ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
FRANÇOIS ROBERT

DIMINUTION DU TEMPS DE CALCUL D'UN POSTE DU SIMULATEUR HYPERSIM

MONTRÉAL, LE 1ER MARS 2005

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Michel Lavoie, directeur de mémoire
Département de génie logiciel à l'École de technologie supérieure

M. Louis A. Dessaint, codirecteur
Département de génie électrique à l'École de technologie supérieure

M. Jean-Claude Soumagne, chercheur
Laboratoire de simulation des réseaux à l'Institut de recherche d'Hydro-Québec

M. Pierre Jean Lagacé, président du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY

LE 19 JANVIER 2005

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

DIMINUTION DU TEMPS DE CALCUL D'UN POSTE DANS LE SIMULATEUR HYPERSIM

François Robert

SOMMAIRE

Le mémoire présente les facteurs matériels, logiciels ou algorithmiques qui augmentent le temps de calcul d'un poste, sans pour autant augmenter la précision du simulateur Hypersim du laboratoire de recherche d'Hydro-Québec. Hypersim effectue la simulation d'un réseau électrique en utilisant un ordinateur parallèle afin de fonctionner en temps réel. Le réseau est décomposé en postes, lignes et systèmes de commande qui sont associés chacun directement à une tâche informatique qui peut s'exécuter en parallèle sur un des processeurs.

L'étude, réalisée en deux étapes successives, identifie d'abord les facteurs logiciels ou matériels susceptibles de diminuer le temps de calcul, mais sans modifier la nature du calcul. Ensuite, la seconde étape présente les possibilités de parallélisation ou de modification de l'algorithme de calcul.

L'étude démontre que la parallélisation de la décomposition LU donne de pires résultats que la version non parallélisée, mais elle présente une modification de l'algorithme de calcul qui permet de diminuer, de plus de la moitié, le temps de calcul d'une station.

REDUCING COMPUTATION TIME OF A STATION IN THE HYPERSIM SIMULATOR

François Robert

ABSTRACT

This report presents hardware, software or algorithmic factors that can increase computational time without affecting precision of the Hydro-Québec Hypersim simulator. Hypersim simulates in Real-Time, using a parallel computer, the behaviour of Electrical Networks. Networks are broken down in stations, lines and command systems that constitute the numerous computer tasks. Tasks are distributed on different processors of a parallel machine to meet real time constraint.

The project is conducted in two phases. In the first phase, hardware and software factors are looked into. In the second phase, algorithmic factors are examined.

The study demonstrates that parallelization of the LU decomposition is worse than the non-parallel algorithm. However, a new algorithm that cuts by half the computational time is proposed.

REMERCIEMENTS

Je tiens à remercier chaleureusement mon directeur de recherche, Michel Lavoie et mon codirecteur, Louis-A. Dessaint, titulaire de la chaire de recherche de Trans-Énergie, pour leur appui indéfectible. A plusieurs reprises, j'ai eu l'impression de rencontrer un mur. A chaque fois, ils ont fait preuve de patience et ils ont su m'aider à trouver une nouvelle piste de solution. De plus, sans le soutien financier de Louis-A. Dessaint, ce travail n'aurait pas été possible.

Je veux également souligner l'aide précieuse que m'ont apportée les personnes du laboratoire de recherche en simulation de réseaux d'Hydro-Québec, en particulier Jean-Claude Soumagne pour ses précieux conseils et Sylvain Guérette pour son empressement à m'aider à résoudre mes problèmes de simulation et d'accès aux ordinateurs.

Finalement, je remercie ma conjointe, Valérie, pour son encouragement et ses efforts pour me libérer du temps, et ce, malgré la naissance heureuse de notre fille, Constance.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

INTRODUCTION

Le simulateur de réseaux électriques Hypersim est développé à l'Institut de recherche en électricité du Québec (IREQ). Ce simulateur sert à étudier et à prévoir le comportement d'un réseau électrique à haute puissance. En particulier, il peut vérifier le comportement d'un nouvel équipement, avant que celui-ci soit mis en opération.

Le simulateur Hypersim utilise des algorithmes de calcul numériques, c'est-à-dire que les valeurs des courants et des tensions sont discrétisés et les valeurs de chaque nœud du réseau sont calculées pas à pas. Le simulateur a deux modes d'opération : en temps réel ou en temps différé.

Une interface graphique permet de définir le réseau à simuler. Une fois le réseau défini, un générateur de code crée le modèle numérique de la simulation. Ce modèle est principalement composé de vecteurs de données et de calculs matriciels à effectuer sur ces vecteurs. Que la simulation soit effectuée en temps réel ou non, le calcul du réseau est parallélisé selon la topologie du réseau.

La capacité du simulateur d'analyser, en temps réel, des réseaux complexes est déterminée par la quantité de nœuds et par le nombre de variables impliquées pour chaque nœud. De façon plus spécifique, le temps de calcul d'un nœud dépend de la puissance brute de calcul du processeur, mais aussi de l'agencement des données, de l'algorithme utilisé et des options d'optimisation du compilateur.

Ce mémoire présente les facteurs matériels, logiciels ou algorithmiques qui augmentent le temps de calcul d'un poste, sans pour autant augmenter la précision de la simulation. Il présente aussi des modifications qui permettront de diminuer le temps de calcul d'un tel poste.

L'étude a été réalisée en deux étapes successives. La première étape consistait à identifier les facteurs logiciels ou matériels susceptibles de diminuer le temps de calcul, mais sans modifier la nature du calcul. La seconde étape consistait à étudier les possibilités de parallélisation ou de modification de l'algorithme de calcul. Le mémoire est articulé en deux sections distinctes et successives.

En conclusion de l'étude, ce mémoire propose donc une méthode permettant de diminuer de moitié le temps de calcul d'un poste, mais aussi toute une série de résultats intermédiaires permettant des gains.

CHAPITRE 1

NATURE DU PROBLÈME

1.1 Décomposition des réseaux selon la méthode de EMTP[1]

Le simulateur de l'IREQ est basé sur le simulateur de phénomènes transitoires EMTP. Le simulateur EMTP décompose un réseau en différents éléments de base: résistances, inductances, condensateurs, interrupteurs et autres. Par exemple, une ligne de transport va être modélisée par une résistance, une inductance de ligne et une capacité parasite.

1.1.1 Modélisation d'une résistance

Selon l'approche nodale, le courant est donné et la tension est cherchée. Ainsi, dans le cas d'une résistance linéaire, l'équation devient simplement :

$$v(t) = R\, i(t) \tag{1.1}$$

Si la résistance n'est pas constante, c'est-à-dire qu'elle varie dans le temps, alors la valeur R devient une fonction du temps, ce qui est le cas pour l'approximation des prochaines composantes.

1.1.2 Modélisation d'un condensateur

Pour la modélisation d'un condensateur linéaire et invariant dans le temps, l'équation du courant en fonction de la tension est donnée par :

$$i(t) = C \frac{dv}{dt} \tag{1.2}$$

---

[1] Basé sur le mémoire de maîtrise de Abdou-Rahmani Sana, p. 12 et ss.

L'équation aux différences devient

$$\Delta v = \frac{i\,\Delta t}{C} \tag{1.3}$$

Ce qui fait que pour chaque pas de calcul, on a

$$V_{n+1} = \frac{\Delta t\, I_{n+1}}{C} + V_n \tag{1.4}$$

Il est à noter que ce modèle se base sur une intégration numérique de type Euler arrière, c'est-à-dire que l'intégrale est approximée en utilisant

$$\int f(t)\,dt \cong \sum_i f(i)\,\Delta t \tag{1.5}$$

Une méthode plus précise est l'intégration trapézoïdale où l'intégrale est approximée par

$$\int f(t)\,dt \cong \sum_i \frac{(f(i+1) - f(i))}{2}\,\Delta t \tag{1.6}$$

Dans ce texte, nous avons retenu l'intégration de Euler arrière afin de ne pas alourdir le texte. Toutefois, l'intégration trapézoïdale utilisée dans Hypersim, permet de diminuer l'erreur accumulée.

1.1.3 Modélisation d'une inductance

Pour une inductance linéaire et invariante dans le temps, l'équation de la tension en fonction du courant est donnée par:

$$v(t) = L\,\frac{di}{dt} \tag{1.7}$$

L'équation aux différences devient

$$v = L \frac{\Delta i}{\Delta t} \tag{1.8}$$

Ce qui fait

$$V_{n+1} = \frac{L}{\Delta t} (I_{n+1} - I_n) \tag{1.9}$$

1.1.4 Modélisation d'un interrupteur

Les interrupteurs sont modélisés chacun par une résistance ayant deux valeurs possibles. Si l'interrupteur est fermé, alors la valeur de la résistance est très petite. Par contre, si l'interrupteur est ouvert, la valeur de la résistance est alors très grande. La présence de disjoncteurs ou d'interrupteurs rend la simulation plus ardue, car ils peuvent représenter des discontinuités dans le calcul itératif.

1.1.5 Modélisation des autres composantes

Il existe plusieurs autres composantes telles que les transformateurs, les convertisseurs de puissance. Dans tous les cas, ils sont modélisés avec les éléments de base : résistance, condensateur, inductance et interrupteur. Toutefois, il est à noter que certains éléments n'ont pas toujours un comportement linéaire. Par exemple, pour un transformateur entrant en saturation, on devra tenir compte de phénomènes non linéaires. Ceux-ci sont représentés par la présence d'éléments de base aux valeurs variables dans le temps.

1.1.6 Solution d'un réseau par l'approche nodale

L'approche nodale consiste à représenter le réseau comme étant une série de sources et de résistances. L'avantage de cette approche réside dans le fait qu'elle n'utilise que des

valeurs réelles. L'absence de valeurs complexes diminue grandement le nombre de calculs.

Prenons le circuit de la figure 1

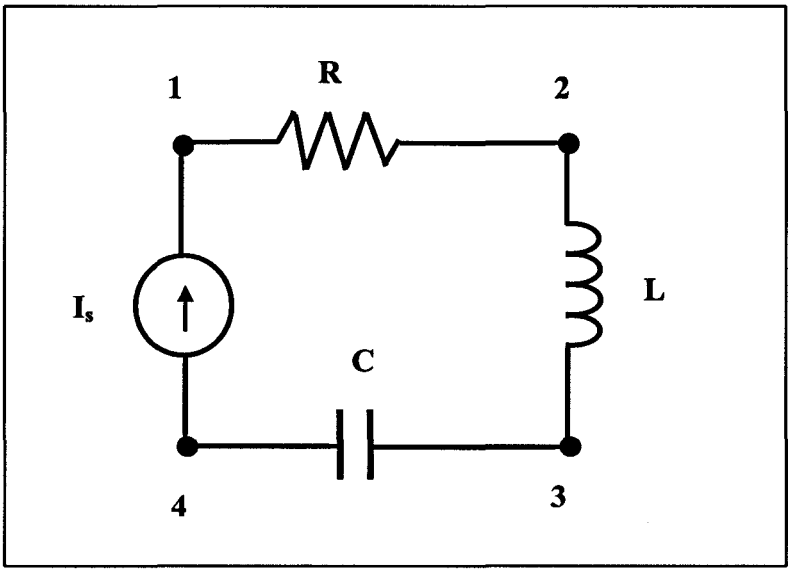

Figure 1 Circuit à modéliser

Le circuit équivalent selon l'approche nodale est donné à la figure 2

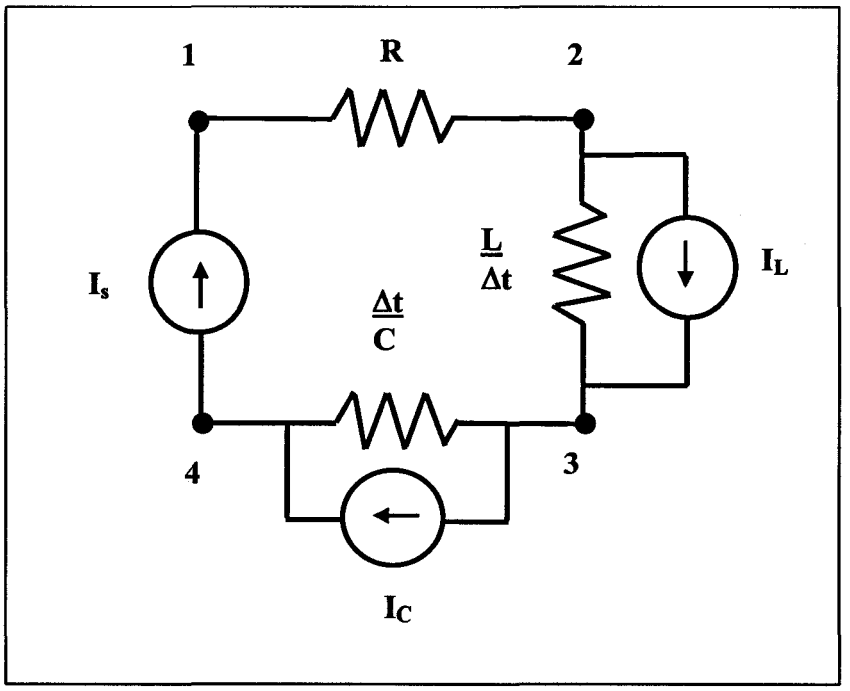

Figure 2 Circuit équivalent

De ce circuit équivalent, on peut tirer les quatre équations de nœuds suivantes à partir de la loi de Kirchoff.

Nœud 1 :

$$(1/R)\, V_1 + (-1/R)\, V_2 = I_S \tag{1.10}$$

Nœud 2 :

$$(1/R)\, V_1 + ((-1/R) + (-L/\Delta t))\, V_2 + (L/\Delta t)\, V_3 = I_L \tag{1.11}$$

Nœud 3 :

$$(\Delta t/L)\, V_2 + ((-\Delta t/L) + (-C/\Delta t))\, V_3 + (C/\Delta t)\, V_4 = I_L + I_C \tag{1.12}$$

Nœud 4 :

$$(C/\Delta t)\, V_3 + (-C/\Delta t)\, V_4 = I_S - I_C \qquad (1.13)$$

Ces quatre équations peuvent être réécrites sous la forme matricielle suivante :

$$\begin{bmatrix} (1/R) & (-1/R) & 0 & 0 \\ (1/R) & ((-1/R) + (-L/\Delta t)) & (L/\Delta t) & 0 \\ 0 & (\Delta t/L) & ((-\Delta t/L) + (-C/\Delta t)) & (C/\Delta t) \\ 0 & 0 & (C/\Delta t) & (-C/\Delta t) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} I_S \\ I_L \\ I_L + I_C \\ I_S - I_C \end{bmatrix} \qquad (1.14)$$

1.1.7 Expression de la solution sous forme matricielle

La forme générale de l'expression matricielle des équations du réseau à simuler est la suivante :

$$I_n = I_S - I_H \qquad (1.15)$$

$$Y_n V_n = I_n$$

Où

Y_n : Matrice d'admittance nodale (purement résistive)

V_n : Vecteur des tensions de nœud cherchées au temps "t_n".

I_n : Vecteur des courants injectés au temps "t_n".

I_S : Vecteur des courants de sources

I_H : Vecteur des courants issus du pas de calcul précédent (historique)

Le simulateur effectue d'abord la somme des courants

$$I_n = I_S - I_H \tag{1.16}$$

pour ensuite résoudre le système

$$Y_n V_n = I_n \tag{1.17}$$

où le courant est donné et la tension cherchée.

1.2 Parallélisation basée sur la topologie du réseau

La plupart des éléments simulés prennent pour acquis la simultanéité des tensions et courants aux interconnections. Toutefois, la modélisation des lignes de transport ne présume pas de cette instantanéité. En effet, la vitesse de propagation de l'onde dans une ligne de transport est donnée par l'équation suivante :

$$u = 1 / \sqrt{LC} \tag{1.18}$$

où L et C sont l'impédance et la capacité par unité de longueur de la ligne

En général, si le pas de calcul est de 50 μs, on devra tenir compte du délai pour une ligne excédant 15 km selon Wong (1999).

Ce délai de ligne permet de séparer en blocs distincts de calcul les différents postes. En effet, le calcul du poste durant le pas de calcul repose alors sur les valeurs du pas précédent pour toutes les lignes entrantes excédant une longueur de 15 kilomètres. Cette technique de simulation permet de découpler les calculs des postes et ainsi de regrouper un certain nombre de postes par processeur (nœud de calcul) et ainsi respecter les contraintes de temps du pas de calcul.

Ce découplage du calcul est essentiellement basé sur la topologie du réseau. Dans le cas d'un réseau de transport comme celui d'Hydro-Québec, le maillage est faible. Toutefois, dans le cas de réseaux américains ou autres, le maillage du réseau ne permet pas toujours un tel découplage. Il y a lieu d'investiguer d'autres méthodes de parallélisation des calculs.

1.3 Historique du simulateur

Au départ, le simulateur de réseaux électriques de l'Institut de recherche d'Hydro-Québec a été construit avec des composantes analogiques. Progressivement, les modules de simulation analogique ont été remplacés par des modèles numériques. Aujourd'hui, le simulateur est entièrement numérique.

Une première approche numérique fut de construire un réseau de nœuds de calcul avec des processeurs DSP TMS320C30/TMS320C40 connecté à une station de travail de Sun Microsystem. Éventuellement, les DSP furent remplacés par des processeurs Alpha de Digital. Les difficultés de maintenance d'un système fait sur mesure ont engendré l'abandon progressif de ceux-ci. Aujourd'hui, les nœuds de calcul Alpha sont remplacés par des ordinateurs parallèles génériques « Origin2000 », « Origin3000 » et « Origin300 » de « SGI » ainsi que par des « Grappes de PC » (PC-Clusters).

Hypersim effectue la simulation d'un réseau électrique en utilisant un ordinateur parallèle afin de fonctionner en temps réel. Le réseau est décomposé en postes, lignes et systèmes de commande qui sont associés chacun directement à une tâche informatique qui peut s'exécuter en parallèle sur un des processeurs. Chaque tâche calcule alors un ou plusieurs postes dépendant de la puissance du processeur. La répartition des tâches sur les différents processeurs se fait sur une base topologique, c'est-à-dire que le délai engendré par les longues lignes de transmission d'énergie sert à découpler les différents maillons de la simulation.

Bien que cette technique s'avère fort efficace pour les réseaux faiblement maillés tels que le réseau d'Hydro-Québec, elle n'est pas utilisable si le réseau est fortement maillé.

CHAPITRE 2

NATURE DU CALCUL MATRICIEL ÉTUDIÉ

Pour chaque poste, Hypersim effectue, à répétition, le calcul suivant :

$$Yv = i \tag{2.1}$$

où : Y est la matrice des admittances

v est le vecteur des tensions

i est le vecteur des courants

Généralement, la matrice des admittances ne change pas, aussi la technique de la décomposition LU (Lower and Upper Matrix) est utilisée, c'est-à-dire que la matrice est divisée en un produit de deux matrices de telle sorte que :

$$LU = Y \tag{2.2}$$

ce qui permet de convertir le calcul de Yv = i par les deux équations matricielles suivantes :

$$Ly = i \tag{2.3}$$

et

$$Uv = y \tag{2.4}$$

L et U étant des matrices triangulaires, le calcul du vecteur v est accéléré par la technique dite de la substitution avant/arrière.

Généralement, le temps de la première phase, la décomposition LU, est de moindre importance, car c'est le calcul de la seconde phase, la substitution avant-arrière, qui est effectuée à répétition.

Toutefois, dans le cas de cette étude, il s'agit d'améliorer le cas complet, incluant les deux phases. C'est-à-dire qu'on doit améliorer ce qu'il a été convenu d'appeler le « pire cas », où non seulement le vecteur « i » change, mais la matrice « Y » change aussi.

2.1 Stabilité de la décomposition LU

La technique de la décomposition LU est très sensible au choix du pivot, c'est-à-dire à chacun des éléments de la diagonale. Si un élément de la diagonale est très petit par rapport aux autres éléments de la même ligne, d'importantes erreurs d'approximation peuvent s'accumuler. Dans le cas où la matrice est diagonalement dominante, c'est-à-dire que chaque élément de la diagonale est égal ou supérieur à la somme des autres éléments de la ligne, aucune permutation de lignes n'est requise. Dans le cas contraire, un réarrangement des lignes est nécessaire.

Les deux aspects, c'est-à-dire avec et sans pivot, ont été étudiés, mais n'ont que très peu d'impact sur le temps de calcul.

2.2 Calcul partiel de la matrice des admittances

À chaque pas de calcul, le vecteur « i » des courants change et les tensions « v » doivent être recalculées. La plupart du temps, la matrice Y décomposée en L et U ne change pas. Toutefois, s'il s'avère que le circuit contienne un disjoncteur, un transformateur en saturation ou tout autre élément non linéaire, il est possible que certains éléments de la matrice doivent être recalculés. Ces éléments sont placés de telle sorte qu'ils aient le moins d'impact possible sur le nombre de calculs à effectuer. De plus, chaque calcul est précédé d'un branchement conditionnel (*if*) qui évite de recalculer si l'élément non linéaire n'a pas changé de valeur.

CHAPITRE 3

MÉTHODE DE RECALCUL PARTIEL DE LA DÉCOMPOSITION LU

Dans le simulateur Hypersim, la décomposition de la matrice des admittances en matrices L et U s'effectue en deux étapes. La première étape consiste à calculer les lignes affectées par un changement de nœud (la diagonale) dans la matrice et la seconde étape consiste à recalculer les lignes en question selon les nœuds qui ont changés.

Ce précalcul des lignes affectées selon le nœud modifié n'est effectué qu'une seule fois au début de la simulation, car il ne dépend que de la configuration de la matrice (c'est-à-dire du réseau électrique) et non de la valeur des éléments de la matrice. En effet, pour les éléments non linéaires, le simulateur ne permet pas la présence d'une valeur tantôt nulle, tantôt non nulle. Par exemple, dans le cas d'un disjoncteur, l'admittance est soit très grande (disjoncteur fermé) ou très petite (disjoncteur ouvert).

La liste des lignes affectées par un nœud est représentée dans le simulateur par une série de condition « if », générées dans le code, indiquant les lignes à recalculer. Cette surcharge de travail est grandement compensée par les calculs économisés à chaque pas de calcul lorsque peu d'éléments non linéaires changent.

À cet effet, une des astuces actuelles pour faire « passer » de grosses simulations (une simulation où beaucoup d'éléments changent en même temps) consiste à légèrement décaler les événements de façon à minimiser le nombre de changements simultanés.

CHAPITRE 4

ÉVOLUTION DE L'ARCHITECTURE DES PROCESSEURS

Au départ, le simulateur analogique a été remplacé par des processeurs spécialisés de traitement des signaux (DSP). Ceux-ci ont été progressivement remplacés par les processeurs Alpha, puis par les processeurs Mips et Pentium V. Toutefois, le code utilisé pour les calculs d'extraction des matrices L et U n'a pas vraiment changé depuis la première version numérique. Il y donc lieu de se demander si le programme, tel qu'il est codé, utilise de façon efficace ces nouveaux types de processeurs.

