Étude de la Distribution, sur Système à Grande Échelle, de Calcul Numérique Traitant des Matrices Creuses Compressées

Olfa Hamdi-Larbi

▶ **To cite this version:**

Olfa Hamdi-Larbi. Étude de la Distribution, sur Système à Grande Échelle, de Calcul Numérique Traitant des Matrices Creuses Compressées. Calcul parallèle, distribué et partagé [cs.DC]. Université de Versailles-Saint Quentin en Yvelines, 2010. Français. NNT : . tel-00693322

HAL Id: tel-00693322

https://theses.hal.science/tel-00693322

Submitted on 2 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE TUNIS EL MANAR
FACULTÉ DES SCIENCES DE TUNIS

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

THÈSE

présentée en vue de l'obtention du

Diplôme de Docteur en Informatique

par

Olfa Hamdi-Larbi

Ingénieur en Informatique, FST – Tunis

Étude de la Distribution, sur Système à Grande Échelle, de Calcul Numérique Traitant des Matrices Creuses Compressées

soutenue le 27 Mars 2010, devant le jury d'examen

MM.	**Rafik BRAHAM**	***Président***
	Mounir MARRAKCHI	***Rapporteur***
	Frédéric MAGOULÈS	***Rapporteur***
	William JALBY	***Examinateur***
Mme	**Nahid EMAD**	***Directrice de thèse***
MM.	**Zaher MAHJOUB**	***Directeur de thèse***
	Yousef SAAD	***Invité***

Préparée sous convention de cotutelle UTM-FST (Tunisie) – UVSQ (France)

DÉDICACES

J'écris cette dédicace tout en ayant aux yeux les mêmes larmes d'émotion que j'ai eue les jours où j'ai eu mes deux chers poupons. Cette thèse m'est aussi chère que mes enfants sauf que la période de gestation pour cet enfant a été assez longue. Cet enfant qu'on porte avec beaucoup de souffrance, d'effort … mais aussi avec beaucoup d'amour et de plaisir, on attend sa naissance avec impatience, on le protège, on est inquiet si on ressent un jour qu'il peut être en danger et qu'on ne finira pas par l'avoir.
Enfin, il est là, pour illuminer notre vie et nous donner encore une poussée pour continuer …

La valeur de cette thèse n'est pas uniquement dans les résultats auxquels elle a abouti, mais aussi dans le défi, le symbole de patience et de sacrifice qu'elle représente. Cette thèse m'est chère parce qu'elle était le témoin de beaucoup de joie mais aussi de moments difficiles dans ma vie.

A cause de tout cela et parce que tu m'as toujours suivie de près ou de loin et parce que depuis le début tu étais à l'origine de mon choix pour le domaine de l'Informatique,
mon cher père,
je te dédie ce travail. Bien qu'il me soit très cher, il ne suffira pas pour te récompenser pour tout le sacrifice que tu as fait pour moi.

Ma chère mère,

mon exemple pour la patience, l'ambition, la force, l'insistance et la résistance. Mon refuge dans les moments les plus durs, mon cher soutien. Tu as essayé par tous les moyens d'être là pour m'aider, puisse-tu trouver dans ce travail l'expression de toute ma reconnaissance.

Mon cher mari,

depuis notre première rencontre tu étais là pour me pousser vers l'avant. Avec toi j'ai appris à lever les défis et à être plus ambitieuse. Je ne peux qu'être reconnaissante pour tous tes sacrifices afin que je puisse arriver là.

Mes chers petits poussins,

vous étiez les premiers à avoir sacrifié pour que ce travail voit le jour. J'espère que vous me pardonnerez et que vous trouverez dans ce mémoire une récompense pour les longs mois de séparation lors de mes voyages. J'espère aussi que cette thèse vous rendra plus fiers de votre mère et qu'elle vous servira comme symbole de défi et de constance pour atteindre vos objectifs et réaliser vos ambitions dans la vie.

Je dédie également ce travail à mon cher frère Anis.

Mon cher frère Nizar, ma chère grand-mère,

j'ai souhaité que vous soyez là à mes côtés et que vous partagiez avec moi la joie du succès. J'espère que Dieu vous accueille dans son éternel paradis avec son infinie miséricorde.

Je dédie ce travail à mes beaux-parents, à mes beaux-frères et à ma belle-soeur, leurs conjoints et leurs enfants.

Enfin je n'oublie pas de dédier ce travail à mes amies. Je cite en particulier mon amie d'enfance Chiraz Khliaa, et toutes les personnes qui m'ont soutenue.

Remerciements

" من لم يشكر الناس لم يشكر الله عز وجل"

حديث شريف

" Celui qui ne remercie pas les gens ne remercie pas Dieu Tout Puissant "

'Hadith' du Prophète

C'est une tradition louable que de remercier au terme d'un tel travail tous ceux qui, plus ou moins directement, ont contribué à son parachèvement. Même si dans mon cas, cette liste peut sembler plus longue que de coutume, c'est avec mon enthousiasme le plus vif et le plus sincère que je voudrais rendre mérite à tous ceux qui, à leur manière, m'ont aidée à mener à bien cette thèse. Je désire ainsi exprimer mon profond remerciement à :

Mr Rafik Braham, Professeur à l'Institut Supérieur d'Informatique et des Techniques de Communication (ISITCOM) à Hammam-Sousse, qui m'a honorée en acceptant de présider ce jury et dont j'étais l'étudiante en DEA,

Mr Frédéric Magoulès, Professeur à l'Ecole Centrale de Paris, pour avoir bien voulu accepter la charge de rapporter ma thèse,

Mr Mounir Marrakchi, Maître de Conférences à la Faculté des Sciences de Tunis, pour avoir accepté de rapporter mon travail de thèse ainsi que pour ses remarques qui m'ont permis d'améliorer ce manuscrit,

Mr William Jalby, Professeur de l'Université de Versailles Saint-Quentin-en-Yvelines, qui m'honore en acceptant d'être membre examinateur dans ce jury.

Je suis très sensible à la participation dans ce jury de Mr Yousef Saad, 'Distinguished Professor' à l'Université du Minnesota, dont les travaux m'ont permis de mieux découvrir le monde du calcul creux.

J'exprime mes profonds remerciements à ma directrice de thèse, Mme Nahid Emad, Maître de Conférences HDR à l'Université de Versailles Saint-Quentin-en-Yvelines, pour l'aide précieuse qu'elle m'a apportée, pour sa patience et ses encouragements à finir ce travail. Je la remercie profondément pour ses efforts et sa patience. Tout au long de ces années, elle a su orienter mes recherches aux bons moments. Son œil critique et avisé

m'a été très précieux pour structurer le travail et pour améliorer sa qualité. Ses nombreuses remarques et suggestions n'ont pu que valoriser ce travail.
Plus qu'une encadrante, j'ai trouvé en elle l'amie et la grande sœur qui m'a aidée aussi bien dans le travail que dans la vie lorsque j'en avais besoin.

Je tiens à remercier exceptionnellement mon directeur de thèse, Mr Zaher Mahjoub, Professeur à la Faculté des Sciences de Tunis. Il m'a beaucoup appris depuis que j'étais son étudiante durant ma formation d'Ingénieur en Informatique. Il m'a fait découvrir le domaine du parallélisme et m'en a transmis la passion grâce à ses qualités pédagogiques particulières. Il m'a par la suite encadrée lors de la préparation de mon projet de fin d'études d'ingénieur, puis de mon stage de DEA. Il m'a enfin honorée en m'encadrant en thèse.
Après toutes ces années, j'ai appris une chose importante, c'est qu'on ne peut cesser d'apprendre de lui.
Cet espace assez limité ne suffira certainement pas pour le remercier pour ses compétences scientifiques qui m'ont permis de mener à bien ce travail, pour sa disponibilité, pour sa rigueur, son engagement et son implication et parce qu'il met toujours en avant l'intérêt de ses étudiants.
Cet espace ne suffira pas aussi pour parler de son soutien particulier un jour quand j'ai du subir la rigueur du destin.
Puisse-t-il trouver dans ce fruit de plusieurs années de travail, toute ma gratitude et ma profonde reconnaissance.

Je ne saurais enfin oublier d'exprimer ma reconnaissance à Monsieur le Professeur Paul Feautrier (UVSQ puis ENS Lyon) dont l'apport a été très important pour le choix et la finalisation de mon thème de recherche.

Je souhaite aussi exprimer toute ma gratitude à tous mes enseignants depuis l'école primaire. J'ai tant attendu ce moment pour citer en particulier mon Maître Mr Ahmed Béjaoui qui était le premier à me faire découvrir la langue de Molière et qui a continué à le faire durant trois années. Il m'a toujours impressionnée par son sérieux, son engagement et son dévouement en exerçant le noble métier d'enseignant. Il n'a cessé d'exprimer sa fierté à mon égard et de suivre les résultats de mes études jusqu'à l'année 1994 quand Dieu l'a choisi, assez jeune, pour être dans un monde meilleur. Je prie Dieu qu'Il le récompense et l'accueille avec son infinie miséricorde.

J'exprime également ma vive reconnaissance à tous mes enseignants au Lycée secondaire du Bardo ainsi qu'à tous mes enseignants du département des Sciences de l'Informatique à la Faculté des Sciences de Tunis.

J'adresse un grand merci aux membres anciens et actuels du laboratoire PRISM de Versailles au sein duquel j'ai passé divers séjours. Je cite en particulier Fazeli Abou-AlFazl, Ani Sedrakian et Marta Gonzalès pour leur gentillesse et leur aide.

Je remercie vivement Mr Jack Dongarra dont les conseils et l'aide durant un séjour au PRISM ont été à l'origine de plusieurs idées développées dans cette thèse.

Je tiens aussi à remercier Mr Nabil Abdennadher pour sa précieuse aide à comprendre et à utiliser l'outil XtremWeb-CH. Je remercie également son étudiant Mohamed Ben Belgacem pour son aide.
Je voudrais aussi vivement remercier Mr Nejmeddine Jaidene, Professeur au département de Physique à la Faculté des Sciences de Tunis, pour m'avoir accueillie dans son laboratoire et pour m'avoir permis d'utiliser sa machine SUN lors de mes expérimentations.

Le personnel administratif du Département des Sciences de l'Informatique de la FST a aussi droit à mes remerciements pour son aide et sa collaboration. Je remercie en particulier Mr Zarrouk pour l'agréable ambiance qu'il maintient au sein du département ainsi que pour sa serviabilité. Je remercie également Mme Rafika, secrétaire du département, pour son accueil toujours souriant.

Mes collègues de l'Institut Supérieur d'Informatique (ISI) de l'Ariana méritent un grand merci de ma part pour leur aide et leur encouragement. Je cite Mr Moncef Temani Directeur de l'ISI, Mr Ezzedine Zagrouba, Mme Olfa Mourali, mes anciens enseignants Mr Adel Khalfallah et Mme Fatma Amira, ainsi que Hend Koubaa, Azza Ouled Zeid, Mme Nejiba Bellaej et tout le personnel administratif.

Je réserve un remerciement chaleureux à mes camarades, membres de l'unité de recherche URAPAD, pour l'affectueuse amitié dont ils ont toujours fait preuve. Chacun d'eux mérite un remerciement particulier :

Mr Hamadi Hasni, notre grand frère qui nous fait toujours profiter de ses conseils et de sa grande expérience,

Wahid Nasri, pour son aide et ses conseils,

Yosr Slama, pour son sens de l'humour et parce qu'elle m'a appris, sans peut être le savoir, à accepter la vie telle qu'elle est, même si on est dans la pire situation. Avec elle je retrouve ma bonne humeur et je ris du fond du cœur. Je n'oublie pas aussi son soutien et ses encouragements ainsi que sa serviabilité.

Afef Fkiri, pour qui j'ai une pensée affectueuse et à qui je dis « tiens bon ! », le bon goût du succès mérite les sacrifices,

Bchira Ben Mabrouk, merci parce que tu participes bien dans la bonne et très conviviale ambiance de notre unité de recherche,

Héla Guesmi, pour ses qualités humaines, bon courage pour la thèse,

Sami Achour, pour sa gentillesse et sa serviabilité,

Sana Zouaoui, la petite qui grandit, mes félicitations pour la soutenance de Mastère,

Aymen Schehaider, le frère cadet dans l'unité, mes félicitations également pour la soutenance de Mastère,

Asma Ben Salem, Ryma Mahfoudhi et Asma Dab, je vous souhaite à toutes une bonne continuation.

Je ne pourrais jamais oublier l'aide et le soutien de mes amis en France. Je réserve une reconnaissance particulière à :

Sana Bachali et son mari pour avoir été les premiers à m'accueillir chez eux lors de mes premiers séjours à Versailles et qui ont bien pris soin de moi alors que j'étais enceinte et souffrante.

Besma Fliss et son mari, pour leur générosité et parce qu'ils ont laissé leur appartement à ma disposition lors de l'un de mes séjours en France.

Mon angélique amie Sana Younes, pour sa générosité et l'immense hospitalité avec laquelle elle m'a accueillie durant plusieurs séjours que j'ai passés au PRISM.

Enfin, un grand merci à toute ma grande famille, mes oncles et mes tantes pour leurs encouragements.

Exprimer ma gratitude envers les personnes en qui j'ai trouvé un soutien est une tâche ardue. La difficulté tient dans la contrainte de n'oublier personne. C'est pourquoi, je remercie ceux dont le nom n'apparaît pas ici et qui m'ont aidée d'une manière ou d'une autre.

Table des Matières

Liste des figures *v*

Liste des tables *ix*

Introduction générale *1*

Partie I **9**

Chapitre 1. Etat de l'art sur les méthodes itératives *11*

1. Introduction 13
2. Principes de base des méthodes de RSL et de CEP 13
   2.1 Résolution de systèmes d'équations linéaires 13
      2.1.1 Méthodes directes 14
      2.1.2 Méthodes itératives 15
      2.1.3 Méthodes de RSL appropriées au calcul creux? 16
   2.2 Calcul d'éléments propres (CEP) 17
3. Méthodes itératives 17
   3.1 Résolution de systèmes linéaires 17
      3.1.1 Méthodes itératives stationnaires $x^{(k+1)}=Tx^{(k)} + c$ 17
      3.1.2 Méthodes itératives non stationnaires 21
      3.1.3 Méthodes itératives hybrides 23
      3.1.4 Méthodes par bloc 23
      3.1.5 Pré-conditionnement d'une matrice 24
   3.2 Calcul de valeurs propres 25
      3.2.1 Méthode de la puissance itérée 25
      3.2.2 Méthodes de projection 26
      3.2.3 Méthodes itératives hybrides de calcul d'éléments propres 29
4. Conclusion 30

Chapitre 2. Le calcul creux : Présentation générale *31*

1. Introduction 33
2. Structures de matrices creuses 33
   2.1 Définitions générales 33
   2.2 Structures régulières 34
      2.2.1 Matrice triangulaire 34
      2.2.2 Matrice bande constante 34
      2.2.3 Matrice de Hessenberg 35

2.2.4 Matrice bloc 35
2.3 Structures irrégulières 36
2.3.1 Matrice bande variable 36
2.3.2 Matrice quelconque 36
2.4 Matrices creuses et applications scientifiques - Etude statistique 37
3. Formats de compression 43
3.1 Formats pour structures régulières 43
3.1.1 Matrice triangulaire 43
3.1.2 Matrice bande constante 44
3.1.3 Matrice Hessenberg 45
3.1.4 Matrice bloc 45
3.2 Formats pour structures irrégulières 46
3.2.1 Matrice bande variable 46
3.2.2 Matrice quelconque 47
3.3 Etude comparative 49
4. Applications creuses de grande taille 53
4.1 Matrice de Google 53
4.2 Simulation électromagnétique tridimensionnelle 55
5. Conclusion 57

Chapitre 3. Systèmes parallèles et systèmes distribués *59*

1. Introduction 61
2. Taxonomie générale 61
2.1 Machines SIMD 62
2.2 Machines MIMD à mémoire distribuée 63
2.3 Machines MIMD à mémoire partagée 64
2.4 Machines MIMD à mémoire hybride 64
3. Les clusters (grappes) 65
3.1 Architecture générale 65
3.2 Les composants d'un cluster 66
3.3 Classification des clusters 67
4. Les grilles 67
4.1 Définition et Architecture générale d'une grille de calcul (GRID) 67
4.2 Classification des systèmes de grilles de calcul 68
4.3 Classification de l'usage des grilles de calcul 69
5. Systèmes distribués à grande échelle et systèmes pair-à-pair 69
5.1 Définition d'un système distribué à grande échelle 69
5.2 Différences entre SDGE, système P2P et grille de calcul 70
6. Le Cloud computing ou 'Informatique dans les nuages' 71
6.1 Définition 71
6.2 Location de serveurs virtuels nus adaptés à la reprise de l'existant 72
7. Un SDGE type : XtremWeb 72
7.1 Architecture d'XtremWeb 73
7.2 Organisation générale 73
7.3 Architecture du Worker 74
7.4 Architecture des serveurs 74
8. Conclusion 75

Partie II **77**

Chapitre 4. Etude de l'optimisation du produit matrice-vecteur creux ***79***

1. Introduction 81
2. Algorithmes du PMVC pour les différents formats de compression 81
   2.1. PMVC-DNS 82
   2.2. PMVC-CSR 82
   2.3 PMVC-COO 83
   2.4 PMVC-BND 83
   2.5 Tableau récapitulatif 84
3. Etat de l'art sur l'optimisation du PMVC 84
   3.1 Opt1. Proposition d'un nouveau format 84
   3.2 Opt2. Optimisations spécifiques à des matrices creuses particulières 86
   3.3 Opt3. Optimisations spécifiques à des formats particuliers 86
   3.4 Opt4. Optimisations par restructuration des données 87
   3.5 Opt5. Optimisations spécifiques à des architectures particulières 87
   3.5 Opt6. Optimisations de bas niveau 87
   3.6 Récapitulation 88
4. Etude de l'optimisation du PMVC 88
   4.1. Techniques d'optimisation 88
      4.1.1 Approche manuelle 89
      4.1.2 Options du compilateur C sous UNIX 90
   4.2 Optimisation du PMVC 92
      4.2.1 PMVC-DNS 92
      4.2.2. PMVC-CSR 92
      4.2.3 PMVC-COO 93
      4.2.4 PMVC-BND 93
5. Etude expérimentale pour les quatre formats 94
   5.1 Environnements de travail 94
   5.2 Résultats Expérimentaux 95
      5.2.1 Résultats pour le PMVC-DNS 97
      5.2.2 Résultats pour le PMVC-CSR 101
      5.2.3 Résultats pour le PMVC-COO 105
      5.2.4 Résultats pour le PMVC-BND 109
6. Récapitulation et conclusion 113

Chapitre 5 : Fragmentation de matrice creuse pour la distribution du PMVC sur SDGE ***115***

1. Introduction 117
2. Présentation de la problématique 118
3. Etat de l'art sur l'équilibrage des charges pour le PMVC 119
4. Fragmentation des données 121
   4.1 Approche 1 : fragmentations NLE et NLEG 121
      4.1.1 Fragmentation NLE 121
      4.1.2 Fragmentation NLEG 122
   4.2 Approche 2: décomposition NEZ 125
   4.3 Approche 3: fragmentations NEZG, NEZGT et NEZGTC 129

4.3.1 Fragmentation NEZG 129
4.3.2 Fragmentations NEZGTC et NEZGT 131
5. Analyse des performances théoriques 132
5.1 Approche 1 : NLE et NLEG 132
5.2 Approche 2: NEZ 133
5.3 Approche 3: NEZG, NEZGT et NEZGTC 134
6. Etude expérimentale 135
6.1 Résultats numériques 135
6.2 Analyse des résultats 140
7. Conclusion 143

Chapitre 6 : Distribution du produit matrice vecteur creux sur un système de calcul à grande échelle 145

1. Introduction 147
2. Définition d'une géométrie virtuelle d'un SDGE 147
3. Distribution du PMVC sur un SDGE 148
3.1 Présentation du problème 148
3.2 Fragmentation des données 150
3.3 Construction du vecteur résultat *y* 150
4. Evaluation théorique des performances 151
4.1 Complexité de calcul 151
4.2 Complexité de communication 151
5. Expérimentations 152
5.1 Plate-forme XtremWeb-CH 153
5.2 Plate-forme matérielle 154
5.3 Paramètres testés 155
5.4 Résultats et interprétations 156
5.4.1 Evaluation des performances en fonction de la taille de la matrice 156
5.4.2 Evaluation des performances en fonction de la densité 157
5.4.3 Evaluation des performances en fonction du facteur de réduction 159
5.4.4 Evaluation des performances en fonction du nombre de fragments 161
6. Conclusion 164

Conclusion générale et perspectives 165

Bibliographie 171

Netographie 179

Annexes 183

Annexe A : Résultats pour le PMVC-DNS 185
Annexe B : Résultats pour le PMVC-CSR 195
Annexe C : Résultats pour le PMVC-COO 205
Annexe D : Résultats pour le PMVC-BND 213

Liste des figures

Figure 0.1 Problématique et contribution pour sa résolution 4

Figure 1.1 Méthode d'élimination de Gauss 14
Figure 1.2 Méthodes de factorisation 14
Figure 1.3 Décomposition de A=D-M-N 18
Figure 1.4 Algorithme d'Arnoldi 27
Figure 1.5 Algorithme Basique d'Arnoldi (BAA) 28
Figure 1.6 Algorithme ERAM 28
Figure 1.7 Algorithme MERAM 30

Figure 2.1 Structures régulières de matrices creuses 35
Figure 2.2 Structures irrégulières de matrices creuses 36
Figure 2.3 Statistiques sur les tailles des matrices de Matrix-Market 41
Figure 2.4 Statistiques sur les densités des matrices de Matrix-Market 41
Figure 2.5 Classement des matrices selon leurs sructures 41
Figure 2.6 Représentation du nombre de matrices et du nombre d'applications pour chaque type de structure 41
Figure 2.7 Matrice Triangulaire stockée dans un tableau 2D 43
Figure 2.8 Matrice A stockée selon le format MSR 43
Figure 2.9 Matrice bande dans un tableau 2D 44
Figure 2.10 Tableau ABD - BND par colonne 44
Figure 2.11 Tableau ABD-BND par ligne 44
Figure 2.12 Matrice bloc représentée par un tableau 2D 45
Figure 2.13 Matrice bloc stockée selon le format BSR 45
Figure 2.14 Matrice bande variable symétrique stockée dans un tableau 2D 46
Figure 2.15 Matrice bande variable symétrique stockée selon le format SSK 46
Figure 2.16 Matrice quelconque A représentée par un tableau 2D 47
Figure 2.17 Matrice A stockée selon le format CSR 47
Figure 2.18 Matrice quelconque représentée par un tableau 2D 48
Figure 2.19 Matrice stockée selon le format COO 48
Figure 2.20 Un exemple de Web avec quatre pages 54

Figure 3.1 Taxonomie de haut niveau des architectures parallèles 62
Figure 3.2 Une configuration SIMD 63
Figure 3.3 Une configuration MIMD à mémoire distribuée 63
Figure 3.4 Une configuration MIMD à mémoire partagée 64
Figure 3.5 Une configuration MIMD hybride 65
Figure 3.6 Architecture d'un cluster 65
Figure 3.7 Classification des systèmes P2P 70
Figure 3.8 Architecture générale de XtremWeb 73

Figure 4.1 Produit matrice-vecteur creux pour le format DNS 82
Figure 4.2 Produit matrice-vecteur creux pour le format CSR 83
Figure 4.3 Produit matrice-vecteur creux pour le format COO 83
Figure 4.4 Produit matrice-vecteur creux pour le format BND 84
Figure 4.5 Loop Unrolling (facteur *u*) 90
Figure 4.6 PMVC-DNS avec unrolling de facteur 4 92
Figure 4.7 PMVC optimisé pour le format CSR 93
Figure 4.8 PMVC optimisé pour le format COO 93
Figure 4.9 PMVC optimisé pour le format BND 94
Figure 4.10 Matrice téléchargée de Matrix Market 96
Figure 4.11 Performances du PMVC-DNS sur la machine 1 (PC Intel) 97
Figure 4.12 Performances du PMVC-DNS sur la machine 2 (processeur AMD) 98
Figure 4.13 Performances du PMVC-DNS sur la machine 3 (SUN) 99
Figure 4.14 Accélérations obtenues par l'unrolling manuel sur la machine 1 100
Figure 4.15 Performances du PMVC-CSR sur la machine 1 (PC Intel) 102
Figure 4.16 Performances du PMVC-CSR sur la machine 2 (processeur AMD) 103
Figure 4.17 Performances du PMVC-CSR sur la machine 3 (SUN) 104
Figure 4.18 Accélérations obtenues par le remplacement scalaire sur la machine 1 105
Figure 4.19 Performances du PMVC-COO sur la machine 1 (PC Intel) 106
Figure 4.20 Performances du PMVC-COO sur la machine 2 (processeur AMD) 107
Figure 4.21 Performances du PMVC-COO sur la machine 3 (SUN) 108
Figure 4.22 Accélérations obtenues par les options –On sur la machine 2 109
Figure 4.23 Performances du PMVC-BND sur la machine 1 (PC Intel) 110
Figure 4.24 Performances du PMVC-BND sur la machine 2 (processeur AMD) 111
Figure 4.25 Performances du PMVC-BND sur la machine 3 (SUN) 112
Figure 4.26 Accélérations obtenues par les options –On sur la machine 2 112

Figure 5.1 Fragmentation de la matrice 118
Figure 5.2 Pseudo-code de la phase 1 de NEZ 127
Figure 5.3 FDR pour matrices quelconques générées aléatoirement, d=5, *f*=512 136
Figure 5.4 FDR pour matrices bande, dlb=100 138
Figure 5.5 FDR pour les matrices téléchargées de Matrix Market 138
Figure 5.6 Temps CPU pour matrices quelconques générées aléatoirement, d=5%, *f*=512 .. 139
Figure 5.7 Comparaison de FD pour les matrices de Matrix Market 139
Figure 5.8 FDR pour l'approche NLE en fonction de *f* et d, N=10000 141
Figure 5.9 FDR pour l'approche NLE en fonction de *f* et dlb, N=10000 141
Figure 5.10 FDR de NLEG - Matrices quelconques aléatoires, *f*=9 142
Figure 5.11 FDR de NLEG - Matrices bandes aléatoires, *f*=9 143

Figure 6.1 Calcul du PMVC sur un système distribué à grande échelle 149
Figure 6.2 Produit fragment-vecteur creux 149
Figure 6.3 Décomposition NLE 150
Figure 6.4 Construction du vecteur résultat 151
Figure 6.5 Architecture de XtremWeb-CH 154
Figure 6.6 Performances du PMVC en fonction de la taille de la matrice, d=10, FR=2 et *f*=64 156
Figure 6.7 Ratios en fonction de la taille de la matrice, d=10, FR=2 et *f*=64 157
Figure 6.8 Performances du PMVC en fonction de la densité de la matrice, N=5000 et *f*=64 158

Figure 6.9 Ratios en fonction de la densité de la matrice, N=5000 et *f*=64 158
Figure 6.10 Performances du PMVC pour la matrice FIDAPM37, *f*=64 et FR variable..... 159
Figure 6.11 Ratios pour la matrice FIDAPM37, *f*=64 et FR variable..... 160
Figure 6.12 Performances du PMVC : N=9000, d=10 (matrice aléatoire), *f*=32, RF=2 et 32160
Figure 6.13 Ratios pour N=9000, d=10 (matrice aléatoire), *f*=32, RF=2 et 32..... 161
Figure 6.14 Performances du PMVC pour N=9000 en fonction du nombre de fragments, FR=2..... 162
Figure 6.15 Ratios pour N=9000 en fonction du nombre de fragments, FR=2..... 162
Figure 6.16 Performances du PMVC pour N=9000 en fonction de *f* avec *f*=FR 163
Figure 6.17 Ratios pour N=9000 en fonction de *f* tel que *f*=FR..... 163

Liste des tables

Table 1.1 Comparaison entre méthodes directes et itératives dans le cas creux 17

Table 2.1 Répartition des structures creuses par application 40
Table 2.2 Statistiques sur les structures creuses 41
Table 2.3 Quelques statistiques sur les matrices de Matrix Market 42
Table 2.4 Tableau récapitulatif des structures creuses 52

Table 3.1 Taxonomie de Flynn pour des architectures parallèles 61

Table 4.1 Complexités des algorithmes du PMVC 84
Table 4.2 Récapitulation de l'état de l'art sur l'optimisation du PMVC 88
Table 4.3 Résultats expérimentaux selon les 4 modes d'optimisation 114

Table 5.1 Récapitulatif de l'étude théorique 135
Table 5.2 Charges extrêmes obtenues pour les matrices quelconques aléatoires 136
Table 5.3 Matrices téléchargées du site Web Matrix Market 137

Table 6.1 Caractéristiques de la matrice Fidapm37 de Matrix Market. 155

INTRODUCTION GÉNÉRALE

De nombreuses applications en calcul scientifique, telles que la simulation de la mécanique des structures, l'étude de la dynamique des fluides, le traitement d'image/signal, aboutissent à la résolution de problèmes matriciels dits *creux* nécessitant l'utilisation d'une structure particulière de stockage des données [DER92]. Les matrices creuses correspondent à des tableaux dont le nombre élevé d'éléments nuls oblige l'utilisateur, pour des raisons de taille mémoire et de performances de calcul, d'utiliser un format de stockage des éléments non nuls uniquement. Beaucoup de problèmes en calcul scientifique creux se ramènent à des problèmes d'Algèbre linéaire creux [Amo07][BrL06] [HSS03]. Il faut préciser que les deux problèmes fondamentaux en Algèbre linéaire sont (i) la résolution de systèmes linéaires [DER92][Jen80] [LaT94][OsZ83] et (ii) le calcul d'éléments propres [Cia82][LaT94].

Contrairement au second problème pour lequel on ne peut utiliser que des méthodes itératives, les méthodes de résolution de systèmes linéaires peuvent être directes ou itératives. Les méthodes directes sont robustes, conviennent plutôt à des problèmes de taille modérée, et sont généralement plus rapides que les méthodes itératives. Cependant, sur des problèmes de très grande taille, les méthodes directes peuvent devenir prohibitives car trop gourmandes en termes de mémoire (à cause du phénomène de remplissage dit '*fill-in*') et de calcul. Sur cette gamme de problèmes, les méthodes itératives constituent ainsi une alternative intéressante [DER92][Wyr01].

Des structures de données particulières sont utilisées pour le stockage des matrices creuses dans l'objectif de bien gérer l'espace mémoire. Le traitement de ces matrices de grande taille par les méthodes itératives nécessite aussi l'utilisation de superordinateurs de haute performance qui, souvent, peuvent ne pas être disponibles. Dans le cas de certaines applications scientifiques, les systèmes distribués à grande échelle (SDGE) constitués de réseaux de machines interconnectées peuvent constituer une bonne alternative. En effet, la structure de ce type de système correspond bien à la mutualisation d'un ensemble de ressources facilement accessibles constituant par synergie un superordinateur virtuel. Néanmoins, certaines propriétés de ces superordinateurs, telles que l'hétérogénéité et la volatilité de leurs ressources, rendent leur exploitation assez difficile.

La problématique à laquelle nous nous intéressons consiste, comme l'illustre la figure 0.1, à étudier la distribution, au sein d'un SDGE, des calculs dans des méthodes itératives de résolution de grands systèmes linéaires et de calcul d'éléments propres et ce, dans le cas creux.

Les méthodes itératives de résolution de systèmes linéaires et de calcul d'éléments propres reposent essentiellement sur le calcul du *produit matrice-vecteur creux* (PMVC) qui constitue le noyau de base pour la plupart d'entre elles. Ainsi, notre problématique se ramène à l'étude de la distribution du PMVC sur un système distribué à grande échelle. Malgré l'intérêt de cet algorithme, sa distribution sur un SDGE n'est pas encore bien maîtrisée.

Généralement, trois étapes sont nécessaires pour accomplir cette tâche. La première est l'étape de *prétraitement* qui consiste en la préparation des données, l'optimisation d'algorithme, la prédiction des performances, etc. La deuxième étape, appelée t*raitement*, correspond à la mise en œuvre des algorithmes conçus sur les architectures ciblées. Concernant la troisième étape, dite de *post-traitement*, elle consiste à analyser les résultats de l'application.

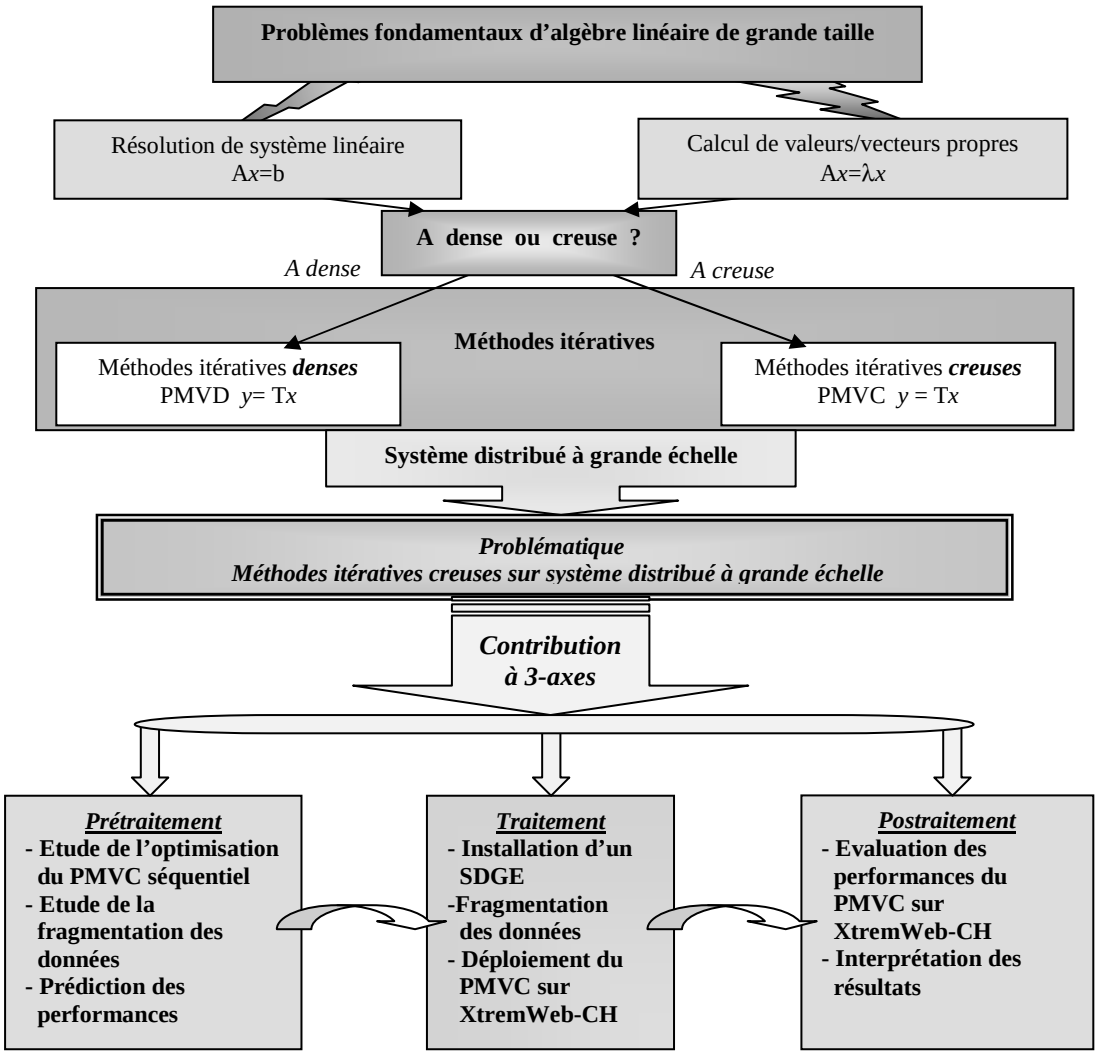


Figure 0.1 Problématique et contribution pour sa résolution

Notre contribution concerne en grande partie la première étape i.e. l'étape de prétraitement, qui a pour objectif de faciliter par la suite l'étude et la mise en œuvre des deux autres étapes.
Un système distribué à grande échelle est en fait formé par des réseaux de machines interconnectées et pouvant avoir des caractéristiques différentes. Par conséquent, à l'étape de prétraitement, nous avons à étudier les performances de l'algorithme du produit matrice-vecteur creux sur des plates-formes séquentielles. Ces dernières, de caractéristiques différentes, représentent ainsi un échantillon de nœuds pouvant constituer un SDGE. Notre étude pour le PMVC s'intéresse à plusieurs versions de cet algorithme relatives à différents formats de compression pour la matrice creuse.
Dans certains travaux, pour optimiser le PMVC, les auteurs se trouvent obligés de proposer un nouveau format de stockage (i.e. différent du format initial) [WiL06][PiH99][MeG04] [KGK08]. La plupart de ces formats constituent des variantes du format dit '*Compressed Sparse Row*' (CSR) [PiH99][MeG04][VuM05][WiL06][KGK08]. A titre indicatif, les optimisations considérées dans [MeG04] sont spécifiques à des matrices creuses où le nombre d'éléments non nuls par ligne est faible. Plusieurs travaux proposent, pour l'optimisation du PMVC, de restructurer la matrice (via des permutations de lignes et de colonnes) [Tol97] [PiC05][PiH99]. D'autres optimisations telles que celles proposées dans [ImY01][Vud02] [Wil07] dépendent de l'architecture de la machine cible. En plus des optimisations spécifiques aux architectures, certains utilisent des optimisations de bas niveau spécifiques à une architecture multi-coeurs [Wil07].
En ce qui nous concerne, nous adoptons une approche générique consistant à appliquer au PMVC des techniques d'optimisation de haut niveau. De telles optimisations sont valables pour n'importe quelle version de l'algorithme et ce, indépendamment de la structure de la matrice traitée, du format de stockage, des structures de donnée utilisées pour ce format ainsi que de l'architecture cible.
Au niveau de la même étape de prétraitement, nous nous focalisons sur l'étude de l'équilibrage des charges pour la distribution des données concernées par notre problématique, plus précisément, de la matrice creuse sur un système distribué à grande échelle (SDGE). Nous nous restreignons dans notre étude aux SDGE constitués par des nœuds homogènes. Une étude des systèmes hétérogènes pourrait en être déduite et constitue en fait une des perspectives de notre travail.

L'équilibrage des charges pour la distribution du PMVC sur des systèmes distribués à grande échelle n'est pas encore bien maîtrisé. La plupart des solutions proposées sont des solutions

dynamiques pour des *clusters* de stations de travail qui utilisent des communications avec passage de message. Les expérimentations ont montré que l'équilibrage des charges dynamique génère un overhead élevé [BiM05][Chr98].
Le problème d'équilibrage des charges que nous traitons peut être formulé comme un problème d'ordonnancement classique NP-difficile pour lequel plusieurs algorithmes d'approximation (heuristiques garanties) sont connus dans la littérature [Che04][BEP07]. Nous utilisons certains de ces algorithmes et nous proposons des heuristiques pour les améliorer.
Nous profitons, par la suite, des résultats des études sur l'optimisation de l'algorithme du PMVC et la distribution des données sur un système distribué à grande échelle (SDGE), pour proposer une étude sur la distribution du PMVC sur un SDGE.

Concernant l'étape de traitement, elle consiste à valider l'étude préalable par une série d'expérimentations réalisées sur un environnement distribué à grande échelle que nous avons installé en utilisant le système distribué à grande échelle XtremWeb-CH. L'approche que nous concevons pour la distribution de ce calcul consiste à utiliser une géométrie virtuelle à une dimension pour les nœuds. La matrice creuse est ainsi décomposée en fragments en se basant sur l'étude de la fragmentation effectuée dans l'étape de prétraitement. L'algorithme du produit matrice-vecteur creux (PMVC) utilisé sur chaque nœud correspond à une version optimisée en se basant sur l'étude effectuée dans la phase de prétraitement [EHM05][EHM09]. Afin d'évaluer les performances du PMVC sur la plate-forme XtremWeb-CH, nous utilisons un échantillon de matrices de tailles et de densités différentes qui sont soit générées aléatoirement, soit téléchargées du site Matrix Market [Mat07]. Les matrices considérées dans tout le document sont de type réel. Il est à préciser que ce travail reste aussi valable pour des matrices de type complexe.

