RDV*}
  $t_a(SR_{j+1}) = t_a(SR_j) + \Delta_{calc}$ {*éq.* 4.5}
finpour
$t_a(ERED_0) = t_a(ER_{n-1}) + \Delta_{calc}$ {*éq.* 4.7}

Le même type d'algorithme peut être utilisé si la primitive d'émission est asynchrone. Au lieu d'utiliser l'équation 4.2 pour le calcul de $t_a(ER_j)$, on se basera sur l'équation 4.4.

4.6.4 L'apport de la réexécution déterministe

Les limites de l'approximation conservatrice

La correction de l'intrusion par la méthode de l'approximation conservatrice n'exploite que les seules informations de la trace T pour approcher la dynamique non-instrumentée T_0. Elle enlève les perturbations directes, tout en conservant l'ordre partiel sur l'ensemble des événements de l'exécution instrumentée. Dans certains cas, on peut exploiter la sémantique d'opérations de plus haut niveau pour dériver des extensions de la méthode de l'approximation conservatrice. Le modèle de correction proposé en section 4.6.3 est basé sur la commutativité de l'opérateur de réduction, ce qui lui permet de reconstituer l'ordre des messages d'une exécution non-instrumentée pour mieux approcher les performances de cette dernière. Cependant, ce type d'extension est assez complexe, et ne fournit pas de solution générale au problème du changement d'ordre des événements de communication point-à-point induit par l'instrumentation.

Le principe de la réexécution déterministe

Les techniques de *réexécution déterministe*[16] (*instant replay*, en anglais), introduites comme support au déverminage d'applications parallèles, utilisent des informations sur l'ordre de réception des messages d'une exécution de *référence* (encore appelée exécution *initiale*). Le mécanisme de réexécution

16. Le lecteur peut se référer à [59] ou [58] pour une description formelle des mécanismes de réexécution déterministe.

déterministe, intégré dans la couche de communication, exploite ces informations d'ordre sur l'exécution de référence pour contraindre d'autres exécutions à respecter ce même ordre. Les messages arrivant dans un ordre différent de celui de l'exécution de référence sont automatiquement reséquencés par le mécanisme de réexécution. L'ordre de lecture des messages étant ainsi conservé, toute réexécution est équivalente à l'exécution de référence [58]. Cette méthode permet d'utiliser des outils de déverminage et de visualisation sans changer le comportement logique de l'exécution par rapport à l'exécution de référence.

L'information sur l'ordre de réception des messages dans l'exécution de référence implique nécessairement un mécanisme de trace spécifique pour capter cet ordre. Par rapport au traceur pour l'évaluation de performances, le traceur pour la réexécution déterministe est beaucoup moins intrusif. En effet, le premier doit enregistrer tous les événements de communication, dont chacun comporte une liste d'attributs de taille importante : dates physiques, délais de blocage, identité des flots partenaires d'une synchronisation et, éventuellement, le masque utilisé pour la réception d'un message. Le deuxième enregistre, pour les seules primitives de réception anonyme, la source et le type du message effectivement reçu, ce qui est suffisant pour le pilotage de la réexécution [58]. Ces informations sont non seulement moins volumineuses (4 octets pour [59]), mais aussi moins fréquentes, en général, que celles enregistrées par le traceur pour l'évaluation de performances. La littérature sur la réexécution déterministe cite généralement, dans le pire des cas, des surcoûts allant de 1,5% à 5% du temps d'exécution [59, 24]. Il paraît donc raisonnable de supposer que *l'ordre reflété par une trace pour la réexécution déterministe est représentatif d'une exécution non-instrumentée.*

La prise de traces sur une réexécution

L'utilisation d'un mécanisme de réexécution permet d'éviter le changement de comportement d'un programme parallèle induit par une instrumentation intrusive. Pour illustrer ceci, considérons une application A qui comprend quatre flots d'exécution (fig. 4.10) :

- un flot récepteur, qui lit 3 messages en utilisant des primitives de réception non-déterministes,
- trois flots émetteurs, dont chacun envoie un message vers le flot récepteur (envoi asynchrone).

Nous supposons que, sur le récepteur, la couche de communication tamponne les messages dans une file et qu'elle les délivre aux primitives de lecture

(réception) de la couche applicative dans l'ordre de leur arrivée (FIFO). La partie supérieure de la figure 4.10 montre une exécution de référence de cette application.

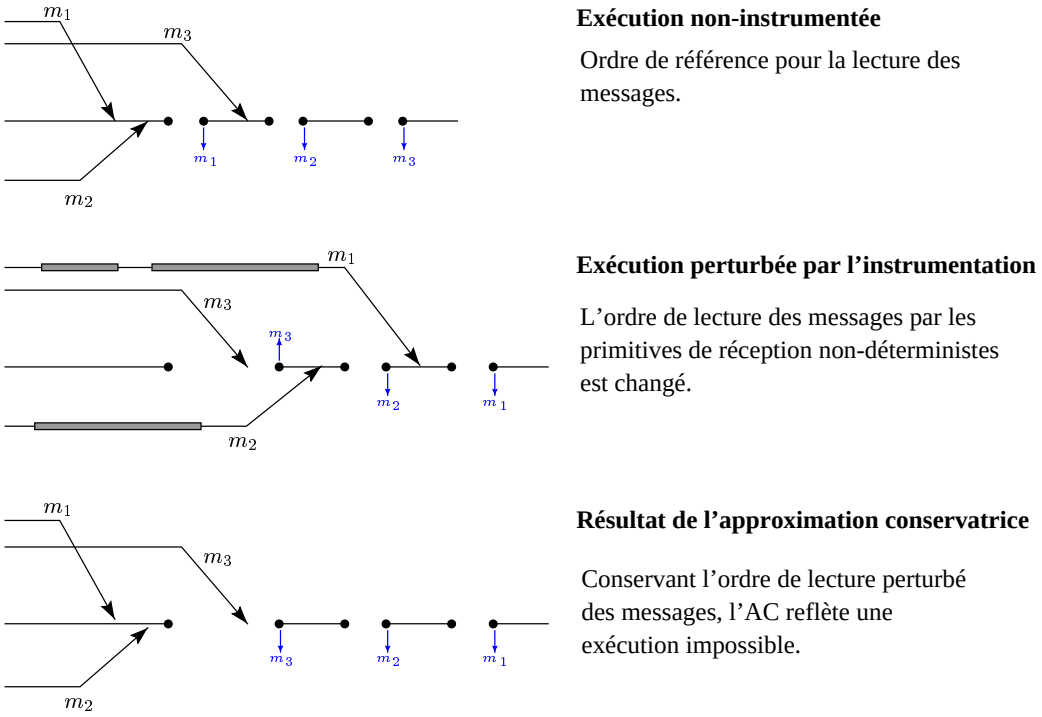


FIG. 4.10 – *Exécution d'une application non-déterministe : illustration du changement d'ordre et de l'échec du mécanisme de correction par approximation conservatrice.*

La partie centrale de la figure 4.10 montre une exécution de cette même application, fortement perturbée par un outil de trace (rectangles ombrés). Notons par T, la trace ainsi obtenue. On note que les messages sont lus dans l'ordre inverse par rapport à celui de l'exécution de référence.

La partie inférieure de la figure 4.10 montre le résultat, T_a, de la correction par approximation conservatrice de la trace T. L'approximation conservatrice retrouve les dates exactes d'émission des messages, reconstitue les dates d'arrivée asynchrone des messages sur le récepteur (événements **B**), mais *conserve* l'ordre de lecture des messages sur le flot récepteur – les performances de celui-ci restent identiques à celles reflétées par la trace brute T. Par ailleurs, dans T_a, l'ordre de lecture des messages est incompatible avec l'ordre d'arrivée des messages au niveau de la couche de communication (violation de la propriété FIFO). T_a représente donc une exécution impossible

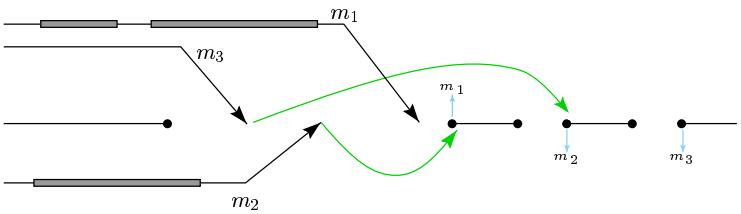


FIG. 4.11 – *Réexécution déterministe selon l'ordre de référence de la figure 4.10.*

---

de l'application analysée A (rappelons qu'elle représente la dynamique de l'application inférée $A_I(T)$).

La figure 4.11 montre une exécution de cette même application A, perturbée par l'instrumentation et *réexécutée* suivant les informations d'ordre disponibles sur l'exécution de référence T_0. La couche de réexécution retarde les messages et ne les rend visibles à la couche applicative qu'à partir du moment où le message satisfaisant l'ordre de référence est lu (flèches courbées). Nous remarquons que ces délais sont directement conditionnés par l'ampleur des perturbations introduites par le traceur. Le comportement logique de l'application est conservé au prix d'une dégradation sensible des performances. L'approximation conservatrice reconstruit, une par une, les dates de fin de réception $t_a(ER)$ en appliquant le modèle de correction présenté dans la section 4.5. Sous l'hypothèse que

- l'exécution ne subit pas d'autres intrusions que celles, mesurables, induites par l'outil de trace,
- le modèle des temps de communication utilisé par l'algorithme de correction est exact,
- le temps de pilotage de la réexécution est nul,
- les perturbations indirectes sont nulles,

l'approximation conservatrice permet de reconstruire exactement les dates de l'exécution de référence.

A chaque primitive de réception non-déterministe, le *temps de pilotage* consiste seulement en un accès à la mémoire pour connaître l'identificateur de l'émetteur du message à recevoir (information obtenue lors de l'exécution initiale) ; la primitive de réception non-déterministe est ensuite redirigée sur une réception explicite en provenance de cet émetteur. Notons que *le temps de pilotage ne comprend pas le délai d'attente de cette réception explicite,*

délai conditionné par les perturbations introduites par le traceur. Dans [58], Leu montre que le temps de pilotage de la réexécution est nul pour les communications par échanges de messages synchrones ou asynchrones. Ce temps est non-nul uniquement dans le cas de la communication asynchrone non bloquante, cas que nous ne traitons pas dans cette thèse.

Ainsi, le champ d'application de l'approximation conservatrice s'étend des applications déterministes de type numérique aux applications non-déterministes, basées sur un découpage dynamique du travail (cf. section 4.6.2), pilotées par un mécanisme de réexécution déterministe.

Informations supplémentaires sur l'exécution de référence

Comme nous l'avons déjà souligné plus haut, le but d'un modèle de correction est de fournir un estimateur de la date effective des événements de base (fin de réception des messages). Les dates de tous les autres événements sont calculées par le modèle de correction très simple de la figure 4.2 (page 81). Les modèles de correction présentés dans les sections 4.4 et 4.5 supposent qu'aucune information n'est disponible sur la dynamique non-instrumentée de l'application (trace idéale T_0). Dans le contexte de la réexécution déterministe, les choses se présentent différemment, car nous supposons que l'ordre de lecture des messages par les primitives de réception anonymes est connu, et que l'obtention de cette information n'altère pas le comportement de l'application. Il est alors tentant d'étendre la prise de traces de réexécution pour qu'elle nous fournisse directement l'information que les modèles de correction doivent estimer – à savoir, les dates des événements de base. Vu que la trace pour la réexécution produit déjà un enregistrement pour chaque exécution de primitive de réception non-déterministe, dont les attributs sont l'identificateur de l'émetteur et le type du message lu, il est très facile d'y rajouter un attribut supplémentaire contenant la date physique de fin de réception. Leu [58] a montré que la perturbation de l'exécution initiale est approximativement de 2% supérieure avec la sauvegarde des informations temporelles[17], et passe ainsi à 3% pour la plupart des applications «réelles».

Les informations supplémentaires de la trace de réexécution fournissent directement les dates de fin de réception, $t_0(ER)$, et la correction d'une trace prise sur une réexécution peut être effectuée très simplement par l'algorithme suivant, simple extension de l'algorithme de correction des flots indépendants

17. Leu était contraint à ajouter une information temporelle à *tous* les événements, y compris les envois de messages. Ceci est dû au fait qu'il était impossible sur l'iPSC/2 de mesurer le temps pris par une intervention du dévermineur: il ne pouvait donc pas appliquer un modèle de correction des dates. Dans notre cas, où l'on ne trace que les réceptions non-déterministes, la perturbation devrait être quelque peu inférieure.

(cf. sous-section 4.3.1) :

$tb^i \leftarrow 0;\ tb^i_a \leftarrow 0;\ acc^i \leftarrow 0; (\forall i)$
tant que $T \neq \{\}$
  $e \leftarrow suiv(T)$;
  $i \leftarrow flot(e)$;
  si $type(e) \neq ER$ **alors**
    $CorrEvt\ (\ t(e), tb^i, tb^i_a, acc^i, t_a(e)\ )$;
  sinon
    $t_a(ER) = t_0(ER)$
    $tb^i \leftarrow t(ER);\ tb^i_a \leftarrow t_a(ER);\ acc^i \leftarrow 0$
  finsi
  $acc^i \leftarrow acc^i + \alpha$;
continuer

Vu qu'elle utilise des informations partielles de l'exécution de référence ($t_0(ER)$) et les informations complètes de la trace prise sur la réexécution, cette technique de correction est dite à phases multiples [87]. Elle est actuellement évaluée dans le cadre de l'environnement de programmation parallèle ATHAPASCAN, basé sur un noyau exécutif de processus légers [76].

4.7 Les résultats expérimentaux

L'analyse de la section 4.6 montre que l'approximation conservatrice joue un rôle central dans les algorithmes de correction de perturbations ; qu'elle soit utilisée directement sur des traces d'applications déterministes, de type numérique, ou sur des traces d'applications non-déterministes, réexécutées selon un ordre de référence. Dans cette section, nous nous proposons de tester la qualité de l'approximation conservatrice sur plusieurs applications numériques et pour les deux modèles de communication étudiés dans ce chapitre.

4.7.1 Jacobi : une première application test

L'algorithme de Jacobi utilise une méthode itérative pour résoudre un système de N équations linéaires à N inconnues, $Ax = b$. Cette méthode construit la suite de vecteurs

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\ (\ b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}), \quad i = 1..N,$$

qui converge vers la solution x du système [18]. Pour paralléliser ces itérations, chacun des P processeurs calcule, à chaque pas d'itération k, N/P termes $x_i^{(k+1)}$. Le coût d'une telle *phase de calcul* est en $\mathcal{O}(\frac{N^2}{P})$ par processeur.

Avant de passer à l'itération suivante il faut que le vecteur $x_i^{(k+1)}$, dont chaque processeur possède un morceau, soit globalement reconstruit. Cette opération de reconstruction, très fréquente dans les applications numériques, est effectuée par un échange total (*all-to-all*, en anglais) lors d'une *phase de communication*. Pour avoir un schéma d'échange déterministe, en vue d'appliquer la correction de traces par approximation conservatrice, nous avons choisi d'implanter l'échange total en utilisant un anneau virtuel. Avec cette implantation, le coût de l'échange total est en $\mathcal{O}(N)$.

Pour détecter la convergence, les processeurs de calcul (esclaves) envoient périodiquement leur erreur locale (maximum de leurs $|(Ax-b)_i|$) à un processeur coordinateur (maître) qui en déduit l'erreur globale et décide si les itérations doivent continuer ou pas. Dans les exécutions que nous étudions ici, le contrôle de l'erreur se fait après chaque itération.

Afin de déterminer la qualité de la correction de traces par approximation conservatrice, nous devons comparer la trace corrigée T_a à une trace idéale, T_0, exempte de toute perturbation. Une telle trace étant inaccessible nous utilisons le temps d'exécution en tant que métrique sur l'ensemble des traces : la correction est précise si elle retrouve correctement le temps d'exécution de l'application non-instrumentée.

Pour la validation des modèles de correction de perturbations, nous avons implanté l'algorithme de Jacobi sur les machines Meganode et IBM-SP2.