Les trois principaux changements qu'ont apportés les nouveaux processeurs sont : la présence d'un pipeline d'instruction, l'utilisation de registres internes et la présence de caches de données (généralement en deux niveaux).

Cette section présente donc l'évolution des architectures de processeurs durant les vingt dernières années et leurs influences sur la nature du code généré.

4.1 Architecture CISC et RISC

Pendant plusieurs années, au fur et à mesure qu'on pouvait entrer de plus en plus de transistors sur une même pastille de circuit intégré, la tendance générale était de complexifier les processeurs afin qu'ils puissent traiter des instructions de plus en plus complexes. En mille neuf cent quatre-vingt deux, Patterson et Sequin brisent cette tendance en proposant de réduire le nombre d'instructions disponibles afin de consacrer plus de transistors à l'accélération de celles les plus fréquemment utilisées. À cette époque, cette idée porte fruit et le monde des processeurs se scinde alors en deux camps : instructions complexes (CISC) et instructions réduites (RISC).

Aujourd'hui, cette distinction n'a plus la même signification, car la plupart des processeurs dit RISC (Mips, PowerPC, Sparc) ont maintenant un jeu d'instructions passablement garni et les processeurs de tendance CISC (Pentium) consacrent de plus en plus de transistors à l'accélération des instructions déjà présentes. Bref, il y a une certaine forme de convergence des technologies.

Il y a toutefois un élément qui distingue toujours ces deux familles de processeurs. Les processeurs RISC utilisés commercialement sont aussi des processeurs de type chargement-rangement. C'est-à-dire que les opérations arithmétiques ou logiques ne peuvent se faire que sur les valeurs des registres internes au processeur et la longueur des instructions est toujours la même. Les processeurs de type CISC, eux, peuvent additionner directement en mémoire (en réalité, dans les caches) deux valeurs et la longueur des instructions est variable de deux octets à onze octets. Cela peut sembler anodin, mais cette différence change grandement la façon dont le pipeline d'instruction est réalisé et change aussi la façon dont les compilateurs optimisent le code machine généré. Dans le cas des architectures à jeu d'instructions réduites de type chargement-rangement, on parle alors d'une disposition des modes d'adressage dite orthogonale, ce qui permet de systématiser l'optimisation du code généré. Dans le cas d'un processeur ayant un jeu d'instructions complexes, l'optimisation du code repose sur un ensemble d'heuristiques plutôt que sur une approche systématique.

Toutefois, dans la pratique, il n'y a pas de différence notable quant aux performances respectives des deux classes de processeurs. Cependant, il est à noter que les processeurs à jeu d'instructions complexes ont une horloge interne généralement deux fois plus rapide que les processeurs à jeu d'instruction réduit pour obtenir une puissance de calcul comparable. Cela s'explique par une technique dite du dédoublement d'horloge utilisée dans les CISC qui permet d'accélérer le décodage des instructions.

4.2 Registres internes

Au départ, les processeurs étaient essentiellement composés d'une unité arithmétique et logique (ALU) et du circuit logique pour l'alimenter. Ils effectuaient directement en mémoire les opérations arithmétiques ou logiques. Avec l'accélération des processeurs et la miniaturisation toujours croissante, la distance entre l'unité arithmétique (ALU) et la mémoire devint un problème. Aussi, on ajouta un registre intermédiaire appelé accumulateur. Celui-ci permit d'accélérer le traitement en évitant un accès long et compliqué à la mémoire. Par la suite, cette solution fût généralisée et on vit l'apparition non plus d'un seul accumulateur, mais d'un ensemble de registres internes. Sur les processeurs de type CISC, les registres ont tendance à avoir une fonction spécifique, tandis que les registres des RISC sont utilisables à plusieurs fins.

La présence de registres permet deux stratégies pour accélérer le traitement. La première stratégie consiste à assigner à un registre une valeur qui est utilisée par plus d'une instruction. De cette façon, la valeur n'a pas à se rendre à la mémoire à chaque fois, ce qui permet d'accélérer substantiellement le traitement. La seconde stratégie, utilisée par les processeurs chargement-rangement, consiste à généraliser cette approche et à toujours assigner un registre, même si la valeur n'est utilisée qu'une seule fois. Dans ce cas, le nombre de façons d'accéder à la mémoire est réduit, ce qui simplifie la logique d'adressage, libère un nombre important de transistors qui peuvent être utilisé pour accélérer d'autres parties du processeur.

L'ordre de chargement et d'utilisation des registres revêt une importance fondamentale pour la performance du programme. Le compilateur, lors de la phase d'optimisation, passera donc beaucoup de temps à chercher l'ordre d'utilisation maximal de ces registres.

4.3 Caches et accès à la mémoire centrale

De la même façon que les registres internes permettent d'agir sur une valeur plus rapidement, l'utilisation de mémoires caches permet d'accélérer le traitement en travaillant temporairement sur un ensemble de valeurs placées à proximité de l'ALU, en utilisant un type de mémoire à accès très rapide. Le résultat est qu'un accès à la première cache de mémoire peut être aussi rapide que seulement trois cycles d'horloge tandis qu'un accès à la mémoire centrale peut prendre plusieurs centaines de cycles.

Bien qu'il existe différentes approches pour gérer les caches de mémoire, la plupart des manufacturiers (Intel, Mips, Motorola) ont opté pour la technique de la cache de mémoire dite associative. De plus, afin de maintenir un bon ratio entre la proximité, le nombre et la nature des transistors utilisés ainsi que l'espace disponible et la quantité de chaleur dissipée par unité de surface, les manufacturiers choisissent généralement d'utiliser deux niveaux de caches, la cache la plus proche de l'ALU s'alimentant à partir de la cache de second niveau et celle-ci, à partir de la mémoire centrale.

Une cache associative fonctionne de la façon suivante. La mémoire centrale est divisée en blocs de la même taille. A chaque fois que le processeur a besoin d'un octet (ou plus) de la mémoire centrale, c'est l'ensemble du bloc qui est transféré vers la cache. Le transfert par bloc plutôt que par mot ou octet permet ainsi d'accélérer le transfert entre la cache et la mémoire. Chaque bloc de la mémoire occupe alors une place bien précise et toujours la même dans la cache. Comme le nombre de blocs que peut accueillir la cache est beaucoup plus petit que celui de la mémoire centrale, on utilise une relation surjective (c'est-à-dire une fonction) pour calculer la place que le bloc doit occuper. La fonction choisie est presque toujours la fonction modulo par un exposant de deux, car sa réalisation matérielle en est grandement simplifiée. En effet, il s'agit de ne retenir (connecter) que les dernières lignes d'adresses et d'oublier les lignes d'adressage de plus grand poids. Cette technique est si simple et efficace qu'il est rare de trouver un

processeur qui n'utilise pas ce système. S'il y a deux niveaux de caches, la cache de premier niveau (proche de l'ALU) fonctionne de la même façon que la cache de second niveau qui s'alimente directement à la mémoire centrale.

4.4 Accès non uniforme à la mémoire (NUMA) sur les systèmes à plusieurs processeurs

Dans le cas de systèmes à plusieurs processeurs partageant la même mémoire centrale, le bus local permettant d'accéder à la mémoire peut grandement limiter la performance de la machine. En effet, si le nombre de processeurs est important, il se peut qu'il y ait une forte contention sur le bus local. La solution choisie par Silicon Graphics consiste à « distribuer » la mémoire sur différents noeuds de processeurs et de mettre en place un mécanisme de transfert entre les nœuds. On parle alors d'un accès non uniforme (NUMA), car certaines cases mémoires sont plus longues à accéder que d'autres.

Le système NUMA repose sur la prémisse suivante : dans un programme conçu pour une machine multiprocesseurs, le code associé à chaque processeur travaille majoritairement avec une portion non partagée (la pile, les variables locales, etc.) de la mémoire et les accès aux parties communes de la mémoire sont plus rares. Cette prémisse est souvent vérifiée, aussi ce système, dans la pratique, permet souvent des gains.

Contrairement à un système distribué, où un accès à une case mémoire éloignée doit de faire de façon explicite dans le code, le système NUMA est transparent à la programmation. Ce n'est que la machine, et non le programmeur, qui tient compte de la localisation (le nœud de processeurs) d'une case mémoire.

Dans le système Origin 2000 et Origin 3000 de Silicon Graphics, chaque nœud de calcul contient un espace mémoire et deux processeurs. Les nœuds sont disposés selon une configuration hypercubique, c'est-à-dire que ce ne sont pas tous les nœuds qui peuvent

directement échanger avec tous les autres nœuds, mais seulement certains nœuds. Les transferts les plus éloignés doivent transiter par des nœuds intermédiaires. La configuration hypercubique permet de s'assurer que le nombre de nœuds à traverser ne dépasse jamais « $\log_2(n)$ » où « n » est le nombre de nœuds de calcul présents dans la machine.

4.5 Pipeline d'instruction

Un autre moyen d'accélérer le traitement consiste à traiter en séquences les instructions, à la manière d'une chaîne de montage, c'est-à-dire que chaque instruction s'exécute en plusieurs étapes. Par exemple, dans le cas d'un processeur de type Pentium, l'instruction est d'abord chargée par un premier module, puis l'instruction est décodée par un second module, un troisième module calcule les opérantes, un quatrième exécute l'opération arithmétique ou logique demandée et, enfin, un cinquième module s'occupe de stocker le résultat. On parle alors de chaîne de montage, car ces cinq étapes peuvent s'exécuter en même temps. Par exemple, pendant qu'on stocke le résultat de la première instruction, on peut exécuter la seconde instruction, calculer les opérantes de la troisième instruction, décoder la quatrième instruction et charger la cinquième. Ce parallélisme permet grandement d'accélérer le traitement (dans le cas du Pentium, par un facteur cinq), toutefois, ce mécanisme n'est pas parfait. Le pipeline d'instruction repose sur la prémisse que l'exécution du code est linéaire, c'est-à-dire sans branchement. Or le code réel comporte plusieurs branchements, ce qui limite l'efficacité d'une telle technique.

Lorsqu'il y a un branchement, il est possible que le pipeline doive être vidé et rechargé avec les instructions pointées par le branchement. Ce rechargement est coûteux, en conséquence, beaucoup d'efforts ont été déployés afin de palier à ce problème. La technique souvent utilisée consiste à tenter de prévoir la voie la plus probable du branchement. Par exemple, sur le branchement d'une boucle « for » on peut présumer que la plupart du temps on doit rester dans la boucle plutôt que d'en sortir. Dans le cas

d'un simple « if », il n'y a pas de préférence particulière. La décision du choix du branchement « probable » peut se faire par le processeur directement (basé sur le type de branchement) ou par le compilateur, en indiquant à l'avance sa préférence au processeur. Cette seconde technique est plus puissante, car le compilateur jouit du contexte de l'instruction, ce que le processeur ne peut faire.

4.6 Parallélisation du pipeline d'instruction (Architecture superscalaire)

Un autre moyen d'accélérer le traitement des instructions dans le pipeline consiste à traiter les instructions en parallèle, c'est-à-dire en même temps ou presque. Ceci s'effectue à l'intérieur du processeur et n'est pas directement visible de l'extérieur. Le code généré n'a pas à tenir compte de cette possible parallélisation.

Deux techniques existent permettant une forme de parallélisation. La première consiste à doubler la vitesse de l'horloge interne du processeur, la seconde, à avoir deux unités permettant un traitement. Souvent les processeurs utilisent une combinaison de ces deux techniques.

La technique du dédoublement d'horloge consiste à permettre le transfert d'une donnée sur un registre, soit sur le front montant, soit sur le front descendant de l'horloge (en réalité, l'horloge est doublée et on utilise toujours le même front) et le transfert prend alors le cycle au complet. Comme il y a beaucoup de registres internes (visibles ou implicites), il est possible d'effectuer deux transferts à un demi-cycle d'intervalle s'il ne s'agit pas des mêmes registres et ainsi doubler la vitesse de traitement.

La technique du doublement des unités de traitement peut apparaître sous différentes formes. Si une unité arithmétique ne traitant que les nombres à point flottant est ajoutée (le « co-processeur mathématique »), il est alors possible d'effectuer une opération arithmétique entière en même temps qu'une opération à point flottant. Une autre

possibilité consiste à créer des modules indépendants pour calculer les adressages indirects indexés, ce qui libère l'ALU. Il peut aussi y avoir deux ALU distincts.

Que ce soit la technique du dédoublement ou celle du doublement d'unités, dans les deux cas, une nouvelle contrainte de performance apparaît. Deux instructions consécutives ne peuvent être parallélisées que si elles sont pleinement indépendantes une de l'autre. Par exemple, si on a les instructions suivantes :

```
R1 = R2 + R3
R4 = R1 + R5
```

Elles ne peuvent être parallélisées, car la seconde instruction dépend du résultat de la première. Maintenant, si on a les instructions suivantes :

```
R1 = R2  + R3
R4 = R1  + R5
R6 = R7  + R8
R9 = R10 + R6
```

Elles ne peuvent pas plus être parallélisées. Toutefois, si on réarrange l'ordre de traitement des instructions de la façon suivante :

```
R1 = R2  + R3
R6 = R7  + R8
R4 = R1  + R5
R9 = R10 + R6
```

On remarque que ces instructions peuvent maintenant être parallélisées sans pour autant changer le résultat des calculs. C'est cette technique de réordonnancement que peuvent utiliser les compilateurs pour permettre un traitement plus rapide. Toutefois, cette technique a ses limites et ses pièges. En effet, il n'est pas toujours possible de découpler les instructions, mais, plus grave, dans certains cas la modification de l'ordre de calcul peut induire une instabilité numérique, particulièrement si on déplace des opérations de multiplication ou de division.

Par exemple, si on prend le calcul suivant :

$$R = (a + b + c + d) / e \quad (4.1)$$

où « e » est une valeur constante

Cette équation prend trois additions et une division. Le compilateur peut récrire cette équation de façon à utiliser quatre opérations « multiplication-accumulation » plus rapide en pré-calculant l'inverse multiplicatif de « e » :

$$f = 1 / e \quad (4.2)$$

$$R = (a \cdot f + b \cdot f + c \cdot f + d \cdot f) \quad (4.3)$$

Si la valeur de « e » est très petite par rapport aux variables « a », « b », « c » et « d », le compilateur induit alors une plus grande erreur numérique, car l'erreur causée par la division de « e » due à la limite de la dimension des nombres d'un ordinateur n'est pas générée qu'une seule fois, mais est répétée quatre fois. De plus, le calcul de l'inverse multiplicatif de « e » induit aussi une erreur.

4.7 Interdépendance Architecture-Compilateur

Comme le montrent les sections précédentes, toutes les techniques d'accélération des processeurs reposent sur une série d'hypothèses. Que se soit dans le cas des registres internes, des caches de données ou du pipeline d'instruction, il doit y avoir une étroite relation entre processeur, système d'exploitation et compilateur pour pouvoir tirer profit de ces mécanismes d'accélération. La section *6.1 : Recension des écrits* présente les techniques d'optimisation des compilateurs.

CHAPITRE 5

MÉTHODOLOGIE

Pour réaliser ce travail, les étapes suivantes ont été réalisées :

- identifier les patrons de programmation associés aux calculs des stations;
- identifier les facteurs qui ralentissent le traitement pour chacun des patrons précédemment identifiés;
- identifier, de façon exhaustive, toutes les options de compilation qui peuvent améliorer la vitesse de traitement des patrons;
- identifier l'ordre de traitement des calculs qui peut améliorer la vitesse de traitement des patrons;
- identifier d'autres solutions qui permettent d'accélérer la vitesse de traitement et le démontrer par des cas typiques.

Dans chaque cas, la démarche suivante a été suivie :

- identifier un problème;
- créer un test (in vitro);
- tester et mesurer;
- analyser les résultats.

Pour chaque facteur étudié, une série de dix mesures dans des conditions identiques et sur un processeur libre ont été prises. L'algorithme étudié est exécuté dix mille fois. De ces dix mesures, la médiane est calculée et c'est ce chiffre qui est présenté dans la section *6.4 : Résultats*.

La médiane a été choisie, car elle permet de ne pas tenir compte des écarts trop grands qui proviendraient d'un événement non contrôlé. Toutefois, les mesures réalisées ne présentent pas de grandes variations et la moyenne est très similaire à la médiane.

CHAPITRE 6

FACTEURS MATÉRIELS ET LOGICIELS

Cette section présente les facteurs logiciels ou matériels susceptibles de diminuer le temps de calcul, mais sans modifier la nature du calcul.

6.1 Recension des écrits

Pour les langages de programmation structurés tel que le C, le C++ ou le Pascal, la structure générale d'un compilateur-optimisateur peut être décrite par la figure 3.

Dans la plupart des cas et en particulier dans les compilateurs étudiés ici dans ce document, l'optimisation se fait en deux passes. La première optimisation s'effectue à partir de la représentation en code intermédiaire de moyen niveau et est indépendante de toute architecture particulière de processeurs. La seconde optimisation s'effectue sur le code intermédiaire de bas niveau et est pleinement dépendante de l'architecture du processeur cible.

La première optimisation, de plus haut niveau, s'attaque à des problèmes fortement liés au langage de programmation, par exemple, l'élimination des variables inaccessibles, l'enlèvement du code mort, le réordonnancement des opérations permettant de ne pas réexécuter deux fois le même code, la création de variables temporaires ayant des valeurs pré-calculées, etc.

En particulier, elle permettra de dérouler des boucles, lorsqu'il y a un gain en vitesse possible. Toutefois, ce déroulage n'est pas complètement indépendant du processeur cible et sera terminé par la seconde passe, la post-optimisation.

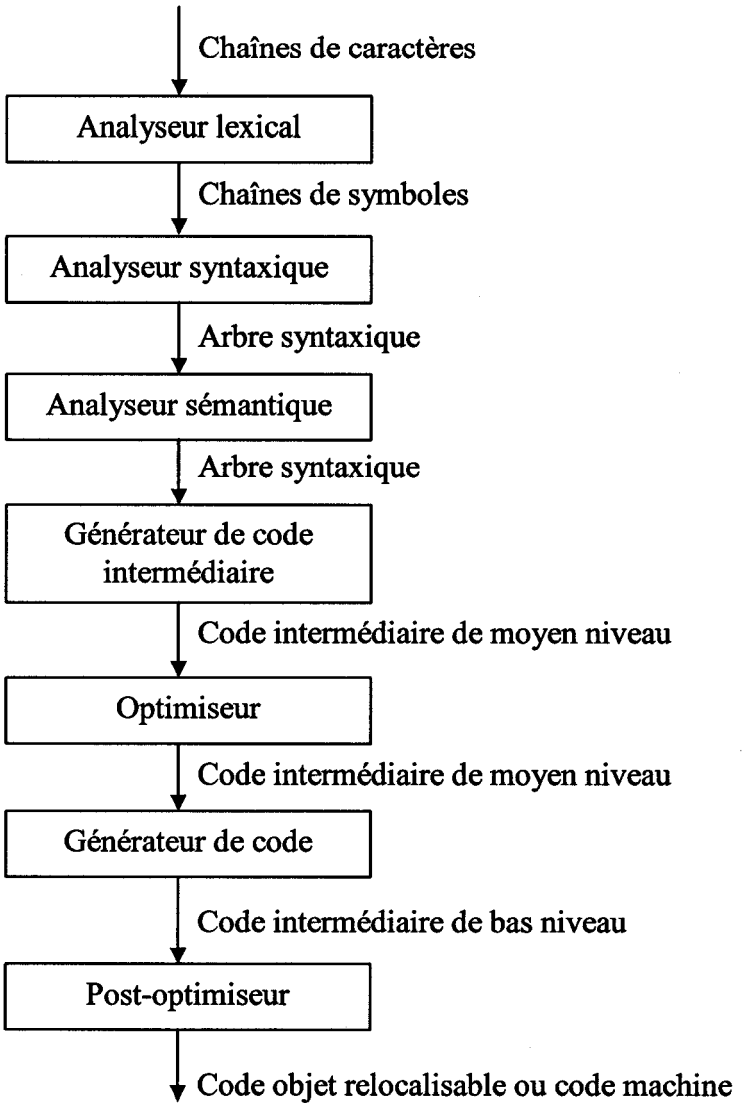

Figure 3 Structure d'un compilateur-optimisateur
(Muchnick, 1997)

La post-optimisation s'attaque aux problèmes liés au code machine et à l'architecture du processeur cible. Elle travaille à partir du code intermédiaire de bas niveau. Plus précisément, elle effectue les trois tâches suivantes :

- allocation efficace des registres du processeur;
- ordonnancement des instructions afin de maintenir le pipeline plein;
- gestion des caches du processeur.

Dans les cas étudiés ici, la plupart des gains en performance seront obtenus par la post-optimisation, car le code généré du simulateur Hypersim est déjà passablement travaillé, de telle sorte qu'il offre peu ou pas de chance d'amélioration à l'optimisateur de moyen niveau.

6.2 Documentation des manufacturiers

Après lecture de la documentation de Silicon Graphics (SGI), de la société MIPS Technology (les processeurs de SGI) et du projet GNU/Linux, il n'apparaît pas qu'il y ait d'options particulières permettant de diminuer la vitesse de traitement du simulateur, autres que les options usuelles des compilateurs.

Dans le cas du Silicon Graphics, les options sont :

- O2 : optimisation de base;
- O3 : ajout de l'optimisation dite "agressive" et du déroulage de boucles;
- O*fast* : ajout d'une analyse inter-procédurale, afin de détecter des fonctions inutiles ou qui peuvent être macro-substituées (*inlined*).

Dans le cas de GNU/Linux, les options sont :

- O2 : optimisation de base;
- O3 : ajout de l'optimisation dite "agressive", mais pas de déroulage;
- funroll-loops : ajout du déroulage de boucle, si jugé nécessaire.

Les ordinateurs de Silicon Graphics ayant des processeurs de type RISC et les ordinateurs de type PC ayant des processeurs de type CISC, les résultats obtenus diffèrent sensiblement.

6.3 Bancs de test

Afin de représenter les différentes situations possibles, une série de tests isolés (in vitro) ont été préparés. Le tableau suivant présente la liste des programmes et leur description.

Tableau I

Liste des programmes de test

Nom	Description
`Testicache`	Teste une multiplication de matrice roulée
`Testunrolled`	Teste une multiplication de matrice, code déroulé
`testdecomplu2`	Teste une décomposition LU, code roulé, sans pivot.
`testdecomplu3`	Teste une décomposition LU, code roulé, avec pivot.
`Testunrolledlu2`	Teste une décomposition LU, code déroulé, sans pivot.
`Testunrolledlu3`	Teste une décomposition LU, code déroulé, avec pivot.
`Testsubstitute`	Teste une substitution avant/arrière, code roulé, sans pivot
`Testsubstitute3`	Teste une substitution avant/arrière, code roulé, avec pivot
`testunrolledsub`	Teste une substitution avant/arrière, code déroulé, sans pivot
`testunrolledsub3`	Teste une substitution avant/arrière, code déroulé, avec pivot
`testunrolledsub5`	Teste une substitution avant/arrière, code déroulé, avec pivot et avec réordonnancement de la matrice LU
`testunrolledsub6`	Teste une substitution avant/arrière, code déroulé, avec pivot et avec un réordonnancement de la matrice LU incluant des trous.