L'étape de post-traitement, quant à elle, consiste à interpréter les résultats expérimentaux obtenus au niveau de l'étape de traitement afin d'en tirer des conclusions pertinentes. L'évaluation des performances du PMVC sur XtremWeb-CH consiste à mesurer quatre paramètres : (i) la durée totale de l'exécution de l'application, (ii) la durée totale de calcul, (iii) la durée totale des E/S et (iv) la durée totale de communication. Nous en déduisons ensuite le ratio d'E/S (rapport temps total d'E/S sur temps total d'exécution), le ratio de calcul (rapport temps total de calcul sur temps total d'exécution) et le ratio de communication (temps total de

communication sur temps total d'exécution). Ces trois derniers paramètres permettent de raffiner l'analyse des résultats.

Le manuscrit présentant notre travail est structuré en deux parties. Dans la partie I, nous détaillons les concepts de base constituant note problématique et dans la partie II, nous présentons notre contribution pour la résolution de cette problématique.

La partie I est constituée de trois chapitres. Le premier est dévolu à l'exposé des méthodes itératives de résolution de systèmes linéaires et de calcul de valeurs et de vecteurs propres. Une justification de notre choix pour les méthodes itératives dans le cas du calcul creux sous la forme d'une comparaison entre les méthodes directes et les méthodes itératives est présentée en premier lieu. Un aperçu détaillé sur les méthodes itératives de résolution de systèmes linéaires (e.g. Jacobi, Gauss-Seidel, méthodes du gradient, méthode GMRES, etc.) et de calcul de valeurs propres (e.g. méthode de la puissance, Arnoldi, méthodes hybrides, etc.) est présenté en second lieu.
Le second chapitre est consacré à la présentation de matrices creuses. Nous exposons leurs différentes structures régulières et irrégulières, les formats de stockage compressés correspondant à ces structures ainsi que certaines applications réelles dont elles proviennent. Ces dernières, reviennent généralement à la résolution de systèmes d'équations linéaires ou au calcul d'éléments propres.
Concernant le chapitre trois, il est dédié à la description de différents types de plates-formes de calcul parallèle et distribué. Ainsi, nous présentons les différentes architectures de machines parallèles, de grappes (clusters), de grilles de calcul, de systèmes distribués à grande échelle (SDGE), de systèmes pair à pair (P2P) ainsi que le récent concept de Cloud computing (Informatique dans les nuages).

En ce qui concerne la partie II du manuscrit, elle est également structurée en trois chapitres. Le premier, le second et une partie du troisième concernent l'étude que nous avons effectuée au niveau de l'étape de prétraitement. Notre travail concernant les phases de traitement et de post-traitement est présenté au troisième chapitre.
Plus précisément, dans le premier chapitre de cette partie, nous étudions les performances de l'algorithme du produit matrice-vecteur creux sur différentes plates-formes séquentielles. Quatre versions du PMVC relatives aux formats de stockage compressé DNS (DeNSe), CSR (Compressed Storage Row), COO (Coordinate) et BND (BaNDed) sont étudiées. Un état de

l’art sur l’optimisation du produit matrice-vecteur creux est aussi effectué. Suit alors une présentation des résultats de l’étude théorique et expérimentale que nous avons réalisée pour optimiser le PMVC.

Notre objectif au chapitre deux est de déterminer, pour une matrice creuse de grande taille et un vecteur dense donnés, le meilleur équilibrage des charges pour la distribution du PMVC sur un SDGE. En conséquence, trois approches heuristiques sont proposées pour la fragmentation par ligne de la matrice creuse. Une analyse théorique des performances des heuristiques suivie par une série d’expérimentations est effectuée afin de mettre en évidence l’intérêt de l’étude.

Dans le troisième chapitre, nous nous intéressons à l’étude de la distribution du PMVC sur le système distribué à grande échelle XtremWeb-CH. Ainsi, nous définissons une géométrie virtuelle pour un SDGE en utilisant une analogie avec un modèle de calcul parallèle appelé modèle data-parallèle [PeE96]. Nous présentons une évaluation théorique des performances validée par la suite par des résultats expérimentaux.

Finalement, nous clôturons ce mémoire par une conclusion qui résume notre travail et dans laquelle nous présentons nos futures perspectives.

Partie I

CHAPITRE 1.

ETAT DE L'ART SUR LES MÉTHODES ITÉRATIVES

1. Introduction

Dans les grandes applications scientifiques, deux problèmes fondamentaux d'algèbre linéaire sont souvent rencontrés, à savoir, la résolution de systèmes d'équations linéaires [DER92] [Jen80][LaT00][OsZ83] et le calcul d'éléments propres de matrices [Cia82][LaT00]. Contrairement au second problème pour lequel on ne peut utiliser que des méthodes itératives, les méthodes de résolution de systèmes linéaires peuvent être directes ou itératives.

Dans ce chapitre introductif, nous commençons, en section 2, par une présentation des principes de base des deux classes de méthodes directes et itératives de résolution de systèmes linéaires. Suit alors, une étude comparative permettant de préciser leurs avantages et inconvénients respectifs, notamment dans le cas du calcul creux qui nous intéresse. Nous présentons succinctement après le problème de calcul d'éléments propres. La section 3 est dévolue à une description détaillée des principales méthodes itératives utilisées pour la résolution des deux problèmes de résolution de systèmes linéaires et de calcul d'éléments propres. Ainsi, nous exposons les méthodes de Jacobi, Gauss-Seidel, de gradient, GMRES, les méthodes hybrides de résolution de systèmes linéaires… pour le premier problème (résolution de systèmes linéaires) et celles de la puissance (itérée), méthodes de projection, méthodes hybrides de calcul de valeurs propres… pour le second (calcul d'éléments propres). Il faut souligner que toutes ces méthodes comportent un noyau de calcul commun, à savoir le produit matrice-vecteur (PMV) dont l'instance creuse (PMVC) est celle autour de laquelle gravite notre contribution.

2. Principes de base des méthodes de résolution de systèmes linéaires et de calcul d'éléments propres

2.1 Résolution de systèmes d'équations linéaires

Un système d'équations linéaires se définit par la donnée d'une matrice $A \in \mathbb{R}^{N\times N}$ et d'un vecteur $b \in \mathbb{R}^{N}$. Il s'agit de déterminer $x \in \mathbb{R}^{N}$ vérifiant :

$$Ax = b$$

Ce problème est au cœur de nombreuses applications scientifiques : discrétisation d'équations aux dérivées partielles, linéarisation de problèmes non linéaires, approximation polynomiale au sens des moindres carrés, etc. [Cia82].

On distingue dans ce cadre deux classes de méthodes de résolution: les méthodes directes et les méthodes itératives.

2.1.1 Méthodes directes

L'idée de base des méthodes directes est de transformer le système d'équations linéaires initial A*x* = b, en un système équivalent, noté T*x*=c, plus facile à résoudre. Selon la méthode choisie, la matrice T obtenue est soit triangulaire (méthode d'élimination de Gauss, voir Figure 1.1), soit diagonale (méthode d'élimination de Gauss-Jordan). T peut aussi s'écrire sous la forme du produit de deux matrices : (i) la première est triangulaire inférieure et la seconde triangulaire supérieure (factorisation LU, Figure 1.2.(b), ou bien factorisation de Cholesky lorsque A est symétrique définie positive, Figure 1.2.(a)) ; (ii) la première est unitaire et la seconde triangulaire supérieure (factorisation QR, Figure 1.2.(c)) [Cia82].

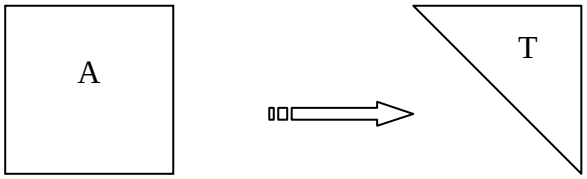
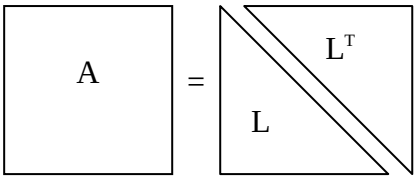
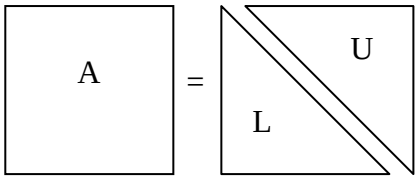


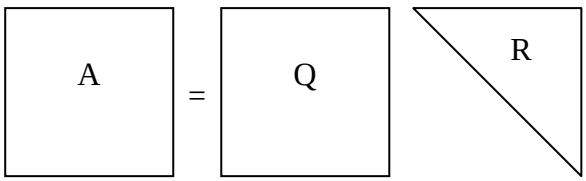
Figure 1.1 Méthode d'élimination de Gauss



(a) Factorisation de Cholesky



(b) Factorisation LU



(c) Factorisation QR

Figure 1.2 Méthodes de factorisation

Les méthodes directes comportent deux phases : une première phase de triangularisation/

diagonalisation/factorisation et une seconde phase de résolution. La résolution du système triangulaire (resp. des deux systèmes triangulaires) obtenu(s) dans le cas des méthodes de Gauss et QR (resp. des factorisations LU et de Cholesky) se fait alors, par une technique dite de remontée (resp. descente puis remontée).

Si les calculs étaient effectués à précision infinie, le résultat obtenu serait la solution exacte du système d'équations linéaires. Cependant, la propagation des erreurs d'arrondi peut faire échouer ces méthodes lorsque la matrice A est mal conditionnée [Gue06].

2.1.2 Méthodes itératives

Le principe de base des méthodes itératives consiste à construire une suite $x^{(0)}$, $x^{(1)}$, …, $x^{(k)}$… qui converge, sous certaines conditions, vers la solution du système. Formellement, partant de $x^{(0)}$ (choisi arbitrairement), nous avons :

$$x^{(k+1)} = T^{(k)} x^{(k)} + c^{(k)} \quad : k=0,1\ldots$$

où $T^{(k)}$ est une matrice construite à partir de la matrice A et le cas échéant du vecteur $x^{(k)}$, le vecteur $c^{(k)}$ dépend en plus (de A et éventuellement de $x^{(k)}$) du second membre b (dans l'équation Ax=b). Plusieurs méthodes itératives sont ainsi connues dans la littérature. Elles diffèrent par le choix des matrices $T^{(k)}$. Les méthodes itératives les plus simples, dites classiques ou stationnaires, correspondent aux cas où $T^{(k)}$ et $c^{(k)}$ sont constants [Cia82][Haz02] [You71].

Les méthodes dites non-stationnaires sont utilisées pour accélérer la convergence des méthodes itératives [Bur98][Haz02][You71]. Dans ce cadre, les plus utilisées sont celles utilisant les polynômes de Chebyshev [Bur98][You71] ou bien les directions conjuguées [Bur98][Cia82].

Le choix de la matrice T dans le cas d'une méthode stationnaire, et des matrices $T^{(k)}$ dans le cas non stationnaire, peut être effectué de diverses manières. Précisons dans ce dernier cas qu'il est possible de construire les matrices de manière à ce que la méthode itérative converge aussi rapidement que possible pour une grande classe de systèmes d'équations linéaires [Bur98][You71].

Il convient de souligner que pour être appliquées, les méthodes directes n'exigent que l'inversibilité de la matrice A, alors que les méthodes itératives exigent des conditions supplémentaires sur A afin de garantir leur convergence. Dans le cas où cette dernière n'est pas garantie, le système à résoudre est transformé en un système équivalent dont la matrice associée satisfait les conditions requises [You71].

2.1.3 Méthodes de résolution de systèmes linéaires appropriées au calcul creux?

Le choix entre une méthode directe et une méthode itérative n'est pas, a priori, simple. En effet, en théorie, cela dépend de la structure ainsi que de la taille de la matrice, et en pratique de l'architecture de la machine cible en plus [Bur98].

Les méthodes directes sont connues pour être robustes et conviennent plutôt à des problèmes de taille modérée. Elles sont généralement plus rapides que les méthodes itératives. Pour des problèmes de très grande taille, le coût de ces méthodes peut devenir prohibitif car trop gourmandes en termes de mémoire (à cause du phénomène de remplissage '*fill-in*') et de calcul. Sur cette gamme de problèmes, les méthodes itératives constituent ainsi une alternative intéressante [DER92] [Wyr01].

Du fait que les méthodes itératives n'affectent pas la matrice A, elles permettent une exploitation totale de son caractère creux. De plus, elles permettent d'éviter le problème du *fill-in* qui, dans le cas d'un stockage compressé de la matrice creuse (i.e. se limitant aux éléments non nuls, voir chapitre 2, section 3), augmente et complique énormément le calcul. Par opposition aux méthodes itératives qui conservent le caractère creux de la matrice A du système, les méthodes directes peuvent fournir à la fin de l'étape de transformation (ou d'élimination dans le cas de la méthode de Gauss notamment) des matrices (triangulaires, dans le cas de la méthode de Gauss) qui peuvent être denses, à l'exception du cas où A est une matrice bande (voir chapitre 2, section 2.2). Néanmoins, dans plusieurs problèmes à matrice bande, la bande elle-même est souvent creuse, ce qui peut rendre l'algorithme assez difficile à implémenter, tel celui de Cholesky-bande [GoV83][INR02].

Le point faible des méthodes itératives réside dans le fait que leur convergence peut-être lente, d'où la nécessité de leur "pré-conditionnement" consistant, en gros, à transformer la matrice A en une matrice satisfaisant mieux les conditions de convergence [You71]**.** Parmi les pré-conditionneurs les plus robustes, on peut citer les méthodes par bloc. Pour ces dernières, des *solveurs* directs creux sont utilisés sur chacun des blocs de la matrice. On peut donc combiner les méthodes itératives et les méthodes directes dans le but d'une résolution plus rapide et plus efficace d'un système linéaire.

Un récapitulatif des avantages et inconvénients des méthodes directes et itératives dans le cas de calcul creux est présenté dans la Table 1.1.

Méthodes	Avantages	Inconvénients
Directes	• robustes • rapides pour des problèmes de taille modérée • fournissent des solutions exactes	• prohibitives car trop gourmandes en termes de mémoire et de calcul • peuvent fournir à la fin de l'étape de transformation des matrices assez denses • problème de *fill-in*
Itératives	• n'affectent pas la matrice A, conservent le caractère creux de A • permettent une exploitation totale du caractère creux de la matrice A • permettent d'éviter le problème du *fill-in*	• convergence pouvant être lente • nécessité de leur "pré-condition-nement"

Table 1.1 Comparaison entre méthodes directes et itératives dans le cas creux

2.2 Calcul d'éléments propres

Etant donnée une matrice carrée A, d'ordre N, déterminer toutes ses valeurs propres, ou bien seulement certaines d'entre elles et, éventuellement, les vecteurs propres correspondants, revient à chercher des scalaires λ (pouvant être nuls) et des vecteurs v non nuls tels que:

$$Av = \lambda v \quad : \quad v \neq 0$$

Le problème du calcul de valeurs et de vecteurs propres de matrices, appelé par contraction calcul d'éléments propres, se rencontre en particulier dans l'approximation des *mouvements stationnaires*, ou des petits mouvements, au sein de divers systèmes physiques [Cia82]. Il se pose également dans le cadre de la recherche d'information [BrL06][Mes06].

Il faut noter que, pour N élevé (en pratique N≥5), le calcul d'éléments propres ne peut se faire, qu'en utilisant des méthodes itératives. Dans ce cas, les valeurs propres sont déterminées, soit une par une (e.g. méthode de la puissance, méthode de déflation, etc.), soit globalement, à travers la construction de suites numériques convergeant, sous certaines conditions, vers les valeurs recherchées (e.g. méthodes LR, QR, Arnoldi, ERAM, etc.). Il en est de même pour les vecteurs propres. Nous exposerons en section 3 certaines méthodes itératives appropriées au calcul d'éléments propres.

3. Méthodes itératives

3.1 Résolution de systèmes linéaires

3.1.1 Méthodes itératives stationnaires $x^{(k+1)}=Tx^{(k)} + c$

- *Méthode de Jacobi*

Considérons le système A*x* = b. La méthode de Jacobi est définie pour des systèmes dont la matrice A possède des éléments diagonaux non nuls. A est d'abord décomposée comme suit : A = D-M-N (voir Figure 1.3).

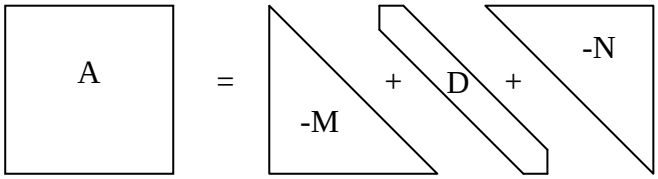


Figure 1.3 Décomposition de A=D-M-N

D est la partie diagonale de A, -M est la partie triangulaire inférieure stricte et -N est la partie triangulaire supérieure stricte. Le système A*x*=b s'écrit alors :

$$(D\text{-}M\text{-}N)\,x = b \Rightarrow Dx = (M+N)x + b \Rightarrow x = D^{-1}(M+N)x + D^{-1}b$$

Soit $J = D^{-1}(M+N)$. J est appelée matrice de *Jacobi*. La méthode de *Jacobi* s'écrit alors :

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = Jx^{(k)} + D^{-1}b : k = 0,1, \ldots \end{cases}$$

Nous avons ainsi l'expression générale $x^{(k+1)} = T\,x^{(k)} + c$ avec T=J et $c=D^{-1}b$.

Cet algorithme converge à condition que J soit contractante i.e. il existe une norme telle que cette norme est strictement plus petite que 1, soit $\|J\| < 1$. Ceci implique que $\rho(J) < 1$, où $\rho(J)$ est le rayon spectral de J i.e. le module de la valeur propre de plus grand module de J. C'est notamment le cas où A est à diagonale dominante i.e. $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$.

Une itération de la méthode de Jacobi se note formellement *y*=J*x*+c i.e. admet comme noyau de base un produit matrice-vecteur (PMV) [Cia82][Jen80].

- *Méthode de Gauss-Seidel*

Partant de la même décomposition précédente de A = D-M-N, nous avons:

$$(D\text{-}M\text{-}N)\,x = b \Rightarrow (D\text{-}M)\,x = Nx + b \Rightarrow x = (D\text{-}M)^{-1}Nx + (D\text{-}M)^{-1}b$$

Soit $\mathcal{L} = (D\text{-}M)^{-1}N$. $\mathcal{L}$ est appelée matrice de *Gauss-Seidel*. La méthode de Gauss-Seidel s'écrit alors:

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = \mathcal{L}\,x^{(k)} + (D\text{-}M)^{-1}b : k = 0,1, \ldots \end{cases}$$

Nous avons ainsi l'expression générale $x^{(k+1)} = T\,x^{(k)} + c$ avec $T=\mathcal{L}$ et $c=(D-M)^{-1}b$.

Cet algorithme converge à condition que $\mathcal{L}$ soit contractante i.e. il existe une norme telle que cette norme est strictement plus petite que 1, soit $\| \mathcal{L} \| <1$. Ceci implique que $\rho(\mathcal{L}) <1$, où $\rho(\mathcal{L})$ est le rayon spectral de $\mathcal{L}$. C'est notamment le cas où A est symétrique définie positive.

Une itération de la méthode de Gauss-Seidel se note formellement $y= \mathcal{L}\,x+c$ i.e. admet comme noyau de base un produit matrice-vecteur (PMV).

Précisons, qu'en pratique, la matrice $\mathcal{L}$ n'est jamais explicitement calculée. C'est plutôt l'expression $(D-M)\,x^{(k+1)} = Nx^{(k)} + b$ qui est utilisée pour déterminer $x^{(k+1)}$ à partir de $x^{(k)}$.

D'un autre coté, il est prouvé que la méthode de Gauss-Seidel est en général au moins aussi rapide que la méthode de Jacobi [Cia82][Jen80].

- *Méthodes de relaxation*

Considérons la méthode itérative $x^{(k+1)} = T\,x^{(k)} + c$. Soit ω un paramètre, appelé coefficient de relaxation et vérifiant $\omega \neq 0$ et 1. On définit la matrice T_ω, dite relaxée de T comme suit :

$$T_\omega = \omega T+(1- \omega)I : I \text{ matrice identité}$$

La méthode itérative associée se note alors comme suit :

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = T_\omega x^{(k)} + \omega c : k = 0,1, \ldots \end{cases}$$

Elle convergera pourvu que $\rho(T_\omega) <1$. On parlera de sur-relaxation (resp. sous-relaxation) lorsque $\omega<1$ (resp. $\omega >1$).

Il convient de souligner que la relaxation a pour but de rendre convergente une méthode itérative qui ne l'est pas ou bien d'accélérer sa convergence.

- La relaxation de la méthode de Jacobi se note comme suit :

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = J_\omega\, x^{(k)} + \omega D^{-1}b : k = 0,1, \ldots \end{cases}$$

avec $J_\omega = \omega J+(1- \omega)I= \omega D^{-1}(M+N)+(1-\omega)I$. La relaxation de la méthode de Jacobi convergera si $\rho(J_\omega) <1$.

- La relaxation de la méthode de Gauss-Seidel se note comme suit :

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = \mathcal{L}_\omega\, x^{(k)} + \omega(D-M)^{-1}b : k = 0,1, \ldots \end{cases}$$

avec $\mathcal{L}_\omega = \omega\, \mathcal{L} +(1-\omega)I=\omega(D-M)^{-1}N+(1-\omega)I$. La relaxation de la méthode de Gauss-Seidel convergera si $\rho(\mathcal{L}_\omega) <1$.

Précisons, qu'en pratique, la matrice $\mathcal{L}_\omega$ n'est jamais calculée. On procède plutôt comme suit :

(i) Détermination de y à partir de : $(D-M)y = Nx^{(k)} + b$ (comme cela se fait dans la méthode de Gauss-Seidel)

(ii) $x^{(k+1)} = \omega y + (1-\omega)x^{(k)}$

Notons que la relaxation de la méthode de Gauss-Seidel est beaucoup plus utilisée en pratique que la relaxation de la méthode de Jacobi [Cia82][Jen80][LaT00].

- *Méthode de directions alternées*

Supposons que la matrice A du système $Ax=b$ s'écrive sous la forme d'une somme de deux matrices H et V telle que la résolution des deux systèmes linéaires $(H+pI)x = h$ et $(V+pI)x = v$ (p étant un réel et I la matrice identité) soit facile et peu coûteuse. Le système initial $Ax=b$ se notant $(H+V)x = b$ i.e. $Hx=b-Vx$ et $Vx=b-Hx$, les derniers systèmes peuvent alors s'écrire comme suit :

$$\begin{cases} (H+pI)x = b+(pI-V)x \\ (V+pI)x = b+(pI-H)x \end{cases}$$

On en déduit la *méthode* itérative dite *de directions alternées* [LaT00] où $x^{(k)}$ (k=0…) étant connu, l'itération k+1 est constituée par deux 'demi' itérations, se ramenant chacune à la résolution d'un système linéaire comme suit :

$$\begin{cases} (H+pI)x^{(k+1/2)} = b+(pI-V)x^{(k)} & : \text{solution } x^{(k+1/2)} \\ (V+pI)x^{(k+1)} = b+(pI-H)x^{(k+1/2)} & : \text{solution } x^{(k+1)} \end{cases}$$

Notons que le paramètre p, pouvant dépendre de l'itération, est à déterminer au mieux e.g afin d'accélérer la convergence de l'algorithme. Il est en fait possible d'écrire les systèmes précédents sous une forme faisant apparaître les résultats qui serviront pour tester l'arrêt des itérations. Soit :

$$\begin{cases} (H+pI)\,(x^{(k+1/2)} - x^{(k)}) = b+(pI-V)\,x^{(k)} - (H+pI)\,x^{(k)} = b-A\,x^{(k)} \\ (V+pI)\,(x^{(k+1)} - x^{(k+1/2)}) = b+(pI-H)\,x^{(k+1/2)} - (V+pI)\,x^{(k+1/2)} = b-A\,x^{(k+1/2)} \end{cases}$$

Autrement écrit :

$$\begin{cases} (H+pI)(x^{(k+1/2)} - x^{(k)}) = b-Ax^{(k)} \\ (V+pI)(x^{(k+1)} - x^{(k+1/2)}) = b-Ax^{(k+1/2)} \end{cases}$$

3.1.2 Méthodes itératives non stationnaires

- *Méthodes du gradient :* $x^{(k+1)} = T^{(k)} x^{(k)} + c^{(k)}$

Lorsque la matrice A est symétrique définie positive, on peut utiliser les méthodes du gradient qui se basent sur la transformation du système en un problème de minimisation d'une fonction f(x), appelée forme quadratique, définie comme suit :

$$f(x) = x^T A x/2 - x^T b$$

On construit alors un algorithme itératif (dit de descente), permettant de minimiser f(x), et qui est de la forme générale [LaT00]:

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = T^{(k)} x^{(k)} + c^{(k)} \quad : k = 0,1, \ldots \end{cases}$$

La matrice $T^{(k)}$ ainsi que le vecteur $c^{(k)}$ sont définis à partir du gradient de f(x) en $x^{(k)}$, $\nabla f(x^{(k)}) = A x^{(k)} - b$. Minimiser f($x$) revient ainsi à annuler son gradient. La convergence de la méthode de gradient se traduit par la convergence du gradient (resp. $x^{(k)}$) vers le vecteur nul (resp. la solution du système Ax=b). Le minimum de f(x) sera alors égal à 0 ici.

- *Méthodes du gradient conjugué*

On peut en fait définir diverses variantes pour la méthode du gradient, en particulier la méthode du gradient conjugué. Cette dernière est souvent utilisée pour la résolution des systèmes à matrice symétrique définie positive. Elle s'explicite comme suit ($x^{(0)}$ donné) :

$$\begin{cases} x^{(0)} \text{ donné} \\ x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \quad : k = 0,1\ldots \end{cases}$$

Nous avons ainsi $T^{(k)} = I$ (matrice identité) et $c^{(k)} = \alpha_k p^{(k)}$, où α_k est un scalaire appelé pas de déplacement et $p^{(k)}$ un vecteur appelé direction de descente, définis comme suit [Gue06][Jen80]:

$$\begin{cases} r^{(0)} = p^{(0)} = b \\ \alpha_k = \|r^{(k)}\|^2 / ((p^{(k)})^T A p^{(k)}) \\ r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}, \\ p^{(k+1)} = r^{(k)} + (\|r^{(k+1)}\| / \|r^{(k)}\|) p^{(k)} \end{cases}$$

Précisons que plusieurs autres méthodes dites de gradient sont connues dans la littérature e.g. la méthode du gradient bi-conjugué (GBiC), la méthode Carré du Gradient Conjugué (CGC) (voir section 3.1.3) [Bar94] [Gre97][Son89].

- *Méthode GMRES (Generalized Minimal RESidual)*

Soit à résoudre le système Ax=b où A est inversible. On choisit un vecteur quelconque z, qu'on suppose être normalisé i.e. $\|z\| = 1$ ($\|\cdot\|$ correspond par exemple à la norme Euclidienne). Soit m << N. Un sous-espace de Krylov de taille m est défini à partir de A et du vecteur z comme suit:

$$K_m(A, z) = [z, A z, A^2 z, \ldots, A^{m-1} z]$$

La méthode GMRES calcule une approximation $x_m \in K_m$ de la solution exacte de Ax=b. Cette solution minimise la norme du résidu $Ax_m - b$.

Certains vecteurs dans la base z, Az, …, $A^{m-1}z$ sont linéairement dépendants [SaS86]. Par conséquent, au lieu d'utiliser cette base, on applique l'itération d'Arnoldi (voir Figure 1.4) pour trouver les vecteurs orthonormaux $q_1, \ldots, q_m$ formant une base orthonormale pour K_m. Ainsi, le vecteur $x_m \in K_m$ peut être écrit sous la forme $x_m = Q_m y_m$ où $y_m \in \mathbb{R}^m$ et Q_m est la matrice de taille (N×m) formée par les vecteurs colonnes $q_1, \ldots, q_m$.

Le processus d'Arnoldi fournit aussi une matrice Hessenberg supérieure (voir chapitre 2, section 2.2.3) $\hat{H}_m$ de taille ((m+1)×m) avec :

$$AQ_m = Q_{m+1} \hat{H}_m$$

Comme Q_m est orthogonale, nous avons

$$\|Ax_m - b\| = \|\hat{H}_m y_m - \beta e_1\|$$

où e_1=(1,0,0,…,0) est le premier vecteur dans la base canonique de $\mathbb{R}^{m+1}$ et $\beta = \|b - Az\|$. Ainsi, x_m peut être calculé en minimisant la norme du résidu suivant :

$$r_m = \hat{H}_m y_m - \beta e_1$$

Ceci correspond en fait à un problème d'approximation linéaire des moindres carrés de taille m. Ainsi, une itération de la méthode GMRES consiste à effectuer les étapes suivantes :

(i) Utiliser l'algorithme d'Arnoldi (voir Figure 1.4) pour déterminer Q_m et $\hat{H}_m$

(ii) Déterminer y_m minimisant $\|r_m\|$

(iii) Calculer $x_m = Q_m y_m$

(iv) Itérer le processus tant que la norme du résidu $\|Ax_m - b\|$ n'est pas suffisamment réduite et ce, en choisissant $z = x_m$

De ce fait, à chaque itération, un produit matrice-vecteur Ax_m doit être calculé en plus des produits matrice-vecteur effectués au niveau de l'algorithme d'Arnoldi [SaS86].

3.1.3 Méthodes itératives hybrides

Plusieurs méthodes de résolution de systèmes linéaires peuvent être vues comme des combinaisons de méthodes de sous espaces de Krylov [VFS93], telles que les méthodes du Gradient Bi-Conjugué (GBiC) et GMRES. De telles méthodes hybrides incluent la méthode Carré du Gradient Conjugué (CGC), celle du gradient bi-conjugué stabilisé (BiCGSTAB), les méthodes QMRES, TFQMR, GMRESR, etc [AlS98] [Saa03] [Son89] [Van03] [VFS93].

A titre d'exemple, le résidu $r^{(k)} = b\text{-}Ax^{(k)}$ dans la méthode du gradient bi-conjugué (GBiC) appliquée à la résolution du système $Ax=b$ avec un vecteur initial $x^{(0)}$, peut s'écrire sous la forme $r^{(k)}=P_k(A)r^{(0)}$ où P_k est un polynôme de degré k. Il a été montré [Son89] qu'avec presque le même volume de calcul, on peut construire des résidus ayant la forme $r^{(k)} = (P_k(A))^2 r^{(0)}$, ce qui constitue la base de la méthode Carré du Gradient Conjugué (CGC).

Par analogie avec la méthode CGC [Van92][Van03], on peut construire des méthodes pour lesquelles $r^{(k)}$ s'écrit sous la forme $r^{(k)}=P_k(A)Q_k(A)r^{(0)}$, P_k étant le polynôme associé à la méthode GBiC et Q_k un polynôme de degré k pouvant être librement choisi sous la condition $Q_k(0)=1$. Il a été proposé dans [Van92] de construire Q_k comme produit de k monômes du type $I\text{-}w_jA$, où w_j est choisi de façon à minimiser localement le résidu. Cette approche constitue le principe de la méthode du gradient bi-conjugué stabilisé (BiCGSTAB) qui peut être vue comme une combinaison des méthodes GBiC et GMRES.

Une variante de la méthode BiCGSTAB est la méthode dite BiCGSTAB2 où le polynôme Q_k est déterminé à partir du trinôme $I\text{-}w'_jA\text{-}w''_jA^2$.

3.1.4 Méthodes par bloc

Toutes les méthodes précédentes, qu'on peut qualifier comme *méthodes par points* peuvent se généraliser en utilisant des décompositions par blocs des matrices traitées [Cia82][LaT00]. On parle alors de *méthodes par blocs*. A cet effet, supposons que la matrice A du système $Ax=b$ admette une décomposition par blocs (rectangulaires) notés A_{ij} (i,j=1…p) de taille (n_i,n_j) et que les vecteurs x et b soient à leur tour décomposés chacun en p sous-vecteurs x^i, b^i de taille n_i (i=1…p). Le système $Ax=b$ de dimension $N = \sum \mathbf{n_i}$ peut alors s'écrire comme suit :

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1p} \\ A_{21} & A_{22} & \dots & A_{2p} \\ \vdots & & & \vdots \\ A_{p1} & A_{p2} & \dots & A_{pp} \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^p \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^p \end{bmatrix} \quad : \sum_{j=1}^{p} A_{ij} x^j = b^i,\ i=1\dots p$$

On peut alors définir les sous matrices diagonale par bloc D et triangulaire inférieure (resp. supérieure) stricte par bloc -M (resp. –N) de A i.e. A=D-M-N comme suit :

$$D = \begin{bmatrix} A_{11} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & A_{pp} \end{bmatrix} ; -M = \begin{bmatrix} 0 & & & \\ A_{21} & \ddots & & \\ A_{31} & A_{32} & \ddots & \\ \vdots & \vdots & & \ddots \\ A_{p1} & A_{p2} & \cdots & A_{p,p-1} \quad 0 \end{bmatrix} ; -N = \begin{bmatrix} 0 & A_{12} & A_{13} & \cdots & A_{1p} \\ & \ddots & A_{23} & \cdots & A_{2p} \\ & & \ddots & & \vdots \\ & & & \ddots & A_{p-1,p} \\ & & & & 0 \end{bmatrix}$$

A titre d'exemple, la méthode de Jacobi par blocs se définit comme suit [LaT00] :

$$A_{ii}\, x^{i,(k+1)} = b^i - \sum_{j \neq i} A_{ij}\, x^{j,(k)} : i=1\ldots p \text{ , } k=0,1\ldots$$

Connaissant $x^{j,(k)}$, j=1…p, la détermination de $x^{i,(k+1)}$, i=1…p, se ramène ainsi à la résolution d'un système linéaire d'ordre n_i. Si on utilise une méthode directe (par points) telle que la factorisation LU (voir section 2.1.1), on peut noter qu'on aura à effectuer la factorisation de chaque bloc diagonal A_{ii} une et une seule fois au départ. On sauvegardera ainsi les facteurs à chaque itération. La résolution effective revient donc à celle de deux systèmes triangulaires successifs comme indiqué auparavant (voir section 2.1.1).

3.1.5 Pré-conditionnement d'une matrice

Soi || . || une norme matricielle subordonnée et A une matrice inversible. Le nombre défini par

$$cond(A)=\|A\|*\|A^{-1}\|$$

s'appelle conditionnement de la matrice A, relativement à la norme matricielle considérée. Dans la littérature anglo-saxonne, ce nombre, appelé '*condition number*', est fréquemment noté $\chi(A)$ [Cia82][LaT00]. Noter qu'on a toujours cond(A) ≥1.

cond(A) mesure en fait la sensibilité de la solution *x* du système linéaire A*x*=b vis-à-vis des variations sur les données A et b, qualité que l'on appelle conditionnement du système linéaire considéré. Ainsi, pratiquement, un système linéaire est dit bien (resp. mal) conditionné si de légères perturbations sur les données d'entrée (du type A+ΔA et/ou b+δb) génèrent de légères (resp. grandes) perturbations sur le résultat (*x*). Dans le premier cas, cond(A) est assez proche de 1 alors qu'on a cond(A) >>1 dans le second cas.

Le pré-conditionnement d'une matrice A (mal conditionnée) consiste en fait à remplacer la résolution du système A*x* = b par celle d'un système équivalent se notant $C^{-1}Ax=C^{-1}b$, la matrice inversible C étant choisie de telle manière que $cond(C^{-1}A) << cond(A)$ i.e. $cond(C^{-1}A)$ assez proche de 1.

En théorie, le meilleur choix est C=A. En effet, dans ce cas, on a cond($C^{-1}A$) =1. En pratique, il s'agit de trouver une matrice C telle que C^{-1} soit '*la plus proche*' de A^{-1} sans que la détermination de C^{-1} soit trop coûteuse. Notons qu'à ce propos diverses techniques appropriées sont connues dans la littérature [Cia82][LaT00].

3.2 Calcul de valeurs propres

3.2.1 Méthodes de la puissance itérée

- *Méthode de la puissance*

La méthode de la puissance peut être utilisée pour calculer la plus grande valeur propre en module ainsi que le vecteur propre correspondant pour une matrice quelconque.

Notons $\lambda_1,..., \lambda_m$ les valeurs propres de A et supposons que $|\lambda_1| \geq |\lambda_2| \geq ... |\lambda_{m-1}| \geq |\lambda_m|$. Si la matrice des vecteurs propres $V=(v_1, v_2, \ldots, v_m)$ est non-singulière, alors tout vecteur $u^{(0)}$ peut être exprimé comme combinaison linéaire des vecteurs propres de la matrice A [Jen80]. Ainsi,

$$u^{(0)}=c_1v_1+c_2v_2+ \ldots +c_mv_m$$

Soit $u^{(1)}=Au^{(0)}$, nous avons :

$$u^{(1)}=Au^{(0)}= \sum_{i=1}^{m} c_i Av_i = \sum_{i=1}^{m} \lambda_i c_i v_i$$

On définit alors $u^{(k)}=Au^{(k-1)} =\ldots = A^k u^{(0)}$: k=1,2…. On aura alors :

$$u^{(k)} =A^k u^{(0)}= \lambda_1^k c_1 v_1 + \lambda_2^k c_2 v_2 + ... + \lambda_m^k c_m v_m$$

$$u^{(k)} = \lambda_1^k (c_1v_1+ (\lambda_2/\lambda_1)^k c_2v_2+\ldots+ (\lambda_m/\lambda_1)^k c_mv_m)$$

Si $|\lambda_1| > |\lambda_i|$, alors, lorsque k est élevé et c_1 est non nul, on a $(\lambda_i/\lambda_1)^k \approx 0$ (en fait tend vers 0 lorsque k tend vers l'infini). De ce fait :

$$u^{(k)} \approx \lambda_1^k c_1 v_1$$

Ainsi, $u^{(k)}$ tend vers un vecteur proportionnel au vecteur propre dominant v_1. D'un autre côté, si nous choisissons des composantes non nulles $u_j^{(k)}$ et $u_j^{(k-1)}$, nous pouvons déduire que :

$$(u_j^{(k)} / u_j^{(k-1)}) \rightarrow \lambda_1 \text{ pour } k \rightarrow \infty$$

En pratique, il est convenu de normaliser le vecteur $u^{(k)}$ après chaque multiplication par la matrice A. Un algorithme itératif pour déterminer v_1 peut alors être exprimé par les deux équations :

$$w^{(k)} = Au^{(k)}, \quad u^{(k+1)} = \frac{1}{\alpha} w^{(k)} \text{ où } \alpha = \pm \|w^{(k)}\|$$

Ainsi, un produit matrice-vecteur $Au^{(k)}$ est effectué à chaque itération de l'algorithme [Jen80].

- *Méthode de la puissance inverse*

C'est la méthode de la puissance appliquée soit à la matrice A^{-1}, ce qui permet d'obtenir la valeur propre de plus petit module de A et un vecteur propre associé, soit à la matrice $(A-\mu I)^{-1}$ ce qui permet d'obtenir la valeur propre la plus proche de μ et un vecteur propre associé. Dans ce dernier cas, on utilise la méthode de la puissance inverse pour calculer efficacement une approximation d'un vecteur propre lorsque l'on connait une approximation d'une valeur propre obtenue par un autre procédé [LaT00].

3.2.2 Méthodes de projection

Soient A une matrice non hermitienne réelle d'ordre N et K un sous espace de $\mathbb{R}^N$. Une matrice Hermitienne (ou auto-adjointe) est une matrice carrée telle que $A=A^H$ (A^H étant la matrice transconjuguée de A i.e. la matrice transposée de la matrice conjuguée de A). Dans le cas où A est réelle, A est dite Hermitienne si et seulement si elle est symétrique.

Une méthode de type sous espace de Krylov permet de calculer une approximation d'une paire (λ_i, x_i) d'éléments propres de A à partir d'une paire $(\lambda_i^{(m)} \in \mathbb{R}, x_i^{(m)} \in K)$ d'éléments de Ritz où le sous espace est défini comme suit :

$$K_m(A,z) = [z, A z, A^2 z, \ldots, A^{m-1} z]$$

z est ici un vecteur quelconque, supposé normalisé i.e. $\|z\| = 1$ ($\|\cdot\|$ correspond par exemple à la norme Euclidienne).

Ces méthodes permettent d'estimer s éléments propres de A par ceux de la matrice d'ordre m obtenue par une projection orthogonale de A sur le sous espace K_m avec $s \leq m << N$.