Le modèle du rendez-vous

Notre plate-forme expérimentale est la machine Telmat Meganode, équipée de 128 Transputers T800, communiquant par des primitives d'émission et de réception synchrones (*rendez-vous*). Les applications sont soumises les unes après les autres (traitement par lots) et les ressources du système (processeurs, mémoire, réseau) sont complètement dédiées à l'application qui est en train de s'exécuter. Pour la prise de traces, on utilise le traceur Tape [2].

Le tableau 4.3 donne les temps d'exécution instrumenté, non instrumenté et corrigé pour une matrice de dimension $N = 512$. Nous avons effectué 10 exécutions non-instrumentées et 10 exécutions instrumentées sur 17 processeurs (les processeurs 0-15 sont les esclaves, le processeur 16 est le maître). Chaque exécution effectue 200 itérations. Le traceur provoque une augmenta-

18. Une condition suffisante pour que la suite des vecteurs converge est que A soit à diagonale dominante [13].

Temps d'exécution	moyenne (ms)	σ (ms)	pert.
non-instrumenté	37 289,5	0,2	–
instrumenté	46 180,2	0,2	25 %
corrigé	37 291,6	0,3	0,006 %

TAB. 4.3 – *Algorithme de Jacobi sur 16 processeurs (Meganode) : statistiques sur les perturbations et la correction représentant le temps d'exécution moyen, l'écart type σ correspondant et le taux de perturbation (pert).*

tion de 25% du temps d'exécution de l'application. Ce taux de perturbation important est typique des outils logiciels de prise de traces pour l'évaluation de performances : tous les événements relatifs aux communications sont tracés. Dans les exécutions qu'on étudie ici, on trace environ

$$200 \text{ itérations} \times 60 \text{ comms/itération} \times 4 \text{ évts/comm} = 48000 \text{ évts}$$

événements, ce qui correspond à une fréquence globale de génération d'événements de 1,04 évts/ms. Le tableau montre que les traces corrigées reflètent le temps d'exécution non-instrumenté avec une précision remarquable (0,006%) ! On en déduit que les perturbations indirectes, qui ne sont pas prises en compte par l'algorithme de correction, sont minimales ; en particulier, les temps de communication ne sont pas affectés par l'instrumentation. Ceci s'explique par le fait que le Meganode utilise un système d'exploitation minimal, limité à une couche de routage, et que ses ressources sont complètement dédiées à l'application. C'est pour cette même raison qu'on a pu mesurer le coût α d'une perturbation directe avec précision ($890\mu s$).

La figure 4.12 montre l'effet de la correction des intrusions en comparant un chronogramme de la trace brute T avec celui de la trace corrigée T_a correspondante. Sur ces diagrammes Paragraph [34], qu'on a adaptés aux communications par rendez-vous, les barres verticales couplent les événements de sortie d'un rendez-vous (événements **ER** et **ES**). La figure montre clairement comment la distorsion temporelle des flots d'exécution est corrigée [19]. On remarquera que la densité des communications est plus élevée dans la trace corrigée que dans la trace brute : il se peut ainsi qu'une exécution non-instrumentée produise une *contention* au niveau des liens de communication. Bien évidemment, une telle contention n'apparaîtra pas dans une

19. Sur l'anneau virtuel, on voit que les temps de communication augmentent lors du premier tour (augmentation de la taille du message) et qu'ils diminuent lors du deuxième (diminution de la taille). Le fait qu'on puisse observer ce phénomène témoigne de la stabilité des temps de communication sur le Meganode.

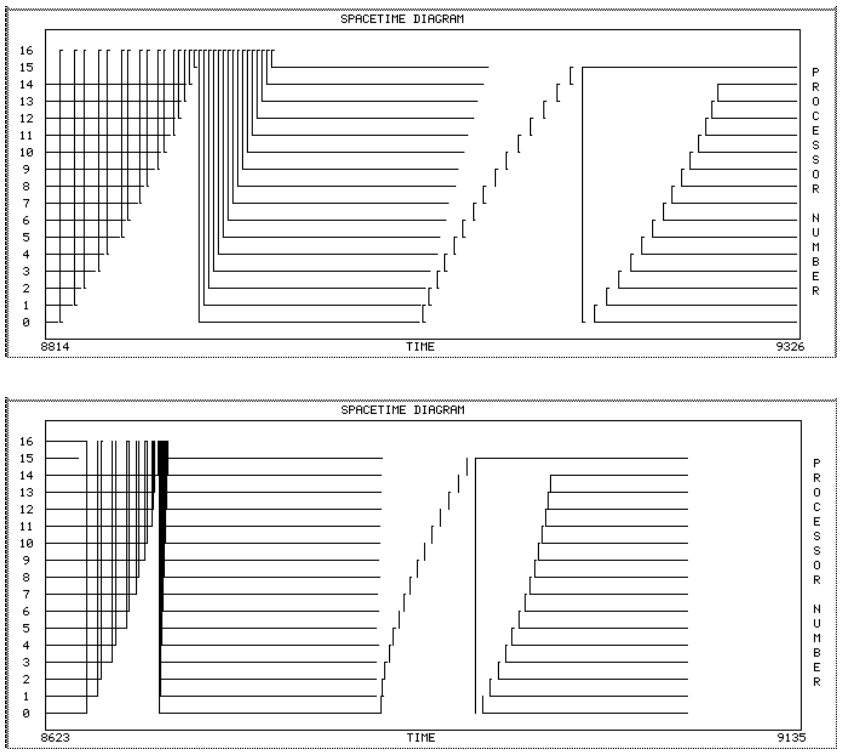


FIG. 4.12 – *Algorithme de Jacobi (Meganode) : comparaison d'une trace brute (partie supérieure) avec la trace corrigée correspondante (partie inférieure). L'unité de temps est de* $500\mu s$ *. Les deux diagrammes sont à la même échelle et représentent 256ms de temps réel.*

---

trace corrigée puisque l'algorithme de correction suppose que l'instrumentation n'a pas d'effet sur les temps de communication (cf. section 4.4). Dans le cas étudié ici, l'égalité des temps d'exécution corrigé et non-instrumenté permet de conclure à l'absence de contention.

Le modèle de l'envoi asynchrone

Notre plate-forme expérimentale est un IBM-SP2 utilisant la bibliothèque de communication par messages PVM. Les communications sont gérées par une version du protocole TCP/IP optimisée pour tirer profit du HPS du IBM-SP2 (*High Performance Switch*, en anglais). Pour que les mesures soient représentatives, la machine nous est réservée lors des expériences. L'application est instrumentée avec le traceur Tape/PVM [63, 65], configuré pour tracer tous les appels à la bibliothèque PVM, y compris les primitives d'empaquetage et de dépaquetage des tampons de messages. Chaque exécution

de l'application effectue 21 itérations, enregistrant environ 1200 événements par exécution sur 4 processeurs et 2100 événements par exécution sur 6 processeurs, ce qui correspond, respectivement, à des fréquences de génération d'événements de 0,4 évts/ms et 0,8 évts/ms pour $N = 1500$. Puisque les événements enregistrés par Tape/PVM sont de taille variable, le coût d'une perturbation directe α n'est pas constant et ne peut pas être précalculé comme sur le Meganode. C'est pourquoi, au moment de l'exécution, Tape/PVM mesure la perturbation directe pour chaque événement et l'enregistre dans la trace avec les autres attributs de cet événement – ainsi, les perturbations peuvent être corrigées par un outil de correction post-mortem.

Sur le IBM-SP2, le coût α est entre 100 et 250μs. Ces délais relativement importants sont dus au fait que Tape/PVM effectue des appels supplémentaires à la bibliothèque PVM pour calculer certains attributs des événements. Par exemple, pour tracer un `pvm_recv`, Tape/PVM appelle la fonction `pvm_probe` pour savoir si le message est présent au moment de l'appel ou pas (cf. sous-section 4.5.5). Par ailleurs, les événements sont encodés, pour occuper moins de place en mémoire tampon et pour réduire le nombre de transferts sur disque [7].

Nous avons réalisé deux ensembles de mesures de temps d'exécution, le premier sur 4 processeurs, le deuxième sur 6 processeurs. Pour chaque ensemble de mesures, la figure 4.13 montre les temps d'exécution instrumentés (courbe supérieure $\mathcal{C}$), non-instrumentés (courbe inférieure $\mathcal{C}_0$) et corrigés (courbe pointillée $\mathcal{C}_a$) pour une taille de matrice N croissante. Les exécutions étant moins stables sur le SP2 que sur le Meganode, nous avons dû faire un grand nombre d'exécutions pour avoir un intervalle de confiance acceptable sur les temps d'exécution[20]. Pour chaque temps d'exécution de la figure 4.13 nous avons effectué 300 mesures : la figure montre les temps d'exécution moyens avec les intervalles de confiance à 95%.

Vu que le nombre d'événements tracés est indépendant de la taille de la matrice, l'écart entre $\mathcal{C}$ et $\mathcal{C}_0$ reste constant. Par contre, cet écart est plus grand pour les exécutions sur 6 processeurs que pour les exécutions sur 4 processeurs, car le nombre d'événements tracés croît avec le nombre de processeurs utilisés. Pour $N = 700$, sur 4 processeurs, le traceur provoque une augmentation de 13% du temps d'exécution. La différence entre le temps d'exécution corrigé et le temps d'une exécution non-instrumentée n'est plus que de 3% et la correction a enlevé 79% des perturbations. Pour $N = 1500$, sur 6 processeurs, l'augmentation du temps d'exécution est de 9%, le temps corrigé est à 2% du temps non-instrumenté et la correction a enlevé 74% des

20. Nous pensons que cette instabilité est due au système d'exploitation ainsi qu'à la gestion des communications par le protocole TCP/IP.

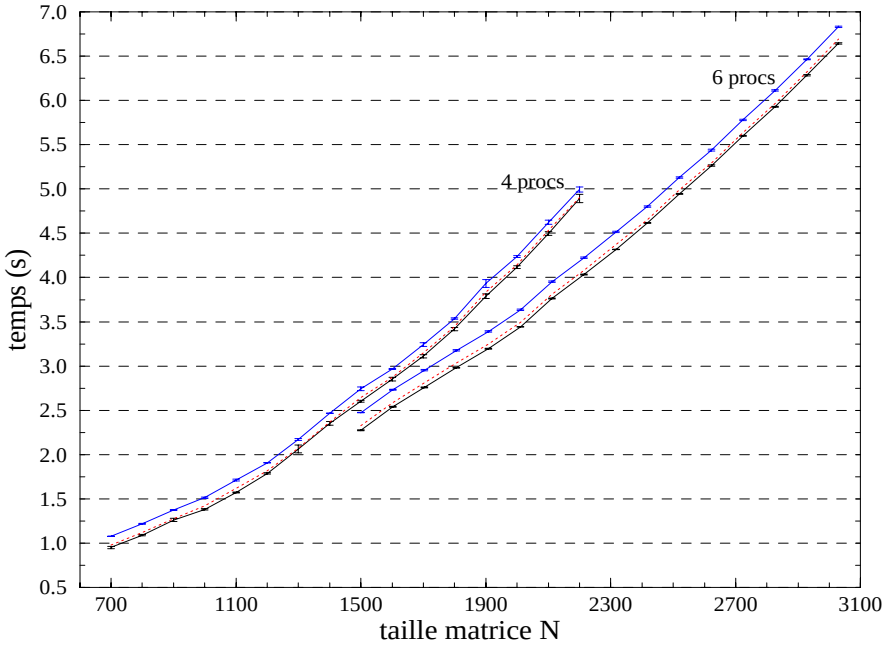


FIG. 4.13 – *Algorithme de Jacobi (IBM-SP2) : temps d'exécution (pour 4 et 6 processeurs) instrumentés, non-instrumentés et corrigés (courbe pointillée) pour une taille de matrice croissante (les barres d'erreur représentent les intervalles de confiance à 95%).*

perturbations.

Le fait que $\mathcal{C}_a$ ne soit pas entièrement située dans la zone de confiance de $\mathcal{C}_0$ implique que la prise de traces induit des *perturbations indirectes*, perturbations que l'on suppose non-mesurables et, *a fortiori*, non-corrigibles (cf. sous-section 4.2.2, page 77). Pour les expériences de la figure 4.13, les perturbations indirectes varient de 10% à 30% de la perturbation totale (la correction enlève de 70% à 90% des perturbations). D'autres mesures montrent que les *phases de calcul*, qui ne comprennent pas d'appels à PVM, ne sont pas affectées par l'instrumentation des *phases de communication*, ce qui implique que les perturbations indirectes affectent seulement les phases de communication. Nous pensons que ceci est dû au fait que le code d'instrumentation contient lui-même des appels à PVM, modifiant l'état interne de la bibliothèque et pénalisant les performances des appels PVM de l'application.

4.7.2 Le jeu d'essais du NAS

Le jeu d'essais du NAS (*Numerical Aerodynamic Simulation*) [8] est un ensemble d'applications réunissant les caractéristiques essentielles du calcul et du mouvement de données des problèmes typiques en simulation aérodynamique. Le jeu d'essai comprend un certain nombre de «noyaux de calcul» (*kernels*, en anglais), dont la résolution de l'équation de Poisson sur une grille 3D (*noyau MG*), le calcul d'une transformée de Fourier rapide en 3D (*noyau FT*), le tri d'un ensemble de clés à valeurs entières (*noyau IS*) ainsi que le calcul de la plus petite valeur propre d'une matrice creuse, symétrique, définie positive, comprenant un schéma aléatoire d'éléments non-nuls (*noyau CG*). Tout comme l'algorithme de Jacobi, les noyaux NAS effectuent un partitionnement statique du travail (cf. section 4.6.2).

A part leur caractéristique d'applications «réelles», ces noyaux ont l'avantage d'être disponibles publiquement sous forme d'applications PVM [91], ce qui les rend directement traçables par Tape/PVM. D'après les résultats des tests effectués dans [91], c'est le noyau CG qui atteint la fréquence de communication la plus élevée et devrait donc également présenter le taux de perturbation le plus élevé. C'est pour cette raison que nous l'avons choisi pour l'étude et la correction de l'effet de sonde. Comme pour les autres applications PVM, le modèle de correction des perturbations à appliquer est celui de l'envoi asynchrone (cf. section 4.5). L'environnement expérimental est identique à celui utilisé pour l'application de Jacobi en sous-section 4.7.1 (envoi asynchrone).

P	N	nb. coms	t_0 (s)	t_a (s)	t (s)	pert. (%)	corr. (%)
4	1400	7124	18,2	18,6	20,7	14	83
9	1400	19774	26,7	27,0	32,6	22	95
4	14000	7124	122,5	122,8	125,4	2	88
9	14000	19774	101,3	101,85	107,9	7	91

TAB. 4.4 – *Noyau CG du jeu d'essais NAS (IBM-SP2) : statistiques sur les perturbations et la correction représentant les temps d'exécution non-instrumentés (t_0), corrigés (t_a) et instrumentés (t) ainsi que le taux de perturbation (pert) et le pourcentage de perturbations corrigées.*

Le tableau 4.4 montre les statistiques d'exécution sur 4 et 9 processeurs pour une matrice de taille 1400 et 14000[21]. Dans tous les cas, la fréquence de

21. On note que la configuration à P=9 et N=1400 conduit à un grain de calcul trop fin. Le programme passe beaucoup plus de temps à communiquer qu'à calculer, ce qui fait qu'il est plus lent que la même configuration à 4 processeurs

génération d'événements est inférieure à 0,6 évts/ms. Le taux de perturbation varie de 2% à 22% ; l'algorithme de correction enlève de 83% à 95% des perturbations totales du temps d'exécution (les perturbations indirectes vont de 5% à 17%). Les résultats expérimentaux sur le noyau CG sont donc tout à fait en accord avec les mesures sur l'application de Jacobi.