Les cinq cas retenus présentant des caractéristiques intéressantes sont :

- substitution avant/arrière en boucles roulées ou en boucles déroulées;
- effet des options de compilation O2 et O3 sur le temps de calcul;
- comparaison entre l'utilisation d'une structure matricielle et d'un vecteur;
- effet de l'ordre d'accès aux données de la matrice;
- effet de la dispersion des données.

Afin de réaliser ces programmes de test, les modules et générateurs suivants ont été préparés.

Tableau II

Liste des modules

Nom	Description
`printmatrix.c`	Imprime une matrice de réels
`printintmat.c`	Imprime une matrice d'entiers
`printvector.c`	Imprime un vecteur de réels
`printintvec.c`	Imprime un vecteur d'entiers
`mulmatrix.c`	Multiplie deux matrices
`mergematrix.c`	Combine deux matrices L et U en une seule matrice
`cmpmatrix.c`	Compare deux matrices
`mulvector.c`	Multiplie un vecteur par une matrice
`cmpvector.c`	Compare deux vecteurs
`decomplu.c`	Effectue une décomposition LU, code roulé, sans pivot, stockée dans deux matrices
`decomplu2.c`	Effectue une décomposition LU, code roulé, sans pivot, stockée dans une seule matrice
`decomplu3.c`	Effectue une décomposition LU, code roulé, avec pivot, stockée dans une seule matrice.
`substitute.c`	Effectue une substitution avant/arrière, code roulé, sans pivot.
`substitute3.c`	Effectue une substitution avant/arrière, code roulé, avec pivot.

Tableau III

Liste des générateurs

Nom	Description
gendimmatrix.c	Génère le fichier « dimmatrix.h » qui définit la grandeur de la matrice courante.
genmatrix.c	Génère une matrice de nombres pseudo-aléatoires
gendommatrix.c	Génère une matrice de nombres pseudo-aléatoires diagonalement dominante
genvector.c	Génère un vecteur de nombres pseudo-aléatoires
gendecomplu.c	Génère le code déroulé d'une décomposition LU utilisant deux matrices de stockage, sans pivot
gendecomplu2.c	Génère le code déroulé d'une décomposition LU utilisant une seule matrice de stockage, sans pivot
gendecomplu3.c	Génère le code déroulé d'une décomposition LU utilisant une seule matrice de stockage, avec pivot
genlu2.c	Génère une matrice décomposée L et U (stockée dans une seule matrice), sans pivot
genlu3.c	Génère une matrice décomposée L et U (stockée dans une seule matrice), avec pivot
genp.c	Génère le vecteur de pivot associé à « genlu3.c ».
gensubstitute.c	Génère une substitution avant/arrière déroulée, sans pivot.
gensubstitute3.c	Génère une substitution avant/arrière déroulée, avec pivot.
gensubstitute4.c	Génère une substitution avant/arrière déroulée, sans pivot, sans un réordonnancement de la matrice LU.
gensubstitute5.c	Génère une substitution avant/arrière déroulée, sans pivot, mais avec un réordonnancement de la matrice LU.
genreorder.c	Génère un tableau de correspondance pour le réordonnancement de la matrice LU
genlu5.c	Génère la matrice LU réordonnancée
genmaxmatrix.c	Génère le fichier « maxmatrix.h » selon la dimension de la matrice et le taux de dispersion
genholes.c	Génère un tableau de correspondance pour créer des trous dans la matrice réordonnée par « genreorder.c ».
genlu6.c	Génère la matrice LU réordonnancée, mais avec des trous
gensubstitute6.c	Génère une substitution avant/arrière déroulée, sans pivot, mais avec un réordonnancement incluant des trous de la matrice LU.

Pour effectuer les différents tests, les scripts suivants sont disponibles :

Tableau IV

Liste des scripts de test

Nom	Description
`runtest.csh` `runpctest.csh`	Compare les décompositions LU avec et sans pivot, la substitution avant/arrière avec ou sans pivot, pour un code à boucles roulées ou déroulées. (SGI et PC)
`runtest02.csh` `runpctest02.csh`	Compare la substitution avant/arrière sans pivot d'un code à boucles roulées ou déroulées. (SGI et PC)
`runtest03.csh` `runpctest03.csh`	Compare la substitution avant/arrière avec pivot d'un code à boucles roulées ou déroulées. (SGI et PC)
`runtest04.csh` `runpctest04.csh`	Compare la substitution avant/arrière avec pivot d'un code à boucles roulées ou déroulées. (SGI et PC)
`runtest05.csh` `runpctest05.csh`	Compare la substitution avant/arrière sans pivot d'un code à boucles déroulées avec une structure de type matriciel ou vectoriel, ordonnée selon l'ordre de calcul (SGI et PC)
`runtest06.csh` `runpctest06.csh`	Vérifie l'effet de la dispersion des données sur une substitution avant/arrière sans pivot d'un code à boucles déroulées, avec un accès selon l'ordre de calcul. (SGI et PC)
`runtest07.csh` `runpctest07.csh`	Vérifie l'effet de la dispersion des données sur une substitution avant/arrière sans pivot d'un code à boucles déroulées, avec un accès selon un ordre matriciel. (SGI et PC)

Tableau V

Liste des scripts de collecte

Nom	Description
`crunchtest.csh` `crunchpctest.csh`	Calcule les médianes des résultats du script `runtest.csh` et les place dans le répertoire `résultat`.
`crunchtest02.csh` `crunchpctest02.csh`	Calcule les médianes des résultats du script `runtest02.csh` et les place dans le répertoire `résultat (PC : resultat02)`.
`crunchtest03.csh` `crunchpctest03.csh`	Calcule les médianes des résultats du script `runtest03.csh` et les place dans le répertoire `résultat (PC : resultat03)`.
`crunchtest04.csh` `crunchpctest04.csh`	Calcule les médianes des résultats du script `runtest04.csh` et les place dans le répertoire `résultat (PC : resultat04)`.
`crunchtest05.csh` `crunchpctest05.csh`	Calcule les médianes des résultats du script `runtest05.csh` et les place dans le répertoire `résultat (PC : resultat05)`.
`crunchtest06.csh` `crunchpctest06.csh`	Calcule les médianes des résultats du script `runtest06.csh` et les place dans le répertoire `résultat (PC : resultat06)`.
`crunchtest07.csh` `crunchpctest07.csh`	Calcule les médianes des résultats du script `runtest07.csh` et les place dans le répertoire `résultat (PC : resultat07)`.
`crunchtest07b.csh` `crunchpctest07b.csh`	Calcule les médianes des résultats du script `runtest07.csh` mais uniquement pour un facteur de dispersion zéro et les place dans le répertoire `résultatB (PC : resultat07B)`.

De plus, afin de vérifier les calculs et de rassembler les résultats, les programmes suivants ont été développés

Tableau VI

Liste des autres programmes

Nom	Description
`Extractor`	Extrait les données provenant de l'outil « perfex »
`Median`	Calcule la médiane à partir d'un fichier de dix valeurs
`testmulvector`	Vérifie la multiplication d'un vecteur par une matrice
`testmulmatrix`	Vérifie la multiplication de matrices
`Verifdecomplu`	Vérifie les différentes décompositions LU
`Verifsubstitute`	Vérifie les différentes substitutions avant/arrière

6.4 Résultats

Cette section présente les résultats.

6.4.1 Boucles roulées ou déroulées

Dans un premier temps, nous avons cherché à savoir si les boucles du calcul de substitution avant/arrière roulées ou déroulées étaient les plus performantes.

6.4.1.1 Exemple d'une boucle roulée et déroulée

Par exemple, pour une matrice trois par trois, le code à boucles roulées permettant d'effectuer la première substitution avant de l'équation "Ly = b" ressemble au cas suivant :

```
for(i = 0; i < 3; i++) {
   y[i] = b[i];
   for(j = 0; j < i; j++) {
      y[i] -= lu[i][j] * y[j];
   }
}
```

Tandis que le code à boucle déroulée ressemble plutôt au cas suivant :

```
y[0] = b[0];
y[1] = b[1];
y[1] -= lu[1][0] * y[0];
y[2] = b[2];
y[2] -= lu[2][0] * y[0];
y[2] -= lu[2][1] * y[1];
```

On observe que dans le cas d'une boucle déroulée, la position dans les vecteurs et matrices est donnée explicitement. Cette dernière méthode élimine les branchements, potentiellement générateurs d'un rechargement prématuré du pipeline d'instruction, mais génère plus d'instructions à exécuter.

6.4.1.2 Mesures obtenues

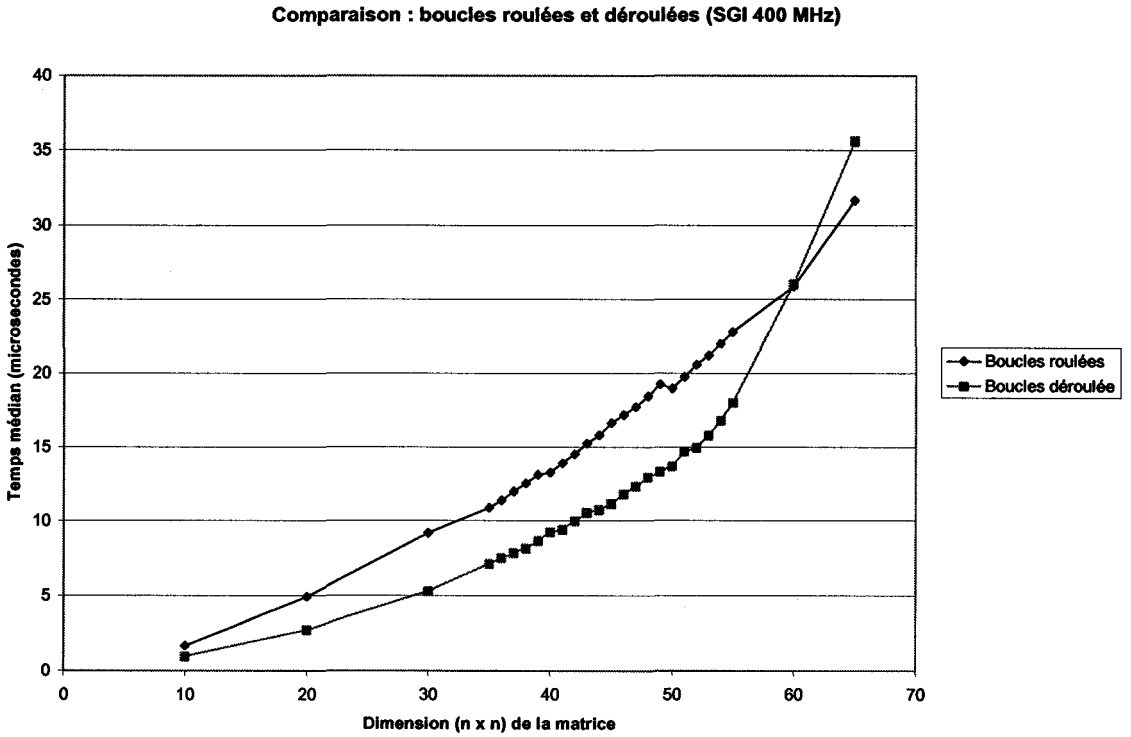

Figure 4 Comparaison boucles roulées ou déroulées (SGI 400 MHz)

On constate que les boucles roulées demeurent une meilleure option, mais seulement pour les matrices excédant 60 par 60 éléments. Cela s'explique par une mauvaise utilisation de la cache d'instruction niveau I.

Le même test a été repris, mais sur un PC GNU/Linux :

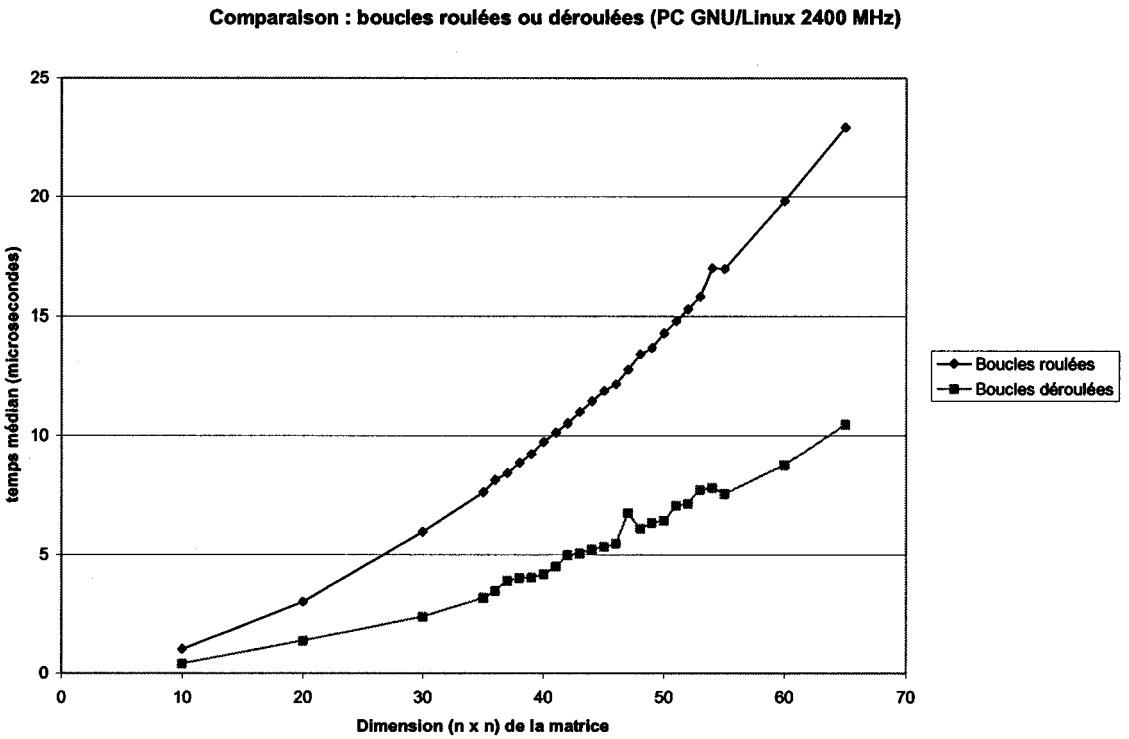

Figure 5 Comparaison boucles roulées ou déroulées (PC GNU/Linux 2400 MHz)

On constate que, sur un PC, le code déroulé est toujours une meilleure option.

6.4.2 Effet des options de compilation

Les figures 6 et 7 montrent l'effet des options de compilation O2 et O3 sur le code roulé et déroulé :

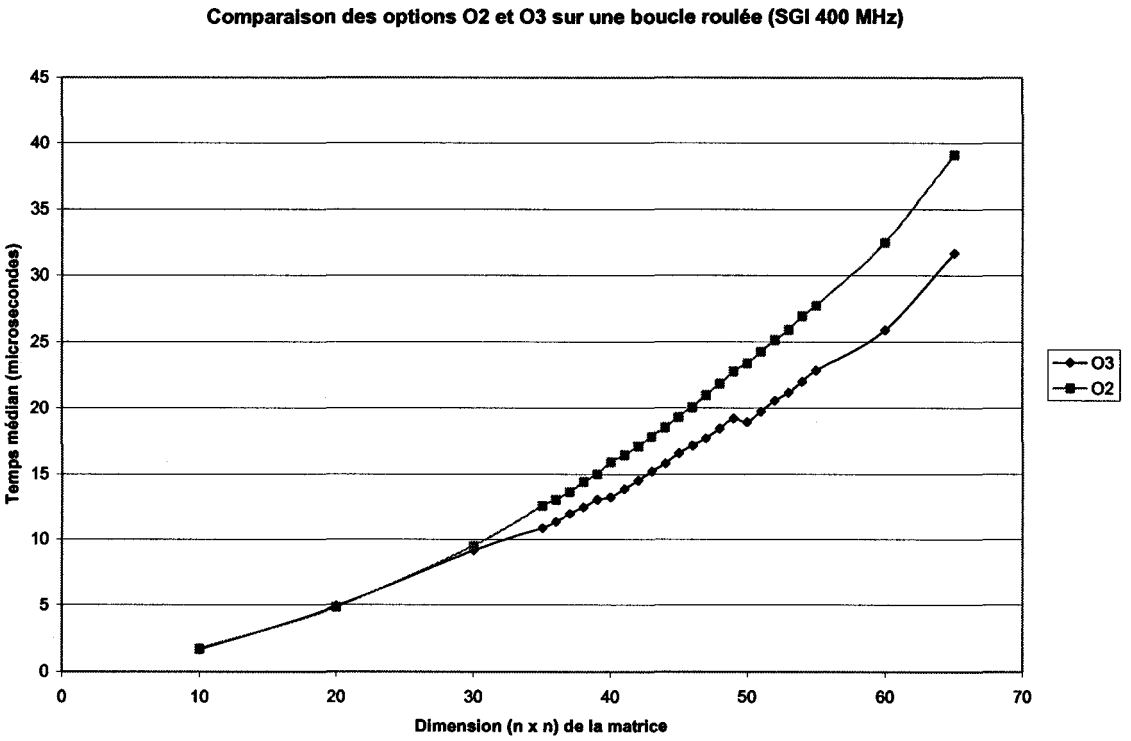

Figure 6 Comparaison des options O2 et O3 sur des boucles roulées

On constate que l'option de compilation O3 a une influence certaine sur le code à boucles roulées, Cela s'explique par le fait que l'optimiseur tient le pipeline d'instruction plein en déroulant juste assez les boucles.

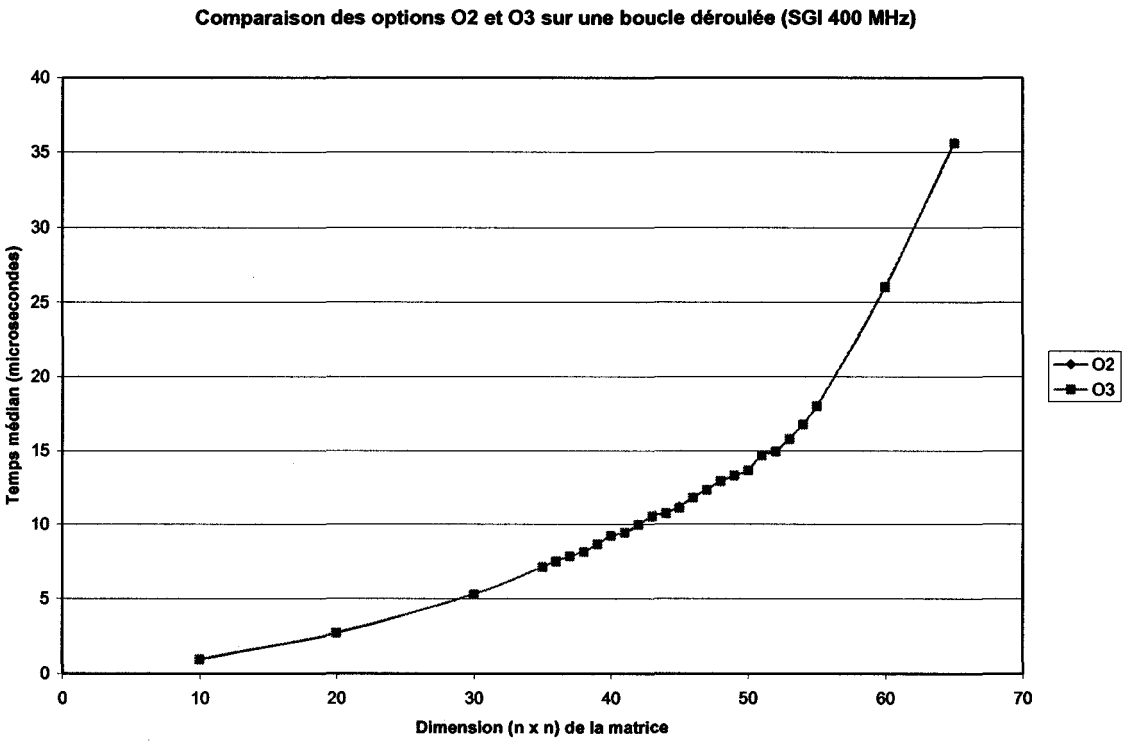

Figure 7 Comparaison des options O2 et O3, boucles déroulées (SGI 400 MHz)

Ici, on constate que l'option de compilation O3 n'a aucune influence sur le code déroulé, car le mode d'optimisation O3 s'attaque au déroulage de boucles et à la prédiction des branchements.

6.4.3 Effet du changement de structure "C" (matrice ou vecteur)

Les données du code produit par Hypersim sont placées dans des vecteurs. Par contraste, nos programmes de tests utilisent des matrices C. On peut facilement passer d'une représentation matricielle à une représentation vectorielle en utilisant la formule de correspondance suivante :

```
Vecteur [z ] = Matrice [x][y]
   si  z = x * DIM + y  et DIM est la dimension de la matrice
```

La figure suivante montre la différence entre l'utilisation d'une matrice ou d'un vecteur pour une substitution avant/arrière :

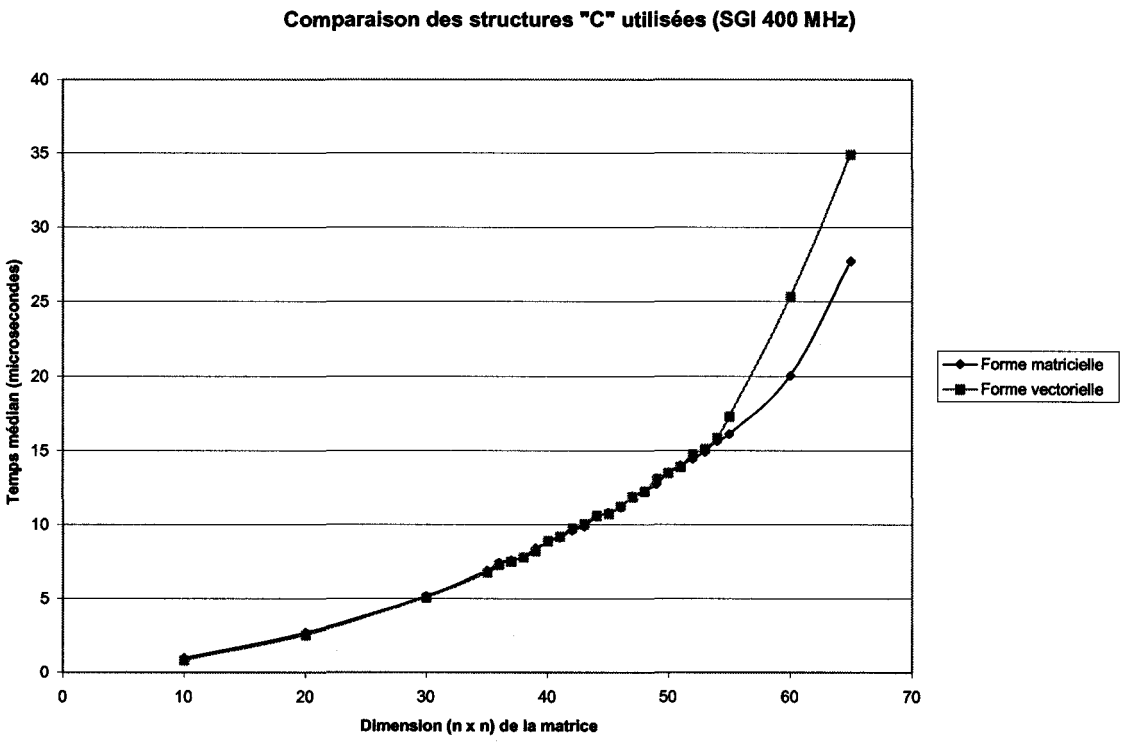

Figure 8 Comparaison de la structure "C" utilisée (SGI 400 MHz)

Dans le cas d'un ordinateur Silicon Graphics, on note que pour les matrices de forte dimension, il est avantageux d'utiliser une matrice plutôt qu'un vecteur. Toutefois, ce résultat ne tient pas sur un processeur de Intel.