Soit Q_m, la matrice dont les colonnes $q_1, \ldots, q_m$ constituent une base orthogonale de K_m. Le problème consiste alors à trouver $\lambda_i^{(m)} \in \mathbb{R}$ et $y_i^{(m)} \in \mathbb{R}^N$ (i=1,…, s) tels que :

$$(H_m - \lambda_i^{(m)} I)\, y_i^{(m)} = 0 \qquad (1.1)$$

où la matrice carrée H_m de taille m est définie par $H_m = Q_m^H A Q_m$ (Q_m^H étant la transposée de la matrice conjuguée de Q_m) et $x_i^{(m)} = Q_m y_i^{(m)}$. Ainsi, certains éléments de A peuvent être calculés approximativement à partir de ceux de la matrice H_m. Ils peuvent être déterminés en construisant une base orthogonale de K_m et en résolvant le problème (1.1). Notons qu'il existe

différentes manières pour construire une telle base. La plus utilisée est le processus d'orthogonalisation d'Arnoldi [EPE05].

- *Algorithme d'Arnoldi*

Soit un vecteur initial q_1 de norme 1. Le processus d'Arnoldi génère une base orthogonale $q_1, \ldots, q_m$ du sous espace de Krylov K_m en utilisant le processus d'orthogonalisation de Gram-Schmidt [EPE05] comme détaillé ci-dessous:

Réduction d'Arnoldi : AR (Entrées : A, m, z ; Sorties : H_m, Q_m)

$q^{(1)}$: vecteur initial de norme 1

Do k = 2, m // Boucle $\mathcal{B}1$ //

$q^{(k)} = Aq^{(k-1)}$

Do j=1,k-1 // projection des composantes de $q^{(k)}$ dans les directions $q^{(1)}, \ldots q^{(k-1)}$ //

$h_{j,k-1} = q^{(j)T}q^{(k)}$, $q^{(k)} = q^{(k)} - h_{j,k-1}q^{(j)}$

EndDo

$h_{k,k-1} = \|q^{(k)}\|$, $q^{(k)} = q^{(k)} / h_{k,k-1}$

EndDo

Figure 1.4 Algorithme d'Arnoldi

Cette méthode a été introduite afin de réduire une matrice donnée à une matrice de forme Hessenberg. Les matrices H_m et Q_m issues de l'algorithme d'Arnoldi et définies à partir de la matrice A satisfont l'équation :

$$AQ_m = Q_mH_m + f_m e_m^H$$

où $f_m = h_{m+1,m}Q_{m+1}$ et e_m est le $m^{ème}$ vecteur de la base canonique de $\mathbb{R}^m$. Une fois le processus d'orthogonalisation effectué, les s valeurs de Ritz désirées (i.e. $\Lambda_m=(\lambda_1^{(m)}, \ldots, \lambda_s^{(m)})$ et les vecteurs de Ritz associés $X_m = (x_1^{(m)}, \ldots, x_s^{(m)})$ peuvent être calculés comme suit [EPE05]:

Algorithme Basique d'Arnoldi : BAA (Entrées : A, s, m, z ; Sorties : r_s, Λ_m, X_m)

1. Calculer une étape AR(Entrées : A, m, z ; Sorties : H_m, Q_m)
2. Calculer les éléments propres de H_m et sélectionner les s éléments désirés.
3. Calculer les s vecteurs de Ritz associés $x_i^{(m)} = Q_m y_i^{(m)}$, i=1, …, s
4. Calculer $r_s=(\rho_1, \ldots, \rho_s)$ avec $\rho_i = \|(A-\lambda_i^{(m)} I)x_i^{(m)}\|_2$

Figure 1.5 Algorithme Basique d'Arnoldi (BAA)

Si la précision des éléments de Ritz calculés n'est pas satisfaisante, la projection peut être redémarrée sur un nouveau sous espace $K_m(A, z)$.

- *Méthode d'Arnoldi avec Redémarrage utilisant un plus grand sous-espace (RAM)*

Une première variante de la méthode d'Arnoldi consiste à définir le nouvel espace $K_m(A,z)$ en utilisant le même vecteur initial z et une plus grande valeur de m.

- *Méthode d'Arnoldi avec Redémarrage Explicite (ERAM)*

Dans la seconde variante de la méthode de projection, la taille du sous-espace est fixée. Cette méthode s'appelle Méthode d'Arnoldi avec Redémarrage Explicite (ERAM). Partant d'un vecteur initial z, elle exécute l'algorithme BBA. Le vecteur initial est mis à jour et un processus basique d'Arnoldi est redémarré jusqu'à ce que la précision de l'approximation de la solution soit satisfaisante (en utilisant les méthodes appropriées pour calculer les vecteurs de Ritz). Cette modification est conçue pour forcer le vecteur désiré dans un sous espace invariant. L'algorithme de la Méthode d'Arnoldi avec Redémarrage Explicite se détaille comme suit [EPE05] :

ERAM (Entrées : A, s, m, z, tol ; Sorties : r_s, Λ_m, X_m)

1. Démarrer. Choisir un paramètre m et un vecteur initial z.
2. Itérer. Calculer une étape BAA (Entrées : A, s, m, z ; Sorties : r_s, Λ_m, X_m).
3. Redémarrer. Si $g(r_s) >$ tol alors utiliser Λ_m et X_m pour mettre à jour le vecteur de départ et aller à 2.

Figure 1.6 Algorithme ERAM

Noter que tol est une valeur de tolérance et la fonction g définit le critère d'arrêt des itérations.

3.2.3 Méthodes itératives hybrides de calcul d'éléments propres

- *Méthode d'Arnoldi avec Redémarrage Implicite (IRAM)*

Une autre variante de la méthode d'Arnoldi basée sur la technique de redémarrage est proposée par Sorensen [Sor92]. Cette version raffinée, appelée Méthode d'Arnoldi avec Redémarrage Implicite (IRAM), est une technique combinant la méthode QR avec décalage implicite [LaT00][Sha05] avec une factorisation Arnoldi. Elle peut être vue comme étant une forme tronquée de la méthode QR avec décalage implicite [LaT00][Sha05]. Elle permet d'appliquer implicitement au vecteur initial un polynôme en A (i.e. (P(A)).*z*). Il a été montré qu'IRAM conduit à de bonnes approximations des éléments de Ritz pour des matrices creuses de grande taille.

- *Méthode d'Arnoldi avec Redémarrage Explicite Multiple (MERAM)*

La méthode proposée par Emad, Petiton et Edjlali [EPE05] pour redémarrer la méthode d'Arnoldi est basée sur une combinaison des deux algorithmes RAM et ERAM décrits ci-dessus. Dans cette version, ni le paramètre m, ni le vecteur initial *z* ne sont fixés. Pour surmonter le problème de stockage dans la méthode d'Arnoldi, la méthode MERAM impose une contrainte sur la taille m du sous-espace. Plus précisément, on suppose que m appartient à l'intervalle discret $\mathcal{B}_m = [m_{inf}, m_{sup}]$. Les bornes m_{inf} et m_{sup} peuvent être choisies en fonction des ressources de stockage et de calcul disponibles et doivent satisfaire $m_{inf} \leq m_{sup} << N$. Soient $m_1 \leq \ldots \leq m_\ell$ avec $m_i \in \mathcal{B}_m$ $(1 \leq i \leq \ell)$, $\mathcal{M} = \{m_i : i=1\ldots\ell \text{ tq } m_1 \leq \ldots \leq m_\ell\}$ et $Z^\ell = [z_1, \ldots, z_\ell]$ la matrice des vecteurs de démarrage. L'algorithme de la méthode MERAM pour le calcul de s ($\leq m_1$) éléments de Ritz de A est explicité dans la figure 1.7 où r_s^i est le vecteur dont les composantes sont les normes des résidus à la $i^{ème}$ itération.
Il est clair que le principe de cet algorithme consiste à utiliser plusieurs ERAMs. Il permet de mettre à jour un vecteur z_i d'un ERAM en prenant en compte les informations intéressantes sur les éléments propres obtenues par les autres ERAMs.

MERAM (Entrées : A, s, M, Z^{ℓ}, tol ; Sorties : r_s, Λ_m, X_m)

1. Démarrer. Choisir une matrice de démarrage Z^{ℓ} et un ensemble de tailles de sous espace $\mathcal{M} = \{m_i : i=1\ldots \ell \text{ tq } m_1 \leq \ldots \leq m_\ell\}$.
2. **Do** i=1, ℓ
   (a) Calculer une étape BAA (Entrées : A, s, m_i, z_i ; Sorties : r_s^i, Λ_{mi}, X_{mi}).
   (b) **IF** ($g(r_s^i) \leq$ tol) **THEN** arrêt.
   EndDo
3. Redémarrer. Mettre à jour les vecteurs $z_1, \ldots, z_\ell$ et aller à 2.

Figure 1.7 Algorithme MERAM

4. Conclusion

Contrairement au problème de calcul d'éléments propres pour lequel on ne peut utiliser que des méthodes itératives, les méthodes de résolution de systèmes linéaires peuvent être directes ou itératives. Concernant la résolution de systèmes linéaires, l'idée de base des méthodes directes est de transformer le système initial Ax = b, en un système équivalent, noté Tx=c, plus facile à résoudre. Le principe de base des méthodes itératives, quant à elles, consiste à construire une suite $x^{(0)}$, $x^{(1)}$, …, $x^{(k)}$… qui converge, sous certaines conditions, vers la solution exacte du système.

Les méthodes directes sont robustes et conviennent plutôt à des problèmes de taille modérée. Sur des problèmes de très grande taille, elles peuvent devenir prohibitives car trop gourmandes en termes de mémoire. Sur cette gamme de problèmes, les méthodes itératives constituent ainsi une alternative intéressante [DER92] [Wyr01]. En effet, elles n'affectent pas la matrice A, et permettent une exploitation totale de son caractère creux.

A travers notre description des principales méthodes itératives pour la résolution de systèmes linéaires et le calcul d'éléments propres, nous avons pu relever qu'elles reposent essentiellement sur le noyau du calcul du produit matrice-vecteur (PMV). Il faut préciser que dans le cadre de notre travail, les matrices étudiées sont creuses. Ainsi, dans le chapitre suivant nous présentons et détaillons les concepts de base du calcul creux, en particulier les structures utilisées pour le stockage (ou la compression) et le traitement de telles matrices.

CHAPITRE 2.

LE CALCUL CREUX : PRÉSENTATION GÉNÉRALE

1. Introduction

Dans ce chapitre, nous présentons un tour d'horizon sur les aspects basiques du calcul creux. De par son appellation, le calcul creux traite des matrices dont la plupart des éléments sont nuls. La distribution de ces éléments au sein de la matrice d'entrée définit sa structure. La section 2 est ainsi consacrée à la description des principales structures de matrices creuses qui peuvent être groupées en deux classes : les structures régulières et les structures non régulières. Cette section se termine par une étude statistique basée sur l'analyse d'un ensemble d'applications scientifiques permettant de relever les structures les plus fréquentes. Dans la section 3, nous décrivons les divers formats dits de compression ou de stockage associés aux diverses structures des matrices creuses. La section 4 est dévolue à la présentation de deux applications spécifiques traitant des matrices creuses et qui se ramènent à la résolution de systèmes d'équations linéaires ou au calcul d'éléments propres. La section 5 conclut le chapitre.

2. Structures de matrices creuses

2.1 Définitions générales

Soit A une matrice réelle d'ordre N (N est supposé assez élevé). Si le nombre d'éléments non nuls de A, noté NZ, est très petit (resp. grand), soit NZ = O(N) (resp. $O(N^2)$), alors A est dite *creuse* (resp. *dense*) [HME07].

Une matrice est donc dite creuse si elle contient beaucoup d'éléments nuls [DER92]. Par opposition, une matrice est dite dense si elle contient beaucoup d'éléments non nuls [Bik96].

On définit la densité comme étant la proportion des éléments non nuls par rapport au nombre total d'éléments dans la matrice [MKJ03]. Pour une application donnée traitant une matrice creuse, on peut réduire considérablement l'espace mémoire ainsi que le temps de calcul nécessaires en stockant uniquement les éléments non nuls et en évitant les opérations redondantes sur les éléments nuls [Bik96].

Que doit être la proportion minimale d'éléments nuls dans une matrice pour la qualifier de creuse et surtout nécessiter un traitement approprié ? En fait, ceci dépendra du calcul à effectuer, de la structure de la partie non nulle de la matrice et, éventuellement, de l'architecture de la machine cible sur laquelle sera implémentée l'application traitant la matrice en question [DER92] [OsZ83].

On distingue en pratique deux classes de structures de matrices creuses : les structures régulières et les structures irrégulières (ou non régulières).

2.2 Structures régulières

La structure de la matrice creuse est dite régulière si tous les éléments non nuls sont regroupés au sein de la matrice de telle manière qu'ils forment une zone limitée permettant de facilement repérer les éléments. Nous en passons certaines en revue dans ce qui suit.

2.2.1 Matrice triangulaire

Une matrice A est triangulaire supérieure (resp. inférieure) si $a_{ij} = 0$ pour tout $i > j$ (resp. $i < j$). Seule donc la moitié des éléments sont nuls (en fait $N(N-1)/2$ si la matrice est de taille N). On rencontre souvent de telles matrices lors de la résolution de systèmes linéaires par une méthode directe (Gauss et factorisations LU et de Cholesky) [DER92][OsZ83], ainsi que dans la méthode itérative du gradient conjugué pré-conditionné [GoV83][Jen80][Saa94].

2.2.2 Matrice bande constante

• Une matrice bande A de largeur $\lambda=2m+1$ est telle que m, appelé demi-largeur de bande, soit le plus petit entier vérifiant:

$$\forall\ i,j\ /\ |i-j| > m \text{ alors } a_{ij} = 0 :\ \ i,j = 1..N,\ m < \lfloor (N-1)/2 \rfloor$$

Dans le cas non symétrique, on définit la demi-largeur de bande inférieure (resp. supérieure) comme étant le plus petit entier m_L (resp. m_U) tel que $\forall\ a_{ij} = 0$, $i-j > m_L$ (resp. $j-i > m_U$). Dans ce cas, la largeur de bande est égale à $\lambda=m_L+m_U+1$.

• Une matrice est dite demi-bande lorsque $m_L = 0$ ou $m_U = 0$ [DER92].

• Une matrice diagonale est une matrice bande de largeur 1.

• Une matrice tridiagonale est une matrice bande de largeur 3.

On rencontre souvent les matrices bandes lors de la résolution de systèmes linéaires ainsi que le produit matrice-vecteur [Saa94]. En fait, on peut toujours ramener une matrice creuse quelconque à une matrice bande par l'algorithme de Cuthill-Mackee [DER92]. Notons que la bande ici peut contenir des éléments nuls. Ajoutons que l'élimination de Gauss sans permutation (dite version à pivot standard, par opposition à la version à pivot maximal partiel qui procède à des permutations des lignes) préserve la structure bande.

2.2.3 Matrice de Hessenberg

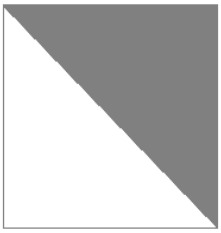
Une matrice est dite de Hessenberg supérieure (resp. inférieure) si $a_{ij} = 0$ pour tout $i > j+1$ (resp. pour tout $i < j+1$). Une telle structure est représentée dans la Figure 2.1.(c).

2.2.4 Matrice bloc

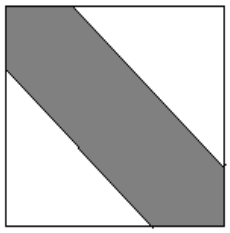
Soient deux ensembles $\{m_1,..,m_p\}$ et $\{n_1,..,n_q\}$ d'entiers positifs. On définit les matrices:

$$A_{ij} \in \mathbb{R}^{m_i \times n_j} \qquad i = 1, .., p \qquad j = 1,..,q$$

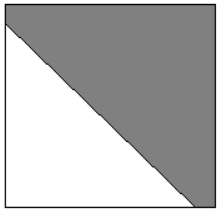
On peut noter A sous la forme dite bloc $A = [A_{ij}]$, i=1…p, j=1…q. A est ainsi une matrice bloc p×q et A_{ij} correspond au bloc (i,j). A titre d'exemple, la matrice de la Figure 2.1.(d) (resp. Figure 2.1.(e)) représente une matrice triangulaire bloc (resp. tridiagonale bloc). D'autres généralisations des définitions d'une matrice triangulaire, d'une matrice bande constante et d'une matrice bande variable sont possibles par simple substitution de a_{ij} par A_{ij}. Chaque bloc est ainsi traité comme étant un bloc dense. On rencontre souvent les matrices bloc lors de la discrétisation d'équations aux dérivées partielles par les méthodes des différences finies ou des éléments finis. On les rencontre aussi dans la méthode directe de Gauss [Bik96][Saa94]. La figure suivante illustre les diverses structures qui viennent d'être décrites.



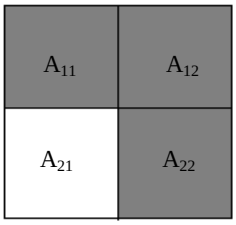
(a**) matrice triangulaire**



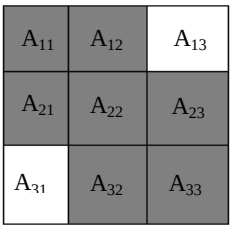
(b) matrice bande constante



(c) matrice Hessenberg



(d) matrice triangulaire bloc



(e) matrice tridiagonale bloc

Figure 2.1 Structures régulières de matrices creuses

2.3 Structures irrégulières

Une matrice possède une structure irrégulière si les éléments sont distribués de manière non uniforme dans la matrice. On peut citer les structures suivantes.

2.3.1 Matrice bande variable

Soit A une matrice d'ordre N et soient ℓ_i et u_j pour $1 \le i \le N$, $1 \le j \le N$ définis par :

$$\ell_i = \text{i-min}\{j / a_{ij} \neq 0\} \; ; \; u_j = \text{j-min}\{i / a_{ij} \neq 0\}$$

ℓ_i et u_j indiquent respectivement la (largeur de) demi-bande inférieure et supérieure dans la ligne i et la colonne j : $a_{ij} \neq 0 \Rightarrow (-u_j \le i-j \le \ell_i)$

La forme bande variable d'une matrice est définie par les vecteurs $\underline{u} = (u_i)$ et $\underline{\ell} = (\ell_i)$. La matrice de la Figure 2.2.(a) représente un exemple de structure bande variable. Bien que des éléments nuls puissent apparaître au sein de la bande variable, la structure non nulle décrite par la bande variable peut être plus précise que celle décrite par une bande constante. Ainsi, les méthodes manipulant des matrices à bande variable constituent une bonne approche pour exploiter les éléments nuls dans une matrice [DER92]. Il est connu que l'élimination de Gauss standard (sans permutation) préserve la structure bande variable [Bik96][DER92] [Jen80].

2.3.2 Matrice quelconque

Une matrice est dite (de structure) quelconque si les éléments non nuls sont distribués de manière tout à fait quelconque (voir Figure 2.2.(b)).

(a) matrice bande variable **(b) matrice quelconque**

Figure 2.2 Structures irrégulières de matrices creuses

Notons que dans toutes les structures de matrices creuses présentées ci-dessus, la zone représentant la partie dense peut contenir des zéros, toutefois en très faible proportion.

2.4 Matrices creuses et applications scientifiques - Etude statistique

Afin d'avoir une idée précise sur les structures creuses les plus fréquemment rencontrées dans les applications scientifiques, nous avons effectué une recherche au sein du site Matrix Market [Mat07]. Ce dernier est un service Web fournissant des informations sur un ensemble de près de 500 matrices creuses provenant de diverses applications, ainsi que des outils et modes de génération de matrices. Cet ensemble comporte les matrices les plus référencées dans la littérature et constitue un riche banc d'essai. Nous en avons retenu quatre collections de matrices : (i) Harwell-Boeing [HaB98], (ii) SPARSKIT Collection [Saa94][SpK00], (iii) Independant Set and Generators [ISG00] et (iv) NEP Collection [NEP98]. Sur 70 applications répertoriées, nous avons ainsi abouti aux résultats présentés dans la Table 2.1 suivante.

N°	**Application**	**Structure**	**Nombre de Matrices**	**Ordre de la matrice**	**Densité (%)**	**Type**	**Collection**
1	ACOUST	BND	4	841	0.57-0.59	S	Harwell-Boeing
2	AIRTFC	QC	1	2873	0.01	S	--
3	ASTROPH	BND	2	180-765	4.16-8.2	NS	--
4	BCSPWR	QC	10	39-5300	0.07-100	S	--
5	BCSSTRUC1	QC / BND / D	QC:9 / BND: 9 / D:8	48-2003	0.06-100	S	--
6	BCSSTRUC2	BND	5	1806-11948	0.1-1.94	S	--
7	BCSSTRUC3	QC / BND / D	QC:3/ BND:4 / D:7	138-15439	0.006-3.65	S	--
8	BCSSTRUC4	BND / D	BND: 4 / D:1	1224-4410	0.05-3.74	S	--
9	BCSSTRUC5	QC / BND	QC:2 / BND: 3	8737-44609	0.04-0.39	S	--
10	BCSSTRUC6	QC / BND	QC: 1 / BND: 1	2132-3345	0.2-0.32	S	--
11	CANNES	QC	18	24-1054	1.08-27.77	S	--
12	CEGB	QC	4	2802-3306	0.68-3.77	S	--
13	CHEMIMP	QC	5	59-425	0.69-7.78	NS	--
14	CHEMIWEST	QC	11	67-2021	0.17-6.54	NS	--
15	CIRPHYS	BND	1	991	0.61	NS	--
16	COUNTERX	QC	3	9-10	61.72-76	NS	--
17	DWT	QC / BND	QC: 20 / BND:10	59-2680	0.34-9.37	S	--
18	ECONAUS	QC	2	1258-2529	0.48-1.4	NS	--
19	ECONIEA	QC	9	Rectangulaire-496	5.07-21.1	NS	--
20	FACSIMELE	QC	10	183-760	0.47-3.19	NS	--
21	GEMAT	QC	3	4929	0.089-0.13	NS	--
22	GRENOBLE	QC / BND	QC:3 / BND: 4	115-1107	0.46-3.18	NS	--
23	JAGMESH	QC	9	939-1440	0.45-0.71	S	--
24	LANPRO	QC / BND	QC:1 / BND:6	100 - 960	0.45-5.94	S	--
25	LAPLACE	BND	1	900	0.95	S	--
26	LAPU	QC	1	25	27.04	S	--
27	LNS	QC / BND	QC: 3 / BND:3	131 - 3937	0.16-3.12	NS	--
28	LOCKHEED	QC	4	700 - 3491	1.31-4.52	S	--
29	LSHAPE	QC	12	265 - 3466	0.19-2.49	S	--

N°	**Application**	**Structure**	**Nombre de Matrices**	**Ordre de la matrice**	**Densité (%)**	**Type**	**Collection**
30	LSQ	QC	4	Rectangulaire	0.65-1.43	NS	Harwell-Boeing
31	MANTEUFFEL	QC	1	5976	0.62	S	--
32	NCENG	QC	1	434	2.22	NS	--
33	NUCL	QC	3	261 - 1374	0.45-2.2	NS	--
34	OILGEN	QC / BND	QC:2 / BND:1	886 - 2205	0.29-0.76	NS	--
35	PLATZ	QC	4	362 - 1919	0.26-4.41	2S / 2 NS	--
36	PROES	QC / BND	QC: 2 / BND: 1	30 - 1224	0.64-20	NS	--
37	PSADMIT	QC	4	494 - 1138	0.31-0.69	S	--
38	PSMIGR	QC	3	3140	5.47-5.5	NS	--
39	SAYLOR	BND	3	238 - 3564	0.17-1.99	NS	--
40	STEAM	QC	3	80 - 600	1.57-4.9	NS	--
41	SMTAPE	QC / BND	QC: 32 / BND (S): 3	32 - 958	0.38-14.35	5S / 30NS	--
42	SHERMAN	BND	5	1000 - 5005	0.07-1.97	NS	--
43	WATT	BND	2	1856 - 1865	0.32-0.33	NS	--
44	DRIVCAV	QC	QC: 30	236 - 17281	0.18-10.49	1S / 29 NS	SPARSKIT Collection
45	DRIVCAVOLD	QC	QC: 26	317 - 4562	0.63-7.27	NS	--
46	FIDAP	QC	QC: 46	27 - 22294	0.05-38.27	NS	--
47	TOKAMAK	QC	QC: 5	300 - 5940	0.23-3.5	NS	--
48	HAMM	QC	3	2395 - 17758	0.03-0.23	NS	Independent Sets and Generators
49	CYLSHELL	QC / BND	QC:1 / BND: 8	5357 - 90449	0.02-0.87	S	--
50	QCD	QC	14	3072 - 49152	0.08-1.26	NS	--
51	AIRFOIL	QC	1	23560	0.08	NS	NEP Collection
52	BFWAVE	QC	6	62 - 782	0.97-11.7	3 S / 3 NS	--
53	BRUSSEL	QC	11	200 - 3200	0.18-2.8	NS	--
54	CHUCK	QC	3	104 - 656	0.9-9.17	NS	--

N°	**Application**	**Structure**	**Nombre de Matrices**	**Ordre de la matrice**	**Densité (%)**	**Type**	**Collection**
55	CRYSTAL	QC	2	2500 - 10000	0.05-0.19	NS	NEP Collection
56	DWAVE	QC	4	512 - 8192	0.06-0.95	NS	--
57	GEDNAY	QC	2	961	0.36-1.14	S	--
58	H2PLUS	BND	2	324 - 2534	7.21-25.46	S	--
59	MHD	BND	8	416 - 4800	0.12-4.94	2 S / 6 NS	--
60	OLMSTEAD	BND	5	100 - 5000	0.08-3.96	NS	--
61	QUEBEC	QC	3	768 - 1484	0.43-0.49	NS	--
62	ROBOTICS	QC	2	480	7.41	NS	--
63	STOCH	BND	1	163	3.52	NS	--
64	TUBULAR	BND	2	100 - 1000	3.96-3.99	NS	--
65	MATPDE	QC	3	225 - 2961	0.16-1.63	NS	--
66	MVMBWM	QC	2	200 - 2000	0.19-1.99	NS	--
67	MVMMCD	BND	6	961	0.1	NS	--
68	MVMODE	QC / D	QC: 1 / D(S): 1	400	0.24-0.25	1S / 1NS	--
69	MVMRWK	BND	3	136 - 5151	0.75-2.58	NS	--
70	MVMTLS	QC	5	90 - 4000	0.05-21.55	NS	--

Table 2.1 Répartition des structures creuses par application

Légendes

BND : matrice bande ; **QC** : matrice quelconque ;**D** : matrice diagonale, **S** : symétrique ; **NS** : non symétrique.

L'analyse des données a permis d'établir les statistiques illustrées dans les Figures 2.3 à 2.6 et les Tables 2.2 et 2.3.

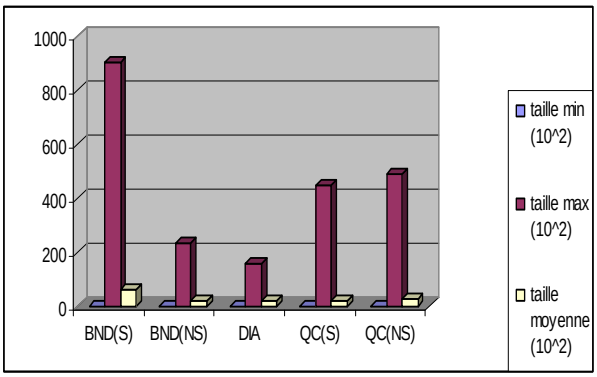


Figure 2.3 Statistiques sur les tailles des matrices de Matrix-Market

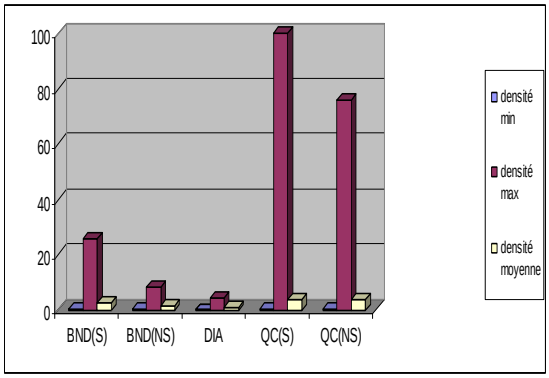


Figure 2.4 Statistiques sur les densités des matrices de Matrix-Market

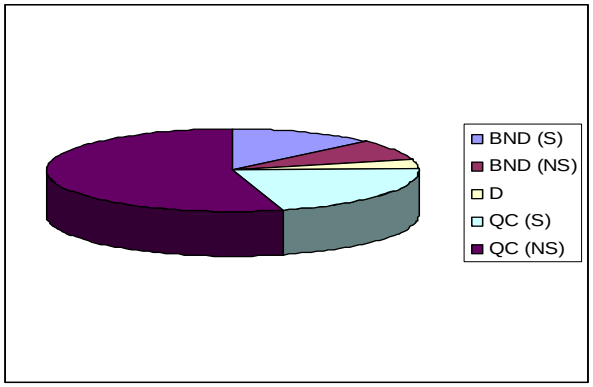


Figure 2.5 Classement des matrices selon leurs structures

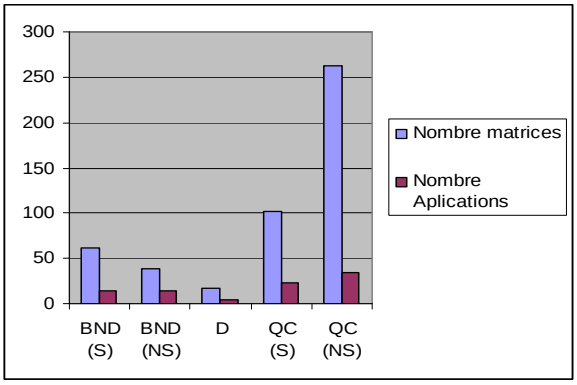


Figure 2.6 Représentation du nombre de matrices et du nombre d'applications pour chaque type de structure

Structure	Type	Nombre de matrices	Nombre d'applications	Total des applications	Pourcentage des applications
BND	S	62	14	28	31.11
	NS	39	14		
D		17	4	4	4.45
QC	S	102	23	58	64.44
	NS	263	35		

Table 2.2 Statistiques sur les structures creuses

Nous constatons ainsi que la structure la plus fréquente est la structure (irrégulière) quelconque (365 matrices sur 483, soit 75.57%), suivie par deux structures régulières (118 matrices sur 483, soit 24.43%) i.e. la structure diagonale (17 matrices, soit 3.52%) et la structure bande (101 matrices, soit 20.91%). Il est à remarquer que pour chacune de ces trois structures, les cas symétrique et non symétrique sont rencontrés.

Dans la Table 2.3, nous présentons des statistiques concernant la distribution des éléments non nuls dans les matrices creuses. Les matrices échantillons ont été également sélectionnées à partir de Matrix Market [Mat07].

Application	**Domaine**	**N**	**Moy**	**$10^4 * R$**
AIRTFC	Modèle de Contrôle du Trafic Aérien	2873	0.46	1.6
ASTROPH	Transfert radiatif non linéaire et équilibre statique en Astrophysique [JeS06][Reg59]	180	15	83.3
			32	400
BCSPWR	Modèles de réseau de puissance [JeC08]	39	3.4	871.8
		49	3.4	693.9
		118	4	339
		274	5.9	215.3
		443	3.7	83.5
		1454	3.6	24.8
BCSSTRUC1	BCS Matrices de l'Ingénierie des Structures (application de calcul de valeurs propres) [ISE07]	48	8.3	1729.2
		66	66	1
		112	5.7	508.9
		132	28	2121.2
		153	16	1045.7
		420	19	452.4
		420	19	452.4
		1074	12	111.7
		1083	17	157

Table 2.3 Quelques statistiques sur les matrices de Matrix Market

Légendes

N: Taille **; Moy:** Moyenne du nombre d'éléments non nuls par ligne **; R**=Moy/N

Nous remarquons que le ratio R varie énormément d'une matrice à une autre. De ce fait, la distribution des éléments non nuls est généralement non uniforme. Ce type de matrices couvre les cas les plus fréquents rencontrés dans les applications du monde réel.

3. Formats de compression

3.1 Formats pour structures régulières

La matrice creuse A ayant N lignes, soit NZ le nombre de ses éléments non nuls. Pour chaque type de structure, nous présentons ci-dessous les modes de stockage adéquats.

3.1.1 Matrice triangulaire

Pour stocker une matrice triangulaire, on peut utiliser le stockage MSR (Modified Sparse Row) [Saa94]. La structure de données correspondante consiste en un vecteur de réels $\mathcal{A}$ et un vecteur d'entiers JA. Les N premières positions dans $\mathcal{A}$ contiennent les éléments de la diagonale de la matrice. La position N+1 du vecteur $\mathcal{A}$ n'est pas utilisée. Commençant à partir de la position N+2, les éléments non nuls de $\mathcal{A}$ sont stockés par ligne, en excluant les éléments de la diagonale. A chaque élément $\mathcal{A}$(k) correspond l'entier JA(k) qui est l'indice de la colonne de $\mathcal{A}$(k) dans la matrice A. Les N+1 premières positions dans JA contiennent les pointeurs sur le début de chaque ligne dans $\mathcal{A}$ et JA.

	1	2	3	4	5	6
1	1.	9.		21.	10.	12.
2		2.	19.			
3			3.	15.	14.	22.
4				4.	20.	
5					5.	17.
6						6.

Figure 2.7 Matrice Triangulaire stockée dans un tableau 2D

$\mathcal{A}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1.	2.	3.	4.	5.	6.		9.	21.	10.	12.	19.	15.	14.	22.	20.	17.

JA

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
8	12	13	16	17	18	18	2	4	5	6	3	4	5	6	5	6

Figure 2.8 Matrice A stockée selon le format MSR

Notons qu'il y a aussi le format orienté colonne homologue à MSR et appelé format MSC (Modified Sparse Column) [Jen80][Saa94].

3.1.2 Matrice bande constante

Le mode de stockage le plus utilisé pour ce type de matrices est le BND (Banded Linpack Format). Il s'agit de stocker les éléments non nuls de la matrice bande A dans un tableau rectangulaire ABD tel que les éléments de la $i^{ème}$ ligne de A soient stockés dans la $i^{ème}$ ligne de ABD. On a aussi besoin de connaître le nombre m_L de diagonales de la bande au-dessous de la diagonale principale ainsi que le nombre m_U de diagonales au-dessus de la diagonale principale. Ainsi, la largeur de bande de A qui est $\lambda=m_L+m_U+1$ représente le nombre minimal de colonnes dans ABD. On parle dans ce cas de format de compression BND par colonne parce que les diagonales constituant la bande sont stockées dans ABD colonne par colonne. Il existe aussi une version BND par ligne consistant à stocker les diagonales ligne par ligne dans le tableau ABD.

	1	2	3	4	5	6
1	1.	21.	16.			
2	9.	2.	19.	15.		
3		23.	3.	0.	27.	
4			20.	4.	22.	12.
5				10.	5.	25.
6					11.	6.

Figure 2.9 Matrice bande dans un tableau 2D

	1	2	3	4
1		1.	21.	16.
2	9.	2.	19.	15.
3	23.	3.	0.	27.
4	20.	4.	22.	12.
5	10.	5.	25.	
6	11.	6.		

Figure 2.10 Tableau ABD - BND par colonne

	1	2	3	4	5	6
1			16.	15.	27.	12.
2		21.	19.	0.	22.	25.
3	1.	2.	3.	4.	5.	6.
4	9.	23.	20.	10.	11.	

Figure 2.11 Tableau ABD-BND par ligne

Précisons que si la matrice bande est symétrique, on se limite au stockage des éléments a_{ij} tels que $j \leq i$ [Jen80].

Dans beaucoup d'applications, les matrices bandes contiennent en général peu de diagonales. Pour stocker de telles matrices, on utilise le format DIA (Diagonal) plus adéquat. Le principe consiste à stocker les diagonales dans un tableau rectangulaire DIAG (1:N,1:NDIAG) où NDIAG est le nombre de diagonales. De plus, on a besoin de connaître le début de chaque diagonale. Ces pointeurs seront stockés dans un tableau IOFF (1:NDIAG). Ainsi, la position (i,k) du tableau DIAG contient l'élément $a_{i,i+ioff(k)}$ de la matrice originale.

Un autre mode de stockage pour matrices bandes et spécialement pour l'implémentation d'algorithmes d'algèbre linéaire sur machines parallèles, est le format JDS (Jagged Diagonal Storage). Son principe consiste, d'abord, à décaler les éléments non nuls vers la gauche,

permuter les lignes de telle manière qu'elles soient ordonnées dans l'ordre décroissant du nombre d'éléments non nuls. Les éléments non nuls sont par la suite stockés dans un vecteur A, colonne par colonne. Leurs indices de colonne dans la matrice originale sont stockés dans un vecteur appelé JA. Un vecteur IA doit contenir les pointeurs sur les débuts des colonnes [Saa94].

3.1.3 Matrice Hessenberg

Une matrice Hessenberg supérieure A d'ordre N peut être stockée par ligne. On aura ainsi besoin d'un vecteur de taille N(N+3)/2-1 où l'élément a_{ij} sera stocké à la position i(2N+3-i)/2-N+j-1 [Jen80].

3.1.4 Matrice bloc

Il existe plusieurs variantes dans les formats utilisés pour stocker les matrices ayant une structure bloc. On peut citer le format BSR (Block Sparse Row) dont le principe est le suivant. Soit NBLK la dimension de chaque bloc, NZR le nombre de blocs non nuls dans la matrice et NR la dimension bloc de cette matrice i.e. NR = N/NBLK. Pour le format BSR, on a besoin de trois tableaux. Le premier est un tableau de réels à trois dimensions $\mathcal{A}$(1:NZR,1:NBLK,1:NBLK) qui contient les blocs non nuls listés ligne par ligne. Un vecteur d'entiers JA(1:NZR) est associé à $\mathcal{A}$. Il contient les indices des colonnes dans la matrice originale des éléments (1,1) des blocs non nuls. Enfin, un vecteur de pointeurs IA(1:NR+1) pointe sur le début de chaque ligne de blocs dans $\mathcal{A}$ et JA [Saa94].

	1	2	3	4	5	6
1	1.	10.			19.	14.
2	9.	2.			20.	11.
3			3.	15.		
4			23.	4.		
5					5.	22.
6					21.	6.

Figure 2.12 Matrice bloc représentée par un tableau 2D

$\mathcal{A}$

1		2		3		4	
1.	10.	19.	14.	3.	15.	5.	22.
9.	2.	20.	11.	23.	4.	21.	6.

JA

1	2	3	4
1	5	3	5

IA

1	2	3	4
1	3	4	5

Figure 2.13 Matrice bloc stockée selon le format BSR

3.2 Formats pour structures irrégulières

3.2.1 Matrice bande variable

Si la matrice à bande variable est symétrique, on utilise le format SSK (Symmetric Skyline). Il s'agit ici de stocker uniquement la partie triangulaire inférieure. C'est une collection de lignes dont la longueur est variable. Une méthode simple utilisée pour stocker une matrice à bande variable est de placer toutes les lignes dans l'ordre 1 à N dans un vecteur de réels $\mathcal{A}$ et d'utiliser, de plus, un vecteur d'entiers qui va contenir les pointeurs sur les débuts des lignes dans $\mathcal{A}$. Les indices de colonne des éléments non nuls stockés dans $\mathcal{A}$ peuvent être déduits facilement. Précisons qu'il existe plusieurs variantes de ce format. Par exemple, on peut utiliser des pointeurs sur les éléments diagonaux au lieu que ce soit sur le premier élément de chaque ligne.