4.7.3 Discussion des résultats

A cause du caractère synchrone des communications par rendez-vous, le taux de perturbation est relativement important sur le Meganode (25%). Par contre, pour les communications à émission asynchrone, les perturbations ne dépassent que rarement 15% du temps d'exécution, comme nos expériences avec PVM l'ont montré. Les autres noyaux du jeux d'essais NAS (MG, IS, FT) provoquent une fréquence de génération d'événements encore bien inférieure à celle du noyau CG, et nos tests ont révélé des taux de perturbation inférieurs à 1% pour ces noyaux. Dans ce cas, l'intérêt des méthodes de correction d'intrusion peut paraître mitigé. Toutefois, le taux de perturbation peut augmenter de façon considérable dans les deux situations suivantes :

1. *Introduction de points de trace supplémentaires*

   L'analyste des performances peut être intéressé par d'autres événements que les seuls événements reliés aux activités de *communication*. Ainsi, l'outil de trace de l'environnement d'instrumentation *Pablo* supporte en plus le traçage des entrées et des sorties de *procédures* et de *boucles* ainsi que des appels système d'*entrée/sortie* (*read, write, seek, open, close*) [75].

2. *Bibliothèque de communications plus performante*

   La bibliothèque de communication PVM utilisant le protocole TCP/IP sur le Switch du IBM-SP2 n'est pas très efficace (la latence des communications est de 3 ms !). D'après nos prédictions,[22] l'utilisation d'une bibliothèque optimisée comme MPI-F permettrait d'exécuter le noyau CG en moitié moins de temps, ce qui doublerait le taux de perturbation.

Il est intéressant de reprendre les expériences de cette section dans ces deux situations pour tester la qualité de l'approximation en présence de taux d'intrusion plus élevés. Ainsi, nous avons repris l'application Jacobi et nous avons inséré manuellement un point d'instrumentation dans la boucle principale de calcul. Le tableau 4.5 montre les résultats obtenus pour une matrice

22. Le traceur Tape/PVM comprend un outil de prédiction de performances que nous présentons dans le chapitre 6.

	t (s)	t_a (s)	évts/ms	pert. (%)	corr. (%)
Exec. non-instrumentée	1,17	–	–	–	–
Exec. instr. sans boucle	1,28	1,19	1,13	9	82
Exec. instr. avec boucle	2,40	1,30	18,1	105	89

TAB. 4.5 – *Jacobi : effet de l'instrumentation de la boucle de calcul.*

de taille $N = 1000$. Le point d'instrumentation supplémentaire fait passer le taux de perturbation de 9% à 105% et la fréquence de génération d'événements de 1,13 à 18,1 évts/ms. Malgré le taux de perturbation élevé, la correction enlève 89% des perturbations.

4.8 Conclusions et perspectives

Dans ce chapitre, nous avons présenté plusieurs modèles pour corriger les perturbations induites par la prise logicielle de traces. Ce réajustement de qualité doit permettre d'approcher la qualité des mesures qu'on pourrait obtenir sur un système de trace hybride ou matériel. Notre algorithme de correction pour le modèle de communication à envoi asynchrone permet d'utiliser un maximum d'informations de la trace en minimisant le nombre de modélisations des temps de communication. Nous avons également abordé le cas des applications dites non-déterministes, dont le comportement logique (décomposition de travail, régulation de charge) peut être affecté par les perturbations. Dans ce contexte, nous avons défini la notion d'approximation conservatrice et nous avons montré qu'elle peut être utilisée conjointement avec un mécanisme de réexécution déterministe pour retrouver la dynamique initiale, non-instrumentée d'une exécution non-déterministe. La stratégie de correction par approximation conservatrice est centrale à toute approche de correction des perturbations d'une application, qu'elle soit à comportement déterministe ou pas. C'est pourquoi la partie expérimentale de ce chapitre est dédiée à la validation de la correction par cette stratégie.

Nos expériences sur des applications numériques ont montré la qualité des traces approchées T_a calculées par les algorithmes de correction sur différents modèles de communication et machines. Sur le Meganode, machine mono-usager, avec un système d'exploitation se limitant à une couche de routage très légère, la correction enlève quasiment l'intégralité des perturbations sur le temps d'exécution. Sur le IBM-SP2, opérant un système d'exploitation distribué AIX sur chaque noeud et utilisant une couche de communication TCP/IP, la correction enlève de 70% jusqu'à 95% des perturbations. Sur ce

dernier système, il y a donc plus de perturbations indirectes, perturbations que nous attribuons à l'importante couche système qui sépare l'application du matériel de communication. Toutefois, nous n'avons pas pu *qualifier* les perturbations indirectes. À ce propos, l'étude des relations entre la valeur agrégée des perturbations indirectes (1-*taux de correction*) et les indicateurs de performance fournis par le système d'exploitation pourrait être intéressante[23]. Quoi qu'il en soit, *sur les applications numériques que nous avons testées, la correction permet un ajustement significatif de la qualité des traces (au sens de la métrique du temps d'exécution).*

Toutefois, malgré le fait que la correction enlève la majorité des perturbations induites par l'instrumentation, on peut se demander, au vu du taux de perturbation relativement faible de certaines exécutions, si l'application d'un algorithme de correction est vraiment essentielle. Par exemple, sur la figure 4.13, l'évolution des temps d'exécution instrumentés (courbes $\mathcal{C}$) reflète correctement le taux de croissance des temps d'exécution non-instrumentés (courbes $\mathcal{C}_0$). Ainsi, il est probable que l'analyse d'une trace d'exécution instrumentée, permet effectivement d'améliorer les performances des exécutions non-instrumentées, comme le confirment les expériences avec la bibliothèque de communication PICL [34]. Cependant, une trace d'application n'est représentative que dans le cas où cette application induit une fréquence de génération d'événements suffisamment faible. L'enregistrement d'informations supplémentaires autres que celles, minimales, liées aux actions de communication, comme par exemple les entrées et les sorties de procédures ou de boucles, peuvent augmenter le taux de perturbation de façon significative. L'intérêt des algorithmes de correction d'intrusion est justement de permettre à l'analyste de performances d'*augmenter le nombre d'informations tracées* bien au-delà des simples événements de communication auxquels un traceur logiciel est traditionnellement contraint à se limiter (principe d'incertitude de l'instrumentation). Finalement, l'intérêt croissant des développeurs d'environnements de programmation parallèles commerciaux et industriels (Aims [94], Annai [23]) pour les techniques de caractérisation et de correction des perturbations démontre l'importance que revêt l'étude de l'effet de sonde en pratique.

En plus de la validation de l'approximation conservatrice sur des applications déterministes de type numérique (Jacobi, NAS-CG), il serait intéressant de tester cette méthode conjointement avec un mécanisme de réexécution déterministe, sur une application non-déterministe, effectuant une décomposition dynamique du travail selon le schéma maître/esclave (cf. sous-section 4.6.2). Pour justifier cette démarche, plus lourde de la part des moyens lo-

23. cf. l'appel système *getrusage*

giciels mis en oeuvre, il faudrait montrer, dans un premier temps, que l'amplitude des perturbations est suffisamment grande pour changer le comportement du mécanisme de régulation c'est-à-dire qu'il y a effectivement un changement dans l'ordre de lecture des messages sur le maître. Nous pensons que les techniques de détection des conditions de conflit (*race conditions*, en anglais), comme celles proposées par Netzer et Miller [74] ou Audenaert et Levrouw [5] peuvent être utilisées pour étendre la correction par approximation conservatrice, afin qu'elle détecte les conditions de conflit et détermine si l'instrumentation a une influence sur leur issue.

Les primitives offertes par la première génération des bibliothèques portables de communication par messages, comme PICL [30] ou PVM[24], n'offraient que des primitives de communication *point-à-point*, dont, en général, une primitive d'émission (bloquante asynchrone), une primitive de réception (bloquante synchrone) et une primitive de sonde pour tester la présence d'un message dans les tampons système. Les algorithmes de correction des perturbations que nous avons proposés dans ce chapitre sont basés sur un modèle général de ces communications point-à-point. Avec ces premières bibliothèques de communication, l'implémentation de communications *collectives*, comme la diffusion et la réduction, était laissée à la charge du programmeur. Afin de faciliter la programmation et d'accroître la portabilité des programmes parallèles, le standard MPI [72] spécifie des primitives de communication collectives dont l'implémentation efficace ne revient plus au programmeur, mais au développeur de machines et des bibliothèques de communication. La correction de traces basées sur ces primitives implique la connaissance d'un modèle de leur implémentation, comme nous l'avons montré sur l'exemple de la réduction ; dans le cadre général, les constructeurs optimisent l'implémentation des opérations collectives pour exploiter au maximum les ressources de l'architecture qu'ils proposent [12]. En présence d'opérations collectives, la correction des perturbations aura donc un caractère moins général que dans le cas des seules primitives de communication point-à-point et nécessitera, le cas échéant, la participation des développeurs de machines et des bibliothèques de communication.

Finalement, il serait intéressant d'étudier l'extension des modèles de correction au cas où plusieurs flots d'exécution sont multiprogrammés sur un processeur (*multithreading*, en anglais). De plus en plus d'applications utilisent ce concept de programmation, car il rend implicite le recouvrement des latences de communication par du calcul.

24. PVM jusqu'à la version 3.2 incluse.

Troisième partie

Le traceur Tape/PVM

Chapitre 5

La génération de traces d'exécution

Ce chapitre décrit l'environnement de prise et d'évaluation de traces Tape/PVM. Les outils de *post-traitement de traces* de Tape/PVM concrétisent les théories de qualité de traces développées dans la deuxième partie de cette thèse ; ces outils sont présentés au chapitre 6. Le présent chapitre décrit le *générateur de traces* de Tape/PVM. Avant de décrire l'outil proprement dit, nous le replaçons dans son contexte de recherche, l'environnement ALPES (section 5.1). Nous décrivons ensuite des travaux connexes, proposant d'autres outils d'instrumentation pour l'évaluation des performances de programmes PVM (section 5.2). Nous procédons par la description des différents composants de l'architecture de Tape/PVM (section 5.3).

5.1 L'environnement ALPES

L'environnement ALPES (*ALgorithmes Parallèles et Evaluation de Systèmes*), développé au sein du projet APACHE [76], comprend un ensemble d'outils facilitant l'analyse, la mise au point et la prédiction des performances d'exécutions parallèles [48, 89]. Le but est de permettre l'évaluation de différentes classes d'applications, de diverses stratégies et de diverses machines parallèles. L'environnement est basé sur un modèle de programmes et sur un modèle de machines.

Le langage de description ANDES (*Algorithms aNd DEScription*) [47, 50] permet de *modéliser une application* par un graphe orienté dont les noeuds représentent des tâches de calcul et dont les arcs représentent les contraintes de précédence. Les noeuds sont étiquetés par les coûts de calcul en nombre d'opérations de base (constante ou variable aléatoire) et par les coûts de

communication en nombre d'octets échangés. La manipulation aisée de ces paramètres permet de décrire une large classe d'applications parallèles.

A partir d'un tel modèle quantitatif d'une application est généré un programme parallèle, dit *synthétique*, qui imite le comportement d'une instance de cette application sur une machine parallèle, dite machine *émulée*. Le programme synthétique est alors exécuté sur une machine parallèle donnée, dite machine *cible*, et imite le comportement qu'aurait cette application sur la machine émulée. Par exemple, pour imiter le temps de calcul d'une tâche, dont le coût est décrit par le modèle de l'application, le programme synthétique effectue une boucle d'opérations flottantes, dont le nombre d'itérations est calculé en fonction du rapport des vitesses (Mflops) des processeurs de la machine émulée et de la machine cible. De même, le programme synthétique joue sur la taille des messages qu'il envoie pour reproduire les temps de communications de la machine émulée. Le programme synthétique ne produit donc pas de résultats, mais imite les temps d'occupation des ressources de la machine émulée sur la machine cible – il permet ainsi à l'analyste des performances de prédire les performances de l'application sur la machine émulée à partir de l'analyse de la charge synthétique de la machine cible. Ce mécanisme implique l'utilisation d'un *modèle de machine* permettant de caractériser précisément la machine cible et la machine émulée. Dans l'environnement ALPES, ce modèle, appelé MADRE (*MAchines, DescRiption et Emulation*) [89], comprend une liste de paramètres caractéristiques des machines dont, la vitesse de calcul (en Mflops ou Mips) et les paramètres du modèle des temps de communication (bande passante, latence, ...). Dans [89], Tron décrit des expériences démontrant la qualité de l'émulation d'une machine Paragon d'Intel sur deux machines cibles différentes, le Meganode de Telmat et le SP1 d'IBM.

En plus des modèles de description de classes d'applications et de machines, l'environnement ALPES permet de décrire certaines *règles d'implantation*, comme par exemple les algorithmes de groupement, d'ordonnancement, de placement et d'équilibrage de charge. Dans [14], Bouvry utilise les outils de l'environnement ALPES pour démontrer l'adéquation de différentes fonctions objectives sur divers algorithmes de placement en étudiant des exécutions de programmes synthétiques. L'environnement permet de faire ces analyses sur une large gamme d'applications et d'architectures.

La figure 5.1 donne une vue d'ensemble de l'environnement ALPES. L'évaluation de la charge synthétique, que ce soit pour prédire les performances d'une exécution sur une machine émulée ou pour valider une stratégie de placement, implique un *mécanisme de trace* qui permet d'instrumenter le programme synthétique. L'outil de trace de l'environnement ALPES est le traceur logiciel Tape. Alors que la première version de ANDES générait

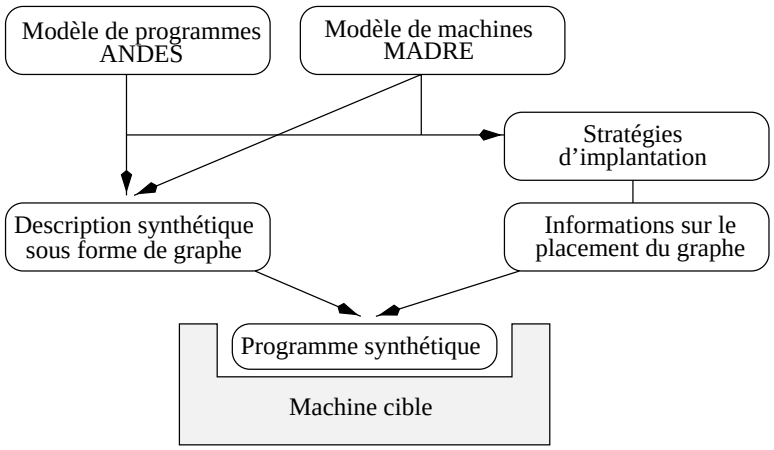


FIG. 5.1 – *ALPES: chaîne de modélisation et d'évaluation.*

---

des programmes synthétiques spécifiques à la machine Telmat Meganode, la version actuelle génère des programmes en PVM, ce qui permet l'accès à une gamme très large de machines cibles. Le traceur Tape a donc également migré de sa première version tournant sur le Meganode [2] vers une version PVM, appelée Tape/PVM [65, 63] que nous décrivons dans ce chapitre.

L'objectif de la conception et de l'implantation du traceur Tape/PVM est double :

1. Tape/PVM doit fournir un support pour l'instrumentation et l'évaluation des performances des exécutions synthétiques de l'environnement ALPES, tout en étant suffisamment générique pour être utilisé dans d'autres contextes que ALPES ;

2. Tape/PVM doit constituer une plate-forme d'expérimentation pour la mise en oeuvre et la validation des théories de qualité des traces présentées dans la deuxième partie de cette thèse, à savoir, la datation globale des événements et la correction de l'effet de sonde.

5.2 Travaux connexes

Le développement de Tape/PVM a tout d'abord bénéficié de nos expériences avec la première version de Tape pour la machine Meganode [2]. Nos techniques de construction de temps global et de correction de l'effet de sonde présentées dans la deuxième partie de cette thèse ont d'abord été intégrées et validées sur la version Meganode de Tape, avant d'être portées et adaptées à la version PVM.