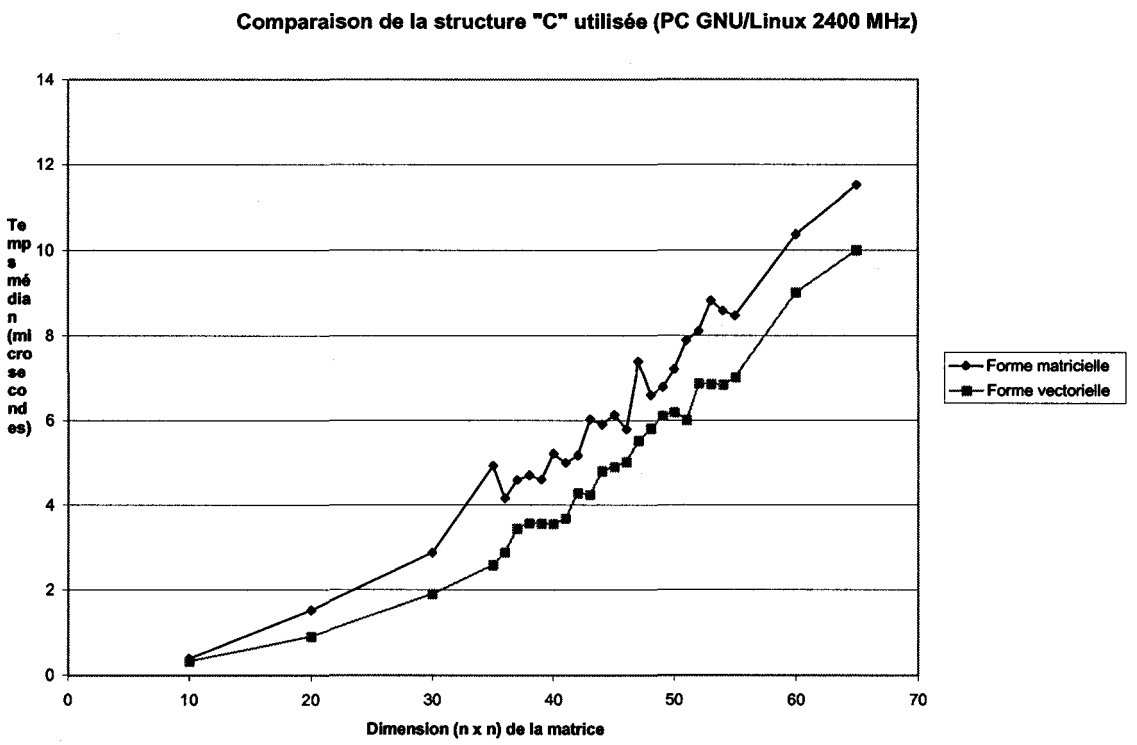

Figure 9 Comparaison de la structure "C" utilisée (PC GNU/Linux 2400 MHz)

On observe que l'utilisation d'un vecteur plutôt que d'une matrice diminue nettement le temps de calcul. Les pics du graphique, qui ne sont pas présents sur le SGI, ne reflètent pas une plus grande variabilité des mesures. Ils illustrent le fait que les caches du processeur Pentium sont plus petites que celles du processeur de MIPS et, par conséquent, ont une plus grande influence sur la performance du système.

6.4.4 Effet de l'ordre d'accès aux données

L'ordre d'accès aux données peut avoir une influence. On peut placer successivement les éléments en parcourant la matrice de gauche à droite et de haut en bas. Les figures 8 et 9 illustrent ce cas par « accès selon la matrice ». On peut aussi placer les éléments en utilisant l'ordre d'accès de la substitution avant-arrière (sans pivot) représenté ici sous le vocable « accès selon ordre de calcul ».

Exemple d'accès selon une matrice :

```
a  b  c
d  e  f   →   [ a  b  c  d  e  f  g  h  i ]
g  h  i
```

Exemple d'accès selon l'ordre de calcul :

```
a  b  c
d  e  f   →   [ a  d  e  g  h  i  f  c  b ]
g  h  i
```

Réécrit, de façon plus synthétique, on a :

```
1  2  3                    1  9  8
4  5  6                    2  3  7
7  8  9                    4  5  6
```

Selon une matrice — Selon l'ordre de calcul

Voici l'effet sur le temps de calcul :

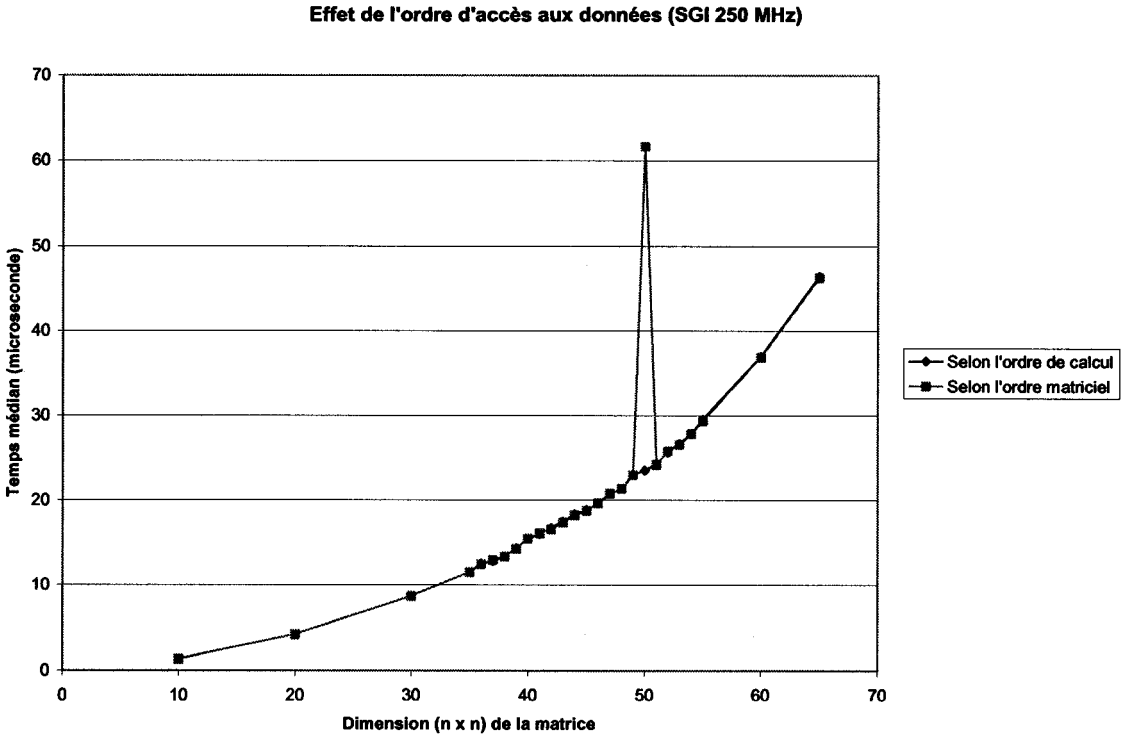

Figure 10 Effet de l'ordre d'accès aux données (SGI 250 MHz)

On constate que l'accès selon l'ordre de calcul ne donne pas de gain direct sur le temps de calcul, mais permet d'éviter une discontinuité dans la courbe du temps de calcul. En effet, le pic présent sur la courbe est un effet de cache, c'est-à-dire qu'il y a une concordance entre la grandeur des caches de premier niveau et la taille de la matrice utilisée, qui provoque une surabondance de remplacement de cache, constatée après l'analyse d'un compteur de remplacement de caches. Les ordinateurs Silicon Graphics ont un outil permettant de compter les remplacements de caches de premier et de second niveau, ainsi que de nombreuses autres variables.

Sur une machine de type PC GNU/Linux, l'effet de cache est encore plus important :

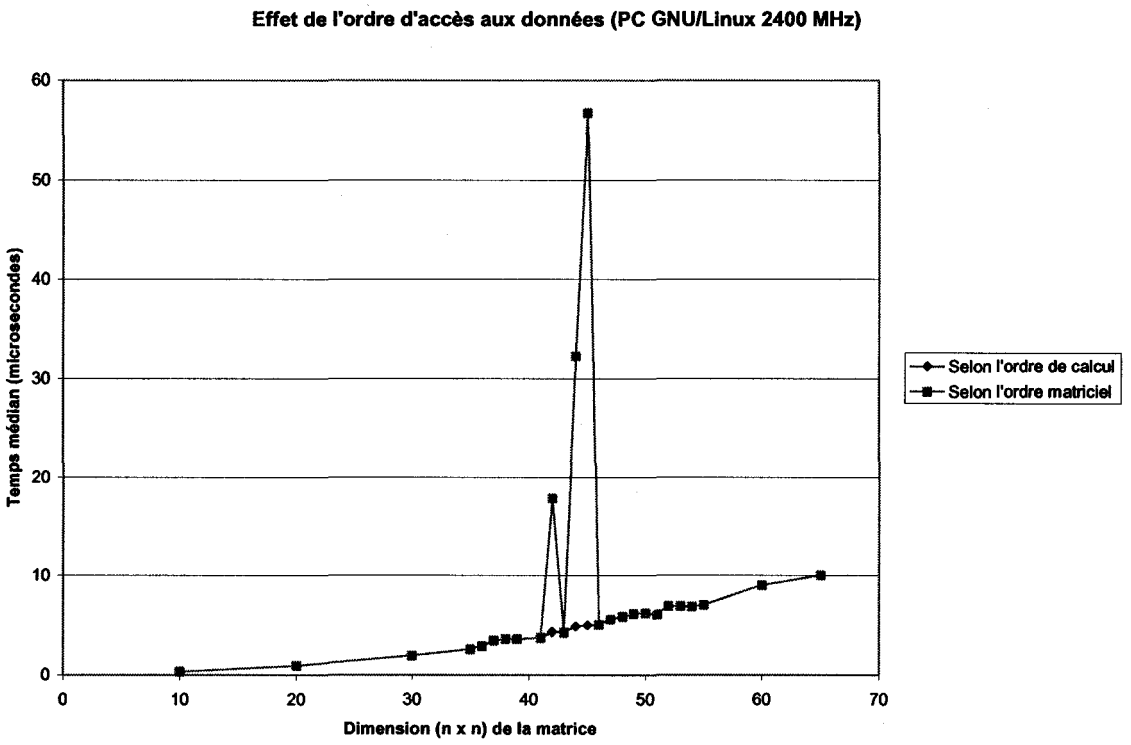

Figure 11 Effet de l'ordre d'accès aux données (PC GNU/Linux 2400 MHz)

6.4.5 Effet de la dispersion des données

En utilisant un ordre d'accès selon le calcul, tel que décrit dans la section précédente, on peut étudier l'effet de la dispersion des données, c'est-à-dire l'influence de la présence de trous (cases non utilisées) dans le vecteur.

$$\text{Taux de dispersion} = \frac{\text{Nombre de cases inutilisées}}{\text{Nombre de cases utilisées}}$$

Les figures 12, 13 et 14 montrent que la dispersion des données n'a que peu d'effets sur le temps de calcul :

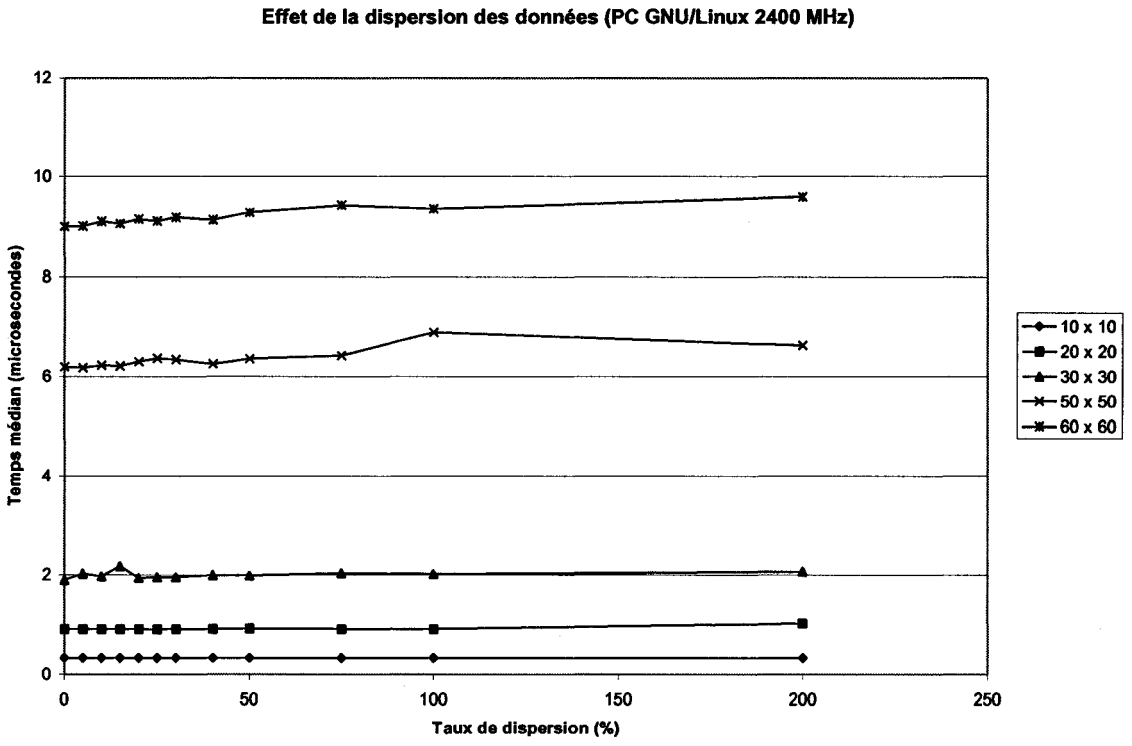

Figure 12 Effet de la dispersion des données (PC GNU/Linux 2400 MHz)

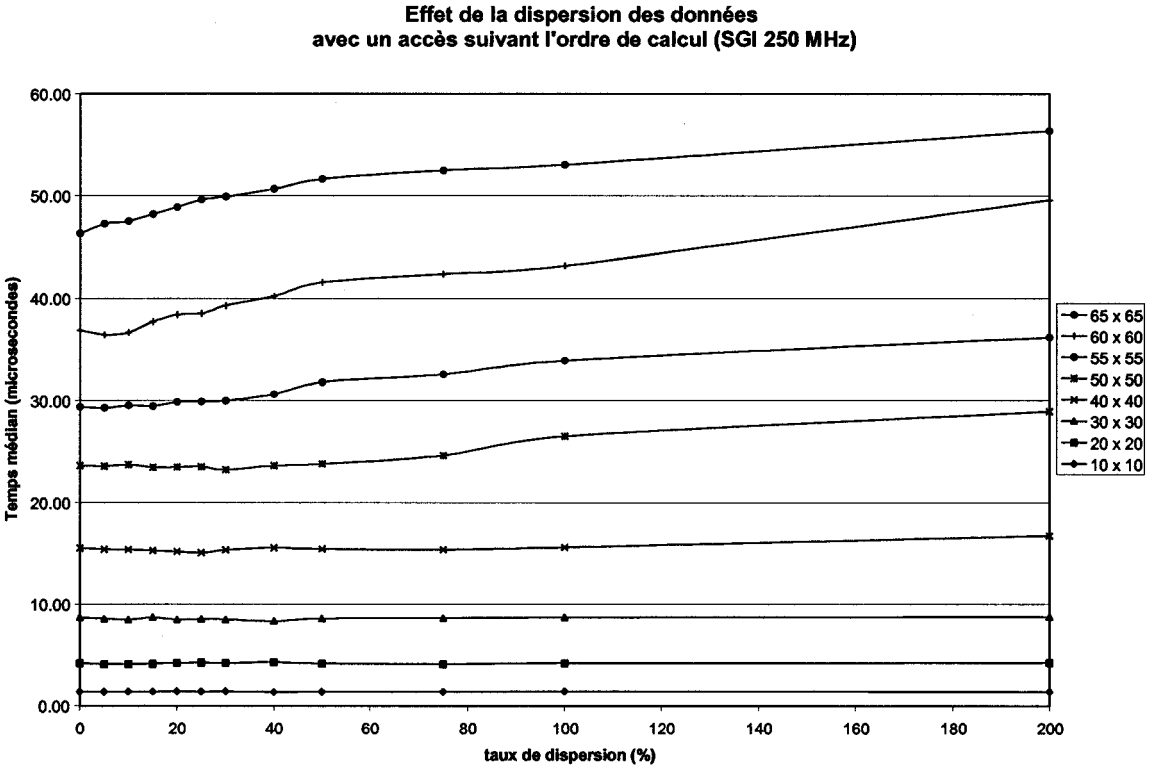

Figure 13 Dispersion des données, accès selon l'ordre de calcul (SGI 250 MHz)

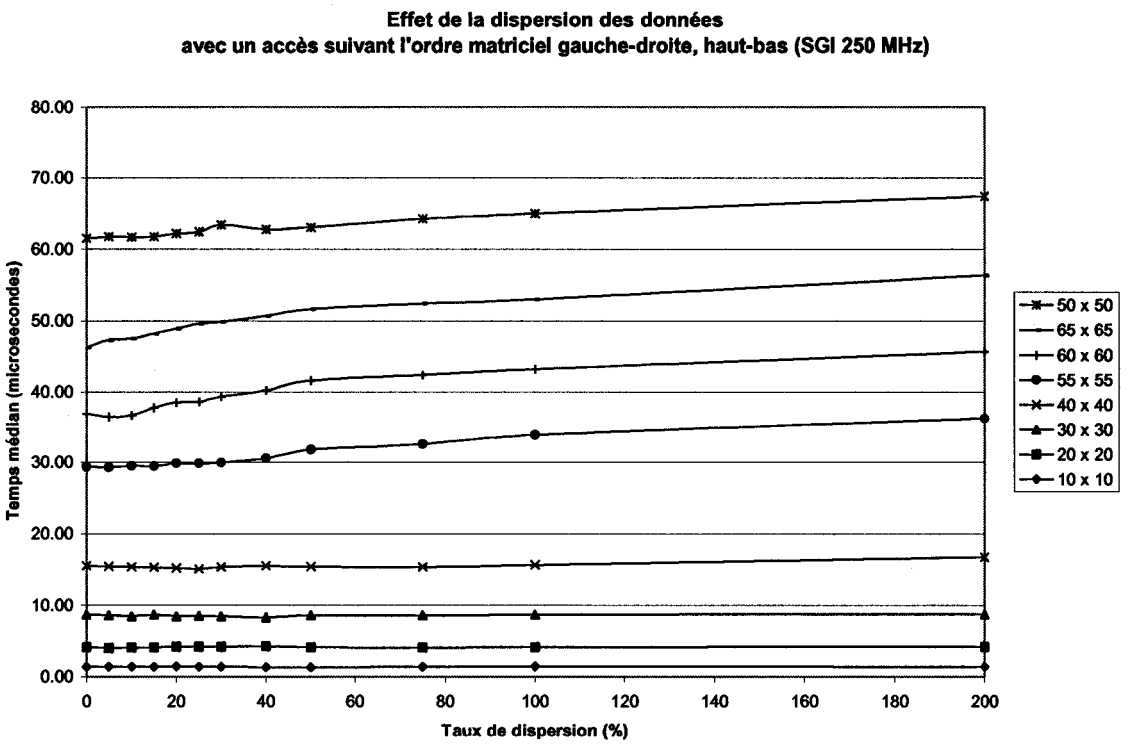

Figure 14 Dispersion des données selon un accès matriciel (SGI 250 MHz)

6.4.6 Comparaison Silicon Graphics et PC GNU/Linux

Un dernier résultat compare la performance d'une machine SGI à 500 MHz par rapport à un PC à 2400 MHz.

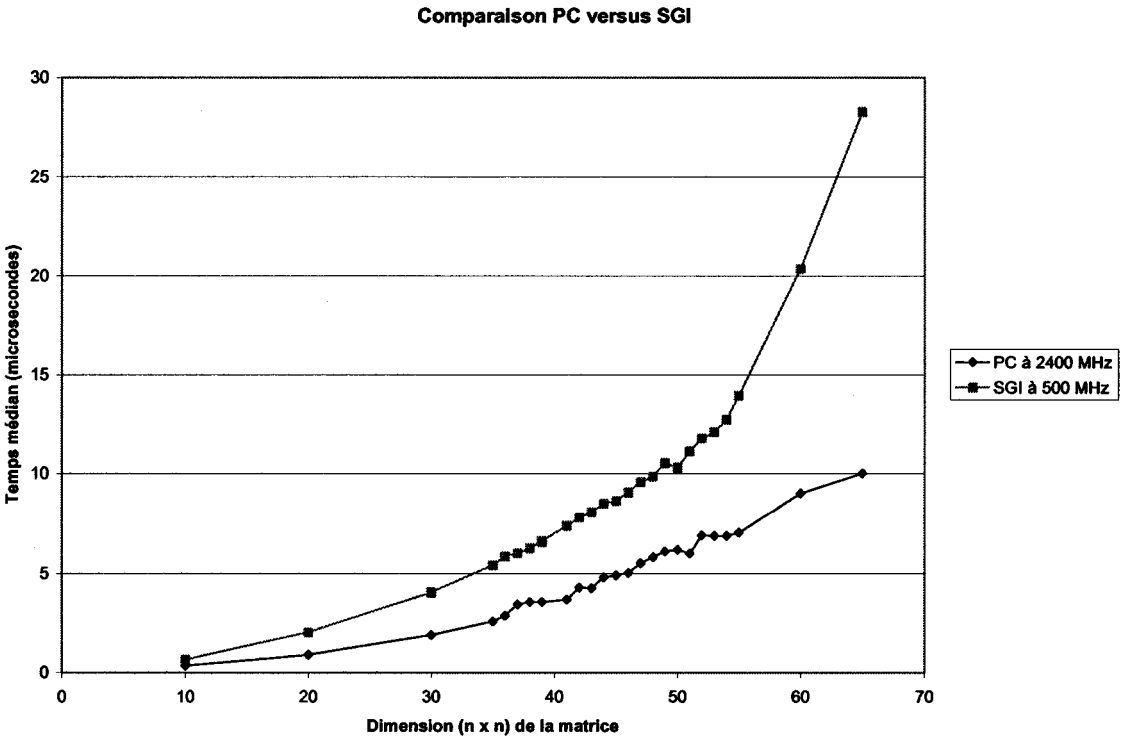

Figure 15 Comparaison d'un PC versus un SGI

Il est intéressant de constater que, bien que la performance du PC soit meilleure, elle n'est pas en proportion de sa vitesse d'horloge. Bien que ça ne soit pas vérifié, il est fort probable que la petitesse des caches du Pentium (PC) et la faible vitesse des bus locaux expliquent cette absence de rapport proportionnel.