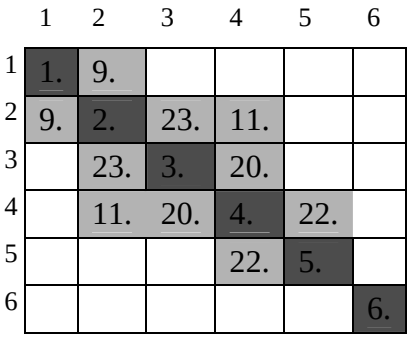


Figure 2.14 Matrice bande variable symétrique stockée dans un tableau 2D

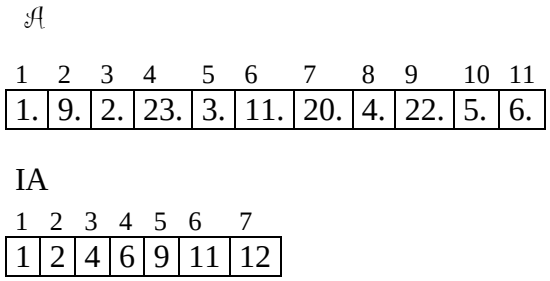


Figure 2.15 Matrice bande variable symétrique stockée selon le format SSK

Dans le cas où la matrice est non symétrique, on utilise le format NSK (Non symmetric Skyline). Son principe se base sur le fait qu'une matrice à bande variable non symétrique est constituée de deux parties : la partie inférieure qui est stockée selon le format SSK (standard) et la partie supérieure stockée selon le format SSK orienté colonne. Plusieurs modes peuvent être utilisés pour regrouper les deux parties ainsi stockées. Une possibilité consiste à utiliser deux tableaux séparés AL et AU pour la partie inférieure et supérieure, respectivement avec les éléments diagonaux dans la partie supérieure. Les structures de données utilisées pour chacune des deux parties sont similaires à celles utilisées dans le cas du format SSK [Saa94].

3.2.2 Matrice quelconque

Pour une matrice creuse quelconque, il existe une variété de modes de stockage. Citons les formats CSR (Compressed Sparse Row), CSC (Compressed Sparse Column), LNK (Linked List), SSS (Symmetric Sparse Skyline), USS (Unsymmetric Sparse Skyline), COO (Coordinate) … [Bik96][DER92][GoV83][Jen80][OsZ83][Saa94]. Nous détaillons ci-après ces divers formats.

(a) La structure de base du format CSR est constituée par trois tableaux [EHM05][EHM06] :

- Un tableau $\mathcal{A}$ de réels contenant les coefficients a_{ij} stockées ligne par ligne, de la ligne 1 à la ligne N. La longueur de $\mathcal{A}$ est NZ.
- Un tableau JA d'entiers contenant les indices de colonne des éléments a_{ij} déjà stockés dans le tableau $\mathcal{A}$. La longueur de JA est NZ.
- Un tableau IA d'entiers contenant les pointeurs sur le début de chaque ligne dans les deux tableaux $\mathcal{A}$ et JA. Ainsi, IA(i) donne la position dans $\mathcal{A}$ et JA où commence la $i^{ème}$ ligne de A.

	1	2	3	4	5	6
1	1.			21.		
2		2.	19.	15.		
3	9.		3.		14.	22.
4		23.		4.		
5	10.		20.		5.	
6		11.				6.

Figure 2.16 Matrice quelconque A représentée par un tableau 2D

$\mathcal{A}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1.	21.	2.	19.	15.	9.	3.	14.	22.	23.	4.	10.	20.	5.	11.	6.

JA

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	4	2	3	4	1	3	5	6	2	4	1	3	5	2	6

IA

1	2	3	4	5	6	7
1	3	6	10	12	15	17

Figure 2.17 Matrice A stockée selon le format CSR

L'ordre des éléments non nuls dans la même ligne est sans importance. Une variante de ce mode consiste à trier les éléments dans chaque ligne tels que leurs indices de colonne soient dans l'ordre croissant.

(b) Le format CSC est similaire au format CSR sauf qu'au lieu des lignes, ce sont les colonnes qui sont stockées.

(c) Les formats SSS et USS constituent une extension du format CSR. Dans le premier qui est symétrique, on utilise les tableaux suivants: DIAG pour stocker la diagonale, AL, JAL et IAL

stockent la partie inférieure stricte dans le format CSR, et AU stocke la partie supérieure stricte dans le format CSC. Dans USS qui est non symétrique, au lieu d'utiliser AU uniquement, la partie supérieure stricte est stockée dans les trois tableaux AU, JAU et IAU selon le format CSC.

(d) Le format LNK (LiNKed list) correspond à l'une des plus anciennes structures de données utilisées pour le traitement de matrices creuses. Il est constitué de quatre tableaux : $\mathcal{A}$, JCOL, LINK et JSTART. Les tableaux $\mathcal{A}$ et JCOL contiennent les éléments non nuls et leurs indices de colonne respectivement. Le tableau d'entiers LINK est généralement un tableau de pointeurs de lien : LINK(k) pointe sur la position de l'élément non nul qui suit l'élément ($\mathcal{A}$(k), JCOL(k)) dans la même ligne. Notons que l'ordre des éléments dans chaque ligne n'est pas important. Si LINK(k) est nul, alors il n'y a plus d'élément suivant i.e. ($\mathcal{A}$(k), JCOL(k)) est le dernier élément de la ligne. Finalement, ISTART pointe sur le premier élément de chaque ligne dans LINK. Ainsi, k=ISTART(1) pointe sur le premier élément de la première ligne, dans les tableaux $\mathcal{A}$ et JCOL. ISTART(2) pointe sur le second élément, etc. Comme convention, ISTART(i)=0, signifie que la $i^{ème}$ ligne est vide.

(e) Le format COO est certainement le mode de stockage le plus simple. Il est constitué de trois tableaux : un tableau de réels de taille NZ contenant les éléments non nuls de A, un tableau d'entiers contenant leurs indices de ligne et un autre tableau d'entiers contenant leurs indices de colonne. L'ordre des éléments non nuls dans le tableau n'est pas important.

	1	2	3	4	5	6
1	1.			21.		
2		2.	19.	15.		
3	9.		3.		14.	22.
4		23.		4.		
5	10.		20.		5.	
6		11.				6.

Figure 2.18 Matrice quelconque représentée par un tableau 2D

$\mathcal{A}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1.	21.	2.	19.	15.	9.	3.	14.	22.	23.	4.	10.	20.	5.	11.	6.

IA

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	2	2	3	3	3	3	4	4	5	5	5	6	6

JA

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	4	2	3	4	1	3	5	6	2	4	1	3	5	2	6

Figure 2.19 Matrice stockée selon le format COO

En terme d'espace mémoire, ce format n'est pas très efficace puisqu'il nécessite le stockage de 3NZ données (NZ réels et 2NZ entiers). Néanmoins, il est très utilisé à cause de sa simplicité. La version modifiée du format COO utilise un tableau d'entiers contenant les valeurs (i-1)N+j pour les éléments non nuls a_{ij}. Bien que ce format soit moins coûteux en espace mémoire, il présente deux inconvénients. D'abord, il nécessite des calculs supplémentaires pour déterminer les indices de ligne et de colonne originaux. De plus, pour des matrices de très grande taille, il peut induire des overflows à cause des entiers élevés (de l'ordre de N^2) qu'on est amené à manipuler dans les calculs.

3.3 Etude comparative

Dans la Table 2.4, nous présentons une récapitulation de l'étude effectuée sur les différentes structures de matrices creuses, les applications traitant chaque type de structure ainsi que les formats de compression associés à ces structures. Nous donnons également à travers cette table, une idée sur la complexité spatiale (espace mémoire) utilisée pour les structures de données correspondant à chaque format de compression et ce, en nombre de réels et en nombre d'entiers.
La même table contient aussi, pour chaque format, le nombre d'accès indirects effectués par l'algorithme du produit matrice-vecteur creux. Nous distinguons le nombre d'indirections simples et le nombre de double indirections (voir chapitre 4, section 2).

L'analyse de la Table 2.4 permet de déduire ce qui suit.

- Les formats CSR, CSC, COO et LNK sont des formats de compression généralistes i.e. pouvant être utilisés pour n'importe quelle structure de matrice. Ils ont tous la même propriété de stocker uniquement les éléments non nuls, nécessitant ainsi, un volume d'espace mémoire de NZ réels. Néanmoins, ils diffèrent entre eux par l'espace nécessaire pour le stockage des coordonnées (indices de ligne et de colonne) de ces éléments non nuls. En effet, les formats CSR, CSC et USS en utilisent moins i.e. NZ+N entiers, alors que les formats COO et LNK utilisent plus i.e. 2NZ (resp. 2NZ+N) entiers pour COO (resp. LNK).
- Les formats généralistes diffèrent aussi par le nombre d'indirections et de doubles indirections effectuées par l'algorithme du produit matrice-vecteur creux. Le format COO qui utilise le minimum d'indirections (NZ) utilise néanmoins le maximum de doubles indirections (3NZ). Par opposition, les formats CSR et LNK utilisent le maximum d'indirections et le

minimum de doubles indirections. Dans le cas du format CSC, le nombre d'indirections est égal au nombre de doubles indirections i.e. 2NZ.

- Notons aussi, que certains couples de formats compressés constituent deux variantes différentes du même format. L'une est orientée lignes alors que l'autre est orientée colonnes (e.g les formats BND ligne et BND colonne, CSR et CSC, MSR et MSC, etc). Généralement, les structures de données utilisées par les deux variantes nécessitent le même volume d'espace mémoire. La différence entre deux variantes réside dans la disposition des éléments non nuls à l'intérieur de ces structures de données. Le choix entre une variante et une autre dépendra donc du type d'accès aux éléments (par ligne ou par colonne) adopté par l'application.

Format	Structure de la matrice	Application utilisant ce type de matrice	Complexité spatiale		Nombre d'accès indirects pour le produit matrice vecteur	
			Nombre de réels	Nombre d'entiers	Indirections simples	Doubles indirections
DNS	quelconque	toute application manipulant des matrices denses	N^2		N^2	--
BND Ligne	bande	résolution de systèmes linéaires	N(ML+MU+1)		2ML(2MU+ML+1)+4(ML+MU+1)(N-MU-ML-1) +(MU+2)(3MU+1)/2 . ML : largeur demi-bande inférieure . MU : largeur demi-bande supérieure	--
BND Colonne		--	N(ML+MU+1)		--	
CSR	quelconque		NZ	NZ+N	3NZ	NZ
CSC	--		NZ	NZ+N	2NZ	2NZ
MSR	triangulaire	méthodes du gradient conjugué préconditionné	(NZ+1)	NZ+1	3NZ	NZ
MSC	--	--	(NZ+1)	NZ+1	2NZ	2NZ
COO	quelconque		NZ	2NZ	NZ	3NZ
ELL Ligne	quelconque avec un nombre d'éléments non nuls par ligne limité		NDIAG$_*$N	NDIAG$_*$N	3N$_*$NDIAG . NDIAG : Nombre diagonales de la matrice	N*NDIAG
ELL Colonne	--		NDIAG$_*$N	NDIAG$_*$N	2N$_*$NDIAG	2NDIAG
DIA	formée par un nombre limité de diagonales	Produit matrice-vecteur sur superordinateurs	NDIAG$_*$N	NDIAG	3NZ	NZ
BSR	Bloc	discrétisation d'équations aux dérivées partielles	NZ	2NZ	3NZ	NZ
SSK	bande variable symétrique	Elimination de Gauss	(NZ+N)/2	N	3/2(NZ+5N)	NZ
NSK	bande variable non symétrique	--	NZ	2N	3NU+2NL+3N . NL : Nombre éléments non nuls dans la partie inférieure . NU : Nombre éléments non nuls dans la partie supérieure	NU+2NL+N

Format	Structure de la matrice	Application utilisant ce type de matrice	Complexité spatiale		Nombre d'accès indirects pour le produit matrice vecteur	
			Nombre de réels	Nombre d'entiers	Indirections simples	Doubles indirections
LNK	quelconque		NZ	2NZ+N	3NZ	NZ
SSS	symétrique		NZ	(NZ+N)/2	4N+3NL+2NU	NL+2NU
USS	quelconque		NZ	NZ+N	--	--

Table 2.4 Tableau récapitulatif des structures creuses

4. Applications creuses de grande taille

On rencontre des matrices creuses dans diverses applications réelles e.g. problèmes de simulation 2D/3D en mécanique des structures, dynamique des fluides, électromagnétique, thermodynamique, acoustique, infographie, robotique/cinématique, etc. Elles peuvent aussi être issues d'autres domaines tels que l'optimisation, la simulation de circuit, la modélisation économique et financière, la chimie théorique et quantique, la simulation de processus chimiques, les mathématiques et les statistiques, la recherche d'information, etc. [Bou02] [Cag00]. Nous présentons ci-après deux applications spécifiques. La première se situe dans le domaine de la recherche d'information et la seconde dans celui de la simulation 3D.

4.1 Matrice de Google

La puissance du moteur de recherche Google réside dans son algorithme Page Rank (rang des pages) [Mes06] qui évalue quantitativement l'importance de chaque page sur le Web. Il permet ainsi de les classer et présenter à l'utilisateur, suite à sa requête, les pages les plus pertinentes d'abord [LaM06][Tom03]. Dans [BrL06], Bryan et Leise exposent l'une des idées fondamentales se trouvant derrière le calcul du classement des pages Web effectué par Google. Ils expliquent comment ce dernier définit et évalue quantitativement l'importance d'une page donnée. Ce classement revient à une application d'algèbre linéaire, plus précisément, à un problème de calcul de vecteurs propres pour une matrice creuse d'ordre N pouvant être supérieur à 8.10^9. L'idée de base consiste à affecter une valeur quantitative (un score) correspondant à l'importance d'une page Web qui pourrait être déduite à partir des liens issus des autres pages Web vers cette même page.

Supposons que le Web en question contient N pages, chaque page étant indexée par un entier k, $1 \leq k \leq N$. Un exemple typique est illustré dans la Figure 2.20, dans laquelle un arc orienté de la page i vers la page j indique un lien de i vers j.

On dénotera par x_k le score d'importance de la page k dans le Web, sachant que x_k est non négatif et $x_j > x_k$ indique que la page j est plus importante que la page k. Si la page j contient N_j liens dont l'un est vers la page k, alors on augmentera le score de la page k de x_j/N_j. Dans cet arrangement, chaque page Web obtient le total d'un vote, pondéré par le score de cette page Web, qui est également partagé parmi tous les liens sortants.

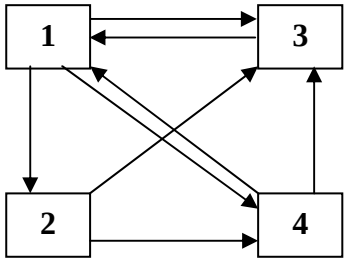


Figure 2.20 Un exemple de Web avec quatre pages

Pour un Web de N pages on note $L_k \subset \{1, 2..., N\}$ l'ensemble des pages ayant un lien vers la page k, c'est-à-dire que L_k est l'ensemble des liens entrants vers k. Pour chaque k on impose la relation

$$x_k = \sum_{j \in L_k} \frac{x_j}{N_j}$$

où N_j est le nombre des liens sortants de la page j.

A titre illustratif, appliquons cette approche aux quatre pages Web de la Figure 2.20. Pour la page 1, on a $x_1=x_3/1+x_4/2$, puisque les pages 3 et 4 ont des liens vers la page 1, et la page 3 contient un seul lien, alors que la page 4 contient deux liens. De manière similaire, on obtient $x_2=x_1/3$, $x_3=x_1/3 + x_2/2 + x_4/2$, et $x_4=x_1/3 + x_2/2$. Ces équations linéaires peuvent être écrites sous la forme $Ax = x$, où $x = [x_1\ x_2\ x_3\ x_4]^T$ et

$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

A, appelée *matrice de liens*, est creuse vu qu'en général le nombre de liens sortant d'une page Web vers d'autres pages est faible. On se ramène ainsi à un problème de calcul de vecteur propre. Il s'agit en fait de déterminer un vecteur propre x correspondant à la valeur propre 1 pour la matrice A.

Dans l'exemple ci-dessus, on calcule le *vecteur propre du score d'importance* [BrL06] tel que la somme de ses composantes soit égale à 1. On obtient ainsi les résultats suivants : $x_1 = \frac{12}{31} \approx 0.387$, $x_2 = \frac{4}{31} \approx 0.129$, $x_3 = \frac{9}{31} \approx 0.290$ et $x_4 = \frac{6}{31} \approx 0.194$.

Plus généralement, pour un Web quelconque, la matrice A doit admettre 1 comme valeur propre si le Web en question n'a aucun noeud puits i.e. une page sans liaison sortante. Pour un

Web de N pages, on se ramène à une matrice A où $a_{ij} = 1/N_j$ si la page j a un lien vers la page i, 0 sinon. La colonne j de A contient alors N_j entrées non nulles, chacune étant égale à $1/N_j$ et la somme des éléments d'une colonne est ainsi égale à 1. La matrice A pour un Web sans nœud puits est dite stochastique par colonne (tous ses éléments sont non négatifs et la somme des éléments d'une colonne est toujours égale à 1).
La résolution d'un tel problème en temps réel dans le cas d'un Web de plusieurs milliards de pages rentre dans le cadre du calcul de haute performance et nécessite un environnement matériel de puissance très élevée.

4.2 Simulation électromagnétique tridimensionnelle

La conception et la réalisation d'un dispositif électromagnétique sont basées sur un processus itératif de prototypage. La phase de conception consiste à définir un comportement et des plans de fabrication. Le prototypage permet de confirmer ou d'infirmer certaines hypothèses de la phase de conception et ce, en construisant et en testant une prévision du dispositif. Les résultats obtenus sont ensuite réinjectés dans la phase de conception afin d'établir de nouveaux plans de fabrication, c'est pourquoi on parle de processus itératif. Le dispositif est considéré comme réalisé lorsque le comportement du prototype est suffisamment proche de celui voulu lors de la phase de conception. De part son unicité, l'usinage d'un prototype tel que celui d'une machine électrique, est une opération longue et coûteuse. De plus, aucun nouveau plan de fabrication ne peut être établi avant la fin de la période d'essai. Dès lors, le processus itératif employé est économiquement onéreux [Cag00].
Il est possible de modéliser de nombreux dispositifs électromagnétiques à partir d'un modèle mathématique bidimensionnel. Cependant, certains dispositifs, du fait de la complexité de leur géométrie, nécessitent un modèle mathématique tridimensionnel. Le problème qui se pose alors est celui de l'explosion du nombre d'inconnues, problème encore aggravé par les phénomènes de nonlinéarité et d'évolutivité dans le temps.
Le problème de modélisation de dispositifs électromagnétiques en trois dimensions peut être formulé en utilisant deux types de formulations. Le premier correspond aux formulations indépendantes du temps, appelées formulations magnétostatiques. Le deuxième correspond aux formulations qui dépendent du temps, appelées formulations magnétodynamiques. La résolution analytique de ces formulations n'est généralement pas possible et le recours aux méthodes de discrétisation doit être envisagé. Une des méthodes de discrétisation qui peut être adoptée est celle des éléments finis et qui aboutit à un système d'équation linéaire creux et de

grande taille. Elle suppose tout d'abord la définition d'une structure mathématique du niveau continu permettant de représenter les champs et les potentiels (complexe de Rham), puis la définition d'une structure mathématique analogue au niveau discret (complexe de Whitney). Les éléments finis de Whitney [Cag00] sont des éléments finis conçus pour l'électro-magnétisme et qui comprennent les éléments finis nodaux, d'arêtes, de facettes et de volumes. Ils permettent de discrétiser les différentes formulations utilisées et d'obtenir les systèmes d'équations correspondants.

Ainsi, pour ce type de problème, il s'agit de résoudre un système d'équations aux dérivées partielles, dérivé des équations de Maxwell, par la méthode des éléments finis. Ce système d'équations présente un certain nombre de caractéristiques qui le rendent difficile à résoudre :

- Les formulations magnétostatiques et magnétodynamiques des équations de Maxwell sont des équations aux dérivées partielles elliptiques et paraboliques pour lesquelles la méthode des éléments finis est la plus appropriée
- La méthode de discrétisation fait appel aux éléments de Whitney faisant porter différentes grandeurs physiques par les noeuds, arêtes, facettes ou encore volume des éléments du maillage ; cette formalisation permet de garder les propriétés de continuité entre éléments du maillage
- Le maillage fait appel à différents volumes élémentaires : tétraèdres, prismes et hexaèdres. Cela offre de la souplesse au mailleur mais complique la construction du système et sa résolution
- Les matrices obtenues par discrétisation par la méthode de Galerkin sont creuses, symétriques et définies ou semi-définies positives ; ce genre de système est résolu par la méthode du gradient conjugué
- Les machines complexes doivent être modélisées en trois dimensions, ce qui induit un grand nombre d'inconnues
- La modélisation n'est pas linéaire, elle fait donc intervenir une boucle supplémentaire dans la résolution pour traiter cette non linéarité
- Il y a évolution dans le temps de la géométrie de la machine simulée, ce qui implique un remaillage éventuel et surtout une reconstruction du système d'équation.

Toutes ces caractéristiques produisent des problèmes creux de grande taille nécessitant un volume de calcul important.

5. Conclusion

Ce chapitre constitue un état de l'art sur les matrices creuses et leurs différents modes de stockage ainsi que sur les applications creuses. Nous avons présenté les différentes structures régulières et irrégulières de matrices creuses.

Afin d'avoir une idée précise sur les structures creuses les plus fréquemment rencontrées dans les applications scientifiques, nous avons effectué une recherche au sein du site Matrix Market [Mat07]. Cet ensemble comporte les matrices les plus référencées dans la littérature et constitue un riche banc d'essai.

Nous avons ainsi constaté que la structure la plus fréquente est la structure (irrégulière) quelconque, suivie par deux structures régulières i.e. la structure diagonale et la structure bande. Il est à remarquer que pour chacune de ces trois structures, les cas symétrique et non symétrique sont rencontrés. Nous avons de plus remarqué à travers notre étude que pour ces matrices, la distribution des éléments non nuls est généralement non uniforme. Ce type de matrices couvre les cas les plus fréquents rencontrés dans les applications du monde réel.

Pour chacune des structures régulières et irrégulières, nous avons détaillé une série de modes de stockage compressé possibles. Nous avons constaté que les formats de compression pouvaient être classés en deux catégories : des formats de compression généralistes et des formats de compression spécifiques. Les formats généralistes tels que le CSR, le CSC, le COO, le LNK, … sont ceux qui peuvent être utilisés pour n'importe quelle structure de matrice. Les formats spécifiques tels que le BND, le MSR, le DIA, le SSK, …ne peuvent être utilisés que pour des matrices ayant une structure particulière.

Les formats généralistes, diffèrent entre eux par la taille de l'espace mémoire utilisé par leurs structures de données. Quand ils sont utilisés par l'algorithme du produit matrice-vecteur creux, ces formats diffèrent aussi par le nombre d'indirections et de doubles indirections effectuées par l'algorithme.

Nous avons remarqué, de plus, que certains couples de formats compressés constituent deux variantes symétriques du même format : l'une est orientée lignes de la matrice et l'autre orientée colonnes (e.g les formats BND ligne et BND colonne, CSR et CSC, MSR et MSC, etc). Généralement, les structures de données utilisées par les deux variantes nécessitent le même volume d'espace mémoire. La différence entre deux variantes réside en fait dans la disposition des éléments non nuls au sein de ces structures de données. Le choix entre une variante et une autre dépendra donc du type d'accès aux éléments (par ligne ou par colonne) adopté par l'application.

Nous avons, en outre, vu à travers l'étude bibliographique effectuée, que plusieurs applications scientifiques [Amo07][BrL06][Cag00][HSS03] effectuent des calculs sur des matrices creuses de grandes tailles [Ase96][INR02][Pot97]. Nous avons présenté deux applications spécifiques, la première se situe dans le domaine de la recherche d'information et la deuxième dans celui de la simulation 3D. Nous avons ainsi constaté à travers ces applications que les calculs scientifiques creux se ramènent souvent à des problèmes d'algèbre linéaire, c'est-à-dire soit à la résolution de systèmes linéaires, soit au calcul de valeurs (et de vecteurs) propres [Cia82][DER92][Jen80][OsZ83]. Nous avons relevé aussi que ces problèmes de grande taille nécessitent un espace de stockage des données ainsi qu'un volume de calcul importants. Ainsi, l'utilisation d'environnements cibles parallèles et distribués offrant des performances élevées s'avère incontournable. C'est leur description qui constitue l'objet du chapitre suivant.

CHAPITRE 3.

SYSTÈMES PARALLÈLES ET DISTRIBUÉS

1. Introduction

Après avoir décrit dans le chapitre 1 les méthodes et algorithmes d'algèbre linéaire auxquels nous nous intéressons, puis dans le chapitre 2 les caractéristiques des matrices à traiter, nous passons dans ce chapitre à la description détaillée des diverses plates-formes matérielles cibles que nous pourrons retenir lors de l'implémentation de notre noyau basique de calcul. Nous passons ainsi en revue les principaux systèmes parallèles et distribués, partant des machines parallèles monolithiques jusqu'aux systèmes pair à pair (P2P), tout en nous arrêtant aux clusters, grilles de calcul et systèmes de calcul distribué à grande échelle (SDGE). La section 2 correspond à une brève présentation des architectures parallèles classiques ainsi que de la taxonomie due à Flynn. Dans la troisième section, nous décrivons l'architecture générale d'un cluster et présentons une classification générale de ses différents types. La section 4 est dévolue aux grilles de calcul et leur classification. La cinquième section est consacrée aux systèmes distribués à grande échelle (SDGE) ainsi qu'aux systèmes pair à pair (P2P). Le tout récent concept de Cloud Computing est décrit en section 6. Enfin, dans la section 7, nous nous intéressons à un SDGE spécifique, à savoir XtremWeb.

2. Taxonomie générale

		Flot de données	
		Simple	Multiple
Flot d'instructions	Simple	SISD (Von Neumann)	SIMD (Vectorielle et cellulaire)
	Multiple	MISD (pipeline)	MIMD (Multiprocesseur et Passage de Message)

Table 3.1 Taxonomie de Flynn pour des architectures parallèles

Les ordinateurs parallèles sont des machines ayant une architecture constituée de plusieurs processeurs identiques concourant tous au traitement d'une seule application. Il existe plusieurs types d'ordinateurs (ou de *processeurs*) parallèles, caractérisés, principalement, par différents modèles d'interconnexions inter-processeurs et entre les processeurs et la mémoire. La classification la plus populaire est celle de Flynn, qui catégorise les ordinateurs parallèles

(et séquentiels) selon le type d'organisation du flot de données et du flot d'instructions. Elle distingue ainsi en particulier les machines SIMD (Single Instruction stream-Multiple Data stream : flot d'instructions unique agissant sur des données multiples) et les machines MIMD (Multiple Instruction stream-Multiple Data stream : flot d'instructions et de données multiples). Dans la classe des machines MIMD, on distingue les machines à mémoire distribuée, les machines à mémoire partagée [CuS99][Mor94] et les machines à mémoire hybride [Sla06].

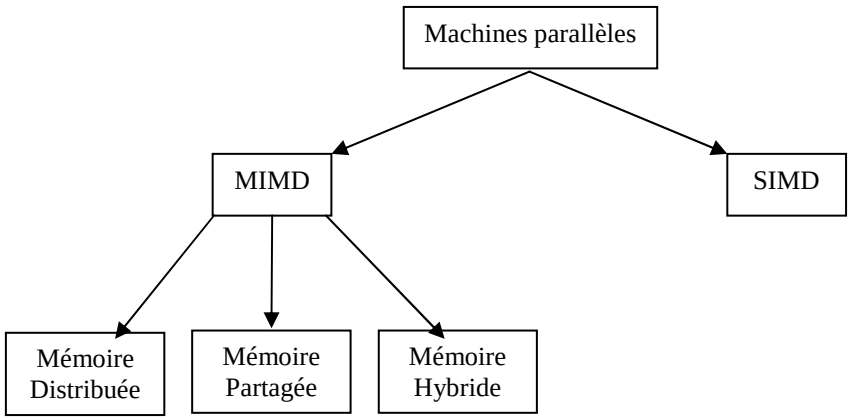


Figure 3.1 Taxonomie de haut niveau des architectures parallèles

Pour présenter les différentes classes de machines, nous rappelons les trois composants de base d'un processeur séquentiel standard.

Le composant appelé $\mathbb{S}$ est le *stockage* ou *mémoire* du processeur. Il stocke les instructions pour le programme à exécuter, les données traitées et les données produites en sortie. Dans les programmes utilisant des E/S, $\mathbb{S}$ sert aussi d'intermédiaire pour le mouvement des données provenant de et allant vers les mémoires secondaires.

Le composant appelé $\mathcal{C}$ est le *contrôleur* ou CPU. Sa tâche est de chercher les instructions dans le composant stockage et de les décoder. Le résultat de ce décodage est un signal de bas niveau envoyé vers le troisième composant, appelé P, le *processeur*. Il reçoit les signaux à partir du contrôleur, cherche les données, les stocke dans le composant stockage $\mathbb{S}$ et les traite selon l'ordre reçu par le contrôleur. Les différentes classes d'architectures parallèles peuvent être caractérisées par différents arrangements de ces trois composants.

2.1 Machines SIMD

L'idée de base d'une machine SIMD est que le contrôleur pilote tout un tableau d'unités processeur/mémoire au lieu d'un processeur unique.

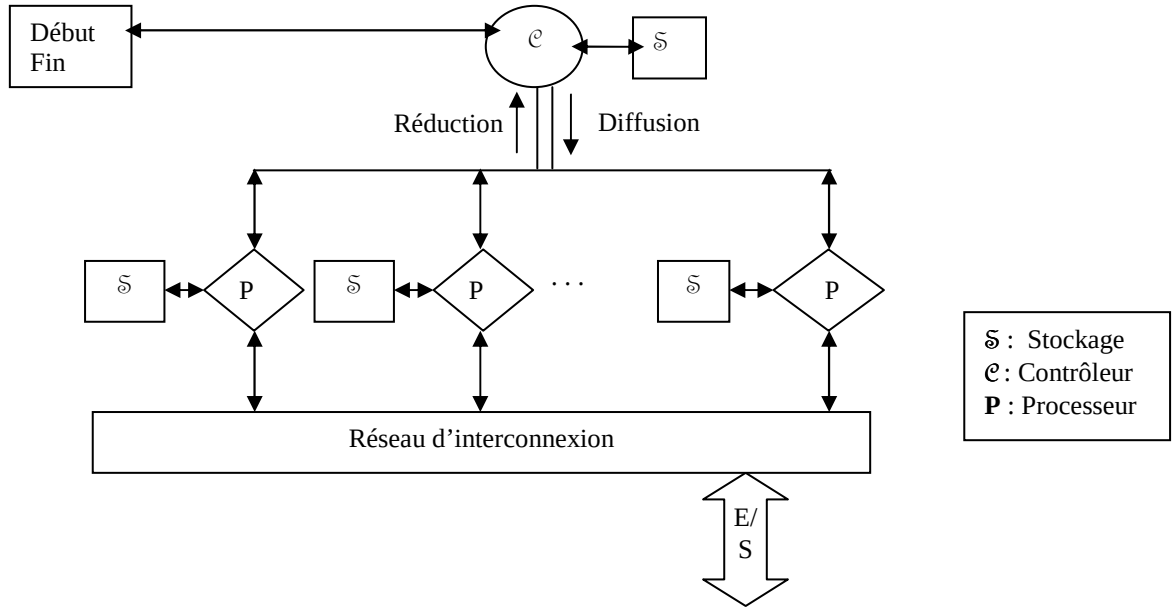


Figure 3.2 Une configuration SIMD

Le contrôleur possède un stockage privé pour contenir le programme et les variables scalaires globales. Les instructions décodées sont par la suite diffusées, une par une, à tout le tableau de processeurs. Chaque processeur possède une banque de mémoire privée qui lui est associée pour stocker les données. Dès que l'instruction est diffusée par le contrôleur, tous les processeurs l'exécutent de manière parfaitement synchrone [CuS99][Mor94].

2.2 Machines MIMD à mémoire distribuée

Dans une machine SIMD, un contrôleur unique pilote une multitude de processeurs esclaves, chacun ayant son propre stockage privé.

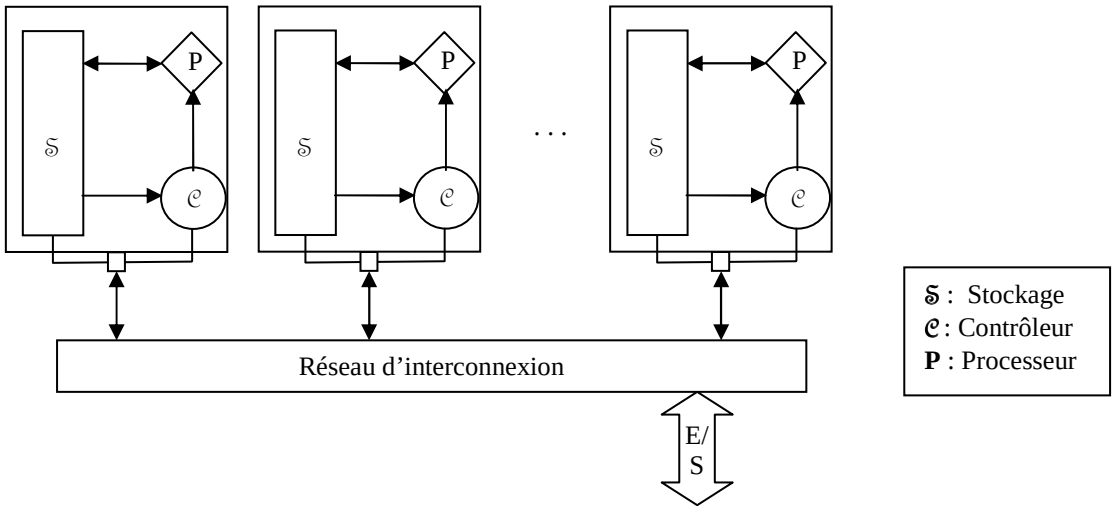


Figure 3.3 Une configuration MIMD à mémoire distribuée

Dans les machines MIMD à mémoire distribuée, non seulement les processeurs et les stockages sont multiples, mais aussi les contrôleurs. Chaque nœud de traitement est capable de stocker et exécuter son propre programme sur ses propres données, de manière complètement indépendante et asynchrone avec tous les autres nœuds de traitement. On utilise aussi le terme multi-ordinateur pour ce type d'architecture [CuS99][Mor94].

2.3 Machines MIMD à mémoire partagée

Dans une machine MIMD à mémoire partagée, le principe consiste à avoir plusieurs copies du contrôleur/processeur avec une instance unique de la banque mémoire uniformément accessible. Ceci est communément appelé SMP (Symmetric MultiProcessor). C'est le modèle le plus réussi et le plus répandu commercialement. La paire C/P est souvent appelée processeur [CuS99][Mor94][Sla06].

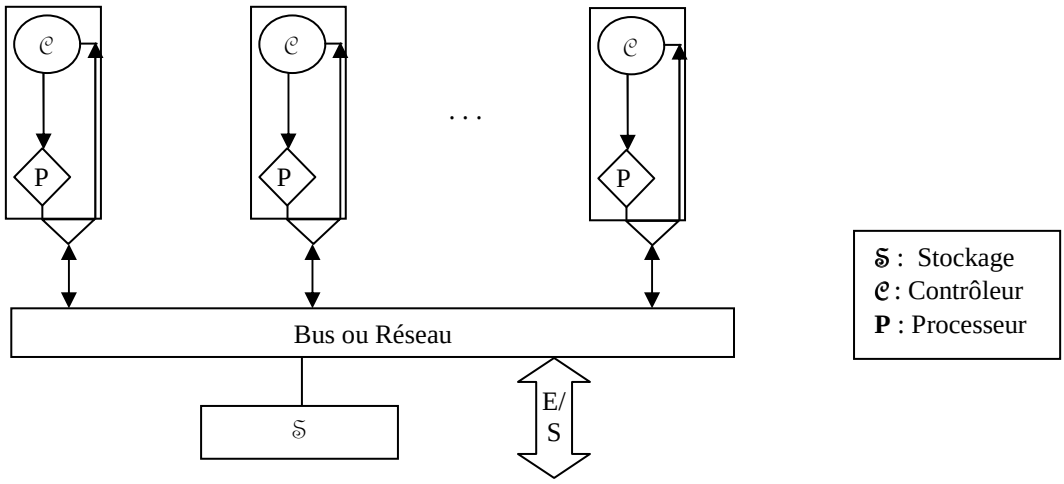


Figure 3.4 Une configuration MIMD à mémoire partagée

2.4 Machines MIMD à mémoire hybride

Pour combiner l'avantage de la rapidité de communication inter-processeurs dans une machine à mémoire partagée, et celui du nombre élevé de processeurs de l'architecture à mémoire distribuée, les constructeurs ont pensé à une nouvelle architecture hybride à mémoire partagée-distribuée. Les machines de ce type sont constituées de nœuds, appelés grappes, dont chacun est composé de plusieurs processeurs partageant une mémoire commune, l'ensemble de ces nœuds étant relié par un réseau d'interconnexion. Ici, l'architecture à mémoire partagée en intra-nœud et l'architecture à mémoire distribuée en inter-nœuds sont assemblées [Sla06].

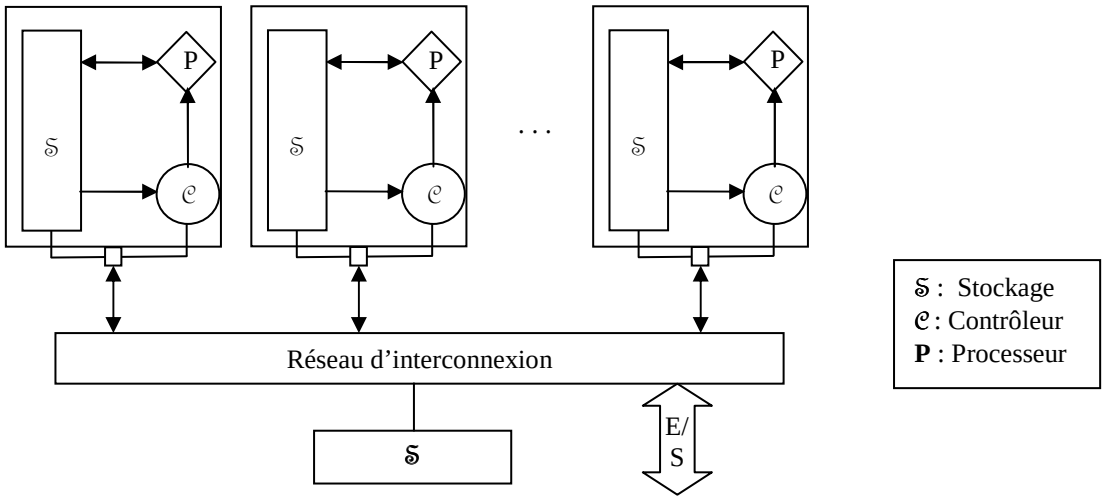


Figure 3.5 Une configuration MIMD hybride

3. Les clusters (grappes)

3.1 Architecture générale

Un cluster consiste en une collection d'ordinateurs autonomes connectés et travaillant ensemble comme une ressource unique et intégrée de calcul.

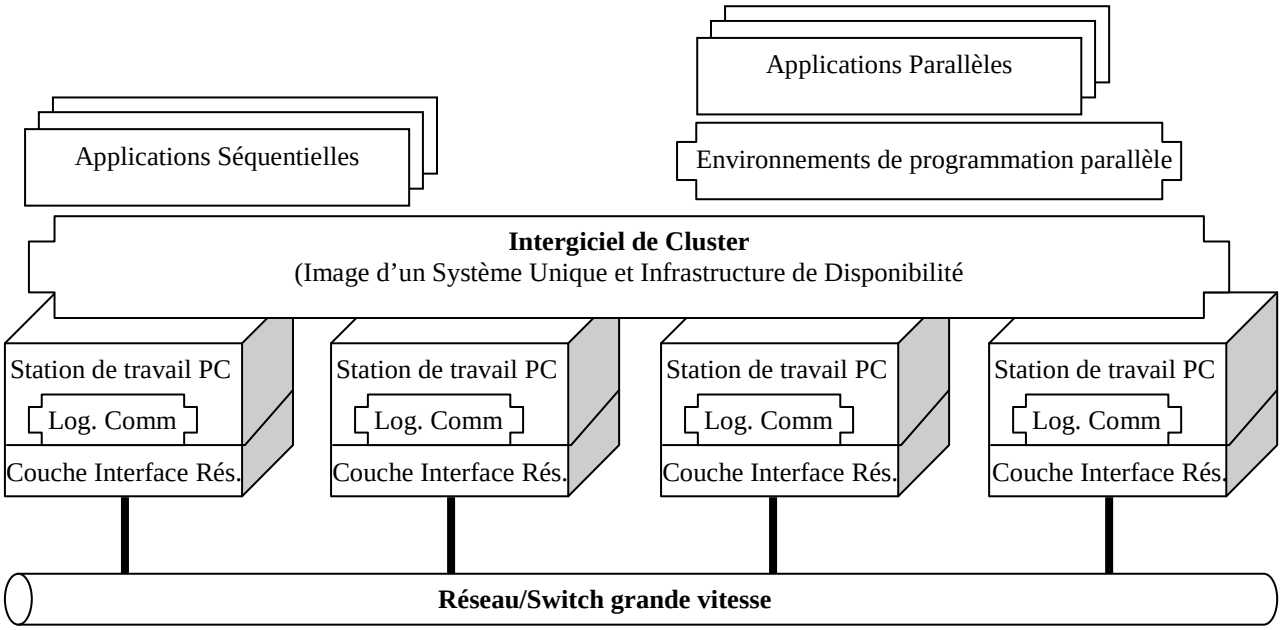


Figure 3.6 Architecture d'un cluster

Un nœud peut être un système unique ou multiprocesseur (PCs, stations de travail ou SMPs) ayant une mémoire, des unités d'E/S et un système d'exploitation. Généralement, un cluster correspond à deux ordinateurs (nœuds) ou plus connectés entre eux. Les nœuds peuvent être sis dans le même local ou géographiquement séparés et connectés via un LAN. Un cluster d'ordinateurs interconnectés (en se basant sur un LAN) peut être vu par les utilisateurs et les applications comme étant un système unique. Un tel système peut avoir des performances équivalentes à celles des systèmes à mémoire partagée qui sont beaucoup plus chers. L'architecture typique d'un cluster est décrite dans la Figure 3.6 [BUY99].