Par ailleurs, l'architecture de Tape/PVM a été largement inspirée de l'environnement AIMS [94, 96] qui comprend un outil d'instrumentation automatisé et un système de traçage supportant différentes bibliothèques de communication par échange de messages, dont PVM. AIMS inclut également des outils de post-traitement de traces permettant de construire un temps global cohérent et de corriger l'effet de sonde. Le mécanisme de correction de l'effet de sonde est basé sur les travaux de Sarukkai et de Malony qu'on a déjà présentés dans le chapitre 4 . L'environnement AIMS conviendrait donc très bien aux besoins d'instrumentation de l'environnement ALPES. Pourtant, AIMS comporte les deux désavantages suivants :

1. l'algorithme de construction de temps global, appelé *compensation de dérive*, consiste simplement à appliquer une version post-mortem de l'algorithme de Lamport [54], dont les désavantages sont bien connus (cf. chapitre 3 de la première partie) ;

2. AIMS est un *produit commercial* de la NASA et sa politique de distribution ne permet pas de le modifier facilement pour y intégrer nos propres techniques de qualité de traces.

PGPVM [88] est un autre outil de traçage pour PVM dont l'architecture est très proche de celle de AIMS et de Tape/PVM. Il utilise le même algorithme de construction de temps global que AIMS. PGPVM est disponible publiquement et aurait pu servir de point de départ pour l'implantation de nos outils. Malheureusement, PGPVM n'était pas disponible au commencement des travaux sur Tape/PVM.

XPVM [52] est une console graphique pour PVM. Suivant le principe de son prédecesseur Xab [10], il utilise le mécanisme de trace intégré dans le noyau de PVM. Ce mécanisme envoie les événements à une tâche collectrice pendant l'exécution de l'application instrumentée afin de tenter une visualisation à la volée de la dynamique de l'exécution. Vu la fréquence élevée de génération des événements lors d'une exécution parallèle, ce mécanisme :

- provoque une charge très élevée de la machine sur laquelle est placée la tâche collectrice, ce qui conduit à des «plantages» fréquents de cette tâche [65] ;

- induit une forte augmentation de la charge du réseau de communication, ce qui pénalise les performances des communications de l'application instrumentée.

Les perturbations, dites indirectes, résultant de la compétition de XPVM et de l'application pour les ressources de communication, sont très difficiles,

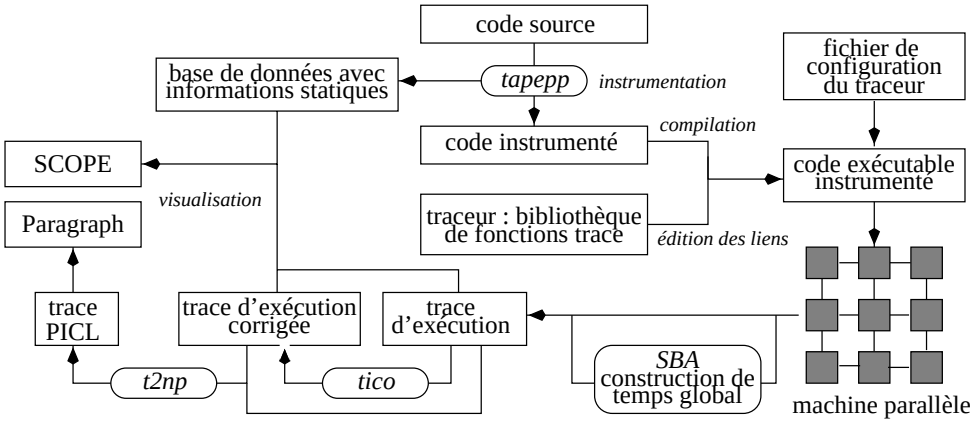


FIG. 5.2 – *Architecture de Tape/PVM.*

sinon impossibles à modéliser et ne peuvent donc pas être évaluées par les algorithmes de correction de l'effet de sonde. De ce fait, l'architecture de XPVM est inintéressante pour l'étude et la validation de nos algorithmes de correction.

5.3 Architecture de Tape/PVM

L'architecture du traceur logiciel Tape/PVM est représentée sur la figure 5.2. Elle comprend plusieurs composants :

1. les outils d'instrumentation du code source (*tapepp*, *tapeppf*),
2. l'exécutif (*runtime*, en anglais) du traceur,
3. la bibliothèque de lecture de traces (*tapereader*),
4. les outils de post-traitement de traces (*t2np*, *tico*) basés sur la bibliothèque de lecture de traces.

La première étape dans la chaîne de traçage de Tape/PVM est l'instrumentation du code source, qui produit une copie instrumentée des modules source de l'application. Ces modules instrumentés sont alors liés à la bibliothèque des fonctions de trace pour produire un code exécutable instrumenté. L'exécution de ce code sur une machine parallèle produit un fichier de trace ; le comportement du traceur peut être configuré par le biais d'un fichier de configuration. Les dates physiques des événements peuvent être rendues globales en appliquant la technique SBA de construction de temps global. La

trace peut ensuite être traitée par l'outil *tico* (*Tape Intrusion COmpensation*) qui implante la stratégie de correction d'intrusions par approximation conservatrice. Dans une dernière étape, l'utilisation d'outils de visualisation permet une évaluation globale des données de la trace – les traces Tape/PVM sont lues directement par SCOPE, l'environnement de visualisation de ALPES [3], ou peuvent être transformées au format PICL pour être exploitées par Paragraph [34].

Les composants d'instrumentation, d'exécutif et de lecture de traces implantent des techniques qui commencent à être bien maîtrisées et qui ont déjà été validées dans d'autres contextes [2, 75, 30, 94]. Les problèmes plus fondamentaux concernent la visualisation des traces, traitée dans [3], ainsi que les aspects de qualité représentative des traces, dont l'étude fait l'objet de la deuxième partie de cette thèse. Pour le présent chapitre, nous avons choisi de nous limiter à une description succincte des composants techniques et nous renvoyons à [7, 65] pour des informations plus détaillées. Le chapitre 6 est dédié à la description des outils de correction et d'analyse de traces.

5.3.1 Instrumentation

On peut envisager plusieurs mécanismes pour l'instrumentation logicielle des applications, dont (cf. aussi le chapitre 2) :

- l'instrumentation du logiciel système (bibliothèque de communication),
- l'insertion de sondes dans le code source.

L'utilisation d'une version instrumentée de la bibliothèque de communication et du système d'exploitation est très commode car elle permet l'enregistrement de traces sans modifier le code source des applications. Cependant, cette technique d'instrumentation demande la collaboration des développeurs du système utilisé et ne permet pas facilement d'activer ou de désactiver la prise de traces pour différentes parties de l'application analysée. XPVM est basé sur ce premier mécanisme d'instrumentation.

L'insertion directe de sondes dans le code source de l'application nous permet *a contrario* de rester indépendant des développeurs de bibliothèques de communication et de contrôler exactement les points d'instrumentation au niveau du code source. Par ailleurs, cette technique d'instrumentation est très générique et peut être utilisée sur toutes les machines pouvant exécuter l'application originale. Opérant au niveau du code source, ce deuxième mécanisme a le désavantage de ne pas pouvoir accéder à des informations système, de plus bas niveau, dont un outil d'analyse pourrait avoir besoin. Évoquons à ce sujet le problème de l'inaccessibilité de l'événement «arrivée

de message» que nous avons traité en détail dans le chapitre 4. Le cas échéant, il est toujours possible de reconstruire l'information manquante par le biais d'un modèle *ad hoc* (cf. sous-section 4.5.5).

Dans Tape/PVM, nous avons choisi le mécanisme de trace par insertion de sondes, principalement à cause de sa généricité. Les outils d'instrumentation, *tapepp* et *tapeppf* sont des préprocesseurs de code source qui insèrent automatiquement des points de trace (*sondes*) dans les modules sources de l'application de l'utilisateur. Le préprocesseur *tapepp* instrumente les sources en C alors que *tapeppf* instrumente les sources en Fortran[1]. L'instrumentation consiste à insérer les appels aux fonctions d'*initialisation* et de *terminaison* du traceur[2] et à remplacer chaque appel à la bibliothèque de communication PVM par un appel à une fonction dite de déroutage. L'entête d'une *fonction de déroutage* est compatible avec celui de la fonction qu'elle remplace : les préprocesseurs complètent cette entête par le numéro de ligne et l'identificateur du module source contenant l'appel dérouté. La fonction de déroutage effectue alors l'appel effectif à la bibliothèque de communication et génère un événement dont les attributs contiennent des informations sur cet appel. Le préprocesseur déroute ainsi tous les appels à la bibliothèque PVM.

En plus de l'instrumentation du code source, le préprocesseur de Tape/-PVM génère également une base de données avec des informations statiques sur le code analysé (cf. fig. 5.2). Actuellement cette base de données consiste simplement en une table affectant un identificateur au nom de chaque module source traité par le préprocesseur.

5.3.2 L'exécutif du traceur

La bibliothèque de l'exécutif Tape/PVM comprend les routines d'initialisation et de terminaison du traceur ainsi que les fonctions de déroutage dont le fonctionnement est décrit en sous-section 5.3.1.

Contrairement à XPVM, Tape/PVM utilise son propre mécanisme de tamponnage d'événements. Les tampons d'événements sont vidés sur les disques locaux, ce qui permet de laisser toute la bande passante du médium de communication à l'application instrumentée. Tape/PVM évite ainsi les perturbations indirectes de l'application au niveau de l'utilisation du réseau.

Lors de l'initialisation, les tampons d'événements sont alloués et une première phase d'échantillonnage pour la synchronisation SBA des horloges est effectuée. La taille des tampons, ainsi que les paramètres pour SBA (durée de la phase d'échantillonnage, nombre de points de mesure, machine dont

1. Les préprocesseurs opèrent par substitution lexicale sans faire d'analyse syntaxique. Ils sont écrits en *lex* et *perl*.

2. Ces appels sont insérés avant le début et après la fin du programme principal (*main*).

l'horloge sert de référence) sont lus depuis le fichier de configuration. Après cette phase d'initialisation, toutes les tâches de l'application franchissent une barrière de synchronisation, puis commencent effectivement leur travail. Pour chaque appel à la bibliothèque PVM, la fonction de déroutage correspondante produit un événement qui est formaté dans le tampon d'événements local à la tâche qui a effectué l'appel.

Lors de la terminaison, les tâches de l'application se joignent à une deuxième barrière après laquelle la deuxième phase d'échantillonnage pour SBA est effectuée depuis la machine dont l'horloge a été choisie comme référence. Une fois l'échantillonnage terminé, la machine de référence calcule les coefficients de dérive et de décalage des horloges et commence à télécharger les traces depuis les autres tâches, «globalisant» leurs dates physiques avant de les écrire dans un unique fichier de traces (on verra des exemples au chapitre 6).

5.3.3 La bibliothèque de lecture de traces

La bibliothèque de lecture de Tape/PVM (*tapereader*) permet de manipuler aisément les enregistrements du fichier de trace tout en faisant abstraction du format de ce dernier. Elle permet au développeur d'outils d'analyse post-mortem de parcourir le fichier de trace de façon séquentielle en utilisant la fonction

```
tape_read_event(fichier_trace, &evt)
```

qui lit l'événement courant du fichier de trace dans la structure *evt* de type `TapeEvent`. Tous les objets de ce type comprennent une partie d'entête et une partie variable. La partie d'entête contient la perturbation directe de génération (α), le type de l'événement, l'identificateur de la tâche génératrice, l'identificateur et le numéro de ligne du fichier source de l'appel générateur ainsi que la date physique globale de génération. La partie variable dépend du type d'événement et contient la liste des attributs de l'événement. La bibliothèque de lecture contient un catalogue donnant pour chaque type d'événement la liste et la sémantique des attributs.

En guise d'exemple, la figure 5.3 illustre la structure de données associée à l'événement de réception de message (fonction `pvm_recv`). Parmi les attributs, on note la présence du booléen *arrived*, indiquant si le message est déjà arrivé au moment de l'appel ou pas (au sens de la fonction de sondage `pvm_probe`). Cette information permet de réduire considérablement le nombre de recours à un modèle pour évaluer les temps de communication lors de la correction des perturbations. Dans l'entête, on note la présence du numéro de ligne et de l'identificateur du fichier contenant l'appel générateur

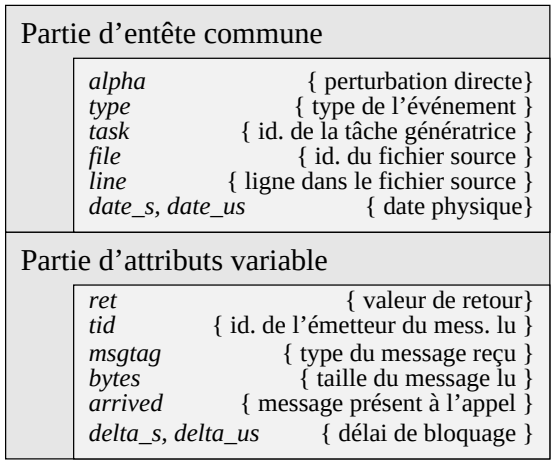


FIG. 5.3 – *Structure de données associée à l'événement de réception de message (`pvm_recv`).*

---

de l'événement – cette information permet aux outils d'analyse de faire le lien avec les modules sources pour localiser plus rapidement l'origine d'un problème de performances (la base de données statiques générée par le préprocesseur permet de retrouver le nom complet du module source à partir de son identificateur).

Un outil d'analyse de traces basée sur la bibliothèque de lecture résiste facilement aux changements du format de trace. Par ailleurs, l'adaptation de la bibliothèque de lecture sur des traces issues d'un autre environnement de trace permet de migrer aisément les outils vers cet environnement (du moment que les modèles de programmation sous-jacents sont compatibles).

5.3.4 Les outils de post-traitement de traces

Dans Tape/PVM, les outils de post-traitement de traces, encore appelés outils d'analyse, comprennent

1. les outils de *conversion* de format de traces,
2. les outils de *correction* de traces.

Les outils de conversion de format de traces permettent l'exploitation des mesures faites par Tape/PVM par des outils d'analyse d'autres environnements. Ainsi, *t2np* transforme les traces Tape/PVM au format *New PICL* [92] ce qui permet de les visualiser avec Paragraph, outil qui fournit plus de 30 diagrammes animés [34]. *t2np* ne supporte non seulement toutes les communications point-à-point de PVM, mais aussi les communications collectives

(réduction, diffusion (*mcast, bcast*), regroupement (*gather*)) et les opérations de groupe (barrière, joindre un groupe, quitter un groupe). Par ailleurs, les traces Tape/PVM sont directement exploitables par l'outil de visualisation SCOPE [3] de l'environnement ALPES.

Les outils de correction de traces concrétisent les travaux théoriques décrits dans les chapitres 3 et 4 et ils ont été utilisés, en partie, pour leur validation. Ils comprennent l'outil de construction de référence globale de temps et l'outil de correction des perturbations. D'autres outils permettent de tester des modèles de communication sur des exécutions réelles ainsi que la prédiction des performances. Nous pensons que cette catégorie d'outils est d'une importance capitale dans tout environnement d'évaluation de performances et nous la traitons en détail dans le chapitre suivant.

5.4 Conclusion

Tape/PVM est un outil d'instrumentation, de prises de traces et d'évaluation de performances d'exécutions d'applications parallèles écrites en PVM. Il doit servir d'instrument de mesure pour les expériences effectuées dans l'environnement ALPES. Grâce à PVM il est suffisamment générique pour être utilisé dans d'autres contextes que ALPES.

Tape/PVM est le fruit d'un travail important d'encadrement [7], de développement[3], de mise au point, de documentation [65] et de distribution. Depuis sa sortie publique[4] début 95, Tape/PVM est utilisé dans plusieurs laboratoires de recherche (Europe, Brésil), principalement à cause de son algorithme de construction de temps global et de la possibilité de visualiser les traces avec Paragraph. Grâce à son système de tampons d'événements il permet d'éviter les problèmes de XPVM liés à l'utilisation intensive du réseau de communication par le mécanisme de trace intégré dans le système PVM.