CHAPITRE 7

MÉTHODE DE CALCUL

7.1 Recension des écrits

Cette section aborde la recension des écrits sur le problème de la parallélisation de la solution d'un système linéaire exprimé sous une forme matricielle. Plus précisément, sont explorées les différentes façons de paralléliser la décomposition LU et ses variantes.

7.1.1 Élimination gaussienne

Étant donné que l'intérêt de la méthode de la décomposition LU est de n'effectuer que la seconde partie du calcul et que ce cas-ci s'attarde au calcul complet, on est en droit de se demander s'il s'agit de la meilleure technique. L'autre technique connue est la méthode de l'élimination gaussienne. Comparons les deux techniques en terme du nombre d'opérations arithmétiques.

L'élimination gaussienne se fait en deux étapes: une triangularisation (en incluant le vecteur de droite « b ») et une substitution avant.

Soit « n » : la taille de la matrice carrée

Pour la triangularisation, le nombre d'opérations est :

$$\sum_{i=1}^{n-1} i\,(i+1) = \frac{n^3}{3} - \frac{n}{3} \quad \text{divisions et soustractions.} \tag{7.1}$$

Pour la substitution avant, le nombre d'opérations est :

$$\sum_{i=1}^{n-1} i = \frac{n^2}{2} - \frac{n}{2} \quad \text{multiplications et soustractions} \tag{7.2}$$

et

n divisions.

Ce qui fait, en tout :

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{n}{2} \quad \text{soustractions} \tag{7.3}$$

$$\frac{n^2}{2} - \frac{n}{2} \quad \text{multiplications} \tag{7.4}$$

$$\frac{n^3}{3} + \frac{2n}{3} \quad \text{divisions} \tag{7.5}$$

7.1.2 Décomposition LU

Soit : $Ax = b$ (7.6)

Posons :

$A = LU$ (7.7)

où L : la matrice triangulaire basse (diagonale à 1)

U : la matrice triangulaire haute (diagonale non unitaire)

A est non singulière

et posons

$Ux = y$ (7.8)

On a alors

$Ly = b$ (7.9)

$Ux = y$ (7.10)

Deux équations faciles à résoudre puisque les matrices sont triangulaires.

La décomposition LU se fait en quatre étapes : le calcul de « L », le calcul de « U », la substitution avant et la substitution arrière.

Pour la décomposition de « L », le nombre d'opérations est :

$$\sum_{i=1}^{n-1}\sum_{j=1}^{i} j = \frac{n!}{3!\,(n-3)!} < \frac{(n-1)^3}{6} \quad \text{multiplications et soustractions} \qquad (7.11)$$

et

$$\sum_{i=1}^{n-1} i = \frac{n^2-n}{2} \quad \text{divisions} \qquad (7.12)$$

Pour la décomposition de « U », le nombre d'opérations est :

$$\sum_{i=1}^{n}\sum_{j=1}^{i} j = \frac{(n+1)!}{3!\,(n+1-3)!} < \frac{n^3}{6} \quad \text{multiplications et soustractions} \qquad (7.13)$$

Pour la substitution avant et arrière, le nombre d'opérations est :

$$n^2 - n \quad \text{multiplications et soustractions} \qquad (7.14)$$

et

$$n \quad \text{divisions} \qquad (7.15)$$

Ce qui fait en tout, en supposant que « n » est grand et que « $n - 1 \cong n$ » :

$$\frac{n^3}{3} + n^2 - n \quad \text{soustractions} \qquad (7.16)$$

$$\frac{n^3}{3} + n^2 - n \quad \text{multiplications} \qquad (7.17)$$

$$\frac{n^2}{2} + \frac{n}{2} \quad \text{divisions} \qquad (7.18)$$

On constate que la méthode de l'élimination gaussienne, bien qu'ayant un plus petit nombre d'opérations, a un plus grand nombre de divisions, ce qui rend cette méthode peu attrayante.

7.1.3 Variante de la décomposition LU utilisée par le simulateur

La variante utilisée par le simulateur Hypersim consiste à inverser immédiatement la matrice L.

Soit : $A = LU$ (7.19)

et

$$Ly = b \quad (7.20)$$

$$Ux = y \quad (7.21)$$

« L » étant une matrice triangulaire basse et ayant les éléments de la diagonale à « 1 », il facile de trouver son inverse multiplicatif.

On obtient alors :

$$L^{-1}Ly = L^{-1}b \quad (7.22)$$

qui devient :

$$Ux = L^{-1}b \quad (7.23)$$

Cette méthode permet de remplacer la substitution avant par une multiplication matricielle, mais ajoute le calcul de l'inverse de « L ».

Une multiplication matricielle par un vecteur, pour une matrice triangulaire basse, coûte:

$$\frac{n^2}{2} \text{ soustractions} \quad \text{et} \quad \frac{(n-1)^2}{2} \text{ multiplications} \quad (7.24)$$

et le calcul de l'inverse de « L » coûte :

$$\sum_{i=1}^{n-1}\sum_{j=1}^{i} j = \frac{n!}{3!\,(n-3)!} < \frac{(n-1)^3}{6} \quad \text{soustractions} \tag{7.25}$$

et

$$\sum_{i=1}^{n-2}\sum_{j=1}^{i} j = \frac{n!}{3!\,(n-3)!} < \frac{(n-2)^3}{6} \quad \text{multiplications} \tag{7.26}$$

Si on ajoute la décomposition de L, toujours requise :

$$\frac{n^3}{6} \quad \text{multiplications et soustractions} \tag{7.27}$$

et

$$\frac{n^2}{2} - \frac{n}{2} \quad \text{divisions} \tag{7.28}$$

la decomposition de U

$$\frac{n^3}{6} \quad \text{multiplications et soustractions} \tag{7.29}$$

et la substitution arrière

$$\frac{n^2}{2} - \frac{n}{2} \quad \text{multiplications et soustractions} \tag{7.30}$$

et

$$n \quad \text{divisions} \tag{7.31}$$

Ce qui fait, en tout :

$$\frac{n^3}{2} + n^2 - \frac{n}{2} \quad \text{soustractions} \tag{7.32}$$

$$\frac{n^3}{2} + n^2 - \frac{n}{2} \quad \text{multiplications} \tag{7.33}$$

$$\frac{n^2}{2} + \frac{n}{2} \quad \text{divisions} \tag{7.34}$$

7.1.4 Stabilité de la méthode

La décomposition LU est très sensible au choix du pivot. Si la matrice est à diagonale dominante, c'est-à-dire que chaque élément de la diagonale est plus grand que la somme des autres éléments de la même ligne, alors le choix du pivot est valide. Si cette condition n'existe pas, le calcul peut devenir instable (non convergent) et il est fortement souhaitable de permuter les différentes lignes de la matrice afin de la rendre diagonalement dominante. Si la matrice n'est pas diagonalement dominante, une accumulation d'erreurs dues à une division par un nombre trop petit peut largement fausser les calculs.

Il est à noter que la méthode de l'élimination gaussienne, les méthodes dérivées de LU et les méthodes itératives ont toutes cette même sensibilité aux éléments de la diagonale.

7.1.5 Décomposition LDLT

Si la matrice « A » est symétrique, il est alors possible de diminuer de moitié le nombre de calculs.

7.1.5.1 Décomposition LDMT

Soit : A = LU où L : la matrice triangulaire basse (diagonale à 1)
U : la matrice triangulaire haute (diagonale non unitaire)
A est non singulière

Posons D : la matrice diagonale des éléments de la diagonale de U.

D^{-1}: est l'inverse multiplicatif de D
(ce qui est trivial dans le cas d'une matrice diagonale)

Il est alors possible de facilement calculer :

$$M^T = D^{-1}U \tag{7.35}$$

La décomposition LU devient :

$$A = LDM^T \tag{7.36}$$

car

$$A = LU = LD(D^{-1}U) = LDM^T \tag{7.37}$$

7.1.5.2 Décomposition LDLT

De plus, si la matrice « A » est symétrique (par rapport à la diagonale), on a :

$$M^T = L^T \tag{7.38}$$

Ce qui fait

$$A = LDL^T \tag{7.39}$$

Ce qui permet de diminuer le nombre de calculs de moitié, puisque le calcul de « L » a déjà été effectué.

Pour la décomposition LU, la recherche de la solution « x » dans le cas de « Ax = b » s'effectue en deux étapes :

Soit :

$$LUx = b \tag{7.40}$$

Posons :

$$Ux = y \tag{7.41}$$

On a alors

$$Ly = b \tag{7.42}$$

« y » est facile à déterminer par substitution avant puisque « L » est triangulaire.

Ce qui permet ensuite de déterminer « x » par une substitution arrière, puisque « U » est aussi triangulaire.

Dans le cas de la décomposition « LDL^T » , on a une étape supplémentaire :

Soit :

$$LDL^T x = b \tag{7.43}$$

Posons :

$$DL^T x = y \tag{7.44}$$

Et :

$$L^T x = z \tag{7.45}$$

On a alors

$$Ly = b \tag{7.46}$$

$$Dz = y \tag{7.47}$$

$$L^T x = z \tag{7.48}$$

L'ordre de calcul de la décomposition LDL^T est similaire à celui de la décomposition LU.

Pour la décomposition de « L », le nombre d'opérations est :

$$\sum_{i=1}^{n-1} \sum_{j=1}^{i} j = \frac{n!}{3!\,(n-3)!} < \frac{(n-1)^3}{6} \quad \text{multiplications et soustractions} \tag{7.49}$$

et

$$\sum_{i=1}^{n-1} i = \frac{n^2 - n}{2} \quad \text{divisions} \tag{7.50}$$

Pour la substitution avant et arrière, le nombre d'opérations est :

$$n^2 - n \quad \text{multiplications et soustractions} \tag{7.51}$$

et

$$n \quad \text{divisions} \tag{7.52}$$

Ce qui fait en tout, en supposant que « n » est grand et que « $n - 1 \cong n$ » :

$$\frac{n^3}{6} + n^2 - n \quad \text{soustractions} \tag{7.53}$$

$$\frac{n^3}{6} + n^2 - n \quad \text{multiplications} \tag{7.54}$$

$$\frac{n^2}{2} + \frac{n}{2} \quad \text{divisions} \tag{7.55}$$

La méthode de la décomposition LDL^T permet d'économiser « $n^3 / 6$ » multiplications et soustractions par rapport à la décomposition LU.

7.1.6 Décomposition de Cholesky

La décomposition de Cholesky est très semblable à la décomposition LDL^T, mais requiert une condition supplémentaire : la matrice doit être définie positivement.

Soit « A » une matrice symétrique, diagonalement dominante, définie positivement et non singulière.

Il existe une matrice L telle que :

$$A = LL^T \tag{7.56}$$

Alors

$$Ax = b \tag{7.57}$$

Devient :

$$LL^T x = b \tag{7.58}$$

Posons :

$$L^T x = y \tag{7.59}$$

On a alors

$$Ly = b \tag{7.60}$$

Ce qui permet de solutionner le système.

L'ordre de calcul de la décomposition de Cholesky est similaire, mais légèrement plus grand que celui de la décomposition LDL^T.

La décomposition de « L » a le nombre d'opérations suivant :

$$\sum_{i=1}^{n-1} \sum_{j=1}^{i} j = \frac{n!}{3!\,(n-3)!} < \frac{(n-1)^3}{6} \quad \text{multiplications et soustractions} \tag{7.61}$$

$$\sum_{i=1}^{n-1} i = \frac{n^2 - n}{2} \quad \text{divisions} \tag{7.62}$$

et

$$\mathbf{n} \quad \text{racines carrées} \tag{7.63))}$$

Pour la substitution avant et arrière, le nombre d'opérations est :

$$n^2 - n \quad \text{multiplications et soustractions} \tag{7.64}$$

mais

$$2n \text{ divisions.} \tag{7.65}$$

Ce qui fait en tout :

$$\frac{n^3}{6} + n^2 - n \quad \text{soustractions} \tag{7.66}$$

$$\frac{n^3}{6} + n^2 - n \quad \text{multiplications} \tag{7.67}$$

$$\frac{n^2}{2} + \frac{3n}{2} \quad \text{divisions} \tag{7.68}$$

$$n \quad \text{racines carrées} \tag{7.69}$$

Au total, si on considère l'ensemble des opérations, la décomposition LL^T suivie de la substitution avant/arrière permet d'économiser par rapport à LU, « $n^3/6$ » multiplications et soustractions mais ajoute « n » racines carrées et « n » divisions.

7.1.7 Comparaison des ordres de calcul

Le tableau suivant montre une comparaison du nombre d'opérations requises par les différentes méthodes. Dans tous les cas, le nombre d'opérations inclut la substitution avant et la substitution arrière.

Tableau VII

Comparaison des ordres de calcul

Méthode	Soustractions	Multiplications	Divisions	Total (non pondéré)
Élimination gaussienne	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$	$\frac{n^2}{2} - \frac{n}{2}$	$\frac{n^3}{3} + \frac{2n}{3}$	$\frac{2n^3}{3} + n^2 - \frac{2n}{3}$
Décomposition LU	$\frac{n^3}{3} + n^2 - n$	$\frac{n^3}{3} + n^2 - n$	$\frac{n^2}{2} + \frac{n}{2}$	$\frac{2n^3}{3} + \frac{5n^2}{2} - \frac{3n}{2}$
Décomposition LU modifiée	$\frac{n^3}{2} + n^2 - \frac{n}{2}$	$\frac{n^3}{2} + n^2 - \frac{n}{2}$	$\frac{n^2}{2} + \frac{n}{2}$	$n^3 + \frac{5n^2}{2} - \frac{n}{2}$
Décomposition LDL^T	$\frac{n^3}{6} + n^2 - n$	$\frac{n^3}{6} + n^2 - n$	$\frac{n^2}{2} + \frac{n}{2}$	$\frac{n^3}{3} + \frac{5n^2}{2} - \frac{3n}{2}$
Décomposition de Cholesky	$\frac{n^3}{6} + n^2 - n$	$\frac{n^3}{6} + n^2 - n$	$\frac{n^2}{2} + \frac{3n}{2}$ et « n » racines carrées	$\frac{n^3}{3} + 3n^2 + \frac{n}{2}$

7.1.8 Ordre du calcul et dépendance des variables

En décomposant le calcul selon les différentes méthodes, il est possible d'établir la dépendance entre les variables. Cet ordre représente la principale limite de la parallélisation.

7.1.8.1 Décomposition LU

La figure suivante illustre la dépendance entre les variables pour la décomposition LU.

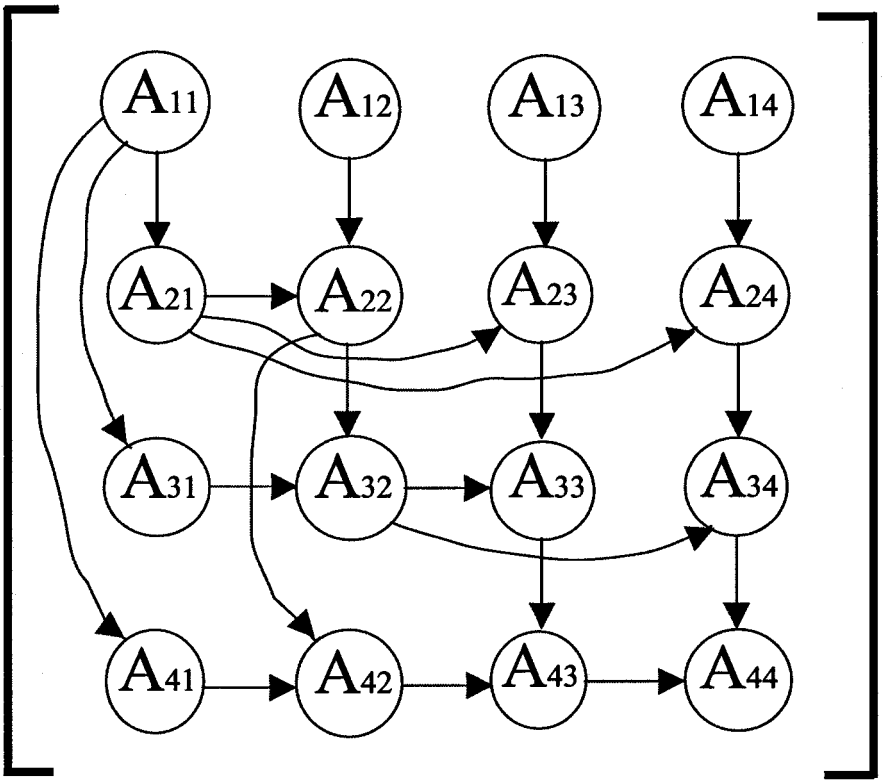

Figure 16 Dépendance des variables de la décomposition LU

De façon plus schématique, l'ordre du calcul est illustré par la figure suivante :

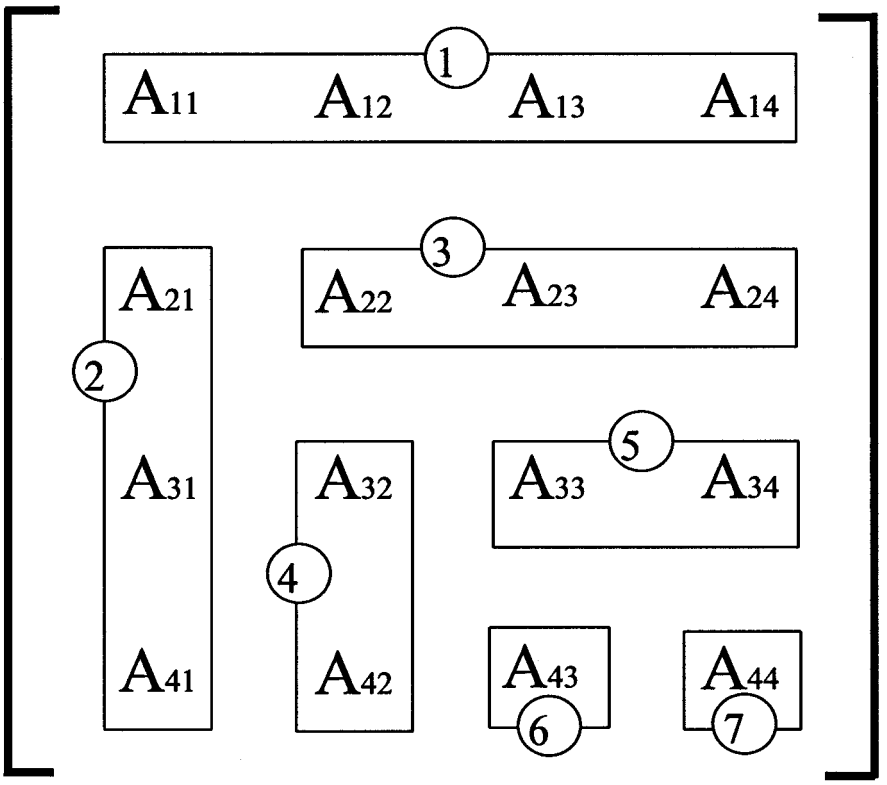

Figure 17 Dépendance des variables de la décomposition LU schématisée

On remarque que le calcul s'effectue en « 2n - 1 » étapes successives.

7.1.8.2 Décomposition de Cholesky

On trouvera à la figure suivante la dépendance entre les variables pour la décomposition de Cholesky.

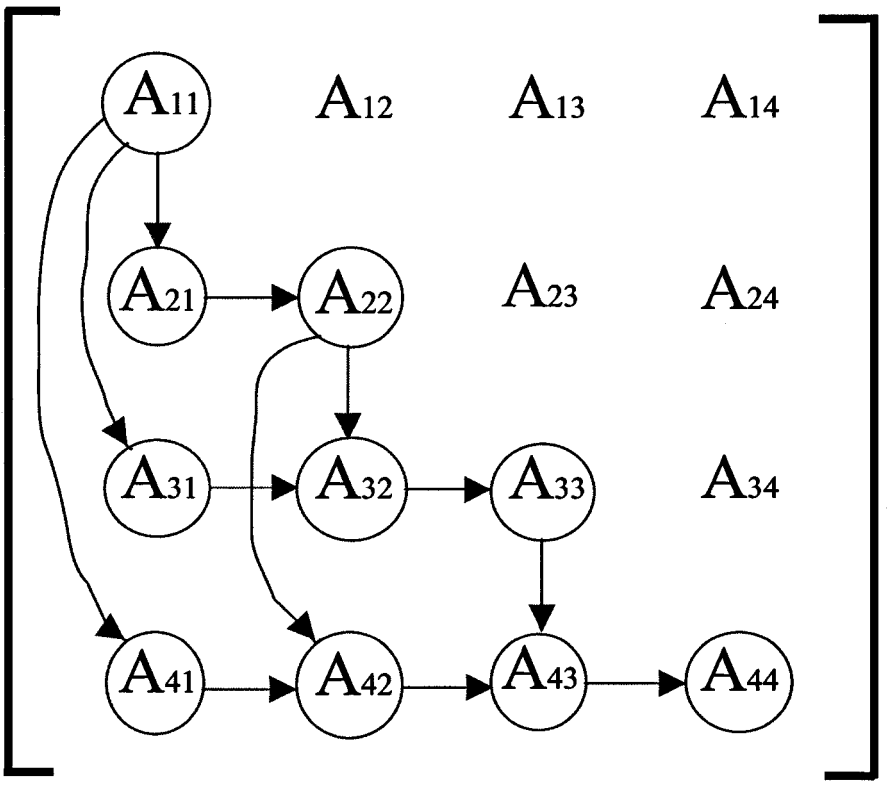

Figure 18 Dépendance des variables de la décomposition de Cholesky

Schématiquement, l'ordre du calcul est illustré de cette façon :

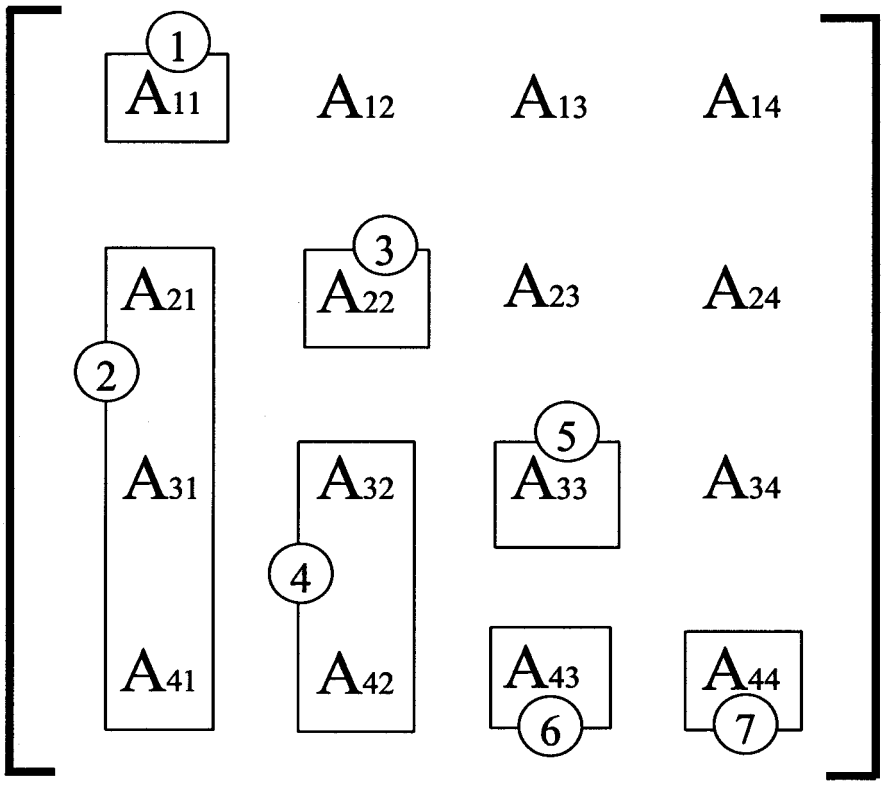

Figure 19 Dépendance des variables de la décomposition de Cholesky schématisée

On remarque aussi que le calcul s'effectue en « 2n - 1 » étapes successives.