3.2 Les composants d'un cluster

Les principaux composants d'un cluster d'ordinateurs sont les suivants :

- Plusieurs ordinateurs haute performance (PCs, stations de travail, ou SMPs)
- Systèmes d'exploitation
- Réseaux/Switchs Haute Performance (tels que Gigabit Ethernet et Myrinet)
- Cartes Interface Réseau
- Protocoles et Services de Communication rapides (tels que Active et Fast Messages)
- Intergiciel Cluster
- Environnements et outils de programmation parallèle (tels que des compilateurs, PVM (Parallel Virtual Machine), MPI (Message Passing Interface))
- Applications (Séquentielles, parallèles ou distribuées)

La couche matérielle de l'interface réseau est responsable de la transmission et la réception des paquets de données entre les nœuds du cluster à travers le réseau/switch.

Le logiciel de communication offre un moyen de transmission de données rapide et fiable entre les noeuds d'un cluster et vers le milieu extérieur.

Les nœuds du cluster peuvent travailler de manière collective, comme une ressource intégrée de calcul. Ils peuvent aussi fonctionner en tant qu'ordinateurs individuels. Le rôle du middleware est de donner l'image d'un système unique.

Les environnements de programmation peuvent offrir des outils portables, efficaces et faciles à utiliser pour le développement d'applications. Ils incluent des bibliothèques à passage de message, des debuggers et des profilers. N'oublions pas que les clusters peuvent être utilisés pour l'exécution d'applications aussi bien séquentielles que parallèles [BUY99].

3.3 Classification des clusters

Les clusters offrent les caractéristiques suivantes avec un coût relativement bas [BUY99]:

- Haute performance
- Expansibilité et adaptabilité
- Haute Disponibilité

Les clusters sont classés en plusieurs catégories en se basant sur divers facteurs comme indiqué ci-dessous [BUY99] :

- *Application cible* : application de calcul scientifique, application cruciale.
- *Propriété du nœud* : possède un propriétaire ou dédié en tant que noeud du cluster.
- *Caractéristiques matérielles des nœuds* : PC, station de travail ou SMP.
- *Système d'exploitation au niveau des nœuds* : Linux, Windows, Solaris, AIX, etc.
- *Configuration du nœud* : architecture du nœud et type du système d'exploitation installé.
- *Niveau de clustering* : classe basée sur la localisation des nœuds et leur nombre.

Des clusters individuels peuvent être interconnectés pour former un système large (clusters de clusters). En fait, Internet lui-même peut être utilisé comme un cluster de calcul. L'utilisation de ressources connectées par des réseaux à large échelle pour le calcul de haute performance a conduit à l'émergence du calcul de haute performance à large échelle [BUY99].

4. Les grilles

4.1 Définition et Architecture générale d'une grille de calcul (GRID)

Grid signifie littéralement 'grille'. Une grille peut être vue comme un environnement informatique et de collaboration sans frontière. Les grilles de calcul permettent d'utiliser les cycles CPU durant lesquels les machines sont inactives pour exécuter une application. Les capacités de stockage des différents utilisateurs sont les ressources agrégées au sein de la grille. Une caractéristique importante des grilles de calcul est de fournir une importante capacité pour le calcul parallèle. Les domaines académique et industriel utilisent énormément de telles capacités. Les ressources partagées rendent possible l'accès à des ressources spéciales et des logiciels dont le prix de licence est élevé. Il s'agit d'une mise en commun des ressources logicielles et matérielles. En raison de la disponibilité d'importantes quantités de ressources, si certaines ressources deviennent inaccessibles localement, la continuité du service reste assurée.

Une des caractéristiques de la grille de calcul est qu'elle se base sur le 'Push model', c'est-à-dire qu'il y a un coordinateur qui est responsable du partage du travail sur les différentes ressources. Elle est aussi généralement symétrique, c'est-à-dire qu'un nœud de la grille peut être à la fois consommateur et producteur [HOD98].
L'architecture générale d'une grille peut être décomposée en quatre couches. La première correspond à l'infrastructure matérielle. Elle contient les ressources interconnectées au travers des réseaux. Elle comprend des PCs, des systèmes de stockage, des bases de données, etc.
La seconde couche comporte les intergiciels de niveau noyau. Elle offre des services de noyau tels que le contrôle de processus à distance, la co-allocation des ressources, l'accès aux stockages, l'enregistrement de l'information, les mécanismes de sécurité, etc.
La troisième couche contient les intergiciels de niveau utilisateur. Elle regroupe tous les outils qui peuvent aider les développeurs à écrire des applications pouvant tourner sur la grille de calcul. On y trouve ainsi des compilateurs, des librairies, des outils de conception d'applications, etc.
La dernière couche, appelée couche applications, regroupe les applications utilisateurs comme les projets scientifiques, médicaux, d'ingénierie, etc.

4.2 Classification des systèmes de grilles de calcul

Les systèmes de grilles de calcul peuvent être employés pour fournir les services suivants [BBL02] :

- *Services calcul (Computational Services)*

Il s'agit de fournir des services sécurisés pour exécuter les tâches des applications sur les ressources informatiques distribuées. Quelques exemples de grilles informatiques sont NASA IPG [Bau02][Sha05], World-Wide Grid [Bau02][Sha05], et NSF TeraGrid [Bau02].

- *Services données (Data Services)*

Ils cherchent à permettre le transfert sécurisé aux ensembles de données distribuées et à leur gestion. Un exemple d'application qui a besoin d'un tel service pour la gestion, le partage et le traitement de grands ensembles de données est la Physique des Hautes Energies [BBL02].

- *Services application (Application Services)*

Ceux-ci ont pour objectif de gérer des applications et de fournir un accès à distance aux logiciels et aux bibliothèques d'une manière transparente. Un exemple qui peut être employé pour développer de tels services est NetSolve [Sha05].

- *Services information (Information Services)*

Ceux-ci se spécialisent dans l'extraction et la présentation des données significatives en employant les services données et application.

- *Services connaissance (Knowledge Services)*

Ils se focalisent sur la manière d'acquérir, d'employer, de rechercher, d'éditer, et de maintenir des connaissances pour aider les utilisateurs à atteindre leurs buts et objectifs particuliers. Un exemple est le datamining pour établir automatiquement de nouvelles connaissances.

4.3 Classification de l'usage des grilles de calcul

Les utilisateurs des grilles de calcul désirent obtenir le plus simplement possible les solutions de leurs problèmes en tirant profit de façon transparente des ressources de calcul les plus adaptées. Le calcul distribué repose sur une architecture client serveur traditionnelle. On peut aussi classer l'usage des grilles de calcul en trois catégories :

- Utilisation pour la diffusion d'information qui permet un plus grand partage des informations.
- Utilisation pour l'augmentation des capacités de stockage.
- Utilisation pour le calcul ou l'augmentation de la puissance des ordinateurs.

5. Systèmes distribués à grande échelle et systèmes pair-à-pair

5.1 Définition d'un système distribué à grande échelle

Les termes *système distribué à grande échelle* (SDGE) ou *système volontaire* ou encore *système de calcul global* sont utilisés pour désigner les environnements de développement et d'exécution pour les applications sur Internet. On peut considérer les SDGE comme une extension du concept de vol de cycles à l'échelle d'Internet i.e. des machines volontaires qui se connectent à un serveur pour recevoir des tâches à exécuter. De tels systèmes se basent sur le 'Pull model' c'est-à-dire qu'ils demandent périodiquement au serveur, du travail puisque les ressources participantes utilisent généralement des 'Firewalls'. Le système volontaire n'est pas symétrique car ce sont les (nœuds) volontaires qui fournissent les ressources informatiques aux projets, et non le contraire. Comme exemples de SDGE, citons Distributed.net [Dis04], SETI@home [ACK02][Bau02], XtremWeb [Bau02] et XtremWeb-CH [AbB07].

5.2 Différences entre SDGE, système P2P et grille de calcul

Dans la littérature [Bau02], les SDGE sont considérés comme une forme de système pair à pair (peer-to-peer ou P2P). Néanmoins, plusieurs paramètres sont discriminants. En effet, un système est dit P2P lorsqu'il autorise la communication directe entre entités d'un réseau, sans passer par une autorité centrale, telle qu'un serveur. Dans un réseau P2P, chaque entité se comporte à la fois comme un client et un serveur. L'architecture des systèmes P2P est donc généralement décentralisée. Les SDGE comme SETI@home [Bau02] ne respectent pas nécessairement ces propriétés.

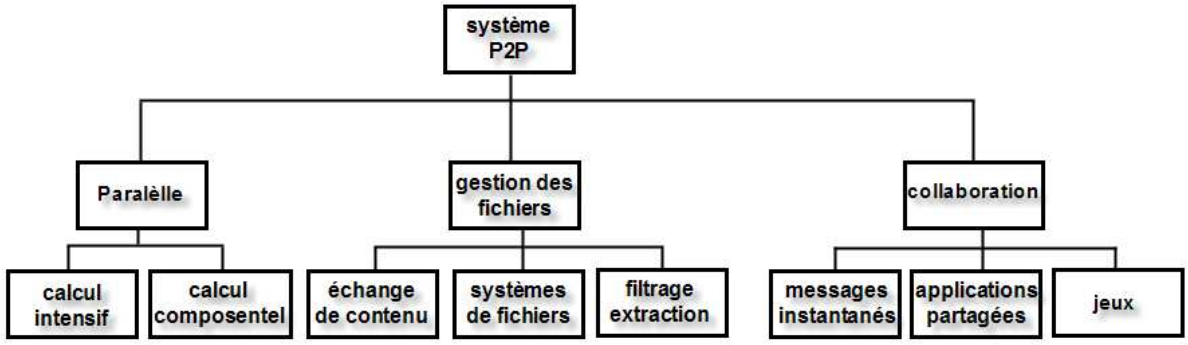


Figure 3.7 Classification des systèmes P2P

D'un autre côté, on confond les systèmes volontaires et P2P avec les grilles de calcul alors qu'ils se distinguent par quatre caractéristiques. En effet, dans un système P2P, nous avons :

1) Le nombre de ressources connectées est plus élevé de plusieurs ordres de grandeur (typiquement 100 000 ressources) et les ressources sont rarement parallèles (biprocesseur au maximum)
2) Les ressources sont extrêmement volatiles
3) Les réseaux connectant les réseaux sont des LAN, des Intranets et l'Internet
4) Les utilisateurs sont aussi en nombre très important (typiquement un utilisateur par ressource).

Comparativement aux grilles de calcul, il faut composer avec une infrastructure matérielle déjà présente, évoluant de façon non coordonnée et en fonction d'impératifs sans relation évidente avec le calcul à grande échelle.

L'ordre de grandeur et l'impossibilité d'action ou d'influence sur l'infrastructure engendrent des problèmes spécifiques qui ne se présentent pas dans le cas des grilles de calcul. Ainsi :

- la sécurité doit être fondée sur des mécanismes extensibles à plusieurs centaines de milliers d'utilisateurs et de ressources et,

- le placement et l'ordonnancement des tâches doivent prendre en compte l'évolution à court terme (connexion et déconnexion des ressources) et l'évolution à long terme (modification d'infrastructure) du système.

Globalement, les techniques adaptées dans le cadre des grilles de calcul sont inapplicables dans celui du calcul à très grande échelle. Les fonctions traditionnellement associées à la procédure de login (identification, sélection des utilisateurs, droits d'accès, priorité d'exécution, protection du site et traçage des opérations) ne passent pas à l'échelle. Inversement, la confiance de l'utilisateur envers les résultats renvoyés et la facturation de l'utilisation des ressources ne peuvent pas reposer, comme dans le cas des grilles de calcul, sur le caractère institutionnel des sites accédés. Les sites de calcul sont des machines quelconques appartenant à des utilisateurs auxquels on ne peut pas accorder a priori une confiance absolue. Le calcul à très grande échelle repose donc sur la capacité de certifier les résultats ou d'être capable de distinguer entre les bons et les mauvais résultats. De même, contrairement au cas des grilles de calcul, l'ordonnancement d'un très grand nombre de tâches sur une centaine de milliers de ressources ne peut être géré à long terme et coordonné à court terme par des mécanismes centralisés.

Ainsi, les grilles et les systèmes pair à pair ont des racines historiques très différentes qui expliquent leur différence d'organisation et de problématique propre [Bau02].

6. Le Cloud computing ou 'Informatique dans les nuages'

6.1 Définition

Le Cloud computing (littéralement l'informatique dans les nuages), c'est la possibilité de disperser un système d'information sur des infrastructures prises en charge par un ou plusieurs prestataires. La localisation géographique de ces ressources virtuellement illimitées n'importe plus. Ce concept était déjà ébauché par les services d'hébergement, le mode SaaS (Software as a Service) ou la location de puissance de calcul. Le Cloud computing a émergé sous l'impulsion de la virtualisation de plates-formes conçues d'emblée pour être mutualisées au travers de vastes grilles de serveurs. Associée à la redondance et à la dispersion géographique, cette mutualisation devient synonyme de montée en charge aisée, de haute disponibilité et de plan de reprise d'activité (PRA) à moindre coût [LEV08][YIE09].

6.2 Location de serveurs virtuels nus adaptés à la reprise de l'existant

Depuis début 2008, les grands de l'informatique ont lancé des nouveaux services dans l'objectif de donner aux entreprises les moyens d'externaliser entièrement leurs salles informatique. Ces offres prennent d'abord la forme de serveurs virtuels hébergés. La virtualisation apporte un premier plus : en cas de panne, un serveur virtuel peut être déplacé presque instantanément. Des prestataires comme Prosodie ou InternetFR intègrent ce type de services à leurs offres d'hébergement classique. Mais les machines virtuelles ne tournent alors sur un matériel mutualisé qu'en phase de développement, ou en cas d'incident ou sinistre.
Dans le cadre du Cloud computing qui sous-tend au contraire une mutualisation en production, la virtualisation présente un avantage supplémentaire : la dématérialisation des serveurs permet, par exemple en cas de dégradation des temps de réponse, de passer dynamiquement sur une machine physique plus puissante, voire même de confier ces serveurs à un autre prestataire ou de les internaliser de nouveau.
Mais la grande nouveauté, c'est que des acteurs majeurs cherchent à généraliser cette idée. HP lance ainsi son offre New Generation Data Center (NGDC) qui sous-tend la location de machines virtuelles sous HP-UX, Linux ou Windows. SUN lui a emboîté le pas avec son projet Hydrazine qui ajoute Solaris à Linux et Windows. De son côté, Amazon avait démarré par S3, un simple service de fourniture de ressources de stockage récemment complété par EC2 (Elastic Computer Cloud), qui inclut la fourniture de serveurs virtuels x86 [LEV08].

7. Un SDGE type : XtremWeb

Le projet XtremWeb vise la conception et le développement d'un environnement de calcul global pair à pair académique et pluridisciplinaire.
L'environnement XtremWeb est une plate-forme généraliste, sécurisée et orientée calcul de haute performance. La conception d'XtremWeb est organisée selon une perspective de dissémination de la technologie logicielle qui sera développée dans le projet [Bau02]. XtremWeb est une plate-forme open source pour étudier :

- ✓ Le calcul Global (extensibilité, organisation des serveurs…).
- ✓ Le calcul P2P (volatilité, communication inter-nœuds, sécurité).
- ✓ Fusion des systèmes de calcul Global et P2P.

Donc XtremWeb est une plate-forme de recherche (étudier de nouveaux mécanismes) et de production (identifier les problèmes des utilisateurs). Cette plate-forme est beaucoup plus

qu'un environnement pour construire des clusters par-dessus Internet ou à l'intérieur d'un réseau propriétaire multi-sites.

7.1 Architecture d'XtremWeb

XtremWeb est conçu pour fournir un environnement de calcul global pour la résolution de différentes applications sur des projets d'institutions, d'entreprises commerciales ou de communautés *open source*. La Figure 3.8 donne un aperçu sur différentes utilisations envisagées d'un système de calcul global comme XtremWeb.

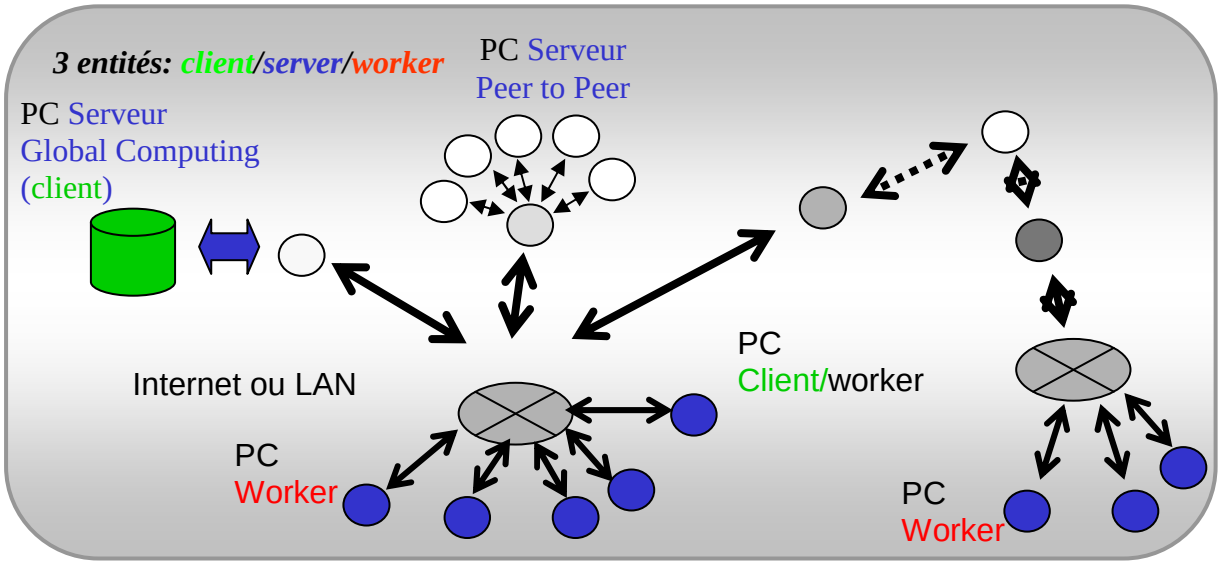


Figure 3.8 Architecture générale de XtremWeb

7.2 Organisation générale

L'architecture générale d'XtremWeb est centralisée, dans le sens où un serveur, qui peut être distribué, organise les calculs sur des ressources distantes géographiquement distribuées ainsi que les échanges d'informations entre ces ressources. Les ressources sont nommées Workers dans la suite.

Dans XtremWeb le serveur est centralisé, il joue aussi le rôle de distributeur de résultats. Le projet XtremWeb, comme tous les environnements de calcul global, doit satisfaire des contraintes sévères telles que l'extensibilité, l'hétérogénéité, la dynamicité, la disponibilité, tolérance aux pannes, l'utilisabilité, la sécurité, etc.

7.3 Architecture du Worker

Le Worker a deux principales fonctionnalités : fournir des ressources de la machine d'un utilisateur pour l'exécution d'un calcul XtremWeb et exécuter l'application fournie par le serveur. Pour qu'un projet de calcul global soit largement accepté auprès du public, il est nécessaire que le logiciel Worker assure au mieux le respect de l'utilisateur, c'est-à-dire qu'il soit configurable, non intrusif et sécurisé.
L'utilisateur décide quand XtremWeb peut lancer une exécution et quelles sont les ressources (CPU, taille mémoire, espace disque) que cette exécution peut obtenir.

7.4 Architecture des serveurs

Les serveurs XtremWeb sont responsables du support des applications globales en (i) stockant les fichiers binaires des applications pré-compilés pour différentes architectures et des tâches soumises par les clients et (ii) fournissant les tâches et les applications aux Workers désirant faire du calcul.
Distribuer une application consiste à fournir un code binaire pré-compilé pour différentes architectures et une description de l'application. La description de l'application comprend son nom, la manière dont l'application prend en compte ses paramètres et la forme que prennent ses résultats. Il est aussi possible de fournir différentes versions de l'application, la mise à jour est alors automatique sur le serveur.
Un client soumet une tâche en fournissant la référence de l'application et l'ensemble des paramètres qui définissent une tâche. Les paramètres peuvent être différents fichiers, un fichier lu sur l'entrée standard de l'application ou des arguments sur la ligne de commandes.
Lorsqu'une tâche est enregistrée dans la base de données, elle reçoit un unique identifiant de tâche. Cet identifiant de tâche réfère la tâche pour toutes les opérations suivantes d'ordonnancement et de récupération des résultats. Un module de surveillance contient les informations sur l'ensemble des tâches telles que (i) les machines ayant traité la tâche ou la liste des machines si la tâche a été interrompue, (ii) le client qui a soumis la tâche et (iii) différentes prises de temps. Les utilisateurs et administrateurs d'XtremWeb peuvent interagir avec le système à travers une interface web pour soumettre des tâches, suivre leur progression, récupérer les résultats. Les utilisateurs peuvent obtenir des statistiques sur l'activité des machines participant à XtremWeb en tant que Worker.

Le placement des tâches sur les Workers est effectué en deux étapes : la sélection des tâches et le placement des tâches. La sélection des tâches à partir de la liste des tâches, remplie par les requêtes des clients, se fait en fonction des applications prioritaires. La priorité est déterminée en attribuant des proportions de tâches à accomplir par application. Les tâches sont placées selon une méthode FIFO. Lorsqu'un Worker demande une tâche, le serveur sélectionne d'abord la tâche qui peut être exécutée par ce Worker particulier selon son environnement d'exécution et l'existence d'un binaire pré-compilé de l'application. Le serveur détecte aussi les tâches abandonnées en relevant des dépassements de délai et les attribue à d'autres Workers si nécessaire. Les politiques de sélection et de placement des tâches sont reconfigurables dynamiquement.

8. Conclusion

Dans ce chapitre nous nous sommes intéressés à la description des principales architectures parallèles et distribuées. Les machines parallèles offrent de hautes performances de calcul mais à coût élevé. Dans le cas de certaines applications, on peut avoir recours à un cluster (grappe) qui n'est autre qu'une collection d'ordinateurs autonomes connectés et qui peut être vu par les utilisateurs et les applications comme étant un système unique. Les nœuds d'un cluster peuvent être situés dans un même local ou géographiquement séparés et connectés via un LAN.

Les grilles de calcul peuvent être considérées comme une évolution du concept de cluster. Une grille de calcul est constituée de nombreuses ressources informatiques hétérogènes, géographiquement distribuées et interconnectés. Un tel système distribué exploite les puissances de calcul de toutes ses ressources et les combine en synergie, créant en quelque sorte un super-calculateur virtuel extrêmement puissant [HOD98]. L'architecture des systèmes distribués à grande échelle (SDGE) diffère de celle des grilles même s'ils remplissent les mêmes fonctions (exécution d'applications de haute performance). Les SDGE offrent des PCs aux utilisateurs connectés sur Internet pour la résolution de (grands) projets scientifiques.

Un nouveau concept dans la famille des architectures distribuées est le Cloud computing qui a émergé sous l'impulsion de la virtualisation et de plates-formes conçues d'emblée pour être mutualisées au travers de vastes grilles de serveurs. Il offre la possibilité de disperser un système d'information sur des infrastructures prises en charge par un ou plusieurs prestataires. Dans cette panoplie de systèmes distribués, les SDGE et plus précisément, le système

volontaire XtremWeb-CH, constitue notre architecture cible. Nous présenterons dans la partie II du manuscrit les caractéristiques de cette plate-forme ainsi qu'une étude expérimentale que nous avons réalisée dessus.

Partie II

CHAPITRE 4.

ETUDE DE L'OPTIMISATION DU PRODUIT MATRICE-VECTEUR CREUX

1. Introduction

Notre objectif dans ce chapitre est d'étudier les performances de l'algorithme du produit matrice-vecteur creux (PMVC) sur diverses machines séquentielles cibles représentant un échantillon de nœuds d'un système distribué à grande échelle (SDGE). La matrice d'entrée étant creuse, notre étude du PMVC s'intéresse à une série de versions de cet algorithme associées à différents formats de stockage compressé pour la matrice creuse.

Du fait de la diversité des formats de compression, nous nous restreignons à quatre principaux formats, à savoir le DNS (Dense), le CSR (Compressed Sparse Row), le COO (COOrdinate) et le BND (BaNDed). En effet, les trois premiers sont des formats généraux s'appliquant à n'importe quelle structure de matrice creuse. De plus, ils sont très utilisés dans la littérature. Nous avons de plus choisi le format BND comme format spécifique s'appliquant généralement à des matrices bandes. Il se trouve que plusieurs applications réelles manipulent des matrices de cette structure (voir chapitre 2, section 4.). De plus, on peut toujours ramener une matrice creuse de structure quelconque à une matrice bande en appliquant l'algorithme de Cuthill-Mckee [DER92].

D'un autre côté, du fait de la multiplicité des architectures séquentielles, nous avons choisi trois machines cibles représentatives i.e. un PC Intel Pentium dual-core, un processeur d'une machine biprocesseur AMD quad-core et une station de travail SUN dual-core.

Notre travail a consisté à étudier l'optimisation de différentes versions de l'algorithme du PMVC. Ainsi, nous appliquons à ces versions des techniques générales d'optimisation de haut niveau et ce, indépendamment de la structure de la matrice traitée, du format de stockage, des structures de données utilisées pour ce format ainsi que de l'architecture cible [EHM05] [EHM09].

Le présent chapitre est organisé comme suit. Quatre versions du PMVC relatives aux formats DNS, CSR, COO et BND sont présentées en section 2. Un état de l'art sur l'optimisation du PMVC est présenté en section 3. La section 4 est consacrée à l'étude de l'optimisation de ces différentes versions. Enfin, une étude expérimentale est décrite dans la section 5.

2. Algorithmes du PMVC pour les différents formats de compression

La structure de l'algorithme du PMVC dépend du format de stockage utilisé pour la matrice creuse. Ainsi, aux quatre formats retenus i.e. DNS, CSR, COO et BND, nous associons les quatre versions PMVC-DNS, PMVC-CSR, PMVC-COO et PMVC-BND qui sont présentées en détail dans la suite.

2.1. PMVC-DNS

Soient A une matrice creuse N × N stockée dans le format DNS (i.e. tableau 2-D), *x* et *y* deux vecteurs de taille N stockés chacun dans un tableau 1-D. Il s'agit de calculer le PMVC *y* =A*x* pour le format DNS. L'algorithme trivial correspondant se détaille comme suit.

```
Do i = l, N
        Do j = 1, N
            y[i]=y[i] +A[i][j] * x[j]
        EndDo
EndDo
```

Figure 4.1 Produit matrice-vecteur creux pour le format DNS

La complexité spatiale de cet algorithme est N^2 +2N réels. La complexité de calcul (temporelle) est égale à $2N^2$. Notons que dans cette version du PMVC (notée IJ), on effectue le produit ligne par ligne (de A). Une autre version du PMVC (notée JI) peut être associée au format DNS. Elle consiste à faire le produit colonne par colonne (de A). Il est connu [JaK99] que la version IJ (resp. JI) est plus adéquate à un mode de stockage de A par lignes (resp. colonnes). Nous nous restreignons ici à la première, vu que dans nos expérimentations, nous utilisons le compilateur C qui stocke les tableaux 2-D ligne par ligne (contrairement à FORTRAN qui adopte le mode de stockage par colonne).

2.2. PMVC-CSR

On stocke A dans le format CSR (voir chapitre 2, section 3-2.2) où le vecteur $\mathcal{A}$ contient les éléments non nuls, le vecteur JA contient les indices de colonne des éléments de $\mathcal{A}$ et IA contient les pointeurs sur le début de chaque ligne dans $\mathcal{A}$ et JA. L'utilisation du format CSR pour le stockage de la matrice creuse induit un algorithme du produit matrice-vecteur différent du précédent. Il a la forme présentée dans la Figure 4.2.

Remarquons les bornes non affines de la seconde boucle et les accès indirects aux éléments du vecteur *x*. La complexité spatiale de cet algorithme est NZ+2N réels en plus de (NZ+N+1) entiers. La complexité de calcul, quant à elle, est 2NZ. Soulignons le gain important, par rapport à la version PMVC-DNS dans le cas où N est élevé et NZ petit e.g. de l'ordre de O(N) [EHM05].

```
Do  i = l, N
        Do j = IA[i] , IA[i+l]-1
            y[i] = y[i] + 𝒜[j] * x[JA[j]]
        EndDo
EndDo
```

Figure 4.2 Produit matrice-vecteur creux pour le format CSR

2.3 PMVC-COO

Dans le cas où la matrice A est stockée dans le format COO (voir chapitre 2, section 3-2.2) où $\mathcal{A}$ contient les éléments non nuls, IA et JA contiennent respectivement les indices de ligne et de colonne des éléments de $\mathcal{A}$, l'algorithme du PMVC devient comme suit :

```
Do i=1,NZ
    y[IA[i]]=y[IA[i]] +𝒜 [i]*x[JA[i]]
EndDo
```

Figure 4.3 Produit matrice-vecteur creux pour le format COO

Notons les accès indirects aux éléments des vecteurs *x* et *y*. La complexité spatiale de cet algorithme est (NZ+2N) réels + 2NZ entiers. La complexité de calcul est 2NZ.

2.4 PMVC-BND

A étant une matrice bande, soit ML sa largeur de demi-bande inférieure et MU sa largeur de demi-bande supérieure. L'algorithme associé au format BND est donné par la Figure 4.4. ABD est un tableau 2D de taille N×(ML+MU+1) contenant les éléments de la bande de A (voir chapitre 2, section 3.1.2).

Notons les expressions d'indices non affines au niveau des éléments du vecteur *x*. La complexité spatiale de cet algorithme est N(ML+MU+3) réels. La complexité de calcul est 2N(ML+MU+1).

```
Do i = 1, N
        Do j = 1 , ML+MU+1
            y[i] = y[i]+ABD[i][j] * x[Min(N,Max(1,i+j-ML-1))]
        EndDo
EndDo
```

Figure 4.4 Produit matrice-vecteur creux pour le format BND

2.5 Tableau récapitulatif

Le tableau suivant récapitule les complexités spatiales et temporelles pour les quatre algorithmes PMVC-DNS, PMVC-CSR, PMVC-COO et PMVC-BND.

FC	Complexité spatiale	Complexité de calcul
DNS	(N^2+2N) réels	$2N^2$
CSR	(NZ+2N) réels + (NZ+N+1) entiers	2NZ
COO	(NZ+2N) réels + 2NZ entiers	2NZ
BND	N(ML+MU+3) réels	2N(ML+MU+1)

Table 4.1 Complexités des algorithmes du PMVC

3. Etat de l'art sur l'optimisation du PMVC

Nous pouvons classer les travaux sur l'optimisation du PMVC selon six axes, notés Opt1… Opt6, qui sont décrits dans ce qui suit [GKA08].

3.1 Opt1. Proposition d'un nouveau format

Dans certains travaux, pour optimiser le PMVC, les auteurs se trouvent obligés de proposer un nouveau format de stockage (i.e. différent du format original) [WiL06]. Ainsi, dans [PiH99], l'optimisation est effectuée grâce à l'utilisation du format BCSR (Block CSR), une variante du format CSR, qui assure une certaine localité des données. L'idée derrière ce format est d'exploiter les éléments non nuls se trouvant dans des positions contiguës en les empaquetant dans des blocs de tailles différentes. Connaissant l'indice de colonne du premier

élément non nul dans un bloc, on peut déduire les indices de colonne de tous les autres éléments non nuls du même bloc. En d'autres termes, un seul accès indirect est nécessaire pour chaque bloc. Le format BCSR utilise un tableau 1-D (de longueur égale au nombre de blocs) supplémentaire en plus des trois tableaux utilisés par le format CSR.

Dans [MeG04] les auteurs proposent aussi une nouvelle variante du format CSR qu'ils appellent L-CSR (Length-grouped CSR). En effet, l'utilisation de ce format est nécessaire pour pouvoir appliquer la technique d'optimisation unroll-and-jam [MeG04]. Cette dernière, consiste à appliquer un unrolling (voir section 4.1.1) à la boucle externe du PMVC-CSR puis une fusion des boucles internes qui en résultent. Unroll-and-jam ne peut pas être appliqué si les corps des boucles internes résultant de l'unrolling ne peuvent pas être fusionnées. En effet, si on déroule la boucle externe sur les lignes, la fusion des différentes boucles internes qui en résultent n'est pas légale parce que leurs lignes peuvent avoir des longueurs (nombre d'éléments non nuls) différentes. Or, dans les matrices creuses concernées par le PMVC dans [MeG04], le nombre de valeurs que peuvent prendre les longueurs des lignes est limité. Si on réorganise la matrice de telle manière que les lignes de même longueur soient regroupées ensemble, on peut alors appliquer un unroll-and-jam au nid de boucles du PMVC et structurer les calculs selon les groupes de lignes de même longueur. De la même manière que le format CSR, le format L-CSR utilise un couple de vecteurs, JA et $\mathcal{A}$ qui contiennent respectivement les indices de colonne et les valeurs des éléments non nuls de la matrice. Comme dans CSR, une ligne est représentée par une suite contiguë d'entrées dans JA et $\mathcal{A}$. Néanmoins, dans L-CSR, les lignes sont regroupées par longueur, alors que dans CSR les lignes apparaissent selon l'ordre initial dans la matrice. Ainsi, un quatrième vecteur supplémentaire est utilisé par L-CSR pour représenter les groupes de lignes.

Dans [KGK08], sont proposés deux formats dénommés CSR-DU (CSR Delta Unit) et CSR-VI (CSR Value Indexed) qui sont aussi des variantes du format CSR. Le principe du premier format consiste à compresser les structures de données correspondant aux indices. Le format CSR-DU se base sur le codage du tableau JA qui contient les indices de colonne du format CSR (voir chapitre 2, section 3.2.2). Pour cela, on utilise le codage delta [WiL06]. Un delta est défini comme étant la différence entre l'indice courant et l'indice précédent dans JA. Il est positif et inférieur ou égal à la valeur de l'indice courant. Une fois codé, le tableau JA est divisé en unités ayant un nombre d'éléments différents. Pour chacune de ces unités, on détermine la valeur maximale de delta et on en déduit la taille (en octet) qui peut représenter

tous les deltas de l'unité. Une unité se termine lorsqu'elle atteint sa taille maximale fixée à l'avance ou bien quand une nouvelle ligne de la matrice est détectée.

Le principe du format CSR-VI consiste à compresser le vecteur $\mathcal{A}$ qui contient les éléments non nuls de la matrice en profitant de l'éventuelle redondance des valeurs non nulles. A la place de $\mathcal{A}$, le format CSR-VI utilise deux vecteurs vals_unique et val_ind. Le premier contient les valeurs non nulles distinctes de la matrice et le second contient, pour chaque élément non nul de la matrice, l'indice de sa valeur dans vals_unique.

Un autre format dénommé UBCSR (Unaligned BCSR), qui est une extension de BCSR, est présenté dans [VuM05] pour optimiser le PMVC. Par rapport au BCSR, il consiste à utiliser un vecteur supplémentaire qui contient les pointeurs sur les débuts des lignes de bloc.

3.2 Opt2. Optimisations spécifiques à des matrices creuses particulières

Les optimisations considérées dans [MeG04] sont spécifiques à des matrices creuses où le nombre d'éléments non nuls par ligne est faible. Les matrices considérées dans ce papier sont issues de SAGE, une application ASCI du Los Alamos National Laboratory (LANL). Dans ce cas, le nombre de valeurs que peuvent prendre les longueurs des lignes dans une matrice creuse est limité [WhB97].

3.3 Opt3. Optimisations spécifiques à des formats particuliers

La plupart des travaux tournent autour du format CSR [KGK08][MeG04][PiH99]. Ainsi, le format L-CSR [MeG04] est une variante du CSR. De la même manière que le format CSR, le format L-CSR utilise une paire de vecteurs, JA et $\mathcal{A}$ qui contiennent respectivement les indices de colonne et les valeurs des éléments non nuls de la matrice. Comme dans CSR, une ligne est représentée par une suite contiguë d'entrées dans JA et $\mathcal{A}$. Néanmoins, dans L-CSR, les lignes sont regroupées par longueur, alors que dans CSR les lignes apparaissent dans l'ordre normal de la matrice. Ainsi, un quatrième vecteur supplémentaire est utilisé par L-CSR pour représenter les groupes de lignes.

CSR-DU (CSR Delta Unit) et CSR-VI (CSR Value Indexed) [KGK08] sont aussi des variantes du format CSR. Le principe du premier format consiste à compresser les structures de données correspondant aux indices. Le principe du format CSR-VI consiste à compresser le vecteur $\mathcal{A}$ contenant les éléments non nuls de la matrice.

Le format BCSR (Block CSR), est aussi une variante du format CSR dont l'idée est d'exploiter les éléments non nuls ayants des positions contiguës en les empaquetant dans des blocs de tailles différentes.

3.4 Opt4. Optimisations par restructuration des données

Plusieurs travaux proposent pour l'optimisation du PMVC de restructurer la matrice (permutations de lignes et ou de colonnes) en utilisant la stratégie de l'algorithme de Cuthill-McKee inversé ou bien la méthode Glouton pour la structuration en blocs [DER92]. Il a été montré expérimentalement [Tol97] que par rapport à la structure quelconque, une structure régulière et/ou par bloc de la matrice creuse permet de réduire les défauts de cache et améliorer les performances du PMVC [PiC05][PiH99].

3.5 Opt5. Optimisations spécifiques à des architectures particulières

D'autres optimisations telles que celles proposées dans [ImY01][Vud02][Wil07] dépendent de l'architecture de la machine cible. Ainsi, dans [ImY01] et [Vud02] sont étudiées des optimisations qui dépendent des tailles de la mémoire cache et du registre mémoire. Les deux travaux décrivent un système appelé Sparsity qui, étant données les performances d'une machine et les caractéristiques de la matrice creuse, peut déterminer une estimation de la taille optimale du bloc pour la décomposition de la matrice creuse.
Nous relevons de même dans [Wil07] une étude d'optimisation du PMVC pour des plates-formes multi-coeurs. Il se trouve que sur ce type d'architecture, l'optimisation peut être assurée par l'utilisation du parallélisme de threads. La matrice creuse est ainsi partitionnée en blocs. Chaque bloc est traité par un thread. Les threads sont traités en parallèle par les différents cœurs de la machine. Un travail d'équilibrage des charges entre les blocs de la matrice est nécessaire pour avoir de bonnes performances. Chaque thread subit, en plus, d'autres optimisations telles que celles par bloc de cache et par bloc de registre [ImY01][Vud02].

3.5 Opt6. Optimisations de bas niveau

En plus des optimisations spécifiques aux architectures, les auteurs dans [Wil07] utilisent des optimisations de bas niveau spécifiques à une architecture multi-coeurs. Parmi ces

optimisations, citons l'utilisation explicite des instructions SSE (Streaming SIMD Extensions) qui sont des instructions de bas niveau permettant la manipulation de registres mémoire.

3.6 Récapitulation

Un tableau récapitulatif de l'état de l'art sur l'optimisation du PMVC couvrant la période 1997-2008 est présenté ci-dessous.