Dans les extensions futures de Tape/PVM on prévoit notamment :

- l'extension des outils d'instrumentation *tapepp* et *tapeppf* pour insérer des sondes pour le traçage des appels de procédures et des durées de boucles ; ces informations supplémentaires impliquent un mécanisme de régulation automatique de la fréquence de génération des événements ;

- l'échantillonnage, à chaque génération d'événement, de la charge des

3. Tape/PVM comporte un peu moins de 10000 lignes de code C, dont 4500 lignes pour la bibliothèque de lecture de traces et les outils de post-traitement basés sur cette bibliothèque.

4. Tape/PVM est distribué gratuitement sur `ftp://ftp.imag.fr/imag/APACHE/TAPE`.

processeurs ;

- la conversion des traces Tape/PVM au format SDDF [6], ce qui permet de les exploiter dans l'environnement PABLO [80] ;
- l'adaptation au standard MPI [72].

Il s'agit là essentiellement de problèmes techniques d'ingénierie qui ont déjà été abordés dans d'autres environnements comme AIMS [94] ou PABLO [75], par exemple. Nous pensons qu'à l'heure actuelle les problèmes délicats se situent plus au niveau de l'exploitation, de la correction et de l'analyse des traces. Ce sujet est abordé dans le chapitre suivant.

Chapitre 6

Le post-traitement des traces

Ce chapitre décrit les outils de post-traitement des traces du traceur Tape/PVM. Ces outils concrétisent nos travaux sur la qualité représentative des traces qui font l'objet de la deuxième partie de cette thèse. Nous introduisons également deux extensions qui s'intègrent naturellement dans l'outil de correction des perturbations. Ces extensions fournissent un support pour le test de modèles de communication et pour la prédiction des performances.

6.1 La datation globale des événements

Tape/PVM utilise une méthode statistique pour calculer un estimateur d'une référence globale de temps physique. Cette méthode est basée sur la technique d'interpolation SBA [67] qui permet d'obtenir une bonne qualité d'estimation tout en minimisant la durée des périodes d'échantillonnage. Un mécanisme de filtrage enlève automatiquement les mesures aberrantes des échantillons. Contrairement à l'outil de correction des perturbations, le mécanisme de construction de temps global est intégré dans l'exécutif du traceur Tape/PVM même. Si la machine parallèle utilisée ne dispose pas d'une horloge globale physique, l'utilisateur peut activer ce mécanisme par le biais du fichier de configuration du traceur, fichier qui est lu à chaque lancement d'une application instrumentée. Tape/PVM effectue alors l'échantillonnage des horloges physiques nécessaire à l'estimation du temps global : à la collecte des traces, les dates physiques des événements sont automatiquement converties des temps locaux (horloges locales) en temps global. A part les délais supplémentaires d'échantillonnage au début et à la fin des exécutions, ce mécanisme reste complètement transparent à l'utilisateur.

En plus du fichier contenant la trace des événements, le traceur pro-

```
******************** TAPE/PVM - SBA Clock Statistics ******************

Reference machine         : geronimo
Length of sample period   : 13 secs
Size of sample            : 10
Window size for r. median : 5

Machine  |   Slope    |  95% Conf  | %Err |  Offset   | 95% Conf | %Err |
-------------------------------------------------------------------------
geronimo | 1.00000000 | 0.00000000 | 0.00 |  0.000000 | 0.000000 | 0.00 |
nsw18    | 0.99999469 | 0.00000005 | 0.00 |  0.156340 | 0.000136 | 0.09 |
nsw19    | 1.00001908 | 0.00000013 | 0.00 | -0.587212 | 0.000344 | 0.06 |
nsw20    | 0.99998769 | 0.00000002 | 0.00 | -0.707214 | 0.000041 | 0.01 |
nsw21    | 0.99999650 | 0.00000002 | 0.00 | -0.471192 | 0.000056 | 0.01 |
nsw22    | 1.00000042 | 0.00000002 | 0.00 |  0.352856 | 0.000043 | 0.01 |
nsw23    | 1.00000032 | 0.00000001 | 0.00 |  0.235829 | 0.000038 | 0.02 |
nsw24    | 0.99999223 | 0.00000002 | 0.00 | -0.826943 | 0.000043 | 0.01 |
nsw26    | 1.00000431 | 0.00000003 | 0.00 | -0.426932 | 0.000065 | 0.02 |
```

FIG. 6.1 – *Fichier de statistiques du temps global produit par Tape/PVM.*

duit un fichier avec le bilan des statistiques du calcul du temps global. La figure 6.1 donne un exemple de ce fichier obtenu sur un IBM-SP2. La première colonne contient le nom des machines de la configuration PVM. Les colonnes suivantes contiennent les dérives (`Slope`) et les décalages (`Offset`) des horloges correspondantes (par rapport à l'horloge de référence, ici celle de la machine *geronimo*). Les intervalles de confiance à 95% sur la dérive et le décalage sont également donnés. Ces intervalles sont calculés en utilisant les résultats présentés en section 3.2.3 (page 40). Au cas où l'intervalle de confiance sur le décalage est anormalement large (plus de 5%), il se peut que le réseau de communication ait été fortement chargé lors de l'échantillonnage ou qu'un mécanisme de resynchronisation périodique des horloges soit activé, comme le *Network Time Protocol* par exemple. Dans le premier cas, il convient d'augmenter la durée des phases d'échantillonnage moyennant le fichier de configuration du traceur. Si l'erreur sur le décalage ne peut pas être réduite à moins de 1%, cela indique la présence d'un mécanisme de resynchronisation périodique[1] qu'il convient à ce moment de désactiver – un tel mécanisme n'est pas suffisamment précis, en effet, pour dater les événements d'un calcul réparti.

1. C'est le cas sur la ferme de DEC Alpha du LIFL à Lille. L'erreur sur le décalage varie entre 5% et 60%.

6.2 La correction de l'effet de sonde

6.2.1 Rappel du principe de la correction de perturbations

Un algorithme de correction applique un *modèle de perturbation* à une trace d'exécution pour approcher la dynamique non-instrumentée de cette exécution. Pour une exécution d'une application en PVM, le modèle à appliquer est celui de l'*envoi asynchrone* que nous avons présenté dans le chapitre 4. Ce modèle donne un estimateur de la date effective de l'événement de fin de lecture d'un message par `pvm_recv`. Tous les autres événements (événements locaux, événements d'envoi et de début de lecture) sont corrigés par le modèle de correction des flots d'exécution séquentiels, enlevant simplement, sur chaque flot, l'accumulation des perturbations directes par rapport au dernier événement de fin de lecture de message (événement de base).

Une telle correction des perturbations se heurte au problème des applications non-déterministes, dont le comportement logique peut être changé suite aux perturbations induites par l'instrumentation. Une trace d'exécution d'une telle application ne contient en effet que les informations d'un seul comportement sur un ensemble de comportements possibles[2]. La stratégie de correction par *approximation conservatrice* consiste à construire une trace approchée, T_a, appartenant à la même classe de comportement que la trace brute T. Vu qu'il y a équivalence entre comportement et ordre partiel causal sur l'ensemble des événements, cela revient à dire que l'approximation conserve l'ordre de l'exécution instrumentée. Comme on l'a vu dans le chapitre 4, l'application de cette stratégie à une trace d'exécution d'une application non-déterministe peut aboutir à une trace corrigée reflétant un comportement improbable, voire impossible de cette application. Dans ce cas, la correction des perturbations doit être combinée avec un mécanisme de réexécution déterministe.

6.2.2 L'algorithme de parcours pour la correction

Vu qu'elle conserve l'ordre de lecture des messages, la stratégie de correction par approximation conservatrice est facile à mettre en oeuvre. Pour mieux comprendre l'algorithme de parcours de la trace, nous considérons d'abord une implantation «à la volée» de l'approximation conservatrice, où la correction de l'intrusion se fait pendant l'exécution instrumentée.

2. On rappelle que, par définition, deux exécutions ont le même comportement si et seulement si les flots d'exécution effectuent les mêmes instructions sur les mêmes données d'une exécution à l'autre.

Pour l'*implantation à la volée*, chaque flot d'exécution accumule les perturbations directes de génération d'événements dans une variable locale. Lors de la génération d'un événement, la valeur courante de cet accumulateur est retranchée de sa date mesurée pour construire sa date approchée. Chaque flot d'exécution procède ainsi, jusqu'au premier événement de début de lecture de message inclus. Au déblocage de la primitive de lecture, la date approchée de fin de lecture est calculée en appliquant le modèle de correction des communications par envoi asynchrone. Ce calcul se fait à partir des dates mesurée et approchée de l'émission, des dates mesurée et approchée du début de lecture et de la date mesurée de fin de lecture. Les dates concernant l'émission n'étant connues que par le flot émetteur, il faut rajouter ces informations dans le message. Une fois la date approchée de fin de lecture calculée, l'événement de base, par rapport auquel les perturbations sont accumulées, devient cet événement de fin de lecture et l'accumulateur des perturbations est mis à zéro.

Très simple à mettre en oeuvre, cette solution à la volée comporte toutefois deux désavantages :

1. elle augmente les perturbations directes en rajoutant des calculs numériques (4 opérations flottantes par événement) et en augmentant la taille des messages (8 octets) ;

2. elle suppose que le temps global est connu lors de l'exécution, ce qui n'est pas le cas lorsqu'on utilise le temps statistique SBA.

Les perturbations supplémentaires étant directes, car mesurables ou calculables, elles peuvent être comptabilisées par la correction, simplement en les additionnant à l'accumulateur des perturbations directes. Toutefois, pour les applications à comportement non-déterministe, accroître les perturbations signifie augmenter le risque d'un changement de comportement.

Afin d'éviter ces problèmes nous avons opté pour une *solution post-mortem*. Elle consiste en un parcours séquentiel de la trace, triée par ordre chronologique suivant l'échelle de temps instrumenté. Lors de ce parcours séquentiel post-mortem, on simule le fonctionnement de la correction à la volée, en utilisant sur chaque flot une file des messages émis. Sous réserve que le temps global est suffisamment précis pour permettre une datation causalement cohérente, les dates d'émission sont connues au moment où un événement de fin de lecture de message est rencontré dans la trace – le modèle de correction de l'envoi asynchrone peut alors être appliqué.

6.2.3 L'outil de correction de Tape/PVM

Dans Tape/PVM, l'outil de correction des perturbations, implantant la version post-mortem de l'approximation conservatrice, s'appelle `tico` (*Tape Intrusion COmpensation*). Il prend en entrée le fichier de trace trié par ordre chronologique (*events*) et produit l'approximation conservatrice sur la sortie standard (*events.tico*) :

```
tico -sm -beta 2902.6 -tau 0.688442 events > events.tico
```

Nous rappelons que le modèle de correction a recours à un *modèle des temps de communication* pour reconstruire la date d'arrivée des messages au niveau du récepteur, date qui est non-mesurable au niveau applicatif. Dans sa version actuelle `tico` ne supporte que les modèles du type $\beta + L\tau$, où β est la latence et τ l'inverse de la bande passante. Les valeurs de ces deux paramètres doivent être fournies à l'appel de `tico` (β en μs et τ en s/Mo) – les valeurs de l'exemple précédant correspondent à PVM sur TCP/IP-Switch sur un IBM-SP2[3]. Dans [89], Tron montre que le temps d'une communication point-à-point dépend non seulement de la *taille* L du message, mais aussi de la *charge moyenne du réseau* et, éventuellement, de la *distance* entre processeurs communiquants. Dans une version future, on va étendre `tico` pour que l'utilisateur puisse spécifier un modèle des temps de communication arbitraire, fonction de ces trois variables.

L'option `-sm` demande à `tico` de minimiser le nombre de modélisations des temps de communication en appliquant la méthode qu'on a développée dans le chapitre 4 (mode `MINSIM`). Cette méthode permet de garder un maximum d'informations de la trace originale. En utilisant l'option `-sf`, on peut également demander la modélisation systématique de tous les temps de communication (mode `FULLSIM`). On reviendra sur cette option en section 6.4.

`tico` produit également un fichier avec le bilan sur le calcul de la correction. La figure 6.2, montre le bilan obtenu pour la correction d'une trace d'exécution d'une transformée de Fourier rapide bi-dimensionnelle (FFT-2D) sur 4 processeurs. La première ligne de ce fichier montre le nombre total de communications (`NBCOM`), le nombre de primitives de lecture qui ont trouvé leur message avant la correction (`NBCOMF`) et après la correction (`NBCOMFA`), le nombre de modélisations des temps de communication avec notre méthode (`NBSIM`) et avec celle de Sarukkai-Malony (`NBSIM_SM`). Les trois lignes sui-

3. A part le réseau Ethernet standard, le IBM-SP2 est muni d'un «Switch Haute Performance». IBM fournit une version de TCP/IP optimisée pour le Switch : nous appelons cette version du protocole *TCP/IP-Switch*.

```
NBCOM=92 NBCOMF=51 (55%) NBCOMFA=49 (53%) NBSIM=42 (46%) NBSIM_SM=86 (93%)

Alpha correction is on
Simulation mode is MINSIM
Packing and unpacking delays are NOT removed

TRSIM MODEL CHECK

  BETA = 2902.616964
  TAU = 0.688442
  NBCHECK = 36
  MEAN OF RESIDUALS = 0.20564
  STDEV OF RESIDUALS = 0.16145
  k_35 (0.05) = 2.44
  t VARIABLE = 7.642 [REJECT]
  MIN:MEAN:MAX OF MEASU TRANSIT = 0.002360:0.372469:0.746255
  MIN:MEAN:MAX OF MODEL TRANSIT = 0.002914:0.163322:0.183374
```

FIG. 6.2 – *FFT-2D: bilan de la correction d'une exécution brute (cf. aussi la partie supérieure de la figure 6.3).*

vantes donnent des informations sur le mode opérationnel de la correction : dans le cas de la figure 6.2 les perturbations directes α ont été enlevées, la stratégie de modélisation des temps de communication est `MINSIM` et les délais de paquetage et de dépaquetage des messages n'ont pas été enlevés. Enlever les délais de paquetage et de dépaquetage devient intéressant lors de la prédiction de performances (cf. section 6.4). La dernière partie du fichier bilan est relative au test d'adéquation du modèle des temps de communication et est décrite en section 6.3

La *qualité* de la trace approchée par `tico` est discutée en détail dans le chapitre 4. Ici, nous rappelons simplement que sur des applications numériques à comportement déterministe, la correction enlève entre 70% et 95% des perturbations totales du temps d'exécution (PVM TCP/IP-Switch sur un IBM-SP2). Finalement, notons que, dans sa version actuelle, `tico` intègre non seulement le modèle de correction des communications point-à-point (envoi asynchrone), mais aussi un modèle de correction de la diffusion (traité comme un ensemble d'émissions asynchrones) et de la barrière de synchronisation (modèle de la barrière de Malony [68]).

6.3 Le test de modèles des temps de communication

6.3.1 Intérêt du test de modèles

Un modèle des temps de communication comprend un certain nombre de *paramètres*, comme la latence β et la bande passante $\frac{1}{\tau}$, ainsi qu'un certain nombre de *variables*, comme la taille L du message envoyé. L'approche classique pour mesurer le temps t d'une communication dans une machine parallèle MIMD consiste à utiliser l'application *ping-pong* [40, 89] qui fournit un ensemble d'observations expérimentales du couple (L, t). Dans le contexte du modèle des temps de communication, L est une variable explicative et t est la variable expliquée. Les paramètres du modèle sont ajustés sur ces données expérimentales par la méthode classique des moindres carrés.

Le modèle le plus simple et fréquemment utilisé est le modèle linéaire à une variable $\beta + L\tau$, valable pour une communication point-à-point sur un lien dédié. S'il est valable pour une communication isolée, il peut ne plus l'être pour les communications d'une application réelle qui peuvent être plusieurs à se partager un même lien.

Tester l'adéquation d'un modèle aux temps de communication observés lors d'une exécution réelle permet d'évaluer le modèle dans d'autres conditions que celles, souvent artificielles, dans lesquelles il fut initialement ajusté. En cas d'échec du test, le modèle peut être raffiné en y rajoutant une ou plusieurs variables, modélisant la charge des liens de communication par exemple.