7.1.8.3 Substitution avant

Les substitutions avant ou arrière ont le même ordre de calcul à une transposition près. La substitution avant peut être parallélisée, à condition d'utiliser des sommes partielles. Dans ce cas, le schème de dépendance devient :

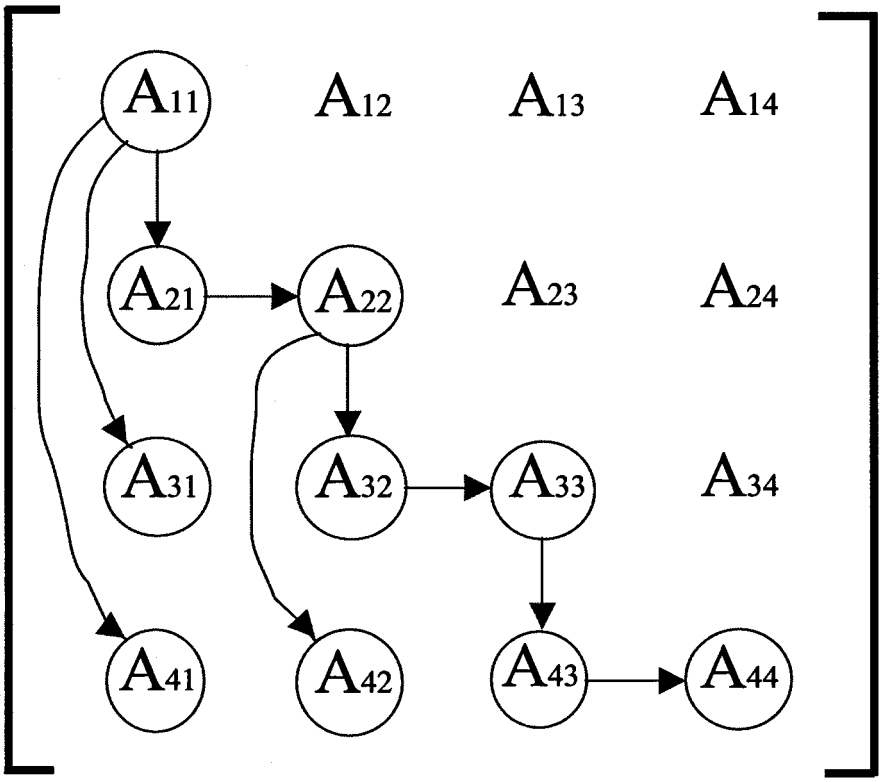

Figure 20 Dépendance des variables de la substitution avant

C'est-à-dire, de façon plus schématique :

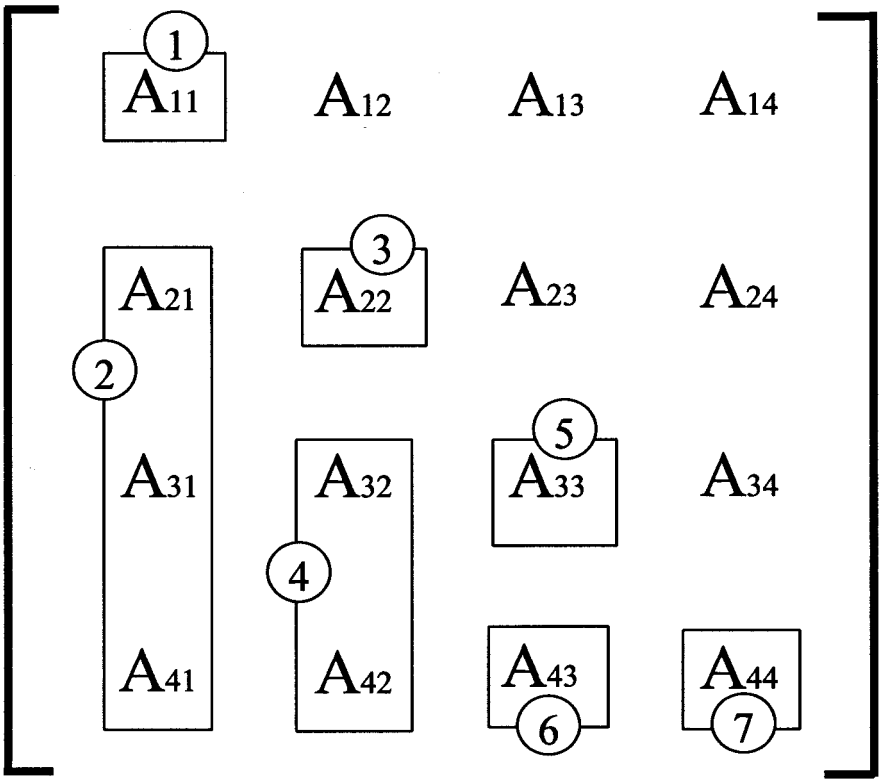

Figure 21 Dépendance des variables de la substitution avant schématisée

Dans ce cas-ci, on a aussi « 2n – 1 » étapes consécutives de calcul.

7.1.9 Limites théoriques de la parallélisation

Cette section présente les limites théoriques de la parallélisation de la décomposition LU et ses variantes.

7.1.9.1 Le cas de base (le cas ¾)

Le premier cas, le cas de base, permettant une première forme de parallélisation, consiste à commencer la substitution avant dès que la matrice L est calculée

Décomposition L Décomposition U

---------- temps ---------->

Substitution avant Substitution arrière

L'avantage de cette technique est qu'elle nécessite un faible nombre de communications. Si on considère qu'une ligne de cache contient 16 nombres à point flottant (de type « double », sur un Silicon Graphics Origine 3000), on peut estimer le coût de communication à « $(n^2/2) / 16$ ».

En supposant que le coût d'une communication est 100 fois supérieur au temps d'exécution d'une instruction[2], on obtient la fonction suivante :

$$\text{Économie (\%)} = \frac{\frac{n^3}{3} + 3n^2 - \frac{3n}{2} + \frac{100}{32} n^2}{\frac{2n^3}{3} + 3n^2 - \frac{3n}{2}} \qquad (7.70)$$

[2] Le temps moyen interprocesseur est de 313 nanosecondes selon *Origin2000 and Onyx2 Performance Tuning and Optimization*, p. 19.

Illustré graphiquement, cela donne :

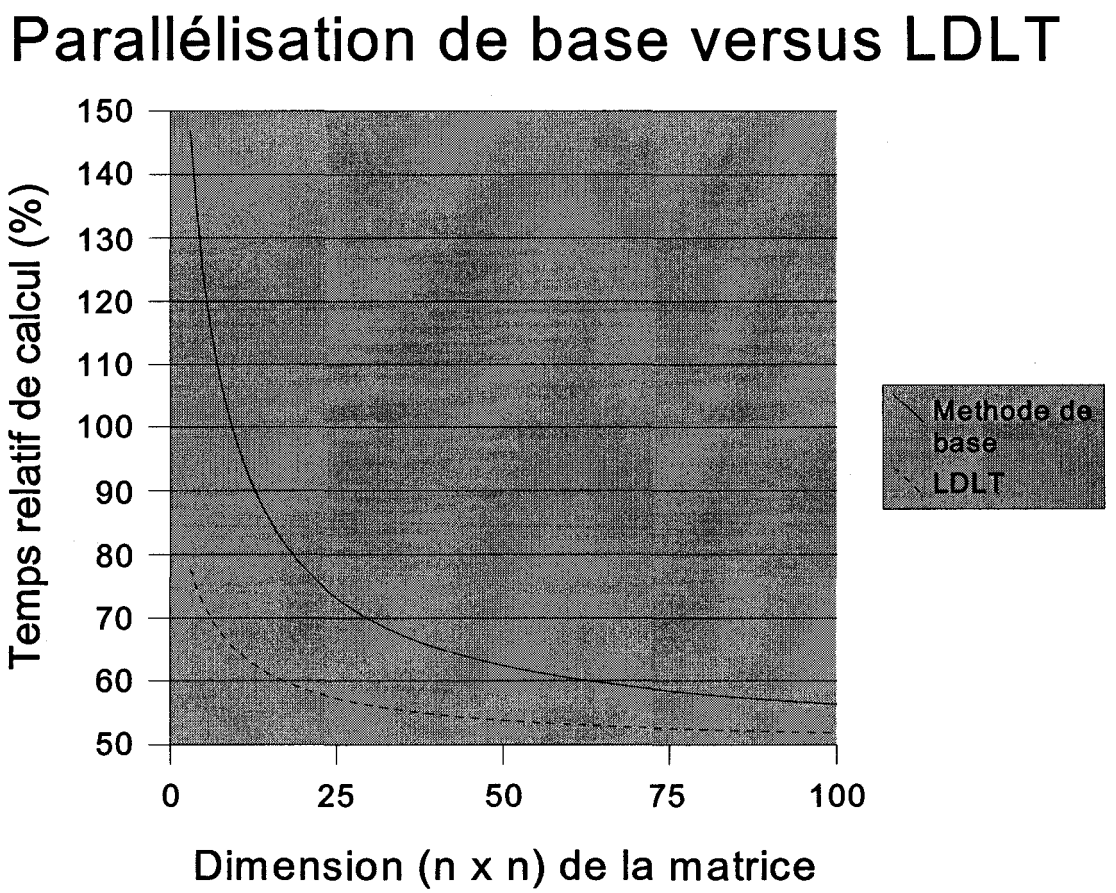

Figure 22 Parallélisation de base versus LDL^T

On remarque que cette technique de parallélisation permet d'économiser beaucoup de temps de calcul. Toutefois, compte tenu de la symétrie de la matrice, il est plus avantageux de remplacer la décomposition LU par une décomposition LDL^T.

7.1.9.2 Parallélisation de la décomposition LU, LDL^T et Cholesky

Dans le cas où il serait souhaitable de paralléliser sur plusieurs processeurs, il existe une autre technique qui consiste à distribuer le calcul par élément, par ligne, par colonne ou par bloc. Toutefois, compte tenu que la principale limite est le coût de communication,

on peut réduire le problème en ne tenant compte que de la méthode offrant le nombre de communications minimum. Dans les trois cas : LU, LDLT et Cholesky, le nombre de communications minimum est « 2n - 1 », pour la décomposition et « 2n - 1 » pour la substitution avant / arrière.

En prenant l'ordre de calcul de la décomposition LU :

$$\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{3}{2}n \tag{7.71}$$

En supposant que l'on ait « n » processeurs disponibles, le nombre de calculs requis est divisé par « n »

$$\frac{2}{3}n^2 + \frac{5}{2}n - \frac{3}{2} \tag{7.72}$$

Mais le coût de communication est :

$$((2n - 1) * n / 16) * 100 \tag{7.73}$$

Ce qui donne l'équation :

$$\text{Économie (\%)} = \frac{\frac{(2n^2 - n)}{16} * 100 + \frac{2}{3}n^2 - \frac{5}{2}n}{\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{3}{2}n} \tag{7.74}$$

Le graphique suivant illustre cette limite théorique pour une décomposition LU incluant la substitution avant/arrière.

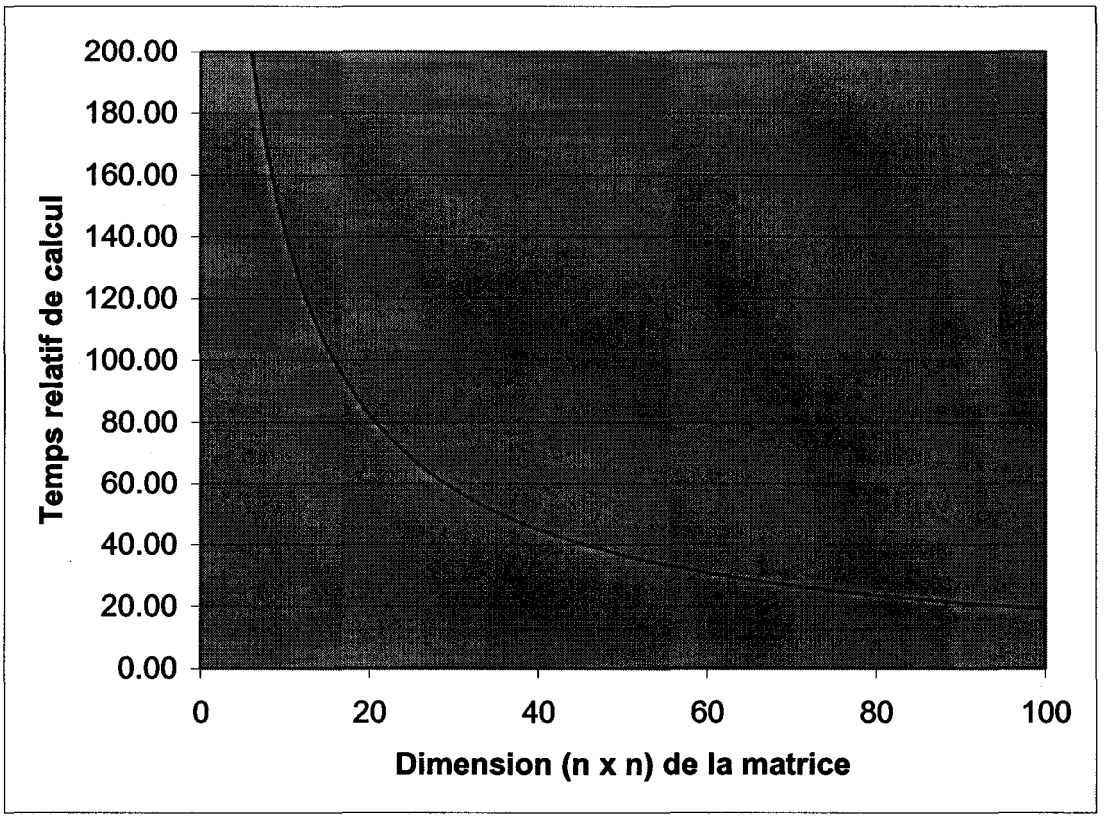

Figure 23 Parallélisation théorique de la décomposition LU

Cette courbe est très optimiste, car un coût de communication qui est seulement cent fois celui d'une instruction arithmétique est fort peu probable[3]. Toutefois, même avec ce coût assez faible, on remarque que cette méthode offre un bien moins bon rendement que la méthode précédente, compte tenu du nombre de processeurs utilisés.

[3] Le temps moyen interprocesseur est de 313 manosecondes selon *Origin2000 and Onyx2 Performance Tuning and Optimization*, p. 19.

7.1.10 Les méthodes itératives

7.1.10.1 La méthode de Jacobi

La méthode de Jacobi consiste à réécrire chaque ligne de la matrice en exprimant la variable de la diagonale comme une fonction des autres variables. Par exemple, pour une matrice trois par trois (Ax = b), les trois équations sont :

$$x_1 = (b_1 - a_{12} x_2 - a_{13} x_3) / a_{11} \quad (7.75)$$

$$x_2 = (b_2 - a_{21} x_1 - a_{23} x_3) / a_{22} \quad (7.76)$$

$$x_3 = (b_3 - a_{31} x_1 - a_{32} x_2) / a_{33} \quad (7.77)$$

À chaque itération, on calcule toutes les variables et on réinjecte le résultat dans la prochaine itération.

7.1.10.2 La méthode de Gauss-Siegel

La méthode de Gauss-Siegel est très similaire à la précédente, sauf que la variable calculée est réinjectée après chaque équation plutôt que chaque itération.

7.1.11 Les autres méthodes

Il existe un autre groupe de méthodes itératives, les méthodes par relaxation (ou sur-relaxation) qui peuvent converger plus vite que Gauss-Siegel.

7.2 Résultats

Cette section présente les résultats de la parallélisation et du changement d'algorithmes.

7.2.1 Décomposition LU

Le graphique suivant présente une mesure réelle de la parallélisation de la décomposition LU. On constate que le temps de communication est bien plus long que celui décrit par la limite théorique, ce qui rend la parallélisation peu intéressante.

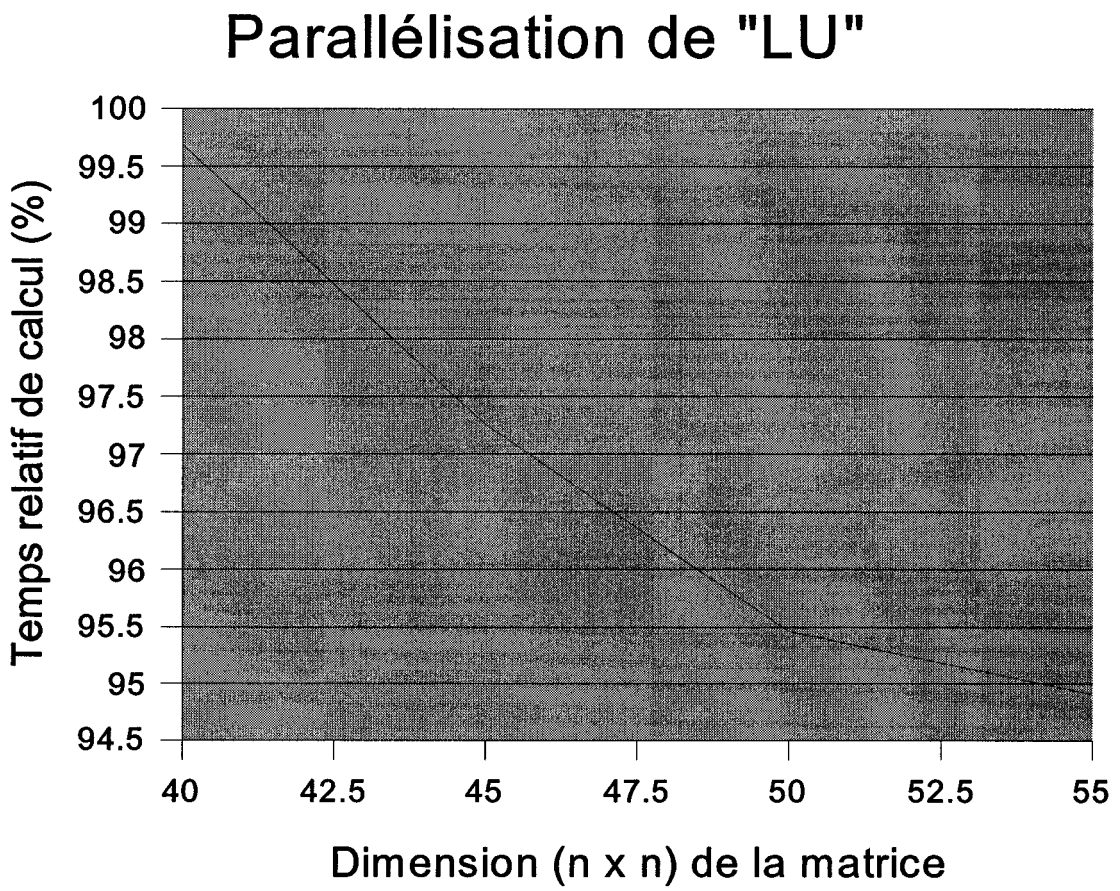

Figure 24 Mesure de la parallélisation de la décomposition LU

Le graphique précédent est le résultat d'une tentative de parallélisation de la décomposition LU (excluant la substitution avant/arrière) sur un ordinateur à mémoire partagée de type SGI Origin 3000 avec une matrice pleine de 55 éléments par 55 éléments. Le graphique suivant illustre cet essai. La première courbe, la courbe « brute », représente le temps de calcul pour une décomposition LU faite à partir d'une

matrice de 55 par 55. La seconde courbe représente des temps de calcul de la décomposition LU, mais faite à partir de structures de données vectorielles conçues de façon à minimiser le nombre d'échanges de données entre les processeurs.