Référence	Opt1	Opt2	Opt3	Opt4	Opt5	Opt6
[Tol97]				*		
[PiH99]	*		*	*		
[ImY01]					*	
[Vud02]					*	
[MeG04]	*	*	*			
[VuM05]	*					
[PiC05]				*		
[WiL06]	*					
[Wil07]					*	*
[KGK08]	*		*			

Table 4.2 Récapitulation de l'état de l'art sur l'optimisation du PMVC

En ce qui nous concerne, nous adoptons une approche générique consistant à appliquer au PMVC des techniques d'optimisation de haut niveau valables pour n'importe quelle version et ce, indépendamment de la structure de la matrice traitée, du format de stockage, des structures de donnée utilisées pour ce format ainsi que de l'architecture cible. Notre objectif est en fait de réaliser cette optimisation des versions du PMVC relatives aux quatre formats de stockage considérés de manière équitable et ce, afin d'effectuer par la suite une étude comparative conséquente entre ces différentes versions.

4. Etude de l'optimisation du PMVC

4.1. Techniques d'optimisation

Pour l'optimisation du PMVC, nous avons choisi d'utiliser des techniques se classant en deux catégories : (i) celles pouvant être appliquées manuellement sur le code, et (ii) celles pouvant être sélectionnées comme options lors de la compilation.

4.1.1 Approche manuelle

Plusieurs optimisations peuvent être appliquées manuellement sur un code donné. On s'intéresse, ici, essentiellement, aux optimisations relatives aux boucles (de type DO). En effet, nous avons vu dans la section 2 que les différentes versions du PMVC ont la structure régulière d'une boucle ou d'un nid de boucles imbriquées [Saad94]. Nous nous intéressons ici aux optimisations, par le remplacement scalaire, l'élimination des invariants des boucles et le déroulement de boucles (loop unrolling).

- *Remplacement scalaire*

II s'agit de remplacer un élément de tableau (accès mémoire indirect) par un scalaire. L'accès est ainsi plus rapide. Cette optimisation est recommandée dans le cas où l'instruction en question se trouve à l'intérieur d'une boucle [EHM05].

Code d'origine

```
Do i=1, N
        Do j=1, N
                A[i]=….
        EndDo
EndDo
```

Code après optimisation

```
Do i=1, N
        Do j=1, N
                Scalaire=….
        EndDo
        A[i]=Scalaire
EndDo
```

- *Elimination des invariants des boucles*

Il s'agit d'éviter un travail qui n'est pas nécessaire en éliminant les invariants de boucle, en particulier si ces invariants induisent des accès mémoire comme l'illustre l'exemple qui suit.

Code d'origine

```
Do j=1,N
        i=3
        Travail(j)= …
EndDo
```

Code après optimisation

```
i=3
Do j=1, N
        Travail(j)= …
EndDo
```

- *Déroulement de boucles (Loop unrolling)* [DoH79]

C'est une technique qui consiste à dupliquer le corps d'une boucle un nombre de fois appelé facteur d'unrolling (*u*), puis à itérer avec un pas égal à *u* au lieu d'un pas égal à 1. Cette

transformation peut améliorer le code en réduisant l'overhead généré par les contrôles des boucles. Elle permet aussi le pipelinage et améliore la localité des données aussi bien au niveau des caches que des registres.

Boucle d'origine

```
Do i=1,N
        Traitement sur A[i]
EndDo
```

Boucle déroulée

```
i = 1
m=N mod u
ne=N-m
Do i=1,ne,u
        Traitement sur A[i+0]
        Traitement sur A[i+1]
        Traitement sur A[i+2]
        Traitement sur A[i+3]
                ….
        Traitement sur A[i+u-1]
EndDo
Do i= ne+1, N
        Traitements sur A[i]
EndDo
```

Figure 4.5 Loop Unrolling (facteur *u*)

- *Autres optimisations*

D'autres optimisations au niveau des boucles sont connues dans la littérature. Nous pouvons citer le remplacement des instructions coûteuses par d'autres moins coûteuses, l'élimination des opérations invariantes des boucles, la permutation des boucles, la fusion et la distribution de boucles, les optimisations des E/S, les optimisations mémoire, etc [Ban93] [EHM05].

4.1.2 Options du compilateur C sous UNIX

Pour nos expérimentations, nous avons utilisé C sous UNIX. Le compilateur C sous UNIX présente plusieurs options permettant une optimisation automatique du code source dans le but d'améliorer ses performances. Parmi ces optimisations, on peut citer l'option -On, où n désigne le

niveau d'optimisation. Les valeurs que peut prendre n dépendent de la version utilisée du compilateur [Alo05][BeA04][EHM05].
Nous nous intéressons aussi à l'option du compilateur qui effectue un unrolling automatique pour le code source. La syntaxe de cette option dépend aussi du compilateur. Généralement, ces options ont les significations suivantes:

- *Option d'optimisation –O0 (par défaut)*

Aucune optimisation n'est effectuée par le compilateur. C'est la valeur par défaut. On l'utilise pour faire du débogage.

- *Option d'optimisation –O1*

II s'agit des optimisations locales, au niveau des instructions MIPS. Elles sont réalisées par les générateurs assembleur et code exécutable. Elles comprennent la superposition des instructions sur les réels, le remplissage des temps d'attente pour les lecture/écriture et les sauts de fonction, la sauvegarde des résultats dans les registres pour éviter des lectures/écritures en mémoire et l'élimination de code inutile car sans effet, dit code *mort*.

- *Option d'optimisation -O2*

II s'agit des optimisations globales qui transforment le code, et optimisent les boucles. Elles comprennent l'étude des dépendances des données à l'intérieur des procédures, la minimisation des lectures/écritures, l'analyse des boucles par déroulement, la fusion (faire plus de calcul avec les mêmes données), le déplacement des lignes de code invariantes à l'extérieur des boucles. L'option -O2 est le plus haut niveau d'optimisation sans risque de perte de précision, puisqu'elle n'active pas tout le potentiel d'optimisation, notamment les opérations sur les nombres en virgule flottante [EHM05].

- *Option d'optimisation -O3*

Il s'agit d'optimisations "agressives" (très poussées) qui prennent plus de temps (run-time) comparées à celles du niveau précédent. Cette option active, entre autres, les optimisations sur les opérations flottantes (-OPT) et l'analyse inter-procédurale (-IPA). Cela peut entraîner des pertes de précision. Toutefois, il est préférable de l'essayer car les performances peuvent être nettement améliorées par rapport au niveau d'optimisation -O2, surtout dans le cas de nids de

boucles imbriquées pour lesquelles ce dernier est nettement moins performant que le niveau -O3 [EHM05].

- *Options funroll-loops et funroll-all-loops*

funroll-loops (resp. funroll-all-loops) est une option du compilateur C qui permet d'appliquer un unrolling à une seule boucle (resp. toutes les boucles) d'un code.

4.2 Optimisation du PMVC

4.2.1 PMVC-DNS

Afin d'optimiser le PMVC pour le format DNS, nous avons d'abord éliminé de la boucle interne j l'élément invariant *y*[i]. Nous avons aussi appliqué un unrolling avec des facteurs différents. A titre illustratif, un unrolling de facteur 4 conduit à l'algorithme de la Figure 4.6.

```
m = N mod 4
ne = N-m
Do  i = l, N
      var = 0
    Do  j = 1, ne, 4
          var = var+A[i][j] * x[j] +A[i][j+1] * x[j+1] +A[j][j+2] * x[j+2] +
          +A[i][j+3] * x[j+3]
    EndDo
    Do  j = ne+1, N
          var = var+A[i][j] * x[j]
    EndDo
     y[i] = var
EndDo
```

Figure 4.6 PMVC-DNS avec unrolling de facteur 4

4.2.2. PMVC-CSR

L'optimisation du PMVC relatif au format CSR consiste, tout d'abord, à éliminer l'élément invariant *y*[i] de la boucle interne j. Dans le but d'éviter le test sur l'expression IA[i+1]-1 pour chaque itération de j, nous remplaçons cette expression par un scalaire. De plus, nous appliquons un unrolling de facteur 4. L'algorithme résultant est donné par la Figure 4.7.

```
Do  i=1,N
    var=0 ; m=( IA[i+1]-IA[i] )mod 4 ; pos=IA[i]+m-1
    Do  j=IA[i], pos
        var=var+ 𝒜[j] * x[JA[j]]
    EndDo
    pt=IA[i+1]-1
    Do  j=pos+1, pt, 4
        var=var+ 𝒜[j] * x[JA[j]] + 𝒜[j+1] * x[JA[j+1]] +
        + 𝒜[j+2] * x[JA[j+2]] + 𝒜[j+3] * x[JA[j+3]]
    EndDo
    y[i]=var
EndDo
```

Figure 4.7 PMVC optimisé pour le format CSR

4.2.3 PMVC-COO

L'algorithme optimisé du PMVC-COO est décrit par la Figure 4.8. Nous l'avons obtenu à partir de celui de la Figure 4.3 en remplaçant, par un scalaire, l'accès indirect IA[i] qu'on utilise deux fois dans la même itération. Ensuite, nous avons appliqué un unrolling de facteur 4.

```
m=NZ mod 4 ; ne=NZ-m
Do i=1,ne, 4
    u=IA[i] ; u1=IA[i+1] ; u2=IA[i+2] ; u3=IA[i+3] ; y[u]=y[u]+ 𝒜[i] * x[JA[i]]
    x[u1]=y[u1]+ 𝒜[i+1] * x[JA[i+1]] ; y[u2]=y[u2]+ 𝒜[i+2] * x[JA[i+2]]
    y[u3]=y[u3]+ 𝒜[i+3] * x[JA[i+3]]
EndDo
Do i=ne+1,NZ
    u=IA[i]  ;  y[u]=y[u]+ 𝒜 [i] * x[JA[i]]
EndDo
```

Figure 4.8 PMVC optimisé pour le format COO

4.2.4 PMVC-BND

Dans cet algorithme, nous remplaçons les expressions ML+MU+1, *y*[i], N-1 et i-ML par des scalaires. Puis, nous appliquons un unrolling de facteur 4. On obtient l'algorithme de la Figure 4.9.

```
L= ML+MU+1 ; m=L mod 4 ; Ne=L – m ;
Do  i=1,N
    var =0 ; w=i-ML-1
    Do  j=1, Ne, 4
        v=w+j
        var=var+ABD[i][j] * x[Min(N,Max(1,v))] +ABD[i][j+1] * x[Min(N,Max(1, v +1))]
            +ABD[i][j +2] * x[Min(N,Max(1, v +2))] +ABD[i][j+3] * x[Min(N,Max(1, v +3))]
    EndDo
    Do  j = Ne+1, L
        v=w+j ; var=var+ABD[i][j] * x[Min(N,Max(1, v))]
    EndDo
    y[i]=var
EndDo
```

Figure 4.9 PMVC optimisé pour le format BND

5. Etude expérimentale pour les quatre formats

5.1 Environnements de travail

Nous avons utilisé trois machines cibles dont les caractéristiques sont décrites ci-dessous.

- **Machine cible 1. PC Intel dual-core**

C'est un PC de marque Acer avec un processeur Intel Pentium Dual-Core de fréquence 1.8 GHz. La capacité de la mémoire cache est 1024 Ko. La RAM est de taille 1 Go. La vitesse du Bus mémoire est 800 MHz. Le système d'exploitation est LINUX (version Fedora Core 7). Le compilateur utilisé est gcc.

- **Machine cible 2. Un processeur d'un biprocesseur AMD quad-core**

La machine est un biprocesseur quad-core (2 processeurs x 4 coeurs) AMD Opteron(tm), processeur 8347 HE, la fréquence horloge d'un cœur est de 1 GHz, la taille du cache est 512 Ko, la RAM est de taille 4Go, le système d'exploitation est GNU-Linux, le compilateur installé est le gcc 4.3. La machine est installée au laboratoire Prism de l'Université de Versailles.

- **Machine cible 3. Station de travail SUN dual-core**

C'est une station de travail SUN SPARC64 dual-core où chaque coeur possède une fréquence de 1.3 GHz. La mémoire est de taille 4 Go. Le système d'exploitation est SUN. Le

langage utilisé sur cette machine est le C. La machine est installée à la Faculté des Sciences de Tunis (Département de Physique).
Précisons que toutes ces machines sont encore en cours d'usage.

Remarques : autres optimisations

Le langage C utilisé permet d'appliquer au PMVC d'autres types d'optimisation. En effet, afin de manipuler des matrices de grande taille, nous avons adopté une allocation dynamique de la mémoire pour implémenter les structures de données dont nous avions besoin. A titre d'exemple, pour une matrice A d'ordre N à deux dimensions, nous allouons dynamiquement un tableau à une dimension A; l'élément A[i][j] correspond ainsi à l'élément A[i∗N+j] (i,j=0,..,N-1). Dans la boucle interne j des algorithmes PMVC-DNS et PMVC-BND (voir Figures 4.6 et 4.9), l'expression i∗N sera considérée comme un invariant, ce qui permet son remplacement par un scalaire. De plus, nous avons utilisé la notation ++ i plutôt que i++. En effet, i++ stocke la valeur i dans une variable temporaire, puis évalue l'expression contenant i en utilisant la variable temporaire, et enfin, elle incrémente i. Tandis que ++i incrémente i et évalue l'expression contenant i [EHM05].

5.2 Résultats Expérimentaux

Nous avons mesuré les variations des temps CPU pour les différentes versions du PMVC en fonction du facteur d'unrolling appliqué à la boucle interne. Nous avons également appliqué les options d'optimisation du compilateur de la machine. Nous avons utilisé un échantillon de matrices de tailles (notées N) et de densités différentes (notées d). Ces matrices ont été soit générées aléatoirement, soit téléchargées à partir du site Matrix Market [Mat07]. Un exemple de matrice creuse de Matrix Market se dénomme Sherman3. Ses caractéristiques sont données dans la Figure 4.10.
C'est une matrice bande de taille N=5005, de densité d=0.08% avec comme largeurs de demi-bande (ML pour inférieure et MU pour supérieure) ML=MU=386, la densité de la bande, notée db, étant égale à 0.15%.

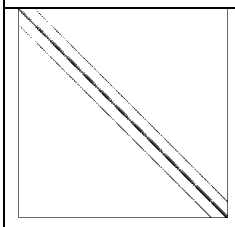
Structure	Nom	Taille	Densité	ML=MU	Densité Bande
	Sherman3	5005	0.08%	386	0.15%

Figure 4.10 Matrice téléchargée de Matrix Market

Précisons que **funroll-loop** (notée 'fun' dans les Figures 4.11 à 4.26) est une option du compilateur C qui applique un unrolling à une seule boucle ; **funroll-all-loops** (notée 'funall' dans les Figures 4.11 à 4.26) est une option qui applique un unrolling à toutes les boucles, **no** indique que le PMVC est exécuté en n'appliquant au code aucune optimisation manuelle ; **nu_rs** indique que le PMVC est optimisé par le remplacement scalaire et sans application d'unrolling ; **u2, u3, …, u256** sont les facteurs d'unrolling. Ajoutons que le temps CPU correspond à une moyenne de 10 exécutions différentes.

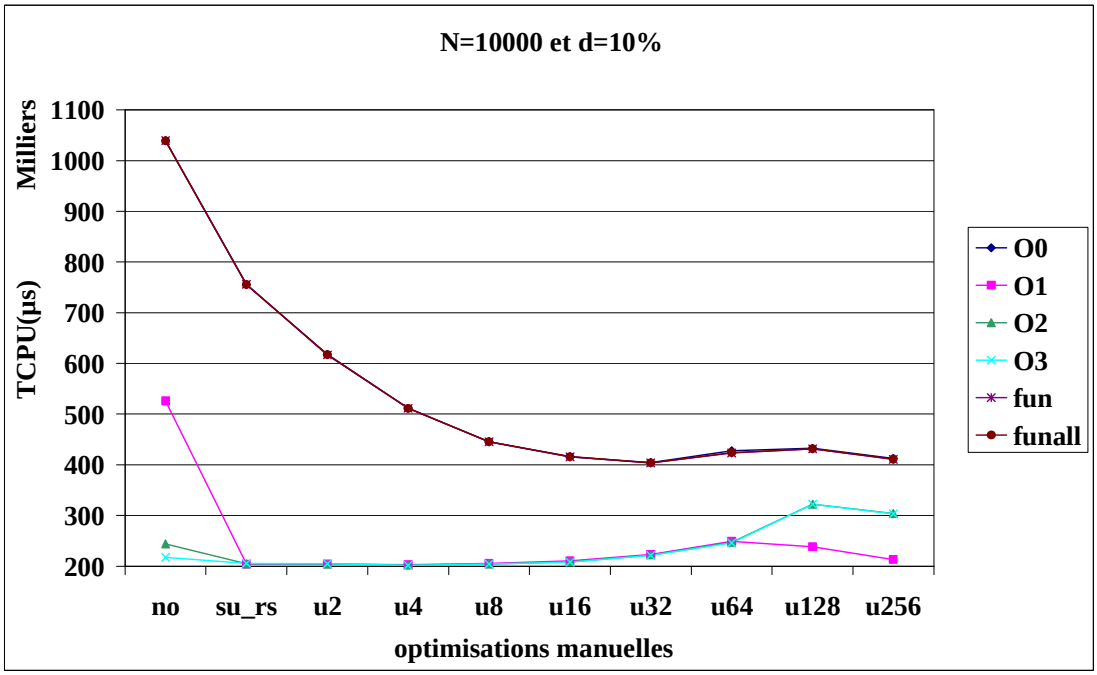
Nous définissons les accélérations comme suit.

- Accélération notée O_0/On, induite par les options –On comme étant le rapport de la durée d'exécution du PMVC en utilisant $–O_0$ (pas d'optimisation) sur la durée d'exécution obtenue lorsqu'une option –On (n=1, 2 ou 3) est utilisée.
- Accélération notée su_rs/u, induite par l'unrolling manuel comme étant le rapport de la durée d'exécution du PMVC en utilisant uniquement l'optimisation par le remplacement scalaire sur la durée obtenue en utilisant, de plus, l'unrolling manuel.
- Accélération notée no/su_rs, correspondant au rapport de la durée d'exécution du PMVC sans aucune optimisation manuelle sur la durée d'exécution du PMVC en utilisant le remplacement scalaire.

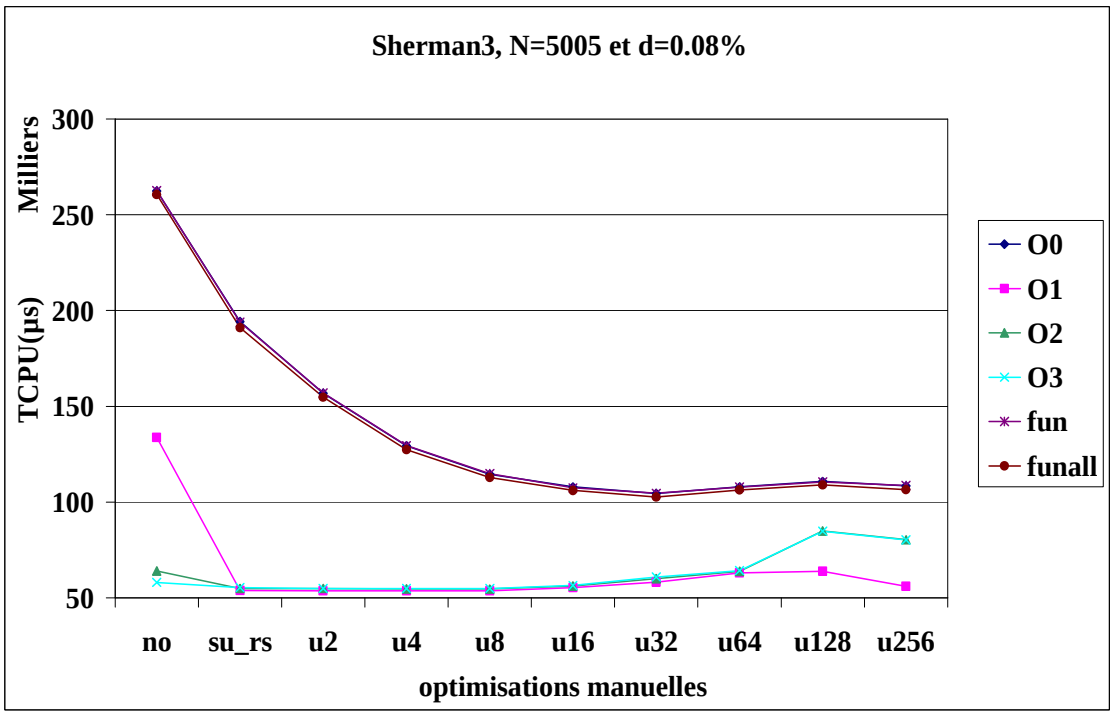
Par souci de concision, les résultats expérimentaux présentés sont restreints à deux matrices : (i) une matrice de taille N=10000 et de densité d=0.01 générée aléatoirement, (ii) la matrice Sherman3 sus décrite. Nous présentons d'abord des courbes d'évolution puis nous passons à l'analyse des résultats. Précisons qu'afin de ne pas allonger outre mesure cette section, nous nous donnons quelques extraits des expérimentations. La totalité des résultats obtenus figurent aux annexes A, B, C et D.

5.2.1 Résultats pour le PMVC-DNS

- *Sur la machine 1*



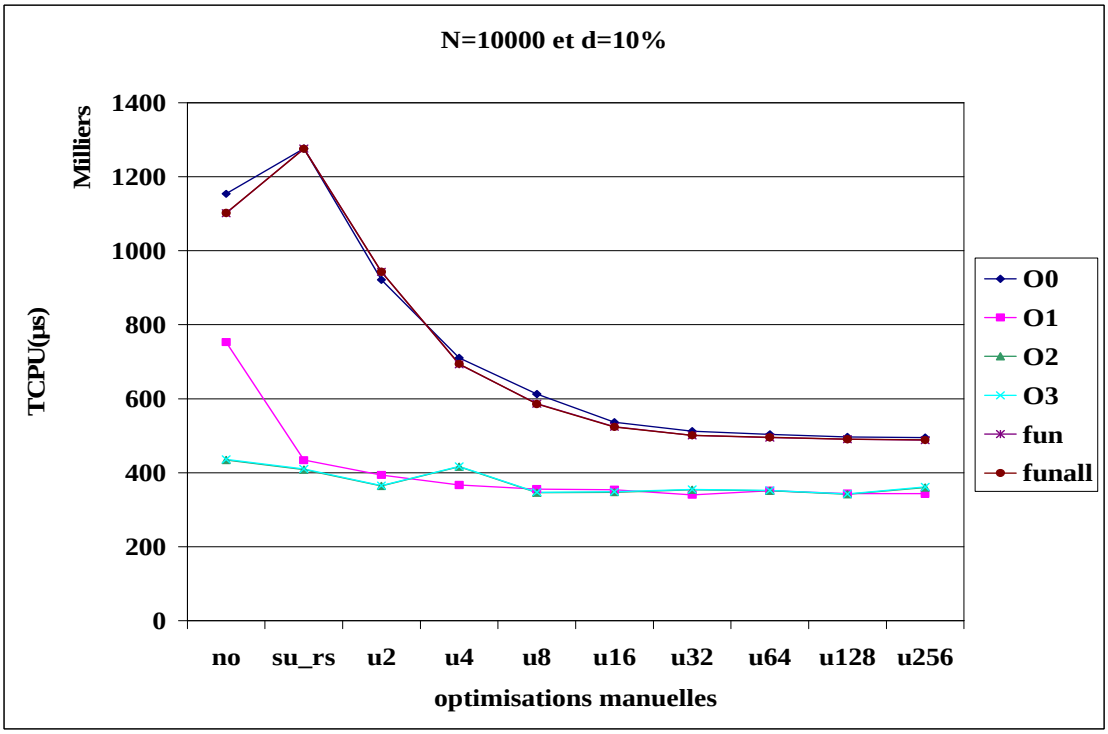
(a) Matrice générée aléatoirement



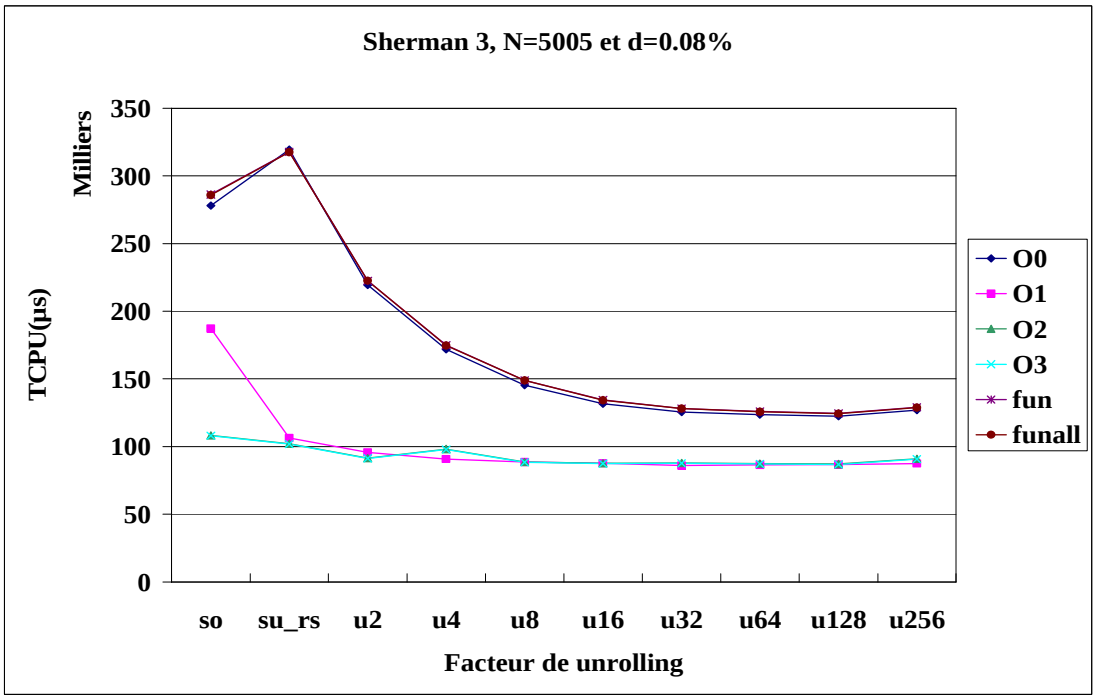
(b) Matrice Sherman3

Figure 4.11 Performances du PMVC-DNS sur la machine 1 (PC Intel)

- *Sur la machine 2*



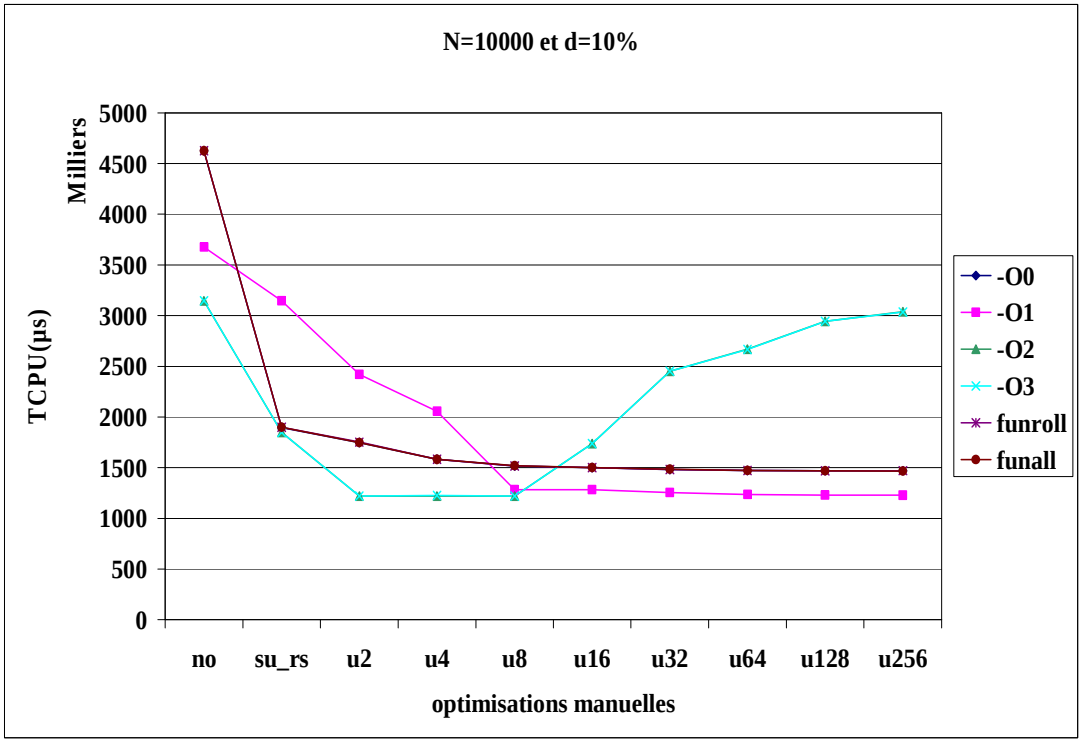
(a) Matrice générée aléatoirement



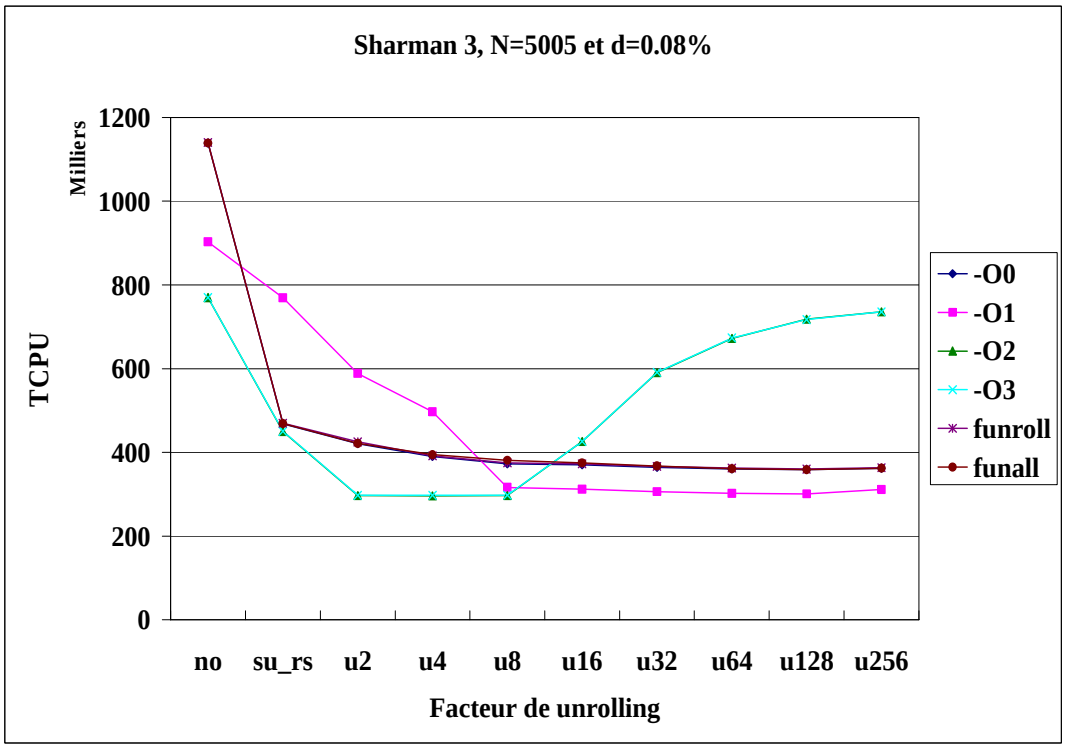
(b) Matrice Sherman3

Figure 4.12 Performances du PMVC-DNS sur la machine 2 (processeur AMD)

- *Sur la machine 3*



(a) Matrice générée aléatoirement



(b) Matrice Sherman3

Figure 4.13 Performances du PMVC-DNS sur la machine 3 (SUN)

Commentaires

- En général, les options **–On** (**n**=1,2,3) fournissent des performances équivalentes et permettent une amélioration i.e. une *accélération* O0/On qui a atteint 4.78 (i.e. le temps est divisé par ce facteur) sur la machine 1 (voir Annexe A, Figure 1.(b)), 3.14 sur la machine 2 (Annexe A, Figure 5.(b)) et 1.48 sur la machine 3 (Annexe A, Figure 11.(b)). De la même manière, le remplacement scalaire améliore les performances avec une accélération autour de 2.5 pour les matrices testées et ce, pour les machines 1 (Annexe A, Figure 4) et 3 (Annexe A, Figure 12). Dans le cas particulier de la machine 2, lorsque nous n'utilisons pas l'option du compilateur **–On**, le remplacement scalaire peut entraîner une dégradation de performances avec un facteur qui peut atteindre 1.15 i.e. multiplication du temps par ce facteur (voir Annexe A, Figure 8).
- D'un autre côté, les options **funroll-loop** et **funroll-all-loops** n'ont aucun impact sur les performances du PMVC.
- Pour la machine 1, lorsque aucune option du compilateur n'est utilisée, nous remarquons que l'unrolling manuel améliore les performances avec une accélération qui atteint 1.87 (Figure 4.14 et Annexe A, Figure 1.(c)). Néanmoins, les améliorations apportées par les options **–On** sont meilleures et masquent celles induites par l'unrolling manuel.

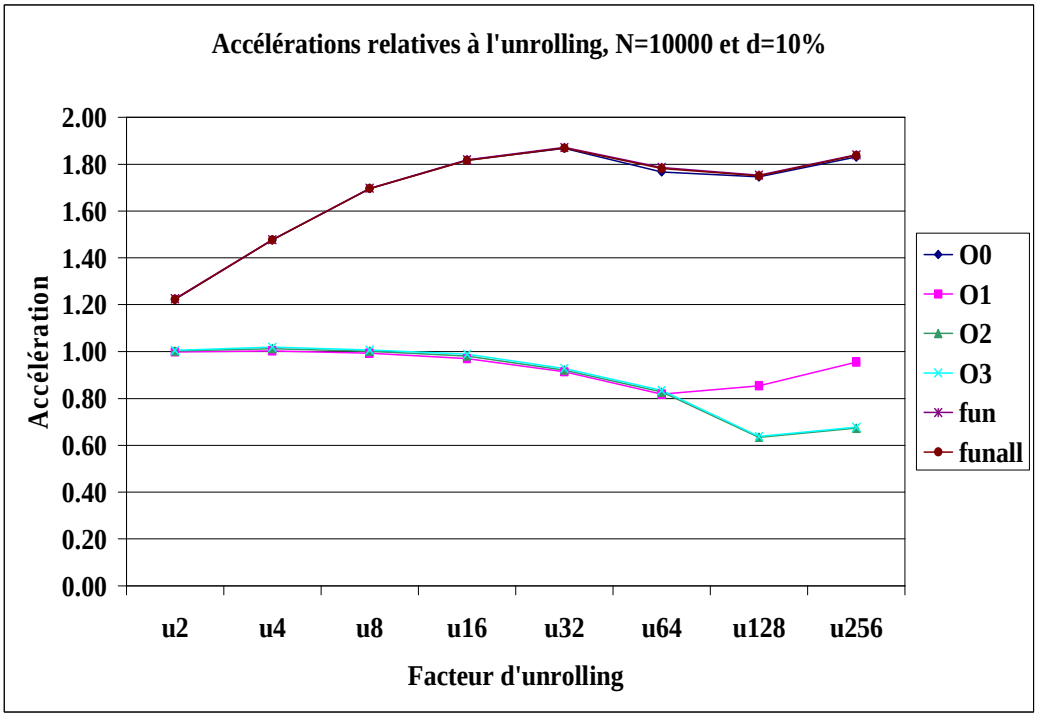


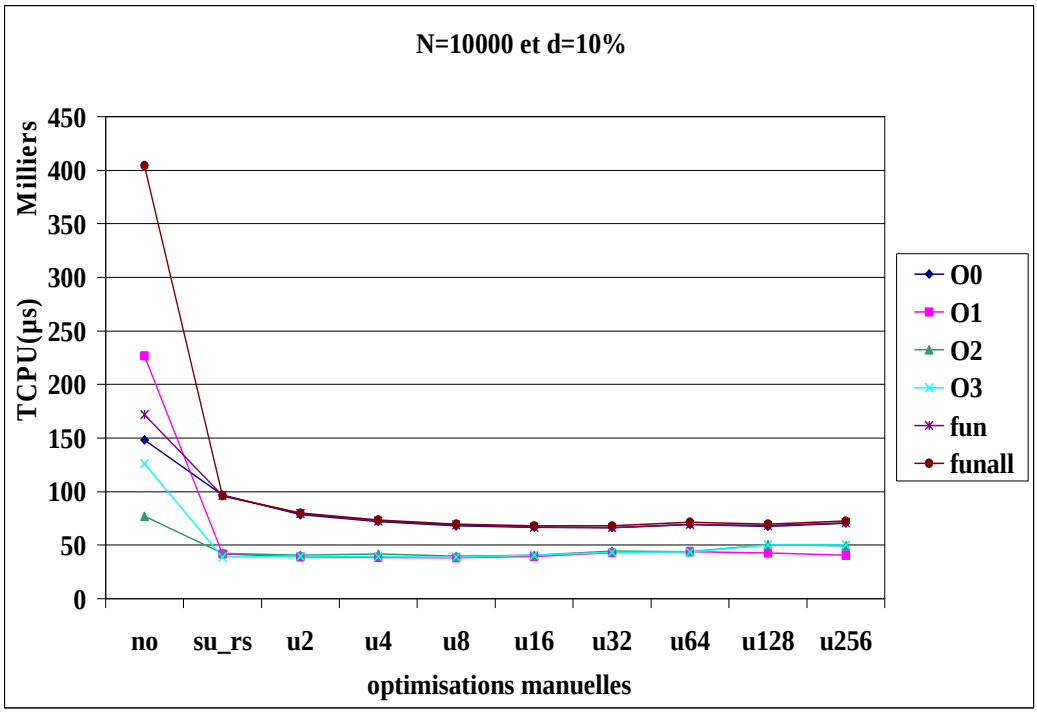
Figure 4.14 Accélérations obtenues par l'unrolling manuel sur la machine 1

- En ce qui concerne la machine 2, l'unrolling manuel permet, dans la plupart des cas, d'améliorer les performances avec une accélération qui atteint 2.62 (Annexe A, Figure 7.(c)).
- Pour la machine 3, en cas de non utilisation des options -O2 et -O3 du compilateur, nous remarquons que l'unrolling manuel améliore les performances avec une accélération qui atteint 2.66 (Annexe A, Figure 10.(c)).

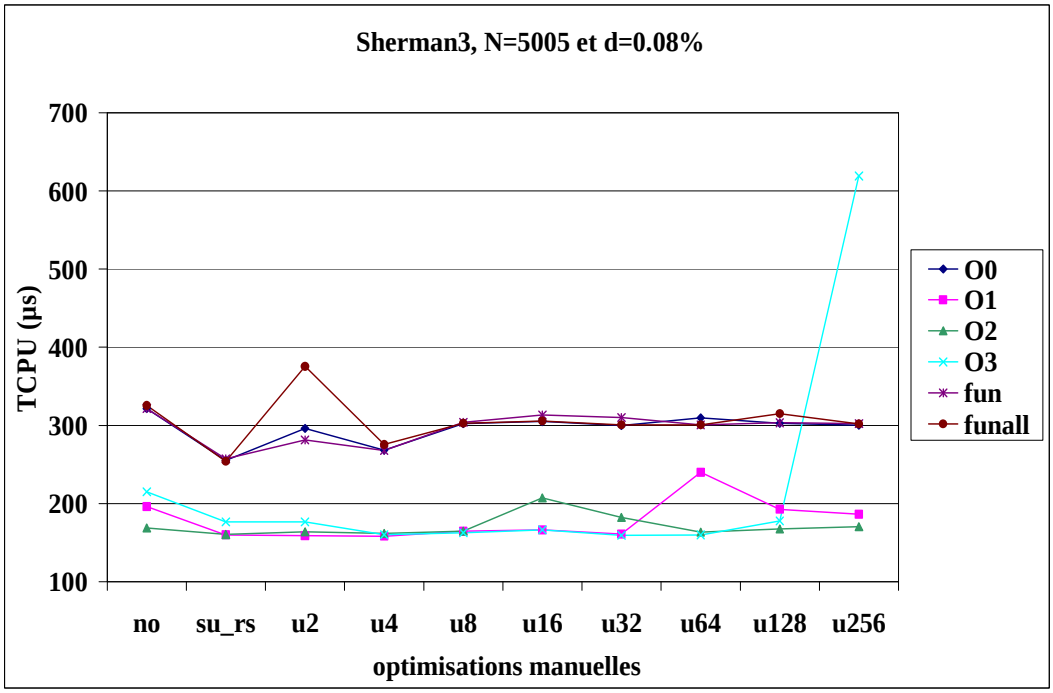
En conclusion, sur les machines 1 et 3, l'unrolling manuel combiné avec les options –On peut dégrader les performances du PMVC-DNS. La Figure 4.14, donne une idée sur l'accélération correspondant à l'utilisation de l'unrolling manuel.