6.3.2 Test statistique utilisé

Nous avons vu que l'algorithme de correction utilise un modèle pour le temps d'une communication quand celui-ci n'est pas calculable depuis la trace. Toutefois, dans le cas où le message arrive *après* le début de la primitive de réception, le temps de communication *est* déductible de la trace : pour un ensemble de messages de taille L, l'algorithme de correction peut alors comparer le temps mesuré t_i^L (i^{e} observation, $i = 1..n^L$) avec le temps estimé par le modèle t^L. En calculant les erreurs d'estimation $e_i^L = t_i^L - t^L$ correspondantes, il est capable d'évaluer l'adéquation du modèle aux valeurs des temps de communication observées sur une trace d'exécution d'une application réelle. Dans `tico`, nous avons intégré un test très simple du modèle des temps de communication. Après la correction de la trace, `tico` fournit une statistique de test renseignant l'utilisateur s'il y a adéquation entre le modèle qu'il a fourni et les temps de communication effectivement observés

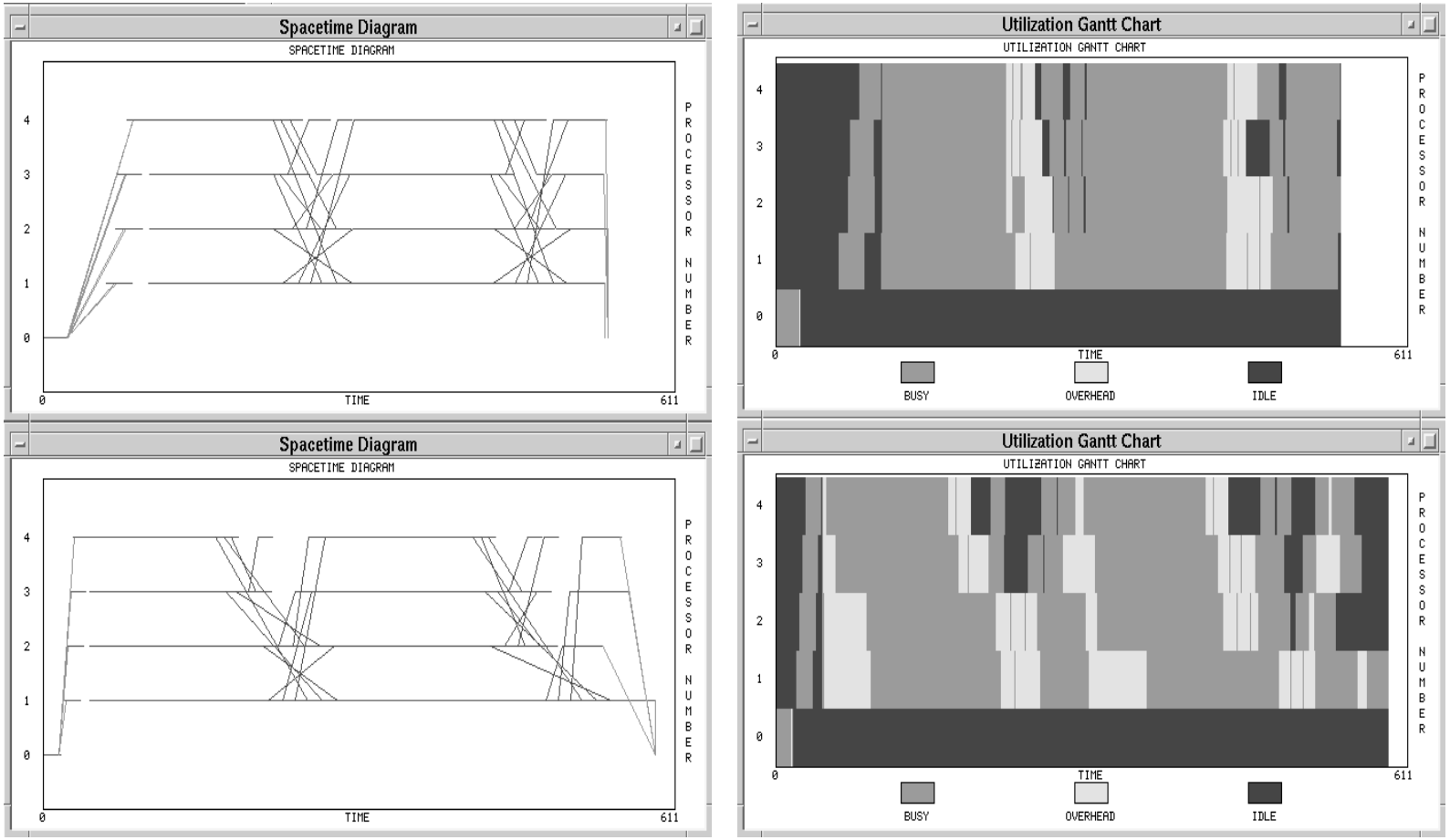


FIG. 6.3 – *FFT2D: diagrammes espace-temps et de Gantt comparant deux exécutions tracées. La partie supérieure représente l'exécution brute alors que la partie inférieure représente une exécution perturbée aléatoirement (l'unité de temps est de 1 ms.*

dans la trace.

Notons X^L la loi parente des e_i^L – la distribution, la moyenne m et l'écart-type σ de X^L sont inconnus. Pour un ensemble de messages de même taille L, nous disons qu'il y a *adéquation* entre le modèle et les observations de la trace si X^L est de moyenne nulle. Il s'agit donc de tester l'hypothèse

$$H_0 : \overline{X^L} = 0 \quad \text{contre} \quad H_1 : \overline{X^L} \neq 0,$$

ce qui est un test paramétrique usel. Vu que l'écart-type σ de X^L est inconnu, la variable de décision est la variable de Student [82] :

$$T_{n-1} = \frac{\overline{X} - m}{S} \sqrt{n-1},$$

la statistique S étant l'écart-type empirique de l'échantillon. Dans notre cas, pour H_0 $(m = 0)$ contre H_1, la région critique est définie par :

$$|T_{n^L-1}| > k \quad \text{avec} \quad P\left(|T_{n^L-1}| > k\right) = \alpha,$$

$$T_{n^L-1} = \frac{\overline{X^L}}{S} \sqrt{n^L - 1}.$$

Ce test n'est valable que si X^L suit une loi de Gauss, ce qui n'est pas le cas dans le contexte de la modélisation des temps de communication, pour lesquels on connait généralement un délai minimum, mais pas de délai maximum [79]. Toutefois, en raison du théorème central limite, le test s'applique encore si n^L est assez grand ($n^L > 30$ environ) [82]. Dans le cas où n^L est trop petit, on pourra toujours se baser sur l'ensemble des observations de plusieurs traces d'exécution de l'application.

Dans la version actuelle de l'outil de correction, le test n'est effectué que pour l'ensemble de messages de taille L dont le cardinal est le plus élevé. Si ce cardinal est inférieur à 30 on ne peut pas, au vu de la trace, conclure quant à l'adéquation du modèle. Dans une version future, on prévoit d'effectuer le test pour tous les ensembles de messages de taille L, dont le cardinal est supérieur à 30 ; le modèle de communication sera accepté si le test est positif pour tous ces ensembles de messages.

6.3.3 Exemple d'utilisation du test

L'outil de correction `tico` écrit le résultat du test statistique dans un fichier bilan. La figure 6.2 montre le fichier bilan obtenu par la correction d'une trace d'exécution d'une FFT bi-dimensionnelle (FFT-2D) [15] sur $P = 4$ processeurs. Dans cette exécution, c'est l'ensemble des messages de taille

L=256Ko qui est de cardinal maximal $(n^L = 36)$[4]. La section `TRSIM MODEL CHECK` du fichier bilan contient les valeurs numériques des paramètres du modèle $\beta + L\tau$ fournies par l'utilisateur (`beta`, `tau`), le nombre n^L de temps de communication observés (`NBCHECK`), la moyenne et l'écart-type des erreurs, ainsi que la valeur de la variable de Student t (7,642). La valeur critique à $\alpha = 0,05$ pour un T_{35} est $k = 2,44$, largement dépassée par t : le test d'adéquation du modèle pour les messages de taille L est donc négatif (`REJECT`). Finalement, les deux dernières lignes donnent le minimum, la moyenne et le maximum des temps de communication mesurés (`MEASU`) et prédits par le modèle (`MODEL`).

La FFT-2D [15], application sur laquelle on teste le modèle, effectue deux échanges totaux comme le montrent les visualisations de la figure 6.3. Lors d'un échange total, l'application envoie sur le réseau, en même temps, 12 messages ($P(P-1)$) d'une taille de 256Ko chacun. La charge du protocole TCP-IP/Switch est donc bien plus élevée que lors de l'exécution du *ping pong* qui a permis d'ajuster les paramètres du modèle. Les valeurs reportées dans les deux dernières lignes du fichier bilan de la figure 6.2 montrent que cette augmentation de la charge induit une augmentation sensible des temps de communication, ce qui conduit au rejet du modèle[5].

Nous avons effectué une deuxième exécution de la FFT-2D en perturbant les différents flots d'exécution aléatoirement et de façon disymétrique. Ceci a pour effet de diluer les communications dans le temps, comme le montre la partie inférieure de la figure 6.3 (sur le diagramme de Gantt, les perturbations aléatoires sont représentées par les états *overhead* supplémentaires). Le fichier bilan du test du modèle est montré sur la figure 6.4. Par le simple fait de diluer les communications dans le temps, les valeurs observées et celles estimées par le modèle sont considérablement rapprochées et le modèle $\beta+L\tau$ reste valable (`ACCEPT`).

Ces expériences démontrent le rôle crucial que joue la charge dans la modélisation des temps de communication avec le protocole TCP/IP-Switch. Dans sa version actuelle, `tico` permet seulement d'utiliser un modèle du type $\beta + L\tau$, insuffisant pour la modélisation de ce protocole. Dans une extension future, `tico` calculera une ou plusieurs variables de charge, comme par exemple le nombre total d'octets envoyés et non encore reçus. L'utilisa-

4. Pour cet exemple, on a du recourir à l'ensemble des observations de 3 traces d'exécution, une seule exécution n'effectuant pas assez de communications pour atteindre un minimum de 30 observations.

5. Notons que ces observations ont été faites sur le protocole TCP/IP-Switch. Des protocoles plus performants, utilisant directement le Switch, comme celui employé par MPI-F [26], permettent d'obtenir des temps de communication «quasiment insensibles au bruit» – le modèle $\beta + L\tau$ est valable même en présence de bruit [40].

```
NBCOM=52 NBCOMF=50 (54%) NBCOMFA=76 (83%) NBSIM=38 (41%) NBSIM_SM=67 (73%)

Alpha correction is on
Simulation mode is MINSIM
Packing and unpacking delays are NOT removed

TRSIM MODEL CHECK

  BETA = 2902.616964
  TAU = 0.688442
  NBCHECK = 34
  MEAN OF RESIDUALS = -0.022589
  STDEV OF RESIDUALS = 0.069625
  k_33 (0.05) = 2.04
  t VARIABLE = -1.891 [ACCEPT]
  MIN:MEAN:MAX OF MEASU TRANSIT = 0.003769:0.162536:0.285395
  MIN:MEAN:MAX OF MODEL TRANSIT = 0.002914:0.166968:0.183374
```

FIG. 6.4 – *FFT-2D: bilan de la correction d'une exécution perturbée aléatoirement de façon non symétrique (cf. aussi la partie inférieure de la figure 6.3).*

teur pourra alors intégrer ces variables explicatives supplémentaires dans ses modèles.

6.4 La prédiction des performances

6.4.1 Intérêt de la prédiction des performances

Jusqu'à présent, nous nous sommes intéressés à la mesure d'une exécution d'une application donnée sur un système donné. Les outils de construction de temps global et de correction de l'effet de sonde nous permettent d'obtenir une trace d'exécution aussi précise que possible. L'exploitation de cette trace permet d'améliorer l'application pour en diminuer le temps d'exécution (*performance tuning*, en anglais).

Cette approche, si elle permet d'augmenter les performances des exécutions, ne permet pourtant pas de prévoir les performances de l'application dans un autre environnement d'exécution (environnement émulé), différent de celui auquel on a accès pour faire les mesures (environnement cible). La technique de *prédiction de performances* que nous traitons dans cette section utilise une trace d'exécution d'une application dans l'environnement cible, pour prédire les performances de cette même application dans un autre en-

vironnement. Cette prédiction se base sur un modèle caractérisant les performances des ressources de l'environnement émulé (vitesse des processeurs, bande passante et latence du médium de communication...). Citons quelques uns des nombreux avantages d'une telle technique de prévision :

- elle permet d'évaluer le gain en performances qu'apporterait l'achat d'une extension coûteuse de l'environnement actuel (acquisition de cartes réseau plus performantes, par exemple) ;
- elle permet d'évaluer le gain en performances d'un portage d'une application existante sur une bibliothèque de communication plus performante avant d'acheter cette bibliothèque ou de se lancer dans le portage effectif.

Bien entendu, la prédiction n'est possible que si l'on dispose d'un modèle caractérisant les performances de l'environnement à émuler. La publication de résultats de jeux d'essais, comme PARKBENCH [35] par exemple, fournit les valeurs numériques des paramètres classiques d'un grand nombre d'environnements.

6.4.2 L'outil de prédiction de Tape/PVM

Dans `tico`, nous avons intégré une extension très simple qui permet de prévoir les performances d'une exécution sur une autre bibliothèque que PVM, mais sur la même machine, en exploitant une trace d'exécution PVM fournie par Tape/PVM. Sur notre environnement cible, le IBM-SP2, il existe différentes possibilités pour développer une application parallèle communiquant par messages : le programmeur a le choix entre PVM ou MPI, par exemple. MPI-F [26] étant beaucoup plus performante que la version standard de PVM utilisant TCP/IP-Switch, l'utilisateur d'un code PVM peut se demander ce qu'il gagnerait par un portage de son code sur MPI-F. En MPI, contrairement à PVM, les données d'un message sont directement lues depuis (envoi) ou recopiées dans (réception) les structures de données fournies par l'utilisateur, ce qui élimine les surcoûts dus à l'empaquetage et au dépaquetage des messages (primitives `pvm_pk`, `pvm_upk`). Par ailleurs, MPI offre une bande passante plus élevée (20.6 Mo/s contre 1.14 Mo/s en PVM) et une latence bien inférieure (230 μs contre 2903 μs en PVM).

L'utilisateur souhaitant faire une prédiction des performances avec `tico` peut ainsi fournir un modèle des performances de la bibliothèque de communication à émuler et demander la modélisation systématique de tous les temps de communication (option `-sf`) :

```
tico -p -sf -beta 230 -tau 0.048414 -betacpy 63 -taucpy 0.024405 events
```

type	temps exec.	gain
instrumenté	125,2s	—
corrigé	122,6s	2,1%
corrigé sans paquetage	89,1s	28,8%
corrigé sans paquetage sur MPI-F	79,6s	36,4%

TAB. 6.1 – *NAS-CG : différentes prédictions de performances à partir d'une trace d'exécution de la version PVM du noyau CG.*

Les options `-beta` et `-tau` donnent les valeurs du modèle pour la durée des temps de communication ($\beta + L\tau$) alors que `-betacpy` et `-taucpy` donnent les valeurs du modèle pour les temps de blocage à l'émission et à la réception ($\beta_{cpy} + L\tau_{cpy}$). Si cette modélisation permet de caractériser avec précision les communications de MPI-F sur le IBM-SP2 [40], elle n'est pas suffisamment générique pour décrire toute la panoplie des bibliothèques et architectures de communication existantes.

Par ailleurs `tico` peut enlever les délais d'empaquetage et de dépaquetage des messages, simplement en considérant ces délais comme des perturbations directes (option `-p`) ; chaque événement d'empaquetage et de dépaquetage contient la durée d'exécution correspondante parmi ses attributs. Bien évidemment, les traces de prédiction construites par `tico` se trouvent, tout comme les traces corrigées, dans la même classe de comportement que l'exécution instrumentée initiale. La prédiction se limite donc aux applications à comportement déterministe.