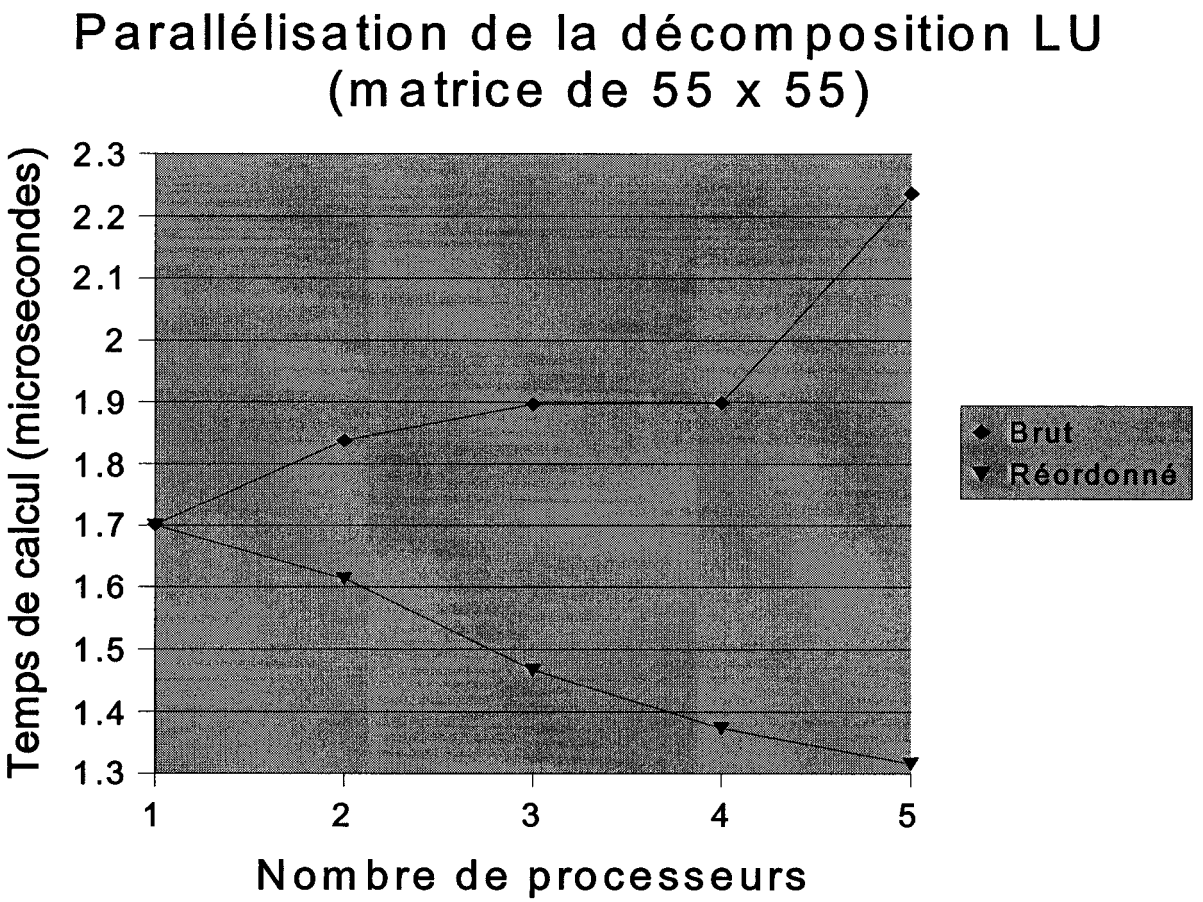

Figure 25 Parallélisation de la décomposition LU (55 x 55)

On remarque que le grand nombre de communications requises entre les processeurs ainsi que le coût de ces communications influencent grandement le temps de calcul. Dans le premier essai, la courbe dite « brute », le nombre de « cache miss » de niveau deux est si grand que le temps de calcul augmente au lieu de diminuer. Dans le second cas, malgré les efforts pour minimiser le mouvement en mémoire des données, la réduction du temps de calcul est bien faible (environ 20 %) par rapport aux ressources utilisées (5 processeurs).

La courbe pour la matrice de 55 par 55 a été choisie, car les matrices plus petites offrent des diminutions (en proportion) du temps de calcul encore moins intéressantes.

Cholesky versus LU sur une matrice creuse

La deuxième série d'essais effectués visait à vérifier si une méthode qui prend avantage de la symétrie de la matrice est valide même si la matrice est creuse.

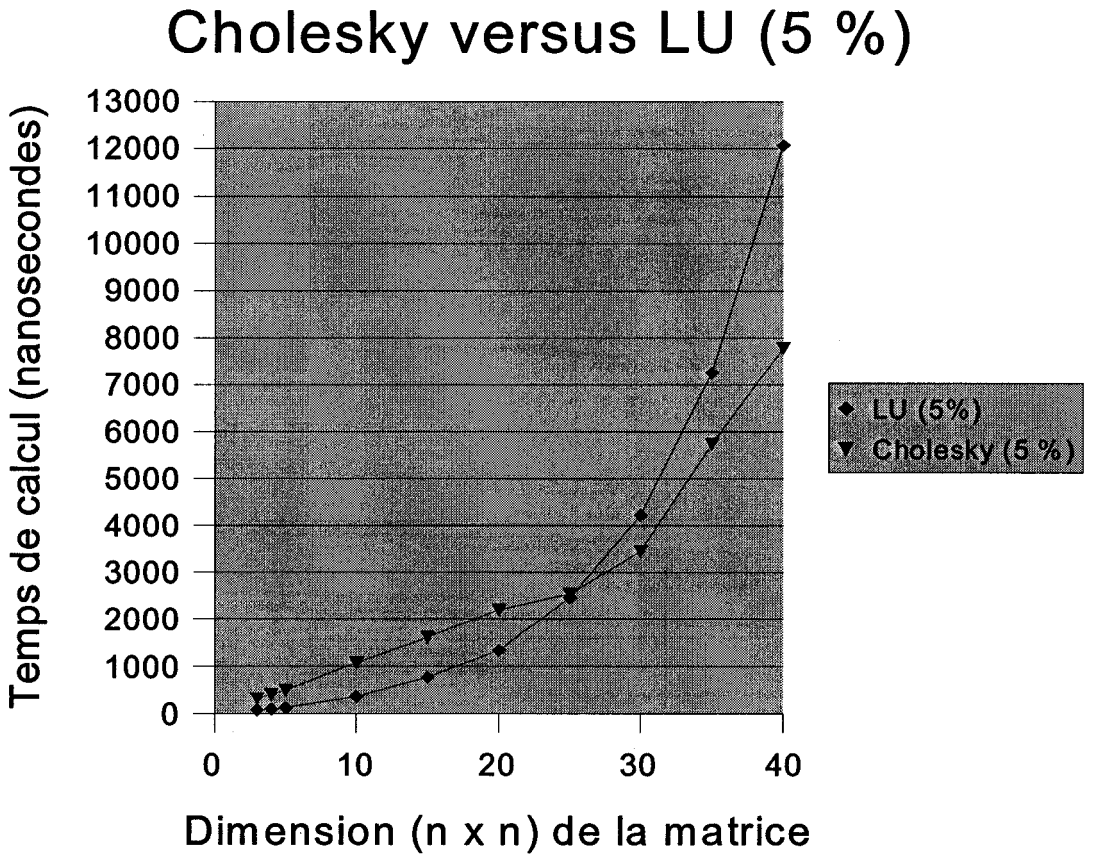

Figure 26 Cholesky versus LU (5 %)

Avec un taux de remplissage des éléments (excluant la diagonale) de 5%, on constate que le calcul selon Cholesky devient rentable si la matrice est supérieure à 25 par 25.

Avec un taux de remplissage de 10 %, l'avantage est autour de 20 par 20.

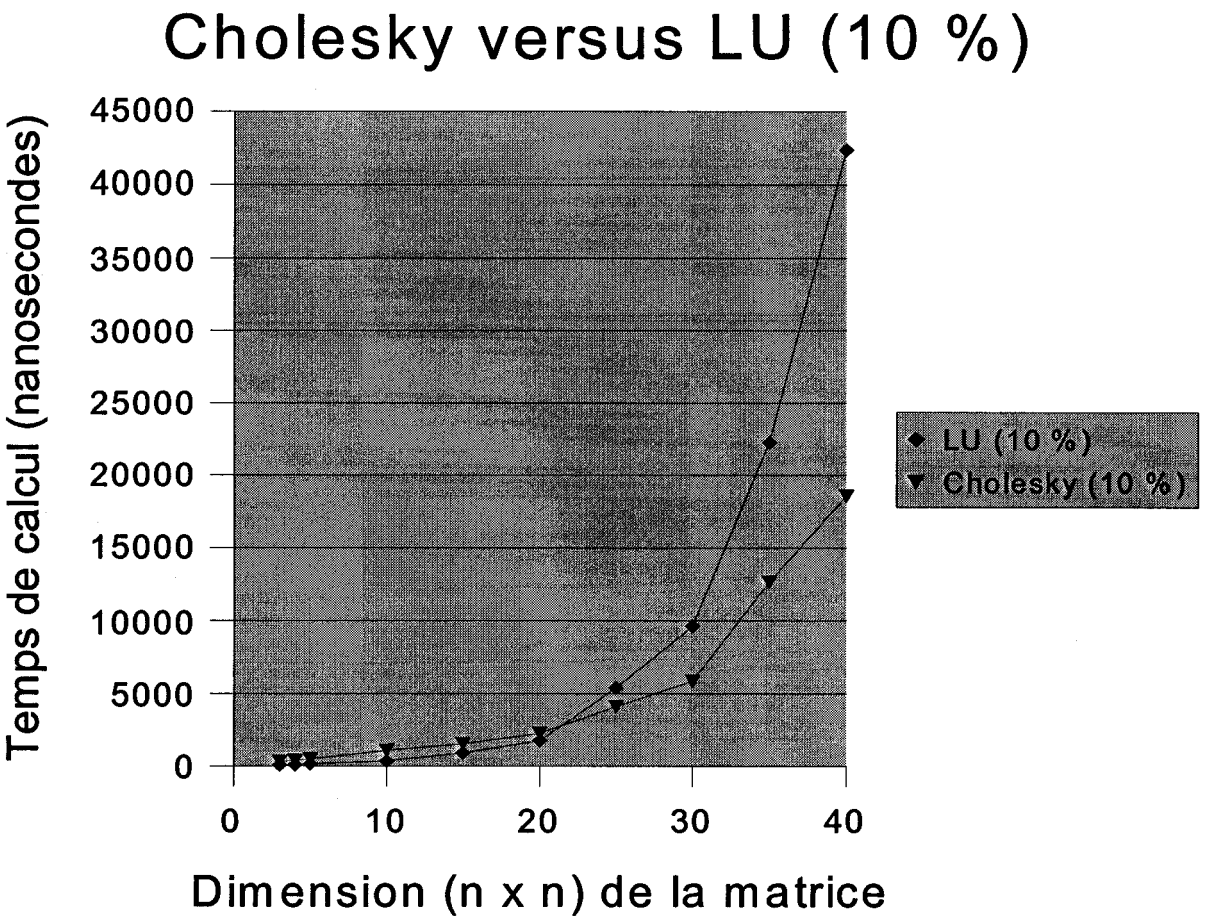

Figure 27 Cholesky versus LU (10 %)

Avec un taux de remplissage de 15 %, l'avantage n'a plus de limite inférieure.

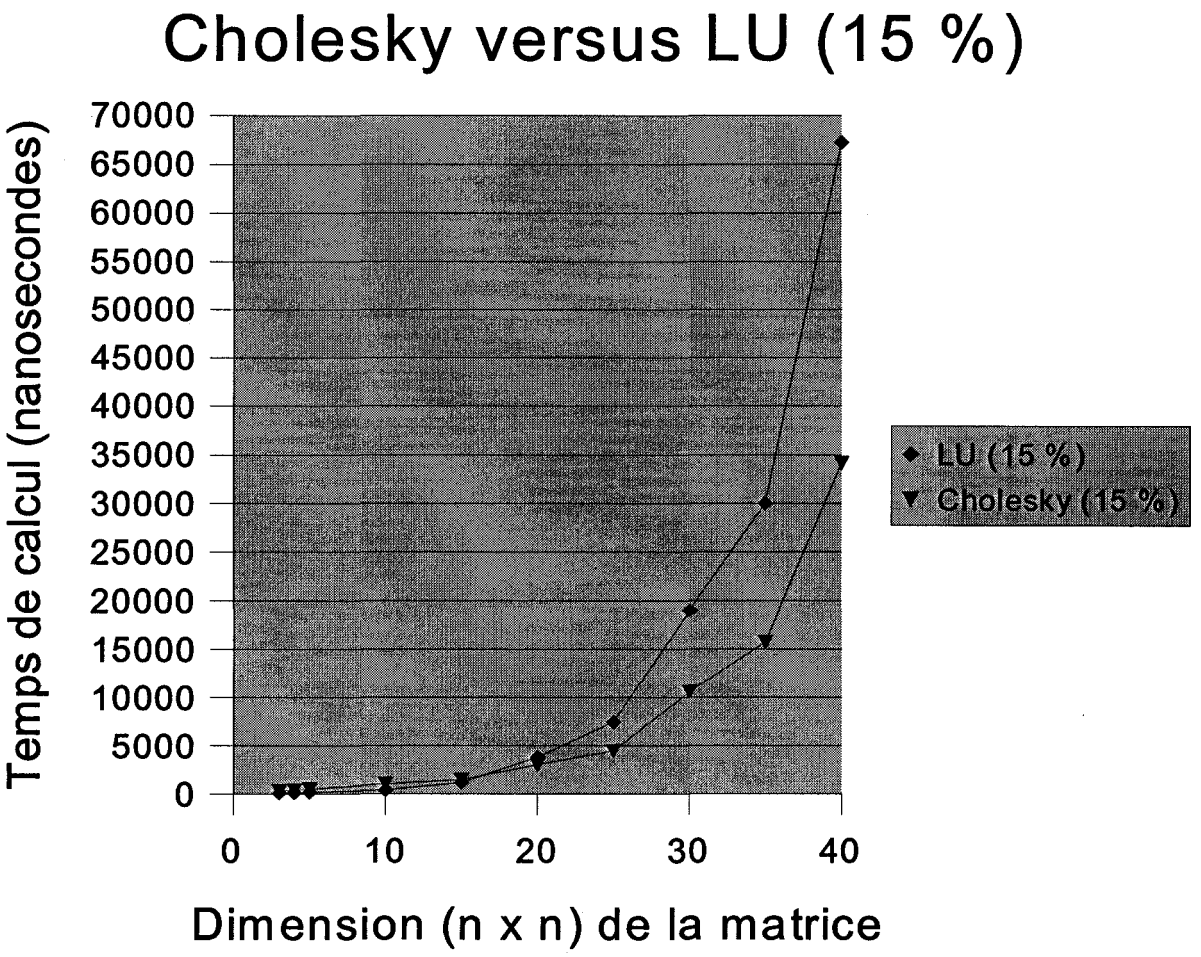

Figure 28 Cholesky versus LU (15 %)

7.2.2 LU, LU modifié et LDL^T sur une matrice creuse

Cette section présente les résultats obtenus en comparant les méthodes LU modifiées (utilisées par le simulateur), LU et LDL^T.

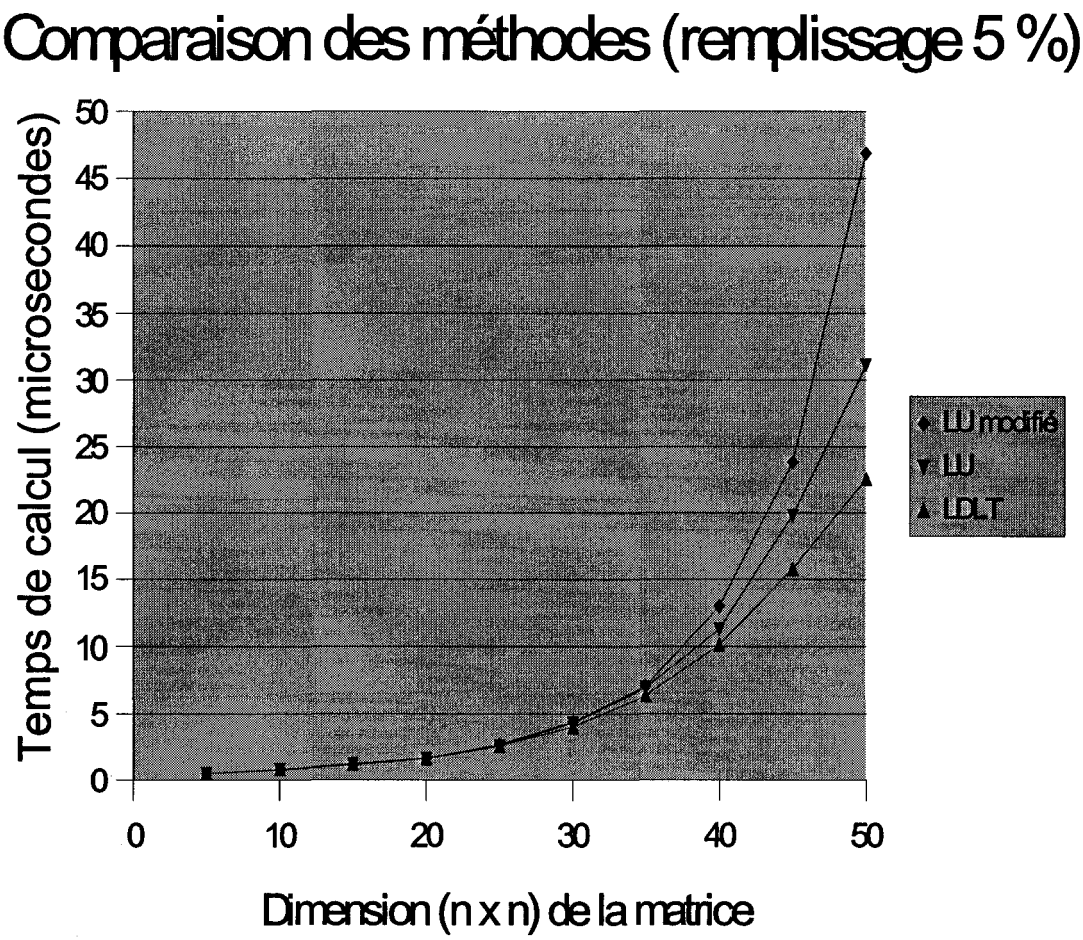

Figure 29 Comparaison des méthodes (5%)

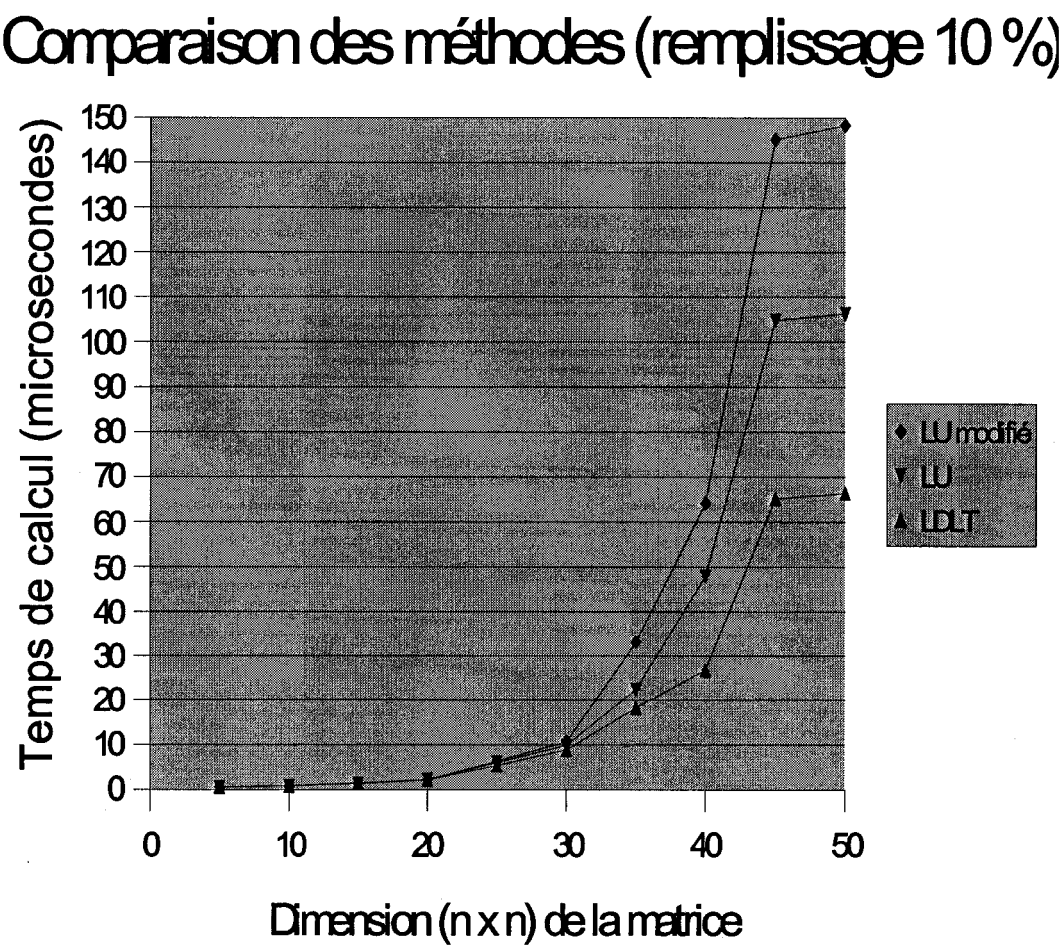

Figure 30 Comparaison des méthodes (10%)

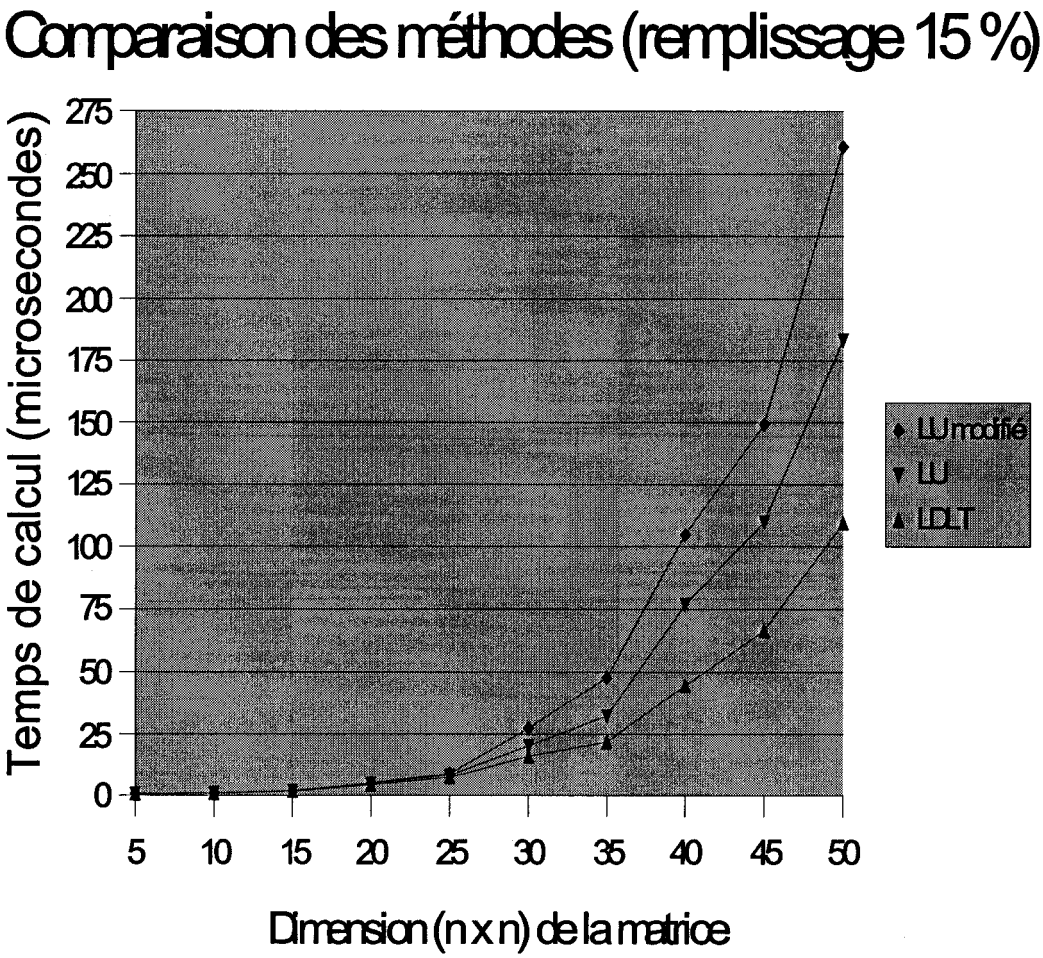

Figure 31 Comparaison des méthodes (15%)