5.2.2 Résultats pour le PMVC-CSR

- *Sur la machine 1*



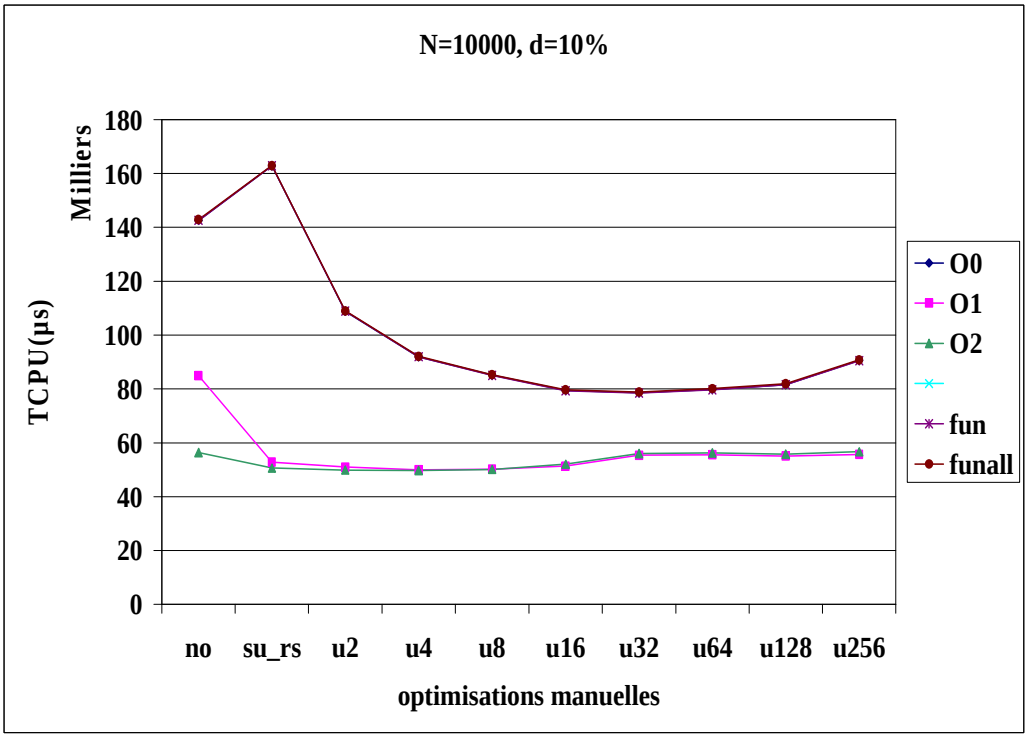
(a) Matrice générée aléatoirement



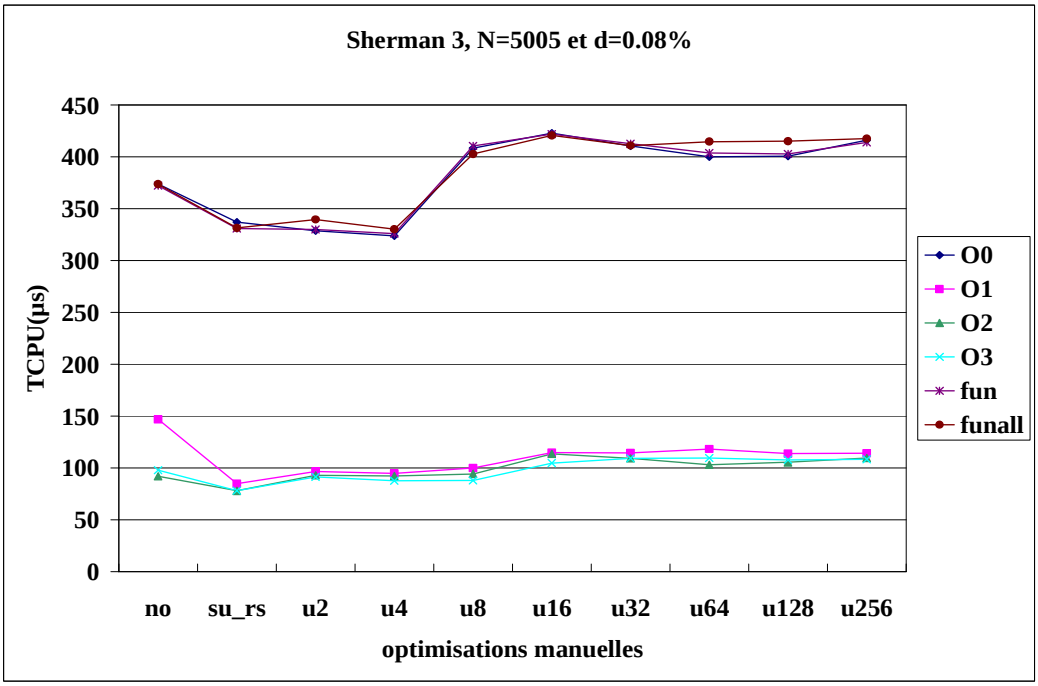
(b) Matrice Sherman3

Figure 4.15 Performances du PMVC-CSR sur la machine 1 (PC Intel)

- *Sur la machine 2*



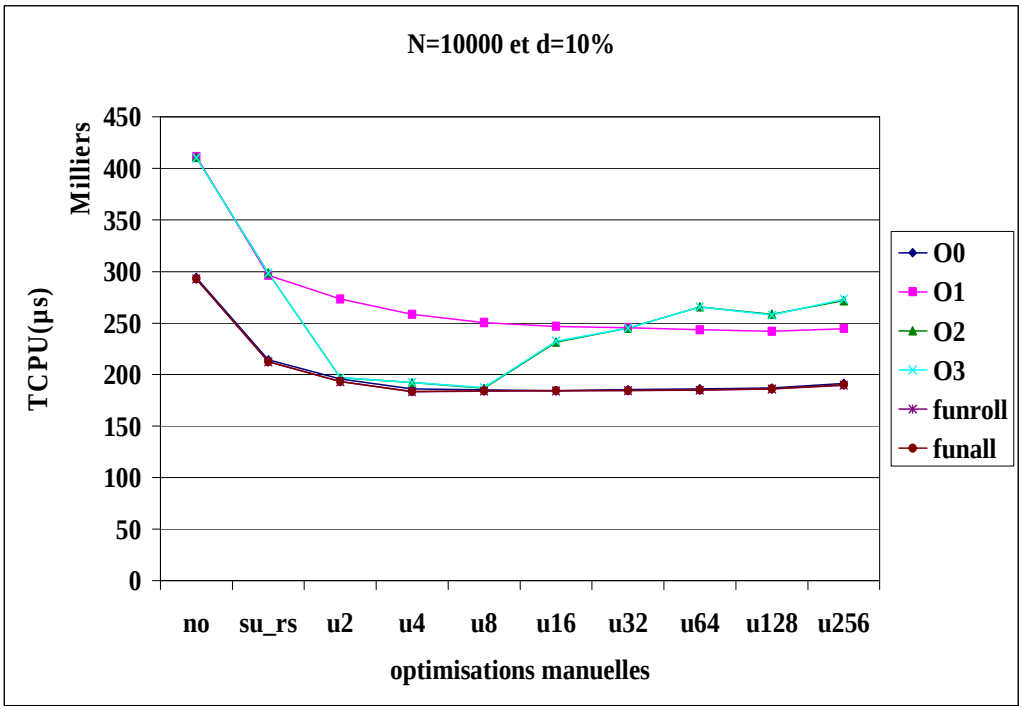
(a) Matrice générée aléatoirement



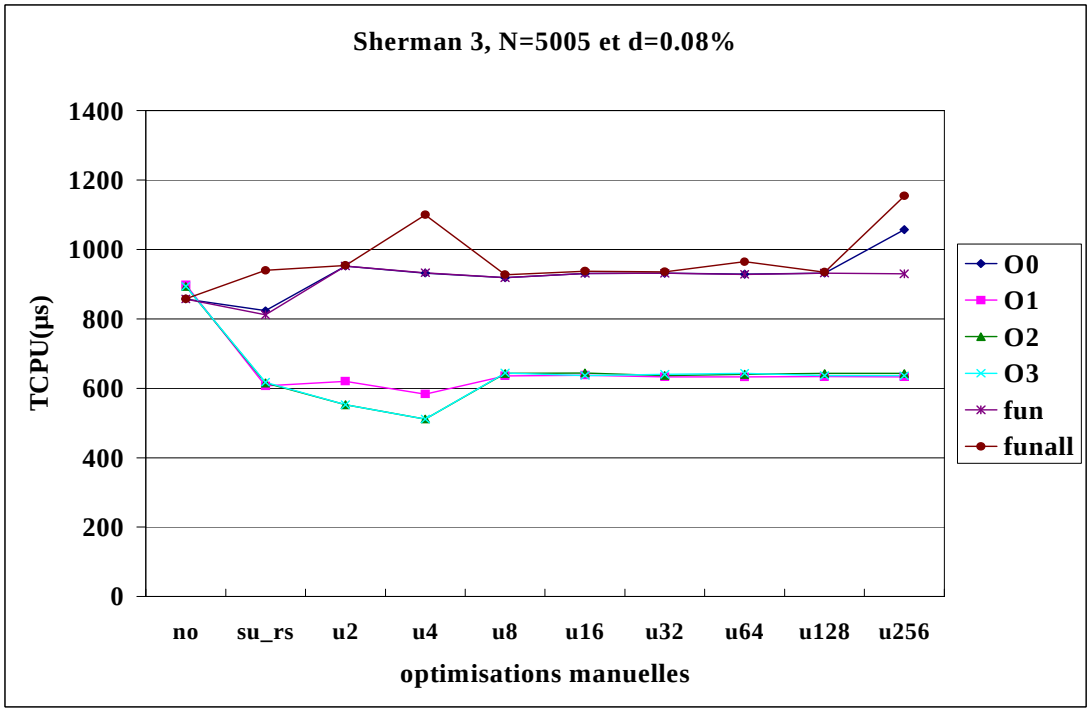
(b) Matrice Sherman3

Figure 4.16 Performances du PMVC-CSR sur la machine 2 (processeur AMD)

- *Sur la machine 3*



(a) Matrice générée aléatoirement



(b) Matrice Sherman3

Figure 4.17 Performances du PMVC-CSR sur la machine 3 (SUN)

Commentaires

- Les options **–On** (**n**= 1, 2, 3) permettent une amélioration avec une accélération qui atteint 2.79 sur la machine 1 (Annexe B, Figure 2.(b)) et 4.64 sur la machine 2 (Annexe B, Figure 7.(b)), mais dégrade les performances sur la machine 3, en particulier lorsque d ≥ 1% (Figure 4.18 et Annexe B, Figure 9.(b)) et ce, avec un facteur qui atteint 1.44 (Annexe B, Figure 10.(b)).
- Le remplacement scalaire est une optimisation très intéressante, en particulier sur la machine 1 où elle permet une nette amélioration avec une forte accélération qui atteint 42.56 (Annexe B, Figure 4). Sur la machine 2 (resp. 3) elle fournit une amélioration qui atteint 1.8 (Annexe B, Figure 8) (resp. 2.58 (Annexe B, Figure 12)).
- Sur les trois machines, les améliorations apportées par l'unrolling manuel ont généralement lieu lorsque les options **–On** ne sont pas utilisées lors de la compilation, en particulier, pour les matrices de densité d ≥ 1%. Dans ce cas, l'unrolling permet d'atteindre une accélération de 1.49 sur la machine 1 (Annexe B, Figure 2.(c)), 2.17 sur la machine 2 (Annexe B, Figure 5.(c)) et 1.16 sur la machine 3 (Annexe B, Figure 9.(c)).

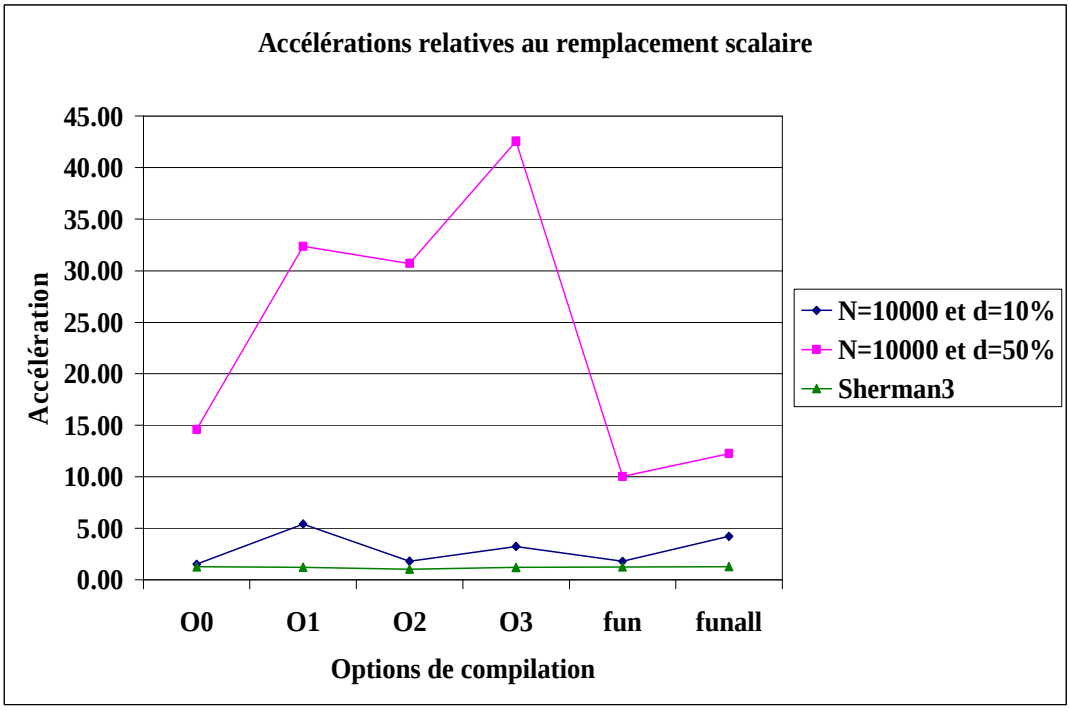
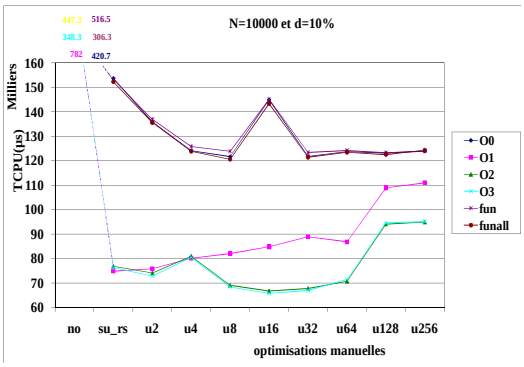


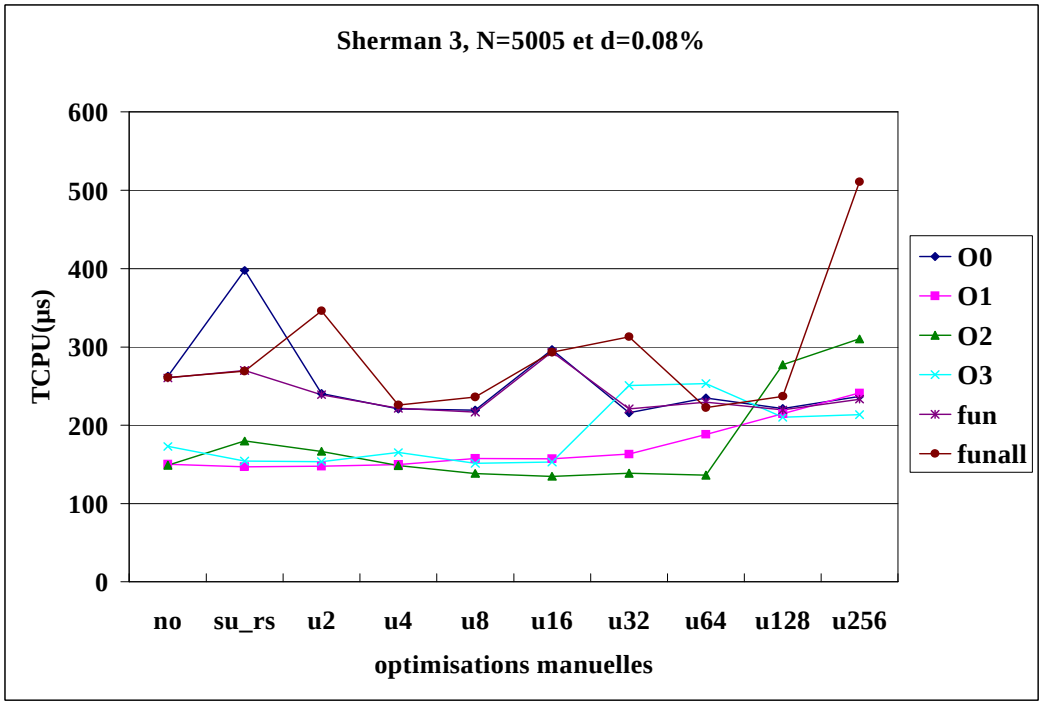
Figure 4.18 Accélérations obtenues par le remplacement scalaire sur la machine 1

5.2.3 Résultats pour le PMVC-COO

- *Sur la machine 1*



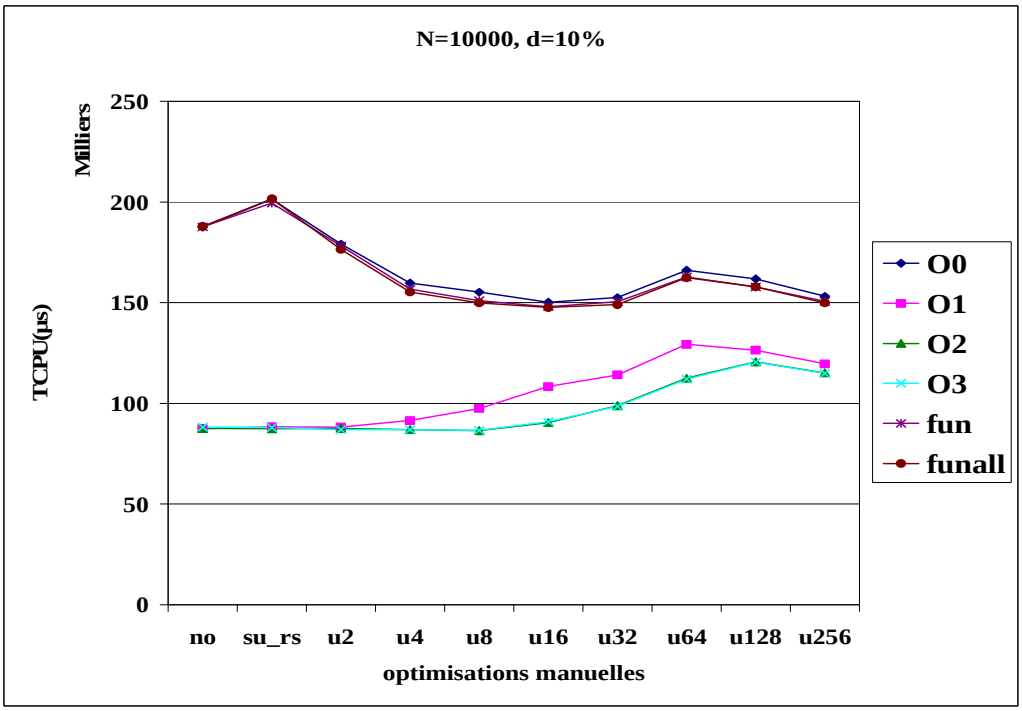
(a) Matrice générée aléatoirement



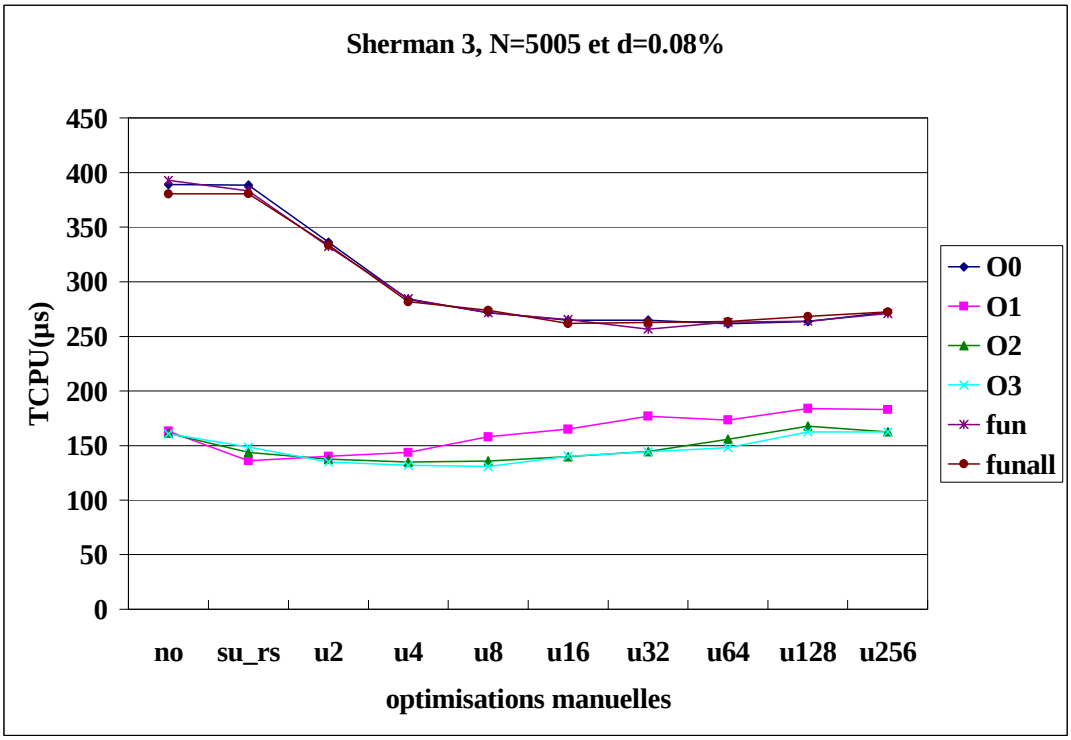
(b) Matrice Sherman3

Figure 4.19 Performances du PMVC-COO sur la machine 1 (PC Intel)

- *Sur la machine 2*



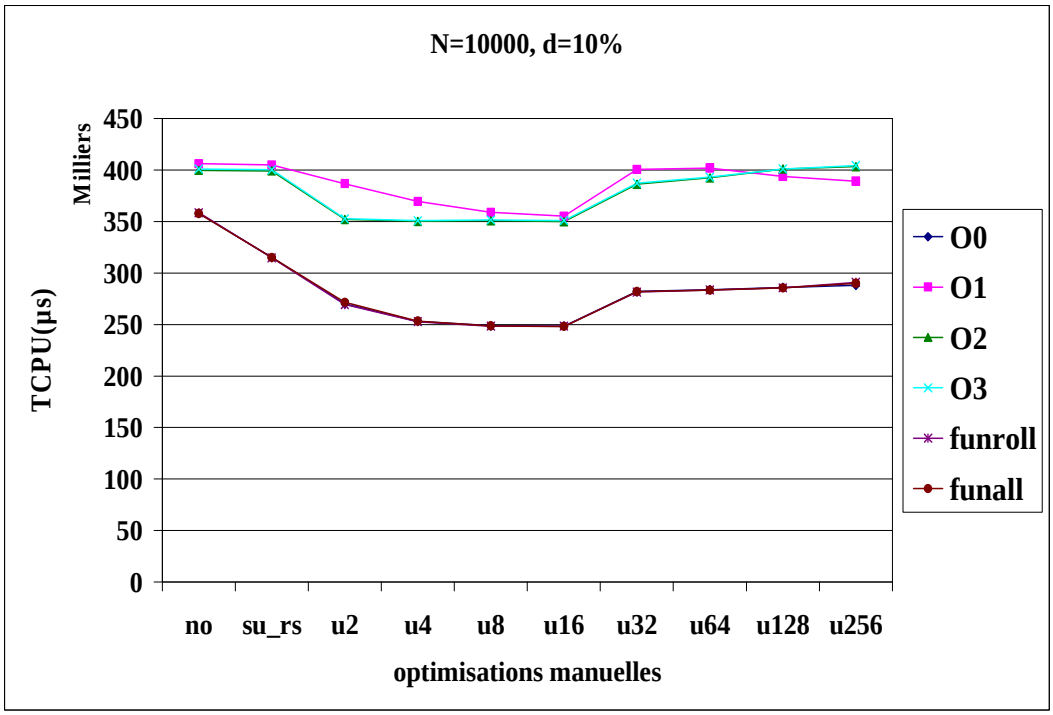
(a) Matrice générée aléatoirement



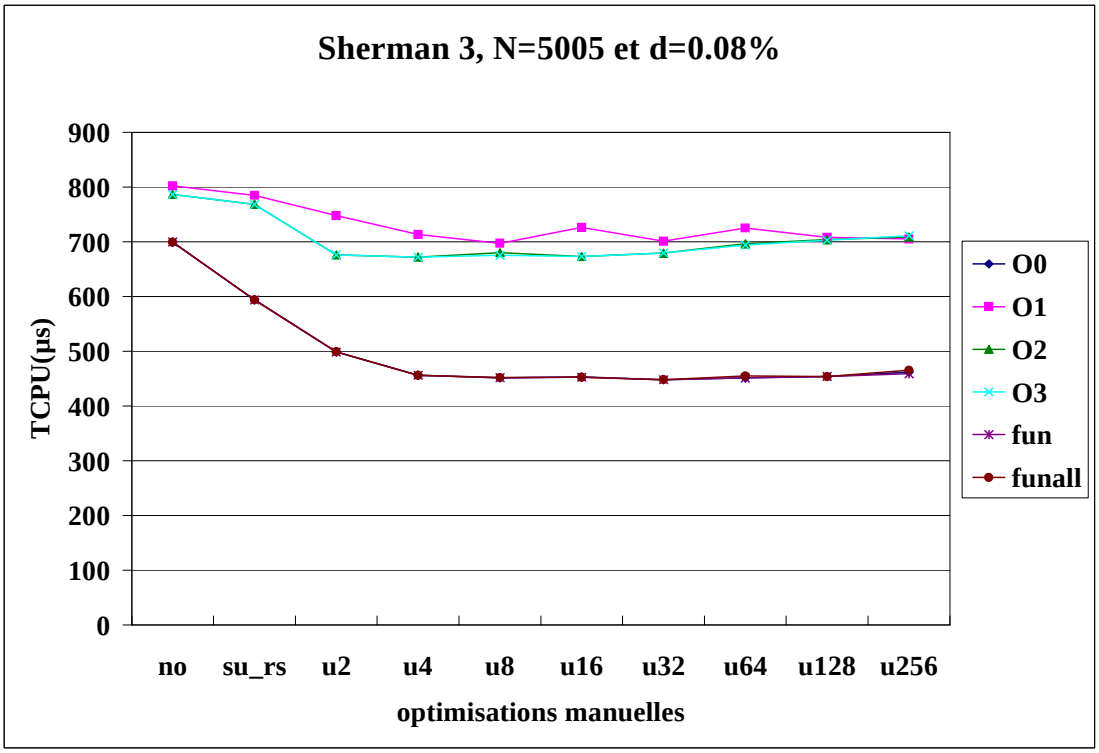
(b) Matrice Sherman3

Figure 4.20 Performances du PMVC-COO sur la machine 2 (processeur AMD)

- *Sur la machine 3*



(a) Matrice générée aléatoirement



(b) Matrice Sherman3

Figure 4.21 Performances du PMVC-COO sur la machine 3 (SUN)

Commentaires

- Les options **–On** (**n**=1, 2, 3) fournissent des améliorations avec une accélération atteignant 2.71 (Annexe C, Figure 3.(b)) sur la machine 1 et 2.85 sur la machine 2 (Annexe C, Figure 7.(b)). Sur la machine 3, elles causent une dégradation avec un facteur qui atteint 1.61 (Annexe C, Figure 11.(b)).
- Sur la machine 1, l'amélioration induite par le remplacement scalaire est négligeable pour les matrices de faible densité. Elle devient importante lorsque la densité est supérieure ou égale à 10% et permet d'atteindre la valeur 45.93 sur la machine 1 (Annexe C, Figure 4). Sur la machine 2, par contre, le remplacement scalaire n'apporte pas d'amélioration mais plutôt, une légère dégradation dans certains cas. Sur la machine 3 le remplacement scalaire peut améliorer les performances avec une accélération de 1.18 (Annexe C, Figure 12).
- Les performances induites par les options **funroll-loop** et **funroll-all-loops** sont généralement similaires à celles obtenues lorsqu'aucune option de compilation n'est utilisée. L'unrolling manuel permet de faibles améliorations en cas de non utilisation des options du compilateur. Ces améliorations permettent d'atteindre une accélération de 1.81 sur la machine 1 (Annexe C, Figure 3.(c)), 1.49 sur la machine 2 (Annexe C, Figure 7.(c)) et 1.33

sur la machine 3 (Annexe C, Figure 11.(c)). L'unrolling manuel avec un facteur élevé peut dégrader les performances du PMVC, en particulier sur la machine 1 lorsqu'on combine l'unrolling manuel avec les options **–On** du compilateur. Cette dégradation atteint un facteur de 1.63 (Annexe B, Figure 3.(c)).

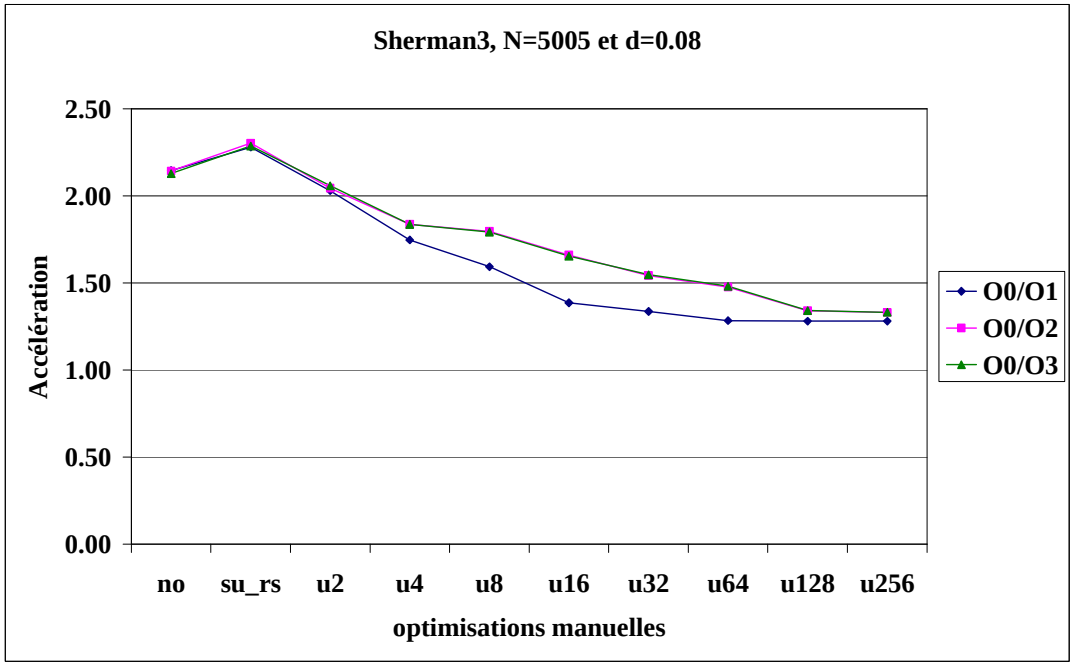
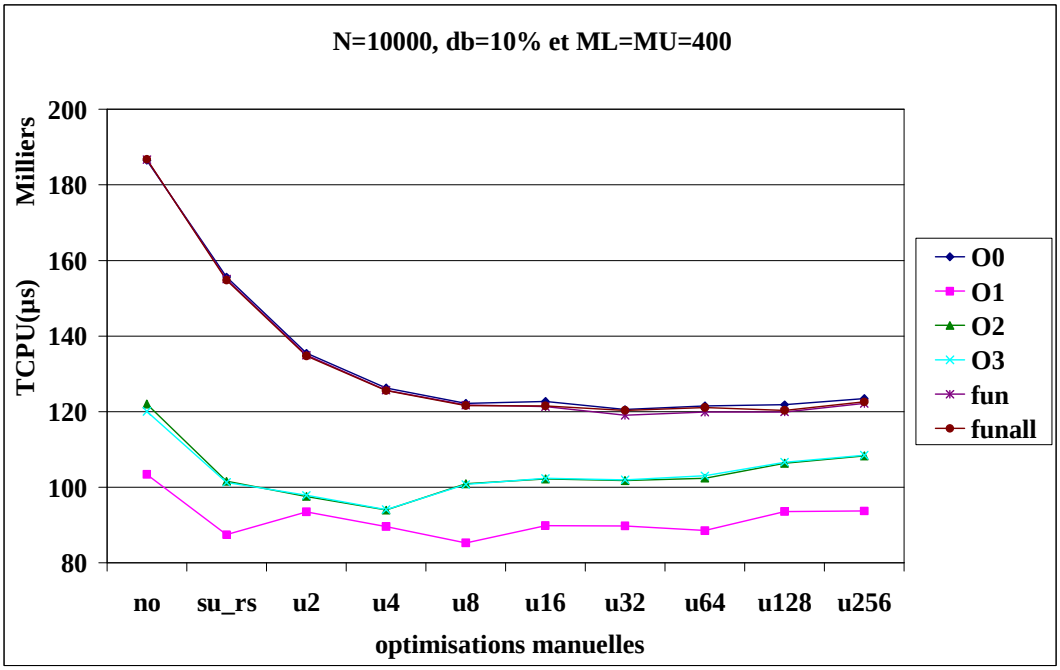


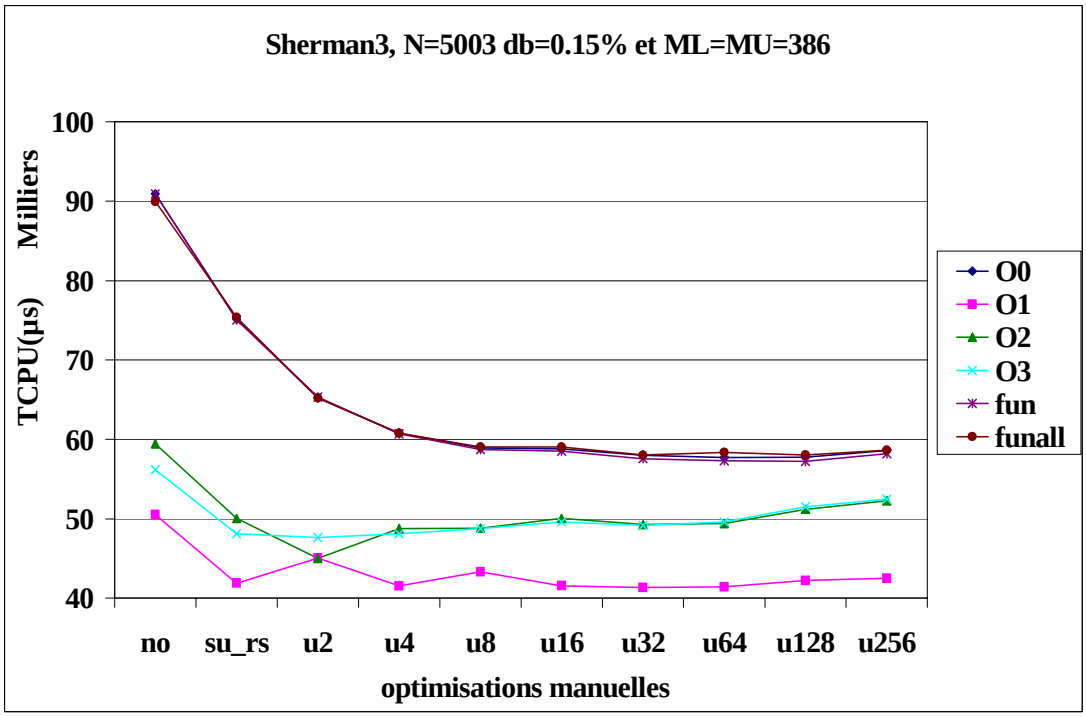
Figure 4.22 Accélérations obtenues par les options –On sur la machine 2

5.2.4 Résultats pour le PMVC-BND

- *Sur la machine 1*



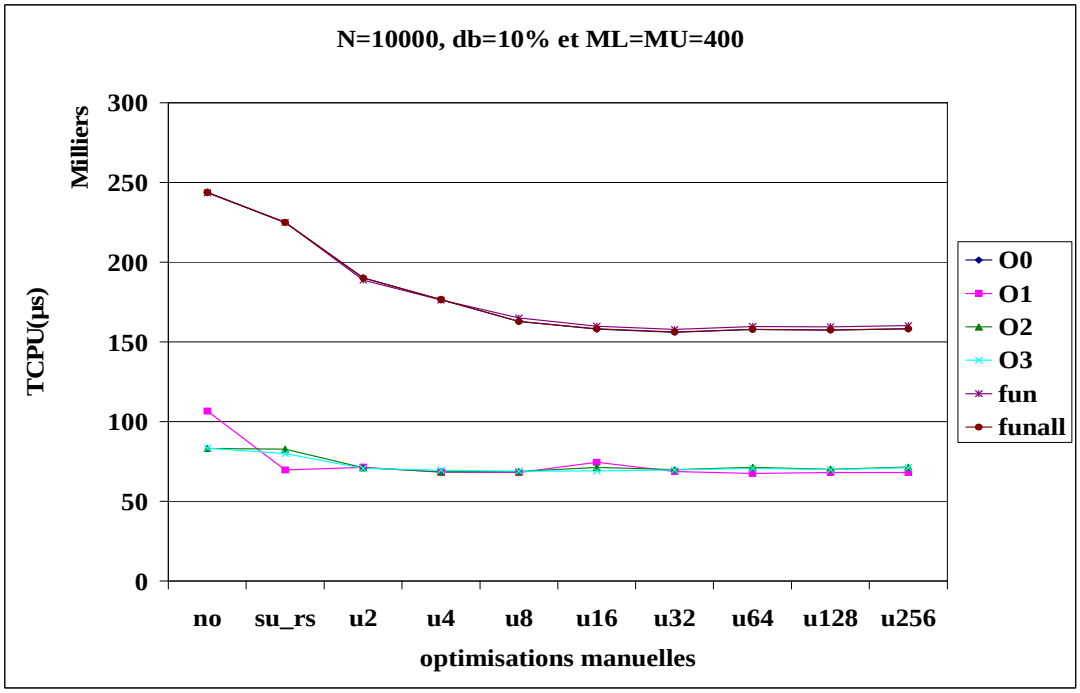
(a) Matrice bande générée aléatoirement



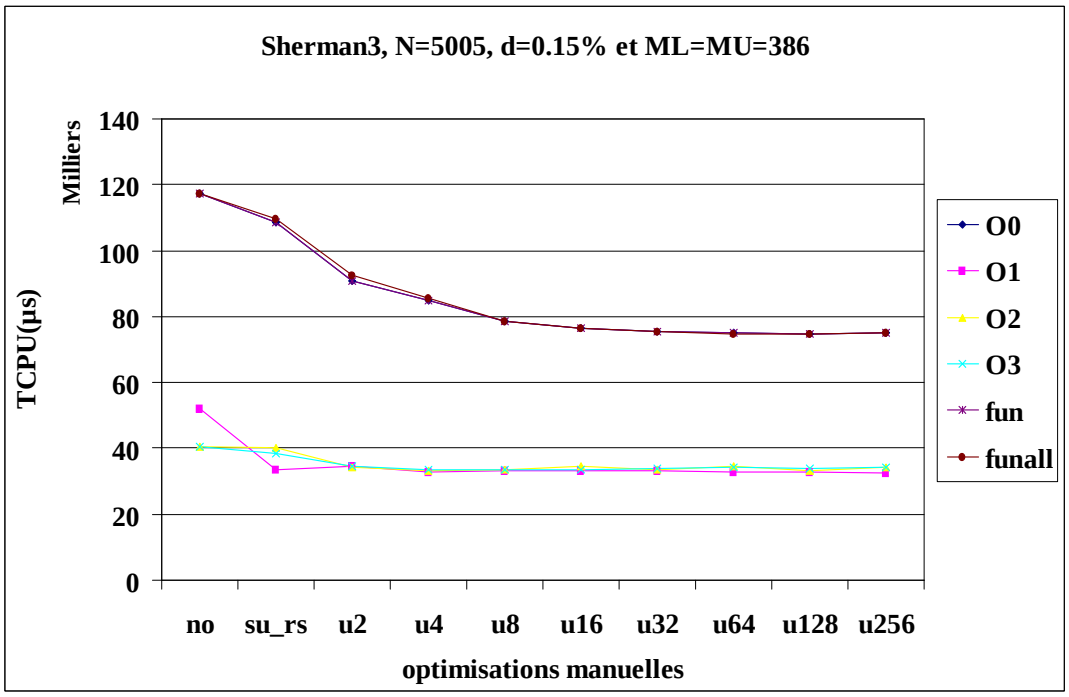
(b) Matrice Sherman3

Figure 4.23 Performances du PMVC-BND sur la machine 1 (PC Intel)