6.4.3 Expériences préliminaires

Le tableau 6.1 donne quelques exemples de prédiction de performances à partir de la trace d'exécution de la version PVM du noyau de calcul NAS-CG [91]. Les premières lignes donnent les temps instrumenté et corrigé. Les lignes suivantes montrent les prédictions du gain obtenu par l'élimination des délais de paquetage de PVM (28,8%), puis par le passage sur la bibliothèque MPI-F (36,4%). On remarque le coût énorme du paquetage des messages en PVM. Au moment de la rédaction de cette thèse nous ne disposons pas d'une version MPI des applications du jeux d'essais du NAS : nous ne pouvons donc pas confronter nos prédictions avec une exécution réelle en MPI.

Remarquons que, tout comme la correction des perturbations, la prédiction des performances fournit une trace d'exécution au format Tape/PVM. Elle est donc exploitable par la bibliothèque de lecture de traces et, *a fortiori*,

par tous les outils d'analyse construits sur cette bibliothèque. En particulier, elle peut être convertie au format PICL, ce qui permet de la visualiser avec Paragraph ou Scope.

6.5 Conclusion et perspectives

Dans ce chapitre, nous avons décrit les outils de post-traitement de traces de Tape/PVM. Le mécanisme de construction de temps global et l'outil de correction des perturbations sont la concrétisation de nos travaux sur la qualité représentative des traces, qui ont fait l'objet de la deuxième partie de cette thèse.

Nous avons également présenté deux extensions de l'outil de correction permettant de tester des modèles de temps de communication sur des traces d'exécutions réelles et de faire de la prédiction des performances des exécutions sur d'autres plate-formes que celle qui est disponible pour faire les expériences. Nous avons donné des exemples d'application de ces extensions.

Tous ces outils sont disponibles et documentés dans la distribution de Tape/PVM, ce qui les rend accessibles et utilisables sur une large gamme de systèmes parallèles.

Sur le plan expérimental, les estimations fournies par l'outil de prévision restent à être confrontées aux performances réellement obtenues sur la bibliothèque de communication émulée. Sur le IBM-SP2, cela implique le portage de nos jeux d'essais (Jacobi, NAS-CG) sur MPI-F. Par ailleurs, l'outil de correction des perturbations doit être testé sur des applications à comportement non-déterministe. A ce propos, nous voulons étendre l'outil de correction pour qu'il détecte un changement d'ordre des messages induit par la perturbation de l'outil de trace. Cette extension permettrait de dégager une ou plusieurs applications dont le comportement est effectivement altéré par la présence du traceur. Le traçage de ces applications peut alors être effectué conjointement avec un mécanisme de réexécution déterministe, comme nous l'avons déjà suggéré dans le chapitre 4.

Une version future des outils de correction, de test de modèles et de prévision des performances supportera d'autres modèles de communication que le simple modèle linéaire d'une seule variable $\beta + L\tau$ supporté actuellement. En particulier, ces modèles pourront utiliser des variables modélisant la charge des liens de communication : lors du parcours de la trace l'outil de correction, de test ou de prévision évaluera automatiquement ces variables de charge en se basant sur le volume de données émises mais non encore reçues.

Actuellement, l'outil de prévision de performances permet seulement d'estimer les performances obtenues avec une autre bibliothèque de communi-

cation sur le même système. Nous pensons qu'il est possible d'étendre ce mécanisme pour prédire les performances d'une exécution réellement parallèle à partir d'une exécution pseudo-parallèle sur un seul processeur. Cela permettrait aux programmeurs de prédire les performances parallèles de leur application à partir d'une trace d'exécution sur un seul processeur, multiprogrammant les différents processus de l'application. Ce cas est très fréquent, car PVM permet justement d'implanter et de mettre au point une application parallèle sur une seule station de travail. La prévision des performances parallèles se baserait sur un modèle simple de la multiprogrammation de cette station de travail et sur un modèle des temps de communication de la machine parallèle à émuler.

Quatrième partie

Conclusion

Bilan et perspectives

Bilan

L'objectif de ce travail de thèse était de proposer une méthodologie d'ajustement de la qualité des traces d'exécution obtenues par voie logicielle. Notre démarche s'est orientée selon les deux principaux désavantages des traceurs logiciels par rapport aux traceurs matériels ou hybrides, à savoir :

1. l'absence de référence globale de temps physique,

2. les perturbations infligées aux applications tracées.

Après une présentation générale du domaine de la mesure et de l'analyse des performances dans la première partie, nous traitons ces deux problèmes dans la deuxième partie. Le chapitre 3 a montré qu'une méthode statistique d'estimation de temps global, combinée avec un mécanisme d'interpolation, donne un accès confortable à une référence globale de temps suffisamment précise pour permettre la datation cohérente des événements répartis. Notre contribution essentielle à ce domaine est l'étude précise des résidus du modèle linéaire, modèle sur lequel se basent les méthodes statistiques. Cette étude nous a permis de proposer une méthode d'échantillonnage réalisant un équilibre, que nous pensons optimal, entre les délais d'échantillonnage et la précision de l'estimation obtenue [67].

Le chapitre 4 a présenté nos modèles de perturbation des applications communiquant par messages et nos algorithmes de correction basés sur ces modèles. En particulier, nous avons proposé une variante de l'algorithme de correction de Sarukkai-Malony [83] qui permet de diminuer considérablement le nombre de modélisations des temps de communication, afin de garder un maximum d'informations des traces originales. Pour les applications à comportement déterministe, effectuant une décomposition statique du travail, la correction des perturbations peut reconstruire exactement la dynamique d'une exécution non-instrumentée (aux perturbations indirectes près). Pour ce type d'application, sur le IBM-SP2, la correction a permis d'enlever entre

70% et 95% des perturbations totales sur le temps d'exécution. Sur la machine Meganode, elle a enlevé quasiment l'intégralité des perturbations. Pour des applications à comportement non-déterministe, comme celles qui décomposent le travail dynamiquement, le mécanisme de correction construit une trace appartenant à la même classe de comportement que la trace de départ (*approximation conservatrice*). Nous avons montré que cette trace peut représenter un comportement peu probable, voire impossible, de l'application. Une solution possible à ce problème est d'appliquer le mécanisme de correction sur une trace d'une exécution réexécutée de façon déterministe selon un ordre de référence, représentatif d'une exécution non-perturbée.

En conclusion, nous pensons que la correction des perturbations jouera un rôle de plus en plus important dans le cadre d'un environnement de trace pour l'évaluation des performances. L'intérêt croissant des développeurs d'environnements (AIMS [94], Annai/PMA [23]) pour la caractérisation et la correction de l'effet de sonde confirme cette tendance.

La troisième partie de la thèse a présenté l'outil de trace Tape/PVM. Nous avons décrit son architecture ainsi que ses outils de post-traitement de traces qui concrétisent nos travaux sur le réajustement de qualité des mesures, présentés dans la deuxième partie. Nous avons également présenté des extensions «naturelles» à l'outil de correction des perturbations, supportant le test d'adéquation de modèles de communication ainsi que la prévision des performances. L'intégration de ces outils dans le traceur Tape/PVM, disponible publiquement, permet de les diffuser largement et on espère avoir un retour d'expériences quant à leur utilisation sur d'autres machines que celles auxquelles on a eu accès durant ce travail de thèse.

Perspectives

A court terme

Nos expériences ont montré que la perturbation sur le temps d'exécution d'une application induite par la présence du traceur Tape/PVM est relativement faible : sur les applications numériques en PVM que nous avons testées, elle ne dépasse que rarement les 15% sur le temps d'exécution non-instrumenté. Nous avons vu que cela était dû au fait que nous n'avons tracé que les appels à la bibliothèque de communication PVM, appels dont la fréquence n'a pas dépassé 0,8 événements par milliseconde. Tape/PVM sera étendu pour *tracer les entrées et les sorties de boucles et de procédures*, comme dans le traceur de PABLO [75]. A part la fonctionnalité supplémentaire qu'une telle extension apportera à l'outil de trace, elle permettra de

tester la correction sur des traces d'exécutions plus fortement perturbées. A ce propos, une expérience préliminaire d'instrumentation d'une boucle de calcul a provoqué une perturbation de plus de 100% du temps d'exécution et l'outil de correction a pu enlever 89% de cette perturbation.

Les outils de correction, de test d'adéquation de modèles de communication et de prévision de performances seront étendus pour traiter des modèles de communication plus complexes que le simple modèle linéaire $\beta + L\tau$, à une variable, supporté actuellement. Ces modèles pourront utiliser des variables modélisant la charge des liens de communication, variables que les outils d'analyse évalueront lors du parcours de la trace. Cela permettra en particulier de tester l'adéquation de modèles plus complexes, comme ceux proposés dans [89], sur des traces d'exécution réelles.

A moyen et à long terme

Dans sa version actuelle, l'outil de correction des intrusions construit une approximation conservatrice d'une dynamique d'exécution non-instrumentée i.e. l'ordre de lecture des messages observé lors de l'exécution instrumentée est conservé. Ce mécanisme sera étendu pour pouvoir *détecter des changements d'ordre de lecture induit par l'instrumentation*. Notre idée est de combiner l'algorithme de correction post-mortem avec la technique de détection de conditions de conflit (*race conditions*, en anglais) proposée par Netzer et Miller [74], ce qui implique la construction post-mortem d'un temps logique vectoriel lors du parcours des traces. Cette extension permettra de *dégager des applications à comportement non-déterministe dont le comportement est effectivement altéré par l'effet de sonde*. Le mécanisme de correction sera alors appliqué aux traces de ces applications réexécutées de façon déterministe. Cette démarche fournira un support expérimental essentiel à la validation de l'approche par phases multiples proposée dans [87] qui est actuellement implantée dans l'environnement ATHAPASCAN.

Actuellement, de plus en plus de développeurs d'applications parallèles se tournent vers la programmation par processus légers (*threads*, en anglais). La multiprogrammation de plusieurs processus légers sur un seul processeur offre un moyen confortable pour recouvrir les latences de communication par le calcul. La présence de processus légers devra être envisagée aussi bien au niveau du modèle de correction qu'au niveau du traceur Tape/PVM.

Bibliographie

[1] Abali (B.) et Stunkel (C.B.). – Time synchronization on SP1 and SP2 parallel systems. *In : Proceedings of the 9th International Parallel Processing Symposium*, pp. 666–671. – Santa Barbara, CA, avril 1995.

[2] Arrouye (Y.). – *The* TAPE *Reference Manual.* – LMC-IMAG, BP53, F-38041 Grenoble Cedex 9, septembre 1992.

[3] Arrouye (Y.). – *Environnements de visualisation pour l'évaluation des performances des systèmes parallèles : étude, conception et réalisation.* – Thèse de PhD, Institut National Polytechnique de Grenoble, novembre 1995.

[4] Arrouye (Y.), Bouvry (F.), Bouvry (P.), Kitajima (JP.), Michallon (P.), Prévost (J.), Roch (JL.) et Villard (G.). – *Manuel du Meganode.* – Laboratoire de Modélisation et de Calcul, BP53, F-38041 Grenoble Cedex 9, avril 1992.

[5] Audenaert (K.) et Levrouw (L.). – Space efficient data race detection for parallel programs with series-parallel task graphs. *In : Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing.* pp. 508–515. – Sanremo, Italie, janvier 1995.

[6] Aydt (R. A.). – *The Pablo Self-Defining Data Format.* – Rapport technique, University of Illinois, Urbana, Illinois 61801, Department of Computer Science, avril 1994.

[7] Azéma (L.) et Collet (M.). – *Développement du traceur Tape/PVM.* – Rapport de stage de 2e année, INPG (Ensimag), 1994.

[8] Bailey (D.), Barszcz (E.), Barton (J.), Browning (D.), Carter (R.), Dagum (L.), Fatoohi (R.), Fineberg (S.), Frederickson (P.), Lasinski (T.), Schreiber (R.), Simon (H.), Venkatakrishnan (V.) et Weeratunga (S.). – *The NAS Parallel Benchmarks.* – Rapport technique n° RNR-94-007, NASA Ames Research Center, mars 1994.

[9] Barreto (R. Menna). – Global Time in Multiprocessor Systems. – février 1993. LMC-IMAG, BP53, F-38041 Grenoble Cedex 9.

[10] Beguelin (Adam L.). – *Xab: a tool for monitoring PVM programs.* – Research paper n° CMU-CS-93-164, Pittsburgh, PA, USA, School of Computer Science, Carnegie Mellon University, 1993.

[11] Bemmerl (T.) et Haunerdinger (J.). – Hardware instrumentation techniques for multimicroprocessors. *In : Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing*, éd. par Cosnard (M.), Barton (M.H.) et Vanneschi (M.). pp. 93–102. – Pisa, Italie, 1988.

[12] Bernaschi (M.) et Iannello (G.). – Efficient Collective Communication Operations in PVMe. *In : EuroPVM'95*, éd. par Dongarra (J.), Gengler (M.), Tourancheau (B.) et Vigouroux (X.). pp. 233–238. – Lyon, 1995.

[13] Bertsekas (D. P.) et Tsitsiklis (J. N.). – *Parallel and distributed computation.* – Prentice-Hall, 1989.

[14] Bouvry (P.). – *Placement de tâches sur ordinateurs parallèles à mémoire distribuée.* – Thèse de PhD, Institut National Polytechnique de Grenoble, octobre 1994.

[15] Calvin (C.). – *Minimisation du sur-coût des communications dans la parallélisation des algorithmes numériques.* – Thèse de PhD, Institut National Polytechnique de Grenoble, juillet 1995.

[16] Carpenter (R.J.). – Performance Measurement Instrumentation for Multiprocessor Computers. *In : High Performance Computer Systems*, éd. par Gelenbe (E.). – Amsterdam, 1988.

[17] Christaller (M.), Castaneda Retiz (M.R.) et Gautier (T.). – Control Parallelism on top of PVM : the ATHAPASCAN Environment. *In : EuroPVM'95*, éd. par Dongarra (J.), Gengler (M.), Tourancheau (B.) et Vigouroux (X.). pp. 71–76. – Lyon, 1995.

[18] Cristian (F.). – Probabilistic Clock Synchronization. *Distributed Computing*, vol. 3, 1989, pp. 146–158.

[19] Denneulin (Y.), Geib (J.-M.) et Méhaut (J.-F.). – A multithreaded-based methodology to solve irregular problems. – 1996. `http://www.lifl.fr/~mehaut/espace.html`.

[20] Dennis (J.E.), Gay (D.M.) et Welsch (R.E.). – An adaptive nonlinear least-squares algorithm. *ACM Transactions on Mathematical Software*, vol. 7, n° 3, septembre 1981.

[21] Duda (A.), Harrus (G.), Haddad (Y.) et Bernard (G.). – Estimating Global Time in Distributed Systems. *In : Proc. 7th Int. Conf. on Distributed Computing Systems.* – Berlin, 1987.

[22] Dunigan (T.H.). – Hypercube Clock Synchronization. *Concurrency Practice and Experience*, vol. 4, n° 3, mai 1992, pp. 257–268.

[23] Endo (A.) et Wylie (B.J.N.). – *Annai/PMA Instrumentation Intrusion Management of Parallel Program Profiling.* – CSCS-TR-95-05, Centro Svizzero di Calcolo Scientifico (CSCS), novembre 1995. `http://www.cscs.ch`.

[24] Fagot (A.) et de Kergommeaux (J. Chassin). – Formal and experimental validation of a low-overhead execution replay mechanism. *In : Euro-Par'95 Parallel Processing*, éd. par Haridi (S.), Ali (K.) et Magnusson (P.). pp. 167–178. – Stockholm, Suède, août 1995.

[25] Fidge (Colin). – Timestamps in message-passing systems that preserve the partial ordering. *Australian Computer Science Communications*, vol. 10, n° 1, février 1988.