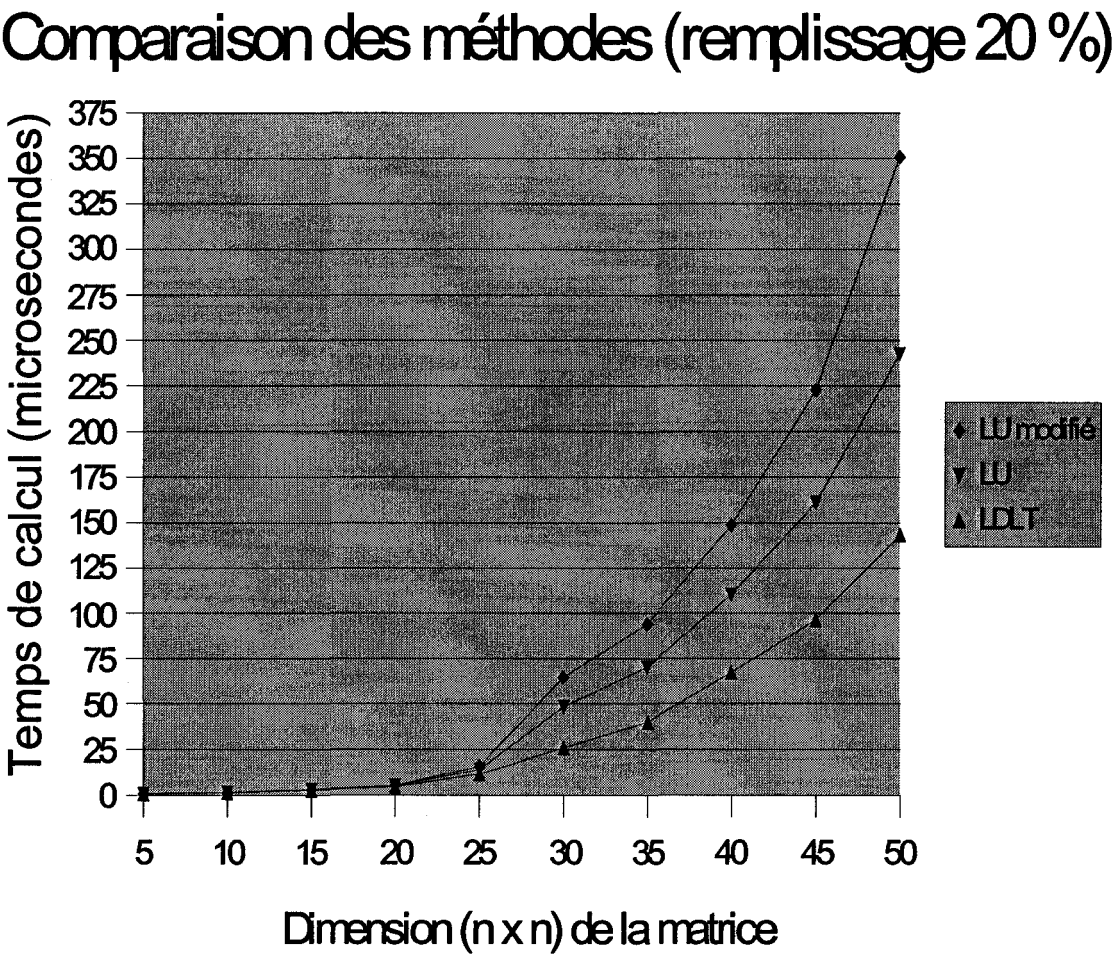

Figure 32 Comparaison des méthodes (20%)

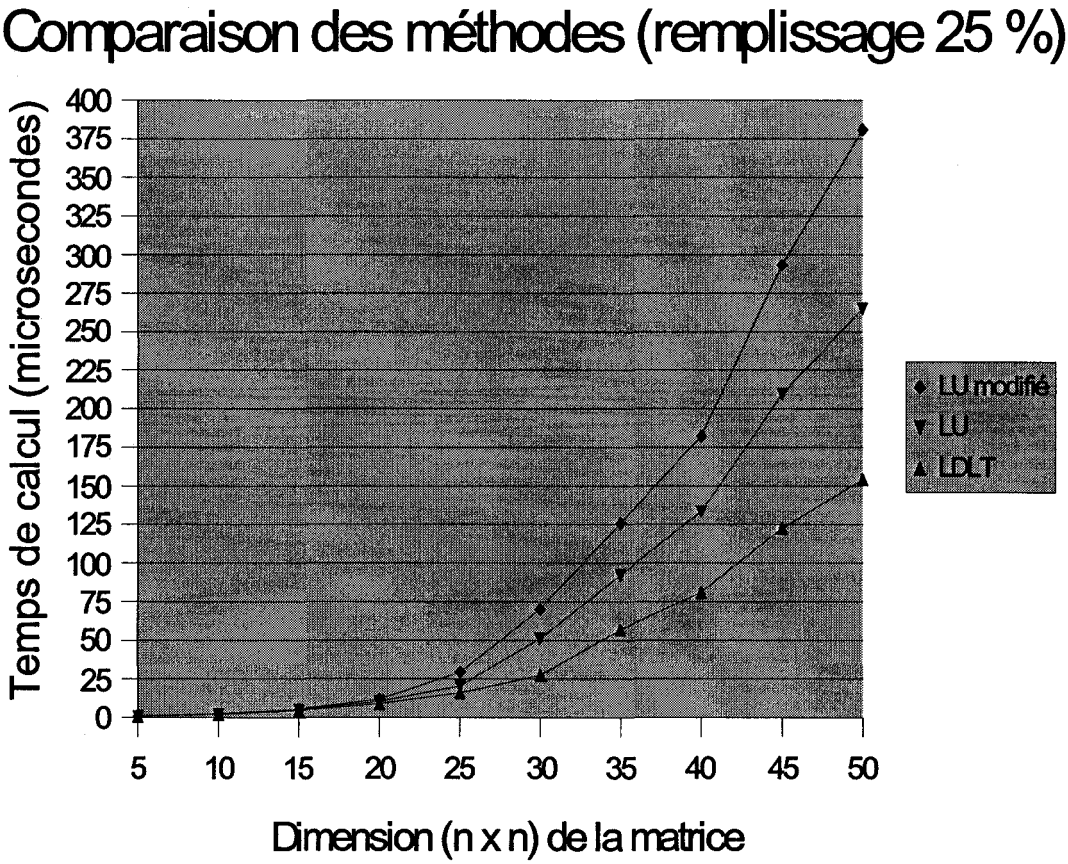

Figure 33 Comparaison des méthodes (25%)

On remarque que peu importe la dimension ou le taux de remplissage de la matrice, l'utilisation de la méthode LDL^T diminue considérablement le temps de calcul.

7.2.3 La méthode de Gauss-Siegel

La méthode de Gauss-Siegel est très similaire aux précédentes, mais la variable calculée est réinjectée après chaque équation plutôt qu'à chaque itération.

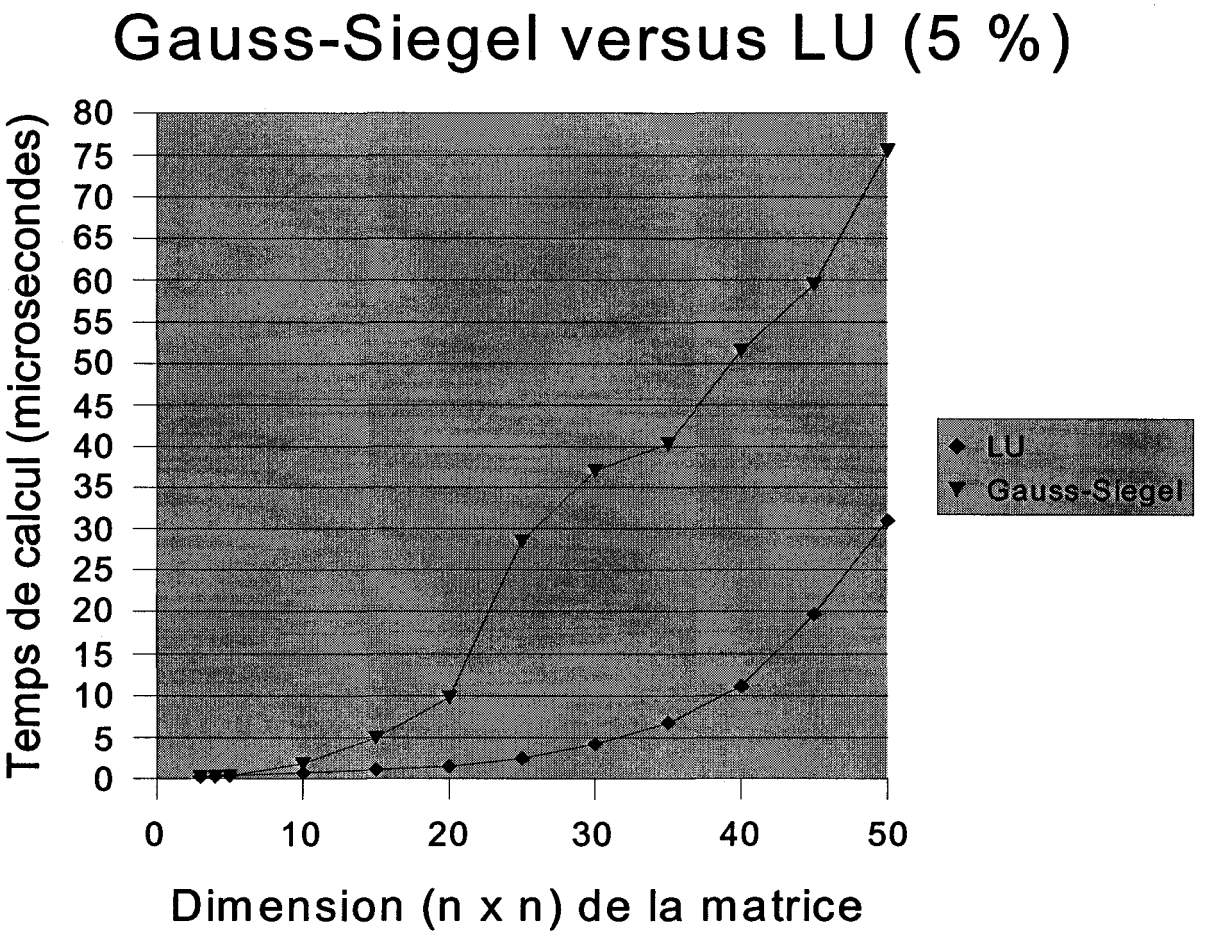

Figure 34 Gauss-Siegel versus LU (5 %)

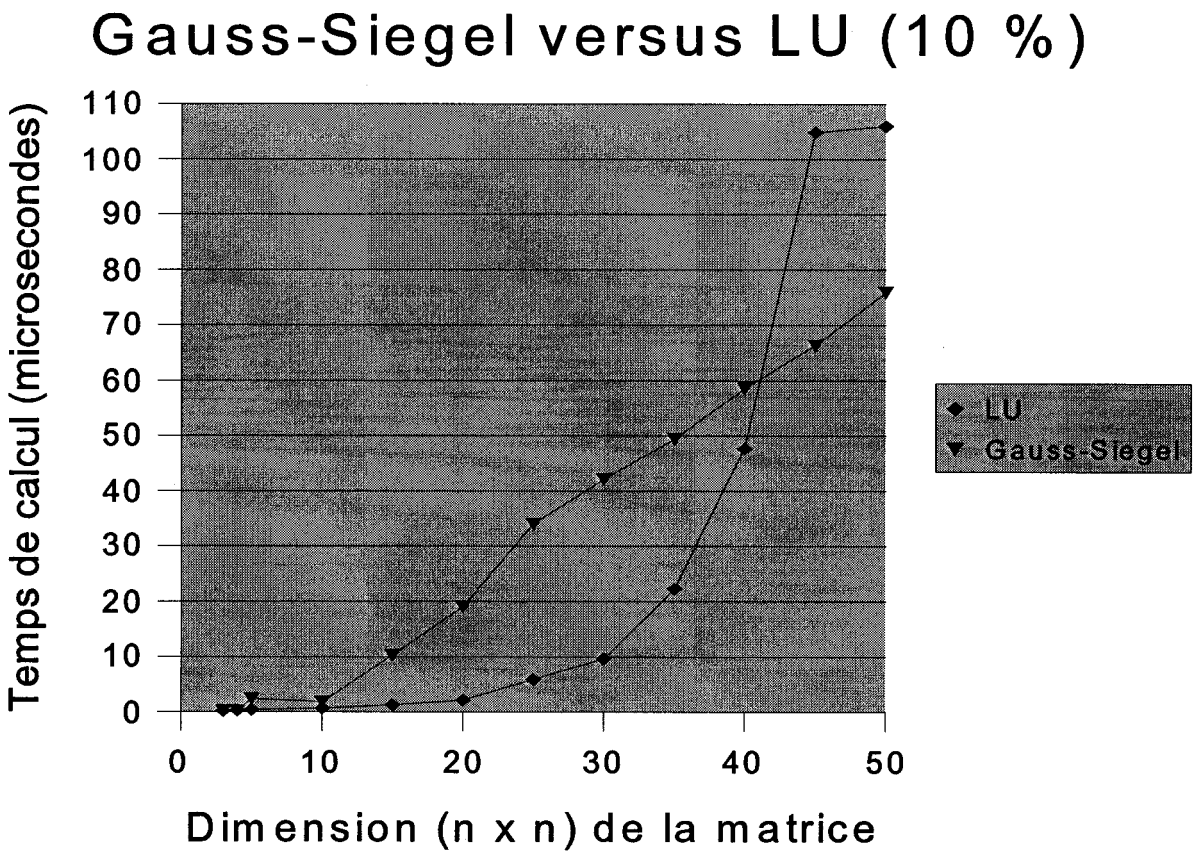

Figure 35 Gauss-Siegel versus LU (10 %)

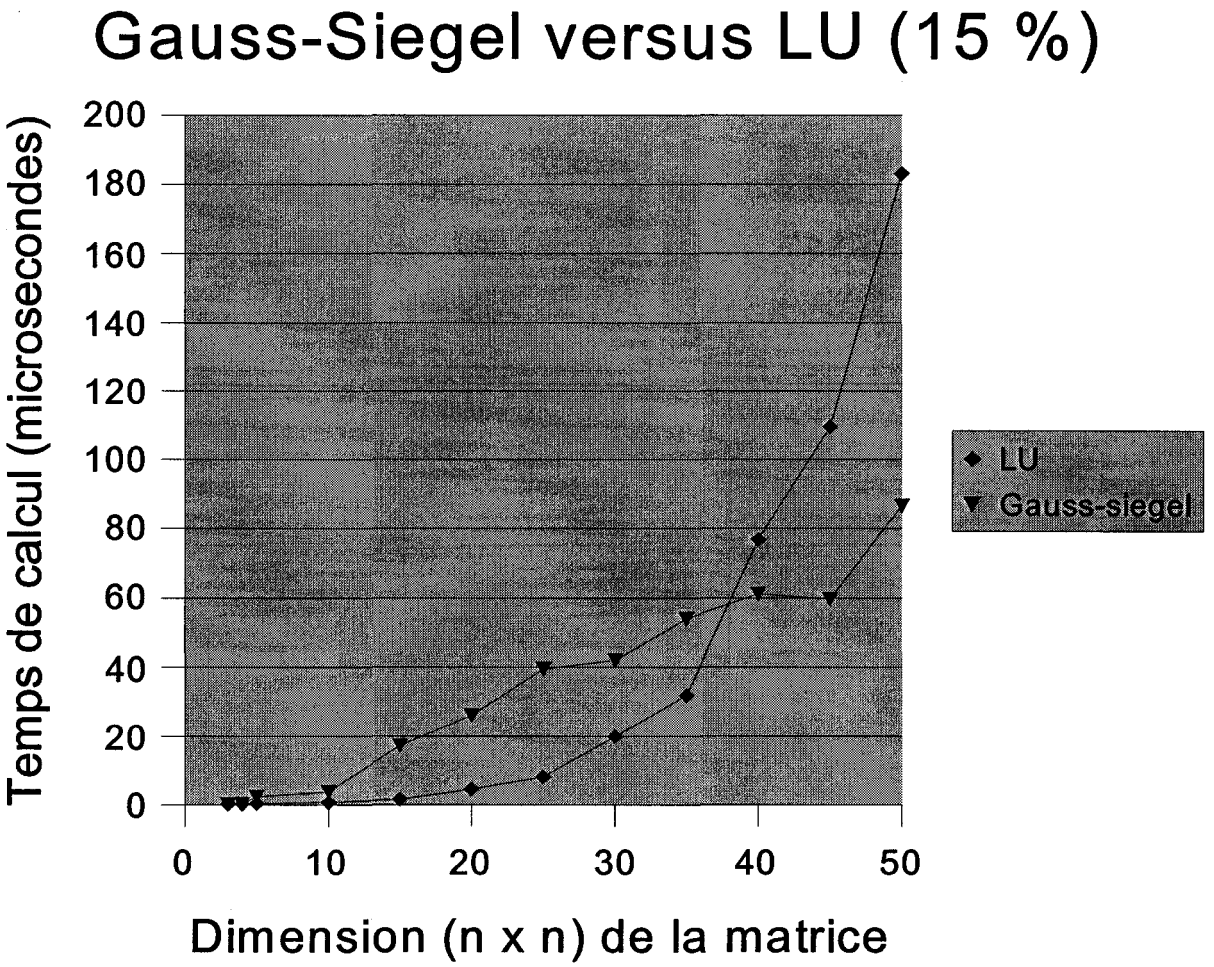

Figure 36 Gauss-Siegel versus LU (15 %)

Les trois graphiques précédents montrent un essai de la méthode de Gauss-Siegel avec une précision de $1 / 10^6$. On remarque que si le taux de remplissage est de 10 % ou plus, le temps de calcul est plus faible que la méthode LU conventionnelle.

CHAPITRE 8

IMPLANTATION DE LA MÉTHODE LDL^T DANS LE SIMULATEUR

La méthode LDL^T a été retenue et implantée dans le simulateur Hypersim. Les mesures de performance montrent bien les gains attendus, soit une diminution de plus de la moitié du temps de calcul. Dans un cas, en prenant six transformateurs triphasés en saturation et aucune communication avec des équipements externes, le simulateur n'a pris que 45 % de son temps de calcul original.

La méthode LDL^T n'est utilisable que si la matrice est strictement symétrique. Le code inclut dorénavant une vérification de la matrice originale (avant décomposition) afin de s'assurer qu'elle est belle et bien symétrique. De plus, comme la matrice change aussi durant la simulation, une inspection du code modifiant cette matrice a été faite et n'indique pas que la matrice puisse devenir asymétrique.

Il est possible que la méthode LDL^T ne soit pas aussi stable numériquement que la méthode actuelle. Plusieurs tests réalisés montrent la même stabilité. Cependant, on ne peut exclure que, dans certains cas, la stabilité du calcul ne soit pas la même qu'avant.

La nouvelle méthode respecte la même technique de recalcul partiel des lignes affectées par le changement d'un nœud. Cependant, lors de la phase de préparation de la matrice, une méthode récursive est appelée pour déterminer les lignes à recalculer. La méthode récursive n'est pas utilisée durant la simulation.

Finalement, seuls deux fichiers sources sont affectés, soit les fichiers :

- hyVNode.cpp
- hyVNode_GenC.cpp

CONCLUSION

À partir de l'analyse des résultats précédents, on peut tirer les conclusions suivantes :

1. Il est préférable de dérouler, par programmation, les boucles des calculs, à moins d'être sur un ordinateur de type SGI et d'avoir de grandes matrices (dans notre cas, plus grand que 60 x 60).

2. Les options de compilation plus grandes que O3 n'ont pas d'effet sur un code dont les boucles de calcul sont déroulées.

3. Le classement dans un vecteur des éléments provenant d'une matrice fait selon l'ordre du calcul ne permet pas de diminuer le temps de calcul, mais est une police d'assurance qui prévient les effets pernicieux de chevauchement de caches.

4. Pour des matrices assez grosses, sur un ordinateur de type SGI, il est préférable d'utiliser une matrice plutôt qu'un vecteur. Cela ne s'applique pas aux ordinateurs de type PC/Pentium.

5. La dispersion des données dans un vecteur n'a que peu d'effet sur le temps de calcul. Bien qu'il soit préférable de grouper les données d'un même calcul, il est inutile de déployer des efforts considérables pour le faire.

Par rapport au simulateur actuel Hypersim, à la lumière des résultats obtenus, il n'y a pas de moyen "informatique" (options de compilation, ordonnancement des données, etc.) qui permet de diminuer davantage le temps de calcul.

La parallélisation de la décomposition LU donne de pires résultats que la version non parallélisée, mais l'utilisation d'une décomposition LDL^T au lieu de LU, lorsque c'est possible, diminue de plus de la moitié le temps de calcul d'une station.

Dans le cas de matrices très creuses, comme c'est le cas dans le simulateur, les méthodes itératives peuvent offrir une très bonne performance, mais elles sont plus sensibles au contexte de la simulation (conditionnement de la matrice, bonne estimation des valeurs de départ, etc). Il serait judicieux de consacrer du temps à l'analyse des conditions de convergence, car les méthodes itératives peuvent représenter des gains importants.

BIBLIOGRAPHIE

Abdou-Rahmani, Sana (1993). *Étude de la représentation des circuits d'électronique de puissance dans EMTP*. Université de Montréal, École Polytechnique de Montréal.

Aho, Alfred, Sethi, Ravi, Ullman, Jeffrey (1986). *Compilers, Principles, Techniques and Tools*. Addison-Wesley.

Gerald, Wheatley (1984). *Applied Numerical Analysis* (3^{e} éd.).

Golub, Gene H., Van Loan, Charles F. (1996). *Matrix Computations* (3^{e} éd.). Johns Hopkins University Press.

Kumar, Grama, Gupta, Karypis (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin Cummings.

MIPS64 Architecture for Programmers, Volume I: Introduction to the MIPS Architecture. Document number : MD00083.

Muchnick, Steven S. (1997). *Advance Compiler Design and Implementation*. Morgan Kaufmann.

Origin 2000 and Onyx2 Performance Tuning and Optimization Guide. Document number : 007-3430-002.

Origin and Onyx2 Programmer's Reference Manual. Document number : 007-3410-001.

Origin and Onyx2 Theory of Operation Manual. Document number : 007-3439-002.

Ortega, J.M. (1988). *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press.

Patterson, D., Sequin, C. (septembre 1982). *A VLSI RISC*. Computer.

Press, William H., Teukolsky, Saul A., Vetterling, Wiliam T., Flannery, Brian P. (1992). *Numerical Recipes in C, The Art of Scientific Computing* (2^{e} éd.). Cambrige University Press.

Stalling, William (2000). *Computer Organization and Architecture* (5^{e} éd.). Prentice Hall.

Stallmann, Richard (1999). *Using and Porting the GNU Compiler Collection*. Free Software Foundation.

Wong, Tony (1999). *Répartition automatique des tâches parallèles : Application dans la simulation de réseaux électriques en temps réel.* Université de Montréal, École Polytechnique de Montréal.