- *Sur la machine 2*



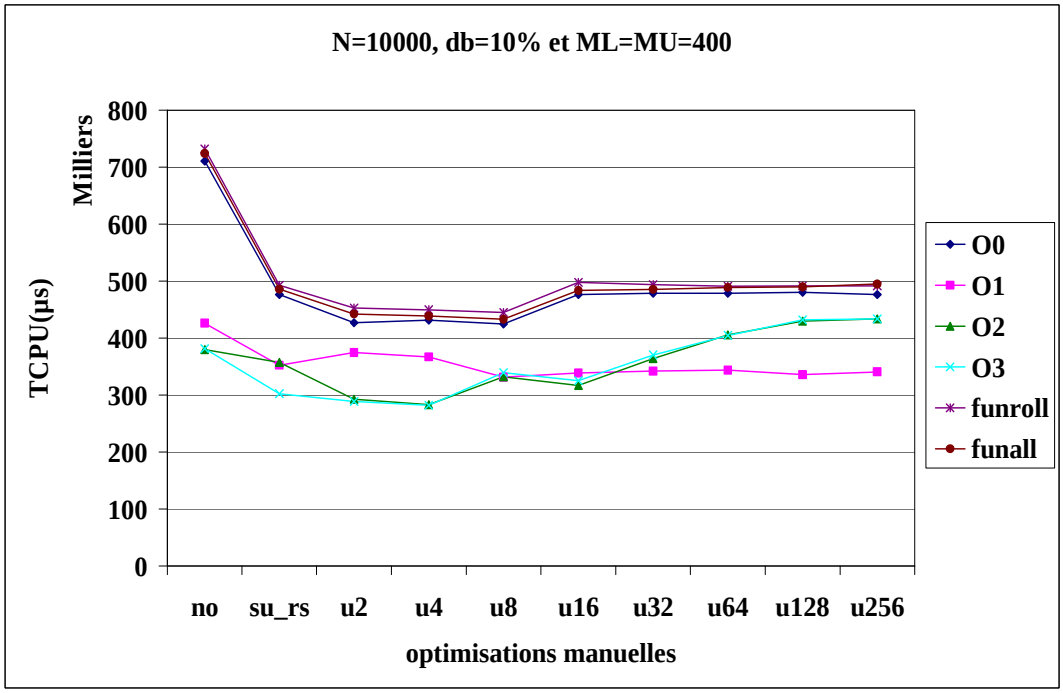
(a) Matrice bande générée aléatoirement



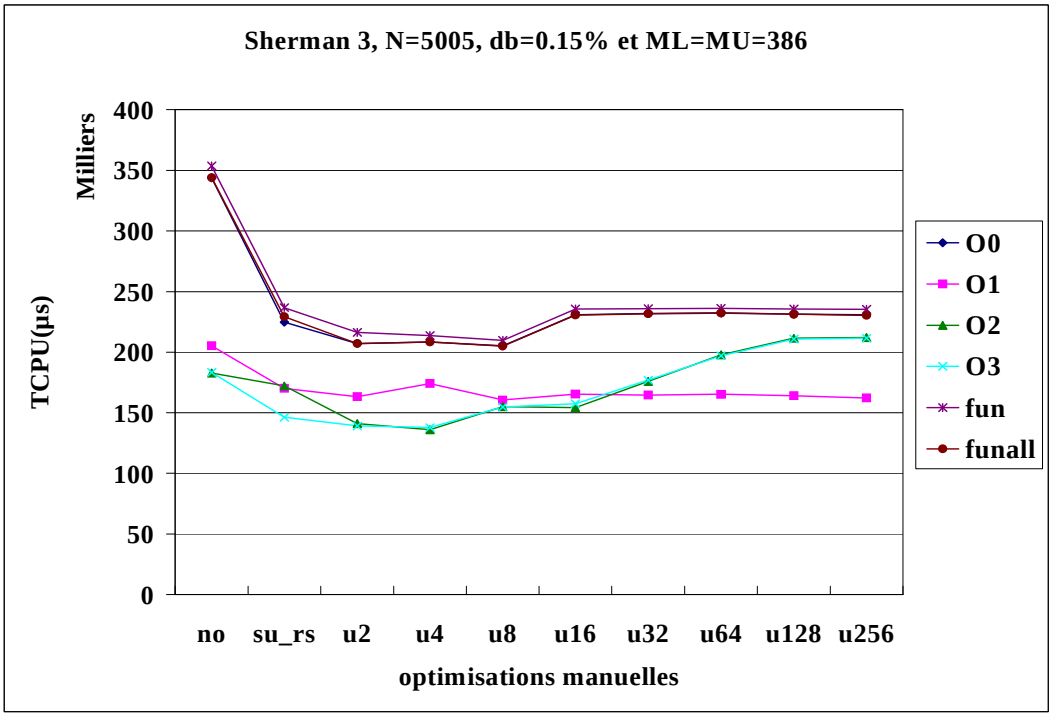
(b) Matrice Sherman3

Figure 4.24 Performances du PMVC-BND sur la machine 2 (processeur AMD)

- *Sur la machine 3*



(a) Matrice bande générée aléatoirement



(b) Matrice Sherman3

Figure 4.25 Performances du PMVC-BND sur la machine 3 (SUN)

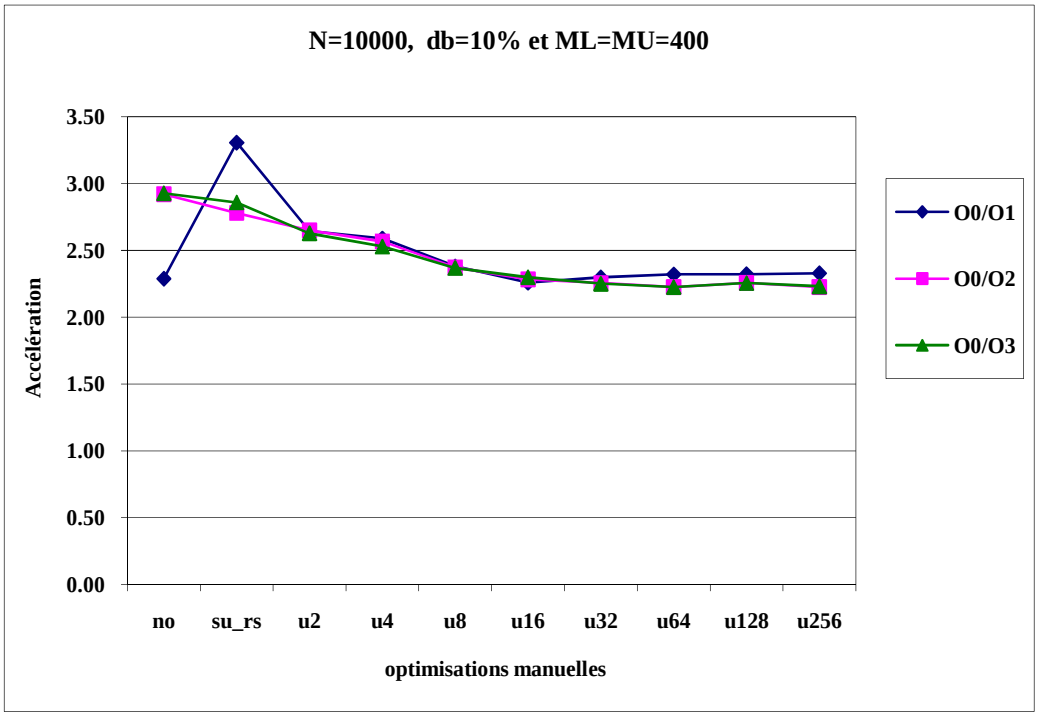


Figure 4.26 Accélérations obtenues par les options –On sur la machine 2

Commentaires

- Les options **–On** (**n**= 1, 2, 3) permettent une amélioration avec une accélération de 1.82 (Annexe D, Figure 2.(b)) sur la machine 1, 3.31 (Annexe D, Figure 6.(b)) sur la machine 2 et 1.72 (Annexe D, Figure 10.(b)) sur la machine 3.
- Le remplacement scalaire améliore les performances avec une accélération qui atteint 1.21 sur la machine 1 (Annexe D, Figure 4), 1.55 sur la machine 2 (Annexe D, Figure 8) et 1.53 sur la machine 3 (Annexe D, Figure 12).
- De la même manière que pour les formats précédents, les options **funroll-loop** et **funroll-all-loops** n'ont aucun effet. L'unrolling manuel améliore les performances lorsqu'aucune option du compilateur n'est utilisée. L'amélioration atteint une accélération égale à 1.3 (Annexe D, Figure 3.(c)) sur la machine 1, 1.46 (Annexe D, Figure 6.(c)) sur la machine 2. Ces améliorations disparaissent lorsqu'on utilise une option **–On**. Sur la machine 3, l'unrolling manuel n'apporte pas d'amélioration remarquable.

6. Récapitulation et conclusion

Les résultats expérimentaux sont récapitulés dans la Table 4.3. Pour chaque format et chaque machine, nous précisons les optimisations qui fournissent les meilleures performances pour le PMVC. Nous rappelons les options associées : Optim 1 (remplacement scalaire), Optim 2 (option –On), Optim 3 (unrolling manuel), Optim 4 (facteur d'unrolling élevé).

L'analyse récapitulative permet de conclure qu'il est intéressant d'utiliser le remplacement scalaire sur les différentes machines et ce, pour les différentes versions du PMVC. Sur les machines 1 et 2, et quelle que soit la version du PMVC, il est préférable d'utiliser, en plus, l'une des options **–On** du compilateur. Dans ce cas, l'unrolling manuel est inutile. D'un autre côté, dans les cas du PMVC-CSR et PMVC-COO sur la machine 3, l'unrolling manuel peut améliorer les performances alors que les options –On les dégradent.

Finalement, nous devons souligner que les résultats obtenus montrent que les critères d'optimisation des performances du PMVC varient d'une machine à l'autre. *Ils doivent donc être bien ajustés aux caractéristiques matérielles et logicielles* des nœuds du système distribué à grande échelle (SDGE) cible et ce, afin d'atteindre des performances globales intéressantes.

Après avoir étudié dans ce chapitre le code du PMVC, nous passons dans le chapitre suivant, à l'analyse de la distribution d'une matrice creuse utilisée comme entrée par le PMVC sur un SDGE.

		PMVC-DNS	PMVC-CSR	PMVC-COO	PMVC-BND
Machine 1 : PC Intel	**Optim 1**	***Amélioration*** *Accélération=2.54*	***Amélioration*** *Accélération=42.5*	***Amélioration*** *Cas* matrice à $d \geq 10\%$. *Accélération=46*	***Amélioration*** *Accélération=1.2*
	Optim 2	***Amélioration*** *Accélération=4.2*	***Amélioration*** *Accélération=2.5*	***Amélioration*** *Accélération=2.2*	***Amélioration*** *Accélération=1.6*
	Optim 3	***Amélioration*** *Cas*: non option -On *Accélération=1.87*	***Amélioration*** *Cas*:non option -On *Accélération=1.41*	***Faible amélioration*** *Cas*: non option -On *Accélération=1.8*	***Amélioration*** *Cas*: non option –On *Accélération=1.3*
	Optim 4	***Dégradation*** *pour –O3 Décélération=1.57*	***Dégradation*** *Ratio=3.5*	***Dégradation*** *Cas* : option -On. *Ratio=1.7*	***Stabilité***
Machine 2 : AMD	**Optim 1**	***Dégradation*** *Cas* : non option -On *Décélération =1.25* ***Amélioration*** *Cas* : option –On Accélération=2.4	***Amélioration*** *Accélération=3.16*	***Dégradation*** *Cas*: non option –On *Décélération* =1.4	***Non amélioration***
	Optim 2	***Amélioration*** *Accélération=3.8*	***Amélioration*** *Accélération=3.9*	***Amélioration*** *Accélération=2.75*	***Amélioration*** *Accélération=4.3*
	Optim 3	***Amélioration*** *Cas*: non option -On *Accélération=2.9*	***Amélioration*** *Cas*: non option -On *Accélération=1.44*	***Amélioration*** *Cas*: non option -On *Accélération=1.67*	***Amélioration*** *Cas*: non option –On *Accélération*=1.18
	Optim 4	***Stabilité***	***Dégradation*** *Ratio=1.12*	***Stabilité***	***Stabilité***
Machine 3 : SUN WS	**Optim 1**	***Amélioration*** *Accélération=2.4*	***Amélioration*** *Accélération=1.37*	***Amélioration*** *Accélération=1.18*	***Amélioration*** *Accélération=1.5*
	Optim 2	***Amélioration*** *Accélération=1.5*	***Dégradation*** *pour d >=1 Décélération =1.37* ***Amélioration*** *pour d<1 si combinée avec Optim1*	***Dégradation*** *Ratio=1.14*	***Amélioration*** *Accélération=1.9*
	Optim 3	***Amélioration*** *Accélération=2.66*	***Amélioration*** *pour d>=1 Accélération=1.6*	***Amélioration*** *Accélération=1.32*	***Amélioration*** *Accélération=1.3*
	Optim 4	***Dégradation*** *si combinée avec* –O2 *ou* –O3. *Décélération =1.66 sinon* ***Stabilité***	***Stabilité***	***Stabilité***	***Dégradation*** *pour* –O2 *et* –O3, *Décélération =1.4 sinon* <u>*Stabilité*</u>

Table 4.3 Résultats expérimentaux selon les 4 modes d'optimisation

CHAPITRE 5.

FRAGMENTATION DE MATRICE CREUSE POUR LA DISTRIBUTION DU PMVC SUR UN SDGE

1. Introduction

Dans ce chapitre, nous nous intéressons à l'optimisation de la distribution du produit matrice-vecteur creux (PMVC) sur un système distribué à grande échelle (SDGE). Notre objectif général est de déterminer, pour une grande matrice creuse et un vecteur dense donnés, le meilleur équilibrage des charges pour la distribution du PMVC sur un tel système. Le vecteur d'entrée étant diffusé à tous les nœuds du système, il s'agit donc de distribuer judicieusement la matrice A sur les nœuds. Dans ce but, trois approches de fragmentation de la matrice sont proposées. Dans la première, la matrice est fragmentée en blocs ayant le même nombre de lignes. La deuxième (resp. troisième) approche consiste à fragmenter la matrice en blocs de lignes contiguës (resp. non contiguës) ayant le même nombre d'éléments non nuls. Les deux dernières approches nécessitant la résolution de problèmes d'ordonnancement connus pour être NP-durs [BEP07][Che04][Ski97], nous avons donc conçu des heuristiques spécifiques. Par la suite, nous effectuons une analyse théorique des performances suivie par une série d'expérimentations mettant en évidence l'intérêt de l'étude. Notons que la plate-forme cible est un SDGE où les nœuds ont des caractéristiques pouvant être différentes, donc constituant un environnement *hétérogène*. Il est toutefois possible, en général, de grouper les nœuds en sous-ensembles homogènes constitués de machines ayant les mêmes performances sur le plan vitesse. C'est donc au sein d'un tel (sous) environnement homogène que nous situons notre étude et ce, en première étape et par mesure de simplification. Nous pensons qu'il est tout à fait possible de la généraliser au cas plus réaliste d'un environnement hétérogène. Cela constitue sans aucun doute une perspective intéressante.

Le présent chapitre est organisé comme suit. Dans la section 2, nous détaillons la problématique. Dans la section 3, nous présentons un aperçu sur les travaux antérieurs qui se sont intéressés à l'équilibrage de charges pour le PMVC. La section 4 est consacrée à la description de notre contribution et qui consiste en la description de trois approches (heuristiques) pour la fragmentation par ligne d'une matrice creuse. La section 5 est dévolue au calcul de complexité et à l'analyse des heuristiques conçues. Dans la section 6, nous exposons et commentons les résultats expérimentaux relatifs aux divers modes de fragmentation et qui ont été testés sur deux séries de données d'entrée i.e. (i) des matrices quelconques et des matrices bande générées aléatoirement et (ii) des matrices issues d'applications réelles. Finalement, dans la section 7, nous concluons le chapitre.

2. Présentation de la problématique

Notre objectif ici est d'étudier la distribution du PMVC sur un SDGE, plus précisément, la distribution d'une matrice creuse et l'équilibrage des charges. En effet, l'équilibrage des charges, au niveau du traitement, peut avoir une grande influence sur les performances de l'application.

Nous nous confrontons en fait à un problème d'ordonnancement particulier qui est NP-difficile [Hoc97]. Par conséquent, pour le résoudre, nous proposons des heuristiques spécifiques conçues selon des approches différentes. Toutes ces approches se basent sur une stratégie à une dimension (1-D) de la matrice (i.e. lignes ou colonnes) comme l'illustre la Figure 5.1.

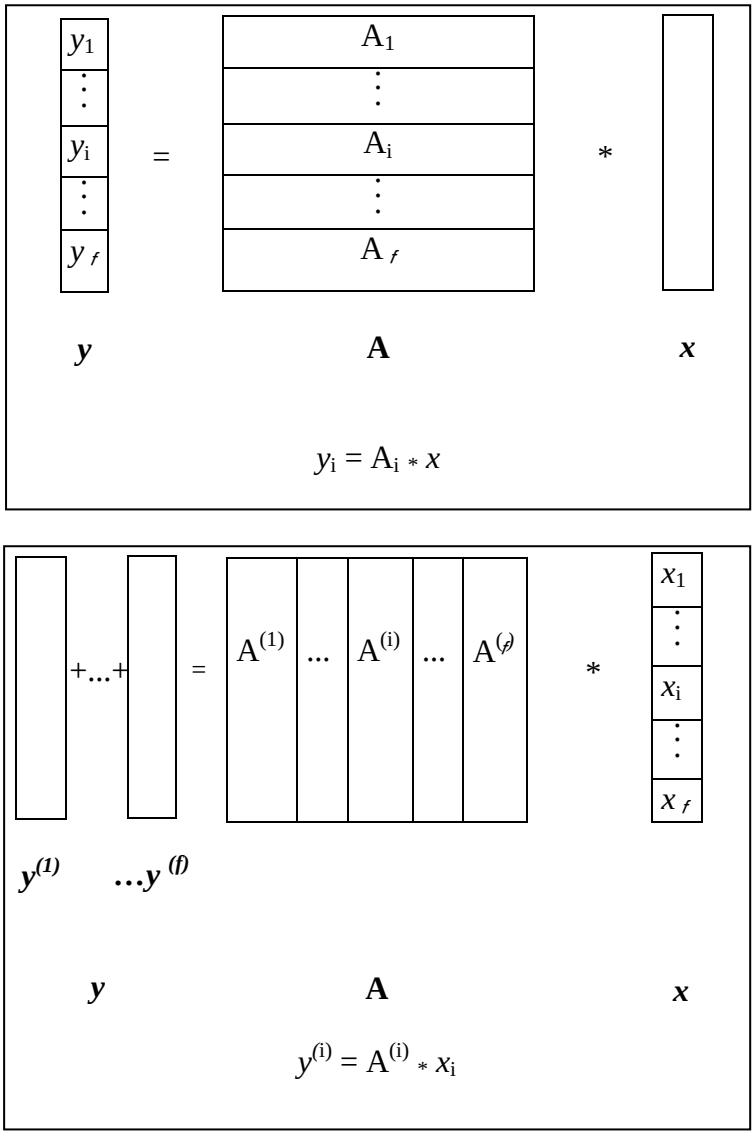


Figure 5.1 Fragmentation de la matrice

En fait, c'est seule la fragmentation par ligne qui est étudiée ici. En effet, la fragmentation par colonne est tout à fait symétrique de celle par ligne et peut en être directement déduite. Le choix d'une fragmentation 1-D, et plus précisément d'une fragmentation par ligne, est justifié par le fait que notre environnement cible est un système distribué à grande échelle (SDGE). Il est connu que les fragmentations 2-D (i.e. par blocs de lignes ou de colonnes) [BiM05] [VaB05] et 1-D par colonne génèrent des communications supplémentaires (par rapport à une fragmentation par ligne) nécessaires pour la réduction du vecteur résultat final. Il faut préciser aussi que dans un SDGE, les communications sont assez coûteuses du fait qu'elles sont généralement effectuées via Internet. Dans l'étude actuelle du prétraitement, nous rappelons que nous ne tenons compte ni de l'hétérogénéité ni de la volatilité des nœuds.

3. Etat de l'art sur l'équilibrage des charges pour le PMVC

A notre connaissance, l'équilibrage des charges pour la distribution du PMVC sur des systèmes à grande échelle n'est pas encore suffisamment étudié. La plupart des solutions proposées sont des solutions dynamiques pour des clusters de stations de travail utilisant des communications par passage de message. Les expérimentations ont montré que l'équilibrage des charges dynamique génère un grand overhead [BiM05][Chr98].

Prenons, à titre d'exemple, PETSc (Portable, Extensible Toolkit for Scientific Computation) [BBG01] qui est un ensemble de structures de donnée et de routines pour l'implémentation parallèle d'applications scientifiques modélisées par des équations aux dérivées partielles (EDP). PETSc est en fait destiné à être utilisé dans le cadre de projets d'applications distribuées à grande échelle. C'est un ensemble d'outils qui utilise le standard interface à passage de message (MPI) pour la communication des messages, intègre des bibliothèques d'optimisation dépendantes de l'architecture telles que BLAS et LAPACK. Il inclut des solveurs parallèles pour EDP linéaires et non linéaires qui peuvent être facilement intégrés dans des programmes C, C++ et Fortran. Néanmoins, il ne présente pas d'outil pour l'équilibrage des charges.

Dans [Chr98], Christen présente un solveur itératif et parallèle pour la résolution de systèmes linéaires sur des multi-ordinateurs et des clusters de stations de travail. Son solveur est capable d'appliquer à la matrice un équilibrage des charges dynamique en redistribuant ses lignes ainsi que les éléments correspondants du vecteur sur les nœuds du système. Il s'agit donc d'ajuster l'état de la charge selon le besoin afin d'obtenir une charge équilibrée. Les expérimentations ont été effectuées sur un cluster de stations de travail connectées par 10

Mbit/s switched Ethernet (un réseau LAN) et les communications ont été effectuées en utilisant le passage de message MPI, ce qui ne correspond pas, réellement, aux caractéristiques d'un système distribué à grande échelle.

Il est facile de remarquer que le problème d'équilibrage des charges que nous étudions peut être formulé comme un problème d'ordonnancement sans préemption consistant à affecter N tâches indépendantes $T_1,\ldots,T_N$ (i.e. les N lignes de la matrice creuse A) de coûts c_1, ..,c_N (c_i étant le nombre d'éléments non nuls dans la ligne i), sur *f* processeurs homogènes. Nous étudions en fait deux instances particulières du problème. La première exige que les tâches affectées au même processeur soient successives (i.e lignes contiguës dans la matrice). Le second ne prend pas en compte cette contrainte de contiguïté et correspond au problème général d'ordonnancement de tâches indépendantes. Soulignons que le *problème de partitionnement linéaire* [Ski97] est assez similaire à la première instance du notre. Toutefois, la différence est que l'objectif du problème de partitionnement linéaire est *la minimisation du makespan* (i.e. la charge du processeur le plus chargé) et non pas l'équilibrage des charges qui se traduit par la minimisation de l'écart entre les charges minimale et maximale. Précisons que la charge d'un processeur est la somme des coûts des tâches qui lui sont attribuées. Notons que les deux objectifs ne sont pas forcément identiques (quoique la réalisation du second entraîne celle du premier) et les solutions optimales des deux problèmes peuvent être différentes (voir section 4.1.2). Ajoutons enfin que le problème de partitionnement linéaire peut être résolu par un algorithme exact de programmation dynamique de complexité $O(f*N^2)$, *f* étant le nombre de processeurs [Ski97].

Concernant le second problème d'ordonnancement auquel nous nous intéressons, c'est un problème classique connu pour être NP-difficile pour lequel plusieurs algorithmes d'approximation (heuristiques garanties) sont connus en littérature. Les plus connus sont le List Scheduling (LS), le Shortest Processing Time (SPT) où les tâches sont d'abord triées par coût croissant et le Largest Processing Time (LPT) où les tâches sont d'abord triées par coût décroissant [BEP07][Che04]. Il est connu que le LPT est (en général) le meilleur des trois et le SPT est le moins performant [BEP07][Che04]. Nous donnerons ci-dessous plus de détails sur ces différents algorithmes d'ordonnancement (voir section 4.3). Ajoutons que nous avons utilisé ces trois algorithmes et que nous avons proposé des heuristiques d'amélioration. Ces dernières, démarrent d'une solution obtenue par l'un des trois algorithmes précédents et adoptent une stratégie itérative d'échange de tâches (lignes de la matrice) entre les processeurs (fragments de la matrice) et ce, afin de réduire le déséquilibre de charge. Cette procédure permet d'aboutir à de meilleurs résultats (voir section 6).

4. Fragmentation des données

Dans ce qui suit, nous présentons trois approches pour décomposer une matrice creuse, notée A, en fragments (blocs de lignes) dont le nombre est noté *f*. La première consiste à décomposer la matrice en blocs (fragments) de lignes ayant la même taille i.e. contenant le même nombre de lignes contiguës. Deux algorithmes sont proposés : (i) NLE (fragmentation avec Nombre Equilibré de Lignes) et (ii) NLEG (NLE Généralisée). La seconde approche, appelée NEZ (fragmentation avec Nombre Equilibré de non-Zéros) consiste à décomposer la matrice en fragments de lignes contiguës (ne contenant pas nécessairement le même nombre de lignes) ayant le même nombre d'éléments non nuls. Concernant la troisième approche, c'est une extension de NEZ où nous permettons aux fragments de contenir des lignes non contiguës. Trois algorithmes sont proposés ici i.e. NEZG (NEZ Généralisée), NEZGT (NEZG avec Tri, décroissant en l'occurence) et NEZGTC (NEZG avec Tri Croissant). Soit 6 algorithmes en tout.

4.1 Approche 1 : fragmentations NLE et NLEG

Considérons la division Euclidienne de N par *f* i.e. N = q∗*f* + p avec 0 ≤ p ≤ *f*-1. Le principe de cette approche consiste à construire *f* fragments contenant le même nombre de lignes contiguës. Deux fragmentations sont proposées : NLE et NLEG.

4.1.1 Fragmentation NLE

C'est une approche naïve qui consiste à partitionner A en *f* fragments (blocs de lignes) où les p premiers contiennent chacun q+1 lignes contiguës alors que les (*f*-p) restants en contiennent q. Nous avons ce qui suit :

- Fragment i (i=1…p) contient les lignes (i-1)(q+1)+1…(q+1)i
- Fragment i (i=p+1… *f*) contient les lignes p(q+1)+q(i-p-1)+1…p(q+1)+q(i-p)

Remarquons que lorsque p=0, il existe une seule fragmentation possible où chaque fragment est constitué de q=N/*f* lignes, alors que dans le cas contraire où p≠0, on peut construire C_N^p fragmentations différentes selon le choix des fragments contenant q+1 lignes. Nous noterons $((q+1)^p, q^{f-p})$ la fragmentation décrite ci-dessus.
La symétrique de la décomposition NLE précédente consiste à décomposer A en *f* fragments

où les (*f*-p) premiers contiennent chacun q lignes contiguës alors que les fragments restants contiennent q+1 lignes. On aura ainsi :

- Fragment i (i=1… *f* -p) contient les lignes (i-1)q+1…i∗q
- Fragment i (i= *f* -p+1… *f*) contient les lignes (i-1)(q+1)-(*f*-p)+1…i∗(q+1)-(*f* –p)+1

Lorsque p≠0, cette partition est notée (q^{f-p},$(q+1)^{p}$).

4.1.2 Fragmentation NLEG

La décomposition NLE vise un équilibrage au niveau du nombre de lignes par fragment. Néanmoins, elle peut présenter un déséquilibrage au niveau du nombre d'éléments non nuls par fragment ce qui peut engendrer, après compression des fragments, de mauvaises performances globales pour le PMVC sur le système cible. Lorsque p≠0, la décomposition NLEG (NLE Généralisée) peut réduire le déséquilibrage. En fait, NLEG est un algorithme à deux phases qui optimise un critère particulier (voir ci-dessous). Contrairement à NLE, les fragments similaires i.e. contenant le même nombre de lignes ne sont pas nécessairement successifs dans NLEG.

La phase 1 de NLEG, consiste à choisir une distribution *aléatoire* des fragments contenant q+1 lignes. Concernant la phase 2, qui est une heuristique itérative, son objectif est de minimiser un critère particulier, noté *Crit*. *Crit* peut correspondre à la charge maximale i.e. le nombre d'éléments non nuls dans le fragment le plus chargé. Il peut aussi correspondre au Facteur de Déséquilibrage (FD) défini par la différence entre les charges maximale et minimale.

Pratiquement, la procédure d'amélioration (phase 2), consiste à simultanément réduire la charge maximale et augmenter la charge minimale à travers des échanges de lignes entre les fragments successifs. Ceci permet, au moyen de raffinements successifs, d'améliorer la fragmentation ultérieure et ainsi, la rendre mieux équilibrée.

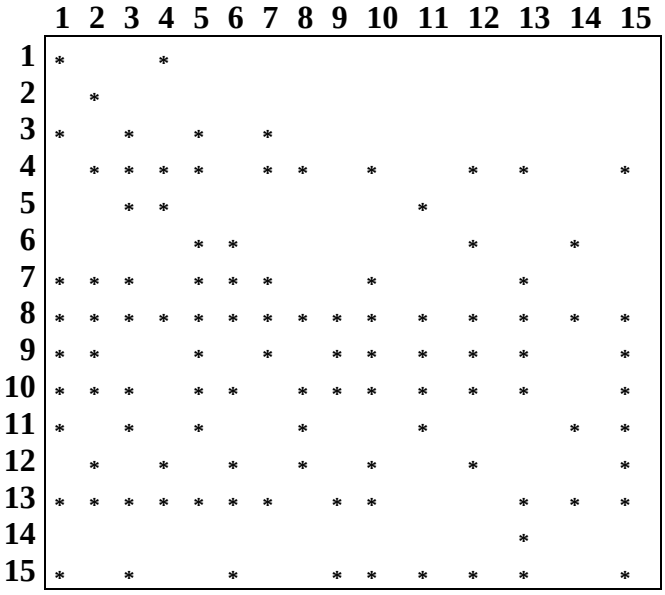
Plus précisément, nous procédons comme suit. Soit *fcmx* le fragment le plus chargé, sa première (ou dernière) ligne est affectée au fragment précédent appelé prédécesseur (ou au fragment suivant, appelé successeur) s'il en existe, pourvu que cette modification n'augmente pas la charge courante du fragment le plus chargé.

De manière symétrique, soit *fcmn* le fragment le moins chargé. On lui attribue la première ligne de son fragment successeur (ou bien la dernière ligne de son fragment prédécesseur) s'il en existe, pourvu que cette modification ne décroisse pas la charge courante du fragment le

moins chargé. Cette procédure est itérée tant que le critère FD peut être réduit et/ou qu'on n'a pas dépassé un nombre maximal d'itérations.

Les algorithmes NLE et NLEG sont illustrés par les exemples suivants où N=15, NZ=104 (NZ étant le nombre d'éléments non nuls de la matrice) et *f*=6. En plus du facteur de déséquilibrage (FD), nous définissons le Facteur de Déséquilibrage Relatif (FDR) qui correspond au rapport de la charge maximale sur la charge minimale.

Exemple 1: Fragmentation NLE



Indice de ligne	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Nombre d'éléments non nuls par ligne	2	1	4	10	3	4	8	15	10	12	6	7	12	1	9

Appliquée à A, la décomposition NLE donne ce qui suit :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (3 lignes)	1(2) ; 2(1) ; 3(4)	7
2 (3 lignes)	4(10) ; 5(3) ; 6(4)	17
3 (3 lignes)	7(8) ; 8(15) ; 9(10)	33
4 (2 lignes)	10(12) ; 11(6)	18
5 (2 lignes)	12(7) ; 13(12)	19
6 (2 lignes)	14(1) ; 15(9)	10

Remarquons que FD=33-7=26 et FDR=33/7=4.71. Cette fragmentation est notée $(3^3,2^3)$.

La fragmentation symétrique $(2^3,3^3)$ correspond à ce qui suit:

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (2 lignes)	1(2) ; 2(1)	3
2 (2 lignes)	3(4) ; 4(10)	14
3 (2 lignes)	5(3) ; 6(4)	7
4 (3 lignes)	7(8) ; 8(15) ; 9(10)	33
5 (3 lignes)	10(12) ; 11(6) ; 12(7)	25
6 (3 lignes)	13(12) ; 14(1) ; 15(9)	22

Ici, FD=33-3=30 et FDR=33/3=11. Elle est donc moins intéressante.

Exemple 2: Fragmentation NLEG (appliquée à la même matrice)

Phase 1 :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (2 lignes)	1(2) ; 2(1)	3
2 (3 lignes)	3(4) ; 4(10) ; 5(3)	17
3 (2 lignes)	6(4) ; 7(8)	12
4 (3 lignes)	8(15) ; 9(10) ; 10(12)	37
5 (2 lignes)	11(6) ; 12(7)	13
6 (3 lignes)	13(12) ; 14(1) ; 15(9)	22

Ici, nous avons FD=37-3=34 et FDR=37/3=12.33 (i.e. pire qu'avec NLE). Cette fragmentation est notée $((2,3)^3)$.

Phase 2 :

En appliquant la phase 2 (stabilisation après une itération) nous faisons croître la charge du fragment 1 (*fcmn*) et décroître la charge du fragment 4 (*fcmx*). Cette fragmentation est notée $(3,2^3,3^2)$.

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (3 lignes)	1(2) ; 2(1) ; 3(4)	7
2 (2 lignes)	4(10) ; 5(3)	13
3 (2 lignes)	6(4) ; 7(8)	12
4 (2 lignes)	8(15) ; 9(10)	25
5 (3 lignes)	10(12) ; 11(6) ; 12(7)	25
6 (3 lignes)	13(12) ; 14(1) ; 15(9)	22

Nous avons FD=25-7=18 et FDR=25/7=3.57. Cette fragmentation n'est pas optimale. En respectant les contraintes de la phase 2 (nombre de lignes dans un fragment égal à q ou q+1), elle ne peut pas être améliorée.

Si nous démarrons la phase 2 de l'algorithme NLEG à partir de la fragmentation particulière (3^3,2^3) qui est fournie par NLE, la fragmentation optimale est atteinte après deux itérations. Elle correspond à ce qui suit :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (3 lignes)	1(2) ; 2(1) ; 3(4)	7
2 (3 lignes)	4(10) ; 5(3) ; 6(4)	17
3 (2 lignes)	7(8) ; 8(15)	23
4 (2 lignes)	9(10) ; 10(12)	22
5 (2 lignes)	11(6) ; 12(7) ;	13
6 (3 lignes)	13(12) ; 14(1) ; 15(9)	22

Maintenant, on a FD=23-7=16 et FDR=23/7=3.28.

4.2 Approche 2: décomposition NEZ

L'approche de fragmentation présentée ci-dessus est assez naïve. Notons que NLE et NLEG peuvent engendrer un déséquilibrage de charge prohibitif pour les nœuds du système distribué quand les éléments non nuls ne sont pas uniformément répartis sur les lignes de la matrice.

Nous avons donc conçu la fragmentation NEZ dans l'objectif d'éviter cet inconvénient. Elle consiste à fragmenter la matrice A en blocs de lignes tels que chaque bloc (fragment) contient (autant que possible) le même nombre d'éléments non nuls. Remarquons que le nombre d'éléments non nuls par fragment doit être autour de la moyenne NZ/*f* et ce, dans le but de garantir une fragmentation équilibrée.

De plus, notons ici que :

(i) une ligne donnée ne peut appartenir qu'à un seul fragment,

(ii) le nombre de lignes n'est pas nécessairement le même pour tous les fragments,

(iii) chaque fragment contient des lignes contiguës. Ainsi, chaque fragment peut être identifié par l'indice de sa première ligne.

Signalons que le principe de l'algorithme de fragmentation NEZ revient, en fait, à la résolution d'un problème d'ordonnancement particulier [Hoc97][SaY99]. Il consiste à

ordonnancer N tâches indépendantes non préemptives $\{T_1,\ldots,T_N\}$ (i.e. les N lignes de A) de coûts $\{c_1, ..,c_N\}$, où c_i est le nombre d'éléments non nuls dans la ligne i, sur f processeurs homogènes sous la contrainte que les tâches (i.e. lignes) affectées à un processeur (i.e. fragment) doivent être successives (i.e. contiguës).

Pour résoudre ce problème particulier, nous avons conçu une heuristique à deux phases que nous détaillons dans ce qui suit. Nous comparerons plus loin les performances de cette heuristique avec celle d'une résolution par programmation dynamique comme mentionné auparavant (voir section 3).

La première phase qui est une phase constructive, permet de construire une fragmentation initiale en un temps O(N). Elle consiste à affecter des lignes contiguës à un fragment donné jusqu'à ce que sa charge (i.e. le nombre total des éléments non nuls de ses lignes) soit très proche du seuil $\lfloor NZ/f \rfloor$ ou l'excède pour *la première fois* (i.e. sa charge sans sa dernière ligne est strictement inférieure à $\lfloor NZ/f \rfloor$ et sa charge avec sa dernière ligne est strictement supérieure). Plus précisément, lors de la construction, un fragment peut rarement avoir exactement $\lfloor NZ/f \rfloor$ éléments non nuls.

Ainsi, nous utilisons un paramètre booléen qui indique que le fragment courant, dont la charge est proche du seuil, est autorisé ou non à prendre une dernière ligne, et par conséquent, excéder le seuil. La charge des lignes non encore affectées (nombre des éléments non nuls restants) est calculée dynamiquement ce qui permet de :

(i) décider si on peut ou non excéder $\lfloor NZ/f \rfloor$,

(ii) garantir que chaque fragment contient au moins une ligne,

(iii) éviter un déséquilibrage élevé.

Le pseudo-code de la phase 1 de NEZ est détaillé dans la Figure 5.2.

Concernant la phase 2, elle est similaire à la phase 2 de NLEG (voir section 4.1.2) avec la seule différence qu'ici le nombre de lignes par fragment n'est pas nécessairement égal à q ou bien q+1.

NEZ est illustré par l'exemple 3 (même matrice que précédemment).

```
/* Seuil: Moyenne du nombre d'éléments non nuls à obtenir par fragment */
/*excès : booléen indiquant si le nombre d'éléments non nuls par fragment peut ou non
dépasser Seuil */
Seuil=⌊NZ/f⌋ ; NZrestant=NZ ; excès=Faux
Do i = 1, f
        /* construction du fragment de numéro i */
        /* NZfrag(i): nombre d'éléments non nuls dans le fragment i */
        /*Parcourir les lignes de la matrice*/
        While (ligne existe And ( NZfrag(i) < Seuil))
                Ajouter la ligne au fragment i
        Endwhile
        If (NZfrag(i) > Seuil And excès = Faux) Then
                Supprimer la dernière ligne ajoutée au