[26] Franke (H.), Wu (E.), Rivière (M.), Pattnaik (P.) et Snir (M.). – *MPI Programming Environment for IBM SP1/SP2.* – Rapport technique, IBM T.J. Watson Research Center, P.O. 218, Yorktown Heights, NY 10598., 1995.

[27] Gait (J.). – A Probe Effect in Concurrent Programs. *Software – Practice and Experience*, vol. 16, n° 3, mars 1986, pp. 225–233.

[28] Gannon (J.A.), Williams (K.J.), Andersland (M.S.), Lumpp (J.E.) et Casavant (T.L.). – Using Perturbation Tracking to Compensate for Intrusion in Message-Passing Systems. *In : Proceedings of the 14th International Conference on Distributed Computing Systems*, pp. 414–421. – Poznan, Poland, juin 1994.

[29] Geist (Al), Beguelin (Adam), Dongarra (Jack), Jiang (Weicheng), Manchek (Robert) et Sunderam (Vaidy). – *PVM 3 Users Guide and Reference manual.* – Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, mai 94.

[30] Geist (G. A.), Heath (M. T.), Peyton (B. W.) et Worley (P. H.). – *A user's guide to PICL: a portable instrumented communications library.* – Rapport technique n° ORNL/TM-11616, Oak Ridge, Tennessee, Oak Ridge National Laboratory, janvier 1992.

[31] Graham (S.L.), Kessler (P.B.) et McKusick (M.K.). – `gprof`: A call graph execution profiler. *SIGPLAN Notices*, vol. 17, n° 6, juin 1982, pp. 120–126. – *Proceedings of the ACM SIGPLAN '82 Symposium on Compiler Construction.*

[32] Guidec (F.) et Mahéo (Y.). – POM : une machine virtuelle parallèle incorporant des mécanismes d'observation. *Calculateurs Parallèles*, vol. 7, n° 2, 1995, pp. 101–118.

[33] Haddad (Y.). – *Performance dans les systèmes répartis : des outils pour les mesures.* – Thèse de PhD, Université de Paris-Sud - Centre d'Orsay, 1988.

[34] Heath (M. T.) et Finger (J. E.). – *ParaGraph: A Tool for Visualizing Performance of Parallel Programs.* – Rapport technique, Oak Ridge National Laboratory, juin 1994.

[35] Hockney (Roger W.). – Public International Benchmarks for Parallel Computers. – février 1994. `http://www.netlib.org/parkbench/parkbench.ps`.

[36] Hollingsworth (J.K.). – An online computation of critical path profiling. *In : ACM SIGMETRICS Symposium on parallel and distributed tools.* – Philadelphia, PA, mai 1996. `http://www.cs.umd.edu/~hollings/papers.htm`.

[37] Hollingsworth (J.K.), Lumpp (J.E.) et Miller (B.P.). – Techniques for performance measurement of parallel programs. *Parallel Computers: Theory and Practice (IEEE Press)*, 1995, pp. 225–240. – `http://www.cs.umd.edu/~hollings/papers.htm`.

[38] Hollingsworth (J.K.) et Miller (B.P.). – Parallel program performance metrics: A comparison and validation. *In : Proceedings of the Conference on Supercomputing.* pp. 4–13. – Minneapolis, MN, USA, novembre 1992.

[39] Irvin (Bruce) et Miller (Bart). – A performance tool for high-level parallel programming languages. *In : IFIP WG10.3 Conference on Programming Environments for Massively Parallel Distributed Systems.* – Ascona, Suisse, avril 94.

[40] Iskander (K.). – *Modélisation des communications dans le SP1.* – Rapport technique, 100 rue des Mathématiques, F-38041 Grenoble Cedex 9, LMC-IMAG, février 1996.

[41] Jard (C.). – La vérification d'exécutions réparties. *In : Actes de l'Ecole de Roscoff sur l'algorithmique répartie.* – CNRS, 1993.

[42] Jard (C.), Jéron (T.), Jourdan (G-V.) et Rampon (J-X.). – A general approach to trace-checking in distributed computing systems. *In : Proceedings of the 14th International Conference on Distributed Computing Systems.* – Poznan, Poland, juin 1994.

[43] Jézéquel (J.M.). – *Outils pour l'expérimentation d'algorithmes distribués sur machines parallèles.* – Thèse de PhD, Université de Rennes 1, octobre 1989.

[44] Jézéquel (J.M.). – *Building a Global Time on Parallel Machines.* – Rapport technique n° 513, Université de Rennes, février 1990. Publication interne.

[45] Johnston (J.). – *Econometric Methods.* – McGRAW-HILL, 1991, third édition.

[46] Jézéquel (J.M.) et Jard (C.). – Building a Global Clock for Observing Computations in Distributed Memory Parallel Computers. *Concurrency Practice and Experience*, vol. 1, n° 8, janvier 1996, pp. 71–89.

[47] Kitajima (J. P.). – *Modèles quantitatifs d'algorithmes parallèles.* – Thèse de PhD, Institut National Polytechnique de Grenoble, 1994.

[48] Kitajima (J. P.) et Plateau (B.). – Modelling parallel program behaviour in ALPES. *Information and Software Technology*, vol. 36, n° 7, 1994, pp. 457–464.

[49] Kitajima (J.P.) et Plateau (B.). – Performance evaluation of vcr1.8c : a virtual router for transputer networks. *Transputers '92*, vol. 26, 1992, pp. 179–186.

[50] Kitajima (J.P.), Plateau (B.), Bouvry (P.) et Trystram (D.). – ANDES: Evaluating Mapping Strategies with Synthetic Programs. *Journal of Systems Architecture*, 1996. – À Paraître.

[51] Klar (R.). – Event-Driven Monitoring of Parallel Systems. *In : Workshop on Performance Measurement and Visualization of Parallel Systems.*

[52] Kohl (J.A.) et Geist (G.A.). – The PVM 3.4 Tracing facility and XPVM 1.1. *In : Proceedings of the 29th Hawaii International Conference on System Sciences.* – Hawaii, 1996.

[53] Krakowiak (S.). – *Principe des systèmes d'exploitation des ordinateurs.* – Dunod, 1987.

[54] Lamport (L.). – Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, vol. 21, n° 7, juillet 1978, pp. 558–565.

[55] Lamport (L.) et Melliar-Smith (P.M.). – Synchronizing Clocks in the Presence of Faults. *Journal of the ACM*, vol. 32, n° 1, janvier 1985, pp. 52–78.

[56] Larus (James R.). – Efficient program tracing. *IEEE Computer*, vol. 26, n° 5, mai 1993.

[57] Lehr (T.). – *Compensating for perturbation by software performance monitors in asynchronous computations.* – Thèse de PhD, Carnegie Mellon University, avril 1990.

[58] Leu (E.). – *La réexécution, pierre angulaire de la mise au point de programmes parallèles.* – Thèse de PhD, Ecole Polytechnique Fédérale de Lausanne, 1992. Thèse No 1049.

[59] Leu (E.) et Schiper (A.). – Execution replay: a mechanism for integrating a visualization tool with a symbolic debugger. *In : Parallel Processing:CONPAR92-VAPPV.* Lyon, France. – Springer-Verlag.

[60] Lewis (L. L.). – An Introduction to Frequency Standards. *Proceedings of the IEEE*, vol. 79, n° 7, juillet 1991, pp. 927–935.

[61] Lumpp (J.E.), Casavant (T.L.), Gannon (J.A.), Williams (K.J.) et Andersland (M.S.). – *Practical Use of Traces for Development of Message-Passing Programs.* – TR-ECE-921020, ECE/UIOWA, octobre 1992.

[62] Lumpp (J.E.), Casavant (T.L.), Gannon (J.A.), Williams (K.J.) et Andersland (M.S.). – Analysis of Communication Patterns for Modeling Message Passing Programs. *In : Proceedings of the International Workshop on Principles of Parallel Computing (OPOPAC).* – Lacanau, France, novembre 1993.

[63] Maillet (E.). – Issues in Performance Tracing with Tape/Pvm. *In : EuroPVM'95, HERMES (ISBN 2-86601-497-9)*, éd. par Dongarra (J.), Gengler (M.), Tourancheau (B.) et Vigouroux (X.), pp. 143–148. – Lyon, 1995.

[64] Maillet (E.). – Les traceurs logiciels d'évaluation de performances. Problématique et solutions. *In : Actes de RenPar7.* – Mons, Belgique, juin 1995.

[65] Maillet (E.). – *TAPE/PVM an efficient performance monitor for PVM applications – User Guide.* – 100 rue des Mathématiques, F-38041 Grenoble Cedex 9, 1995. `ftp://ftp.imag.fr/imag/APACHE/TAPE`.

[66] Maillet (E.). – Issues in Performance Tracing with Tape/Pvm. *Calculateurs Parallèles : numéro thématique PVM*, vol. 8, n° 2, 1996, pp. 189–202. – Version augmentée de l'article de même titre publié dans les actes de *EuroPVM'95*.

[67] Maillet (E.) et Tron (C.). – On Effciently Implementing Global Time for Performance Evaluation on Multiprocessor Systems. *Journal of Parallel and Distributed Computing*, vol. 28, juillet 1995, pp. 84–93.

[68] Malony (A. D.). – *Performance Observability.* – Thèse de PhD, University of Illinois, Urbana, septembre 1990.

[69] Malony (A. D.), Reed (D.A.) et Wijshoff (H.A.G.). – Performance Measurement Intrusion and Perturbation Analysis. *IEEE Transactions on parallel and distriuted systems*, vol. 3, n° 4, juillet 1992.

[70] Malony (A.D.) et Reed (D.A.). – Models for performance perturbation analysis. *ACM SIGPLAN NOTICES*, vol. 26, n° 12, décembre 1991, pp. 15–25.

[71] Mattern (F.). – Virtual Time and Global States in Distributed Systems. *International conference on parallel and distributed algorithms, North Holland*, 1988, pp. 215–226.

[72] MPI Forum. – *MPI: A Message-Passing Interface MPI Forum.* – Rapport technique n° CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, mars 94.

[73] Miller (B.P.), Gallaghan (M.D.), Cargille (J.M.), Hollingsworth (J.K.), Irvin (R.B.), Karavanic (K.L.), Kunchithapadam (K.) et Newhall (T.). – The Paradyn parallel performance measurement tool. *IEEE Computer*, vol. 28, n° 11, novembre 1995, pp. 37–46.

[74] Netzer (R.H.B.) et Miller (B.P.). – Optimal tracing and replay for debugging message-passing parallel programs. *In : Proceedings Supercomputing '92*. pp. 502–511. – Minn., MN, novembre 1992.

[75] Noe (R. J.). – *Pablo Instrumentation Environment User's Guide*. – Rapport technique, University of Illinois, Urbana, Illinois 61801, Department of Computer Science, décembre 1994. `http://www-pablo.cs.uiuc.edu/Projects/Pablo/documents.html`.

[76] Plateau (B.). – *APACHE: Algorithmique Parallèle et pArtage de CHargE*. – IMAG–Équipe APACHE, BP53, F-38041 Grenoble Cedex 9, novembre 1994. Rapport APACHE1.

[77] Poinson (S.), Tourancheau (B.) et Vigouroux (X.). – Distributed monitoring for scalable massively parallel machines. *In : Environments and Tools for Parallel Scientific Computing*, éd. par Dongarra (J.) et Tourancheau (B.). pp. 85–101. – Amsterdam, 1993.

[78] Ramanathan (P.), Kandlur (D.D.) et Shin (K.G.). – Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems. *IEEE Transactions on Computers*, vol. 39, n° 4, avril 1990, pp. 280–283.

[79] Raynal (M.). – *La communication et le temps dans les réseaux et systèmes répartis (tome 1 d'une introduction aux principes des systèmes répartis)*. – 61, Bd Saint-Germain Paris 5e, Éditions Eyrolles, 1991. ISSN 0399-4198.

[80] Reed (D. A.), Olson (R. D.), Aydt (R. A.), Madhyastha (T. M.), Birkett (T.), Jensen (D. W.), Nazief (B. A. A.) et Totty (B. K.). – Scalable Performance Environments for Parallel Systems. *In : Proc. of the 6th Distributed Memory Computing Conference (DMCC-6)*, éd. par Stout (Q.) et Wolfe (M.). pp. 562–569. – Portland, OR, avril 1991.

[81] Reed (D.A.). – Performance Instrumentation Techniques for Parallel Systems. *In : Performance Evaluation of Computer and Communication Systems*, éd. par Donatiello (L.) et Nelson (R.), pp. 463–490. – Springer Verlag, 1993.

[82] Saporta (G.). – *Probabilités, analyse de données et statistiques*. – 27 rue Ginoux, 75737 Paris Cedex 15, Éditions Technip, février 1990. ISBN 2-7108-0565-0.

[83] Sarukkai (S. R.) et Malony (A. D.). – Perturbation analysis of high level instrumentation for SPMD programs. *In: ACM SIGPLAN Notices*, pp. 44–53. – San Diego, CA, juillet 1993.

[84] Seznec (A.) et Mével (Y.). – *Etude des Architectures des Microprocesseurs DEC 21164, IBM POWER2 et MIPS R8000.* – Rapport technique n° 932, IRISA, juin 1995. Publication interne.

[85] Talbi (E.G.). – *Allocation dynamique de processus dans les systèmes distribués et parallèles : Etat de l'art.* – Rapport technique n° TR-162, LIFL, Université de Lille 1, Jan 1995.

[86] Tanenbaum (A.S.). – *Modern Operating Systems.* – Prentice-Hall, 1992.

[87] Teodorescu (F.) et Chassin de Kergommeaux (J.). – Performance evaluation of parallel programs by multiple phases of execution. *In: Proceedings of the 3rd Romanian Conference on Open Systems OSE"95*, pp. 11–14. – Bucharest, Romania, novembre 1995.

[88] Topol (B.), Sunderam (V.) et Alund (A.). – *PGPVM Performance Visualization Support for PVM.* – Rapport technique n° CSTR-940801, Emory University, Atlanta, Dept. of Mathematics and Computer Science, août 1994.

[89] Tron (C.). – *Modèles quantitatifs de machines parallèles : les réseaux d'interconnexion.* – Thèse de PhD, Institut National Polytechnique de Grenoble, décembre 1994.

[90] Tsai (J. J.P.), Kwang-Ya (Fang) et Horng-Yuan (Chen). – A noninvasive architecture to monitor real-time distributed systems. *IEEE Computer*, vol. 23, n° 3, mars 1990, pp. 11–25.

[91] White (S.), Alund (A.) et Sunderam (V.S.). – *Performance of the NAS Parallel Benchmarks on PVM Based Networks.* – Rapport technique n° RNR-94-008, Emory University, Atlanta, Dept. of Mathematics and Computer Science, mai 1994.

[92] Worley (P.H.). – *A new PICL trace file format.* – Rapport technique n° TM-12125, Oak Ridge National Laboratory, juin 1992.

[93] Wylie (B.J.N.) et Endo (A.). – *Design and realization of the Annai integrated parallel programming environment performance monitor and analyzer.* – CSCS-TR-94-07, Centro Svizzero di Calcolo Scientifico (CSCS), août 1994. `http://www.cscs.ch`.

[94] Yan (J.C.). – Performance tuning with AIMS – an automated instrumentation and monitoring system for multicomputers. *In : Proceedings of the 27th Hawaii International Conference on System Sciences.* – Hawaii, 1994.

[95] Yan (J.C.), Sarukkai (S.) et Mehra (P.). – Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit. *Software Practice and Experience*, vol. 25, n° 4, avril 1995, pp. 429–461.

[96] Yan (J.C.), Schmidt (M. A.) et Sarukkai (S.). – Monitoring the Performance of Multidisciplinary Applications on the iPSC/860. *In : Proceedings of SHPCC'94.* – Knoxville, Tennessee, mai 1994.

[97] Yang (C.Q.) et Miller (B.P.). – Critical path analysis for the execution of parallel and distributed programs. *In : Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS).* pp. 366–375. – San Jose, CA USA, 1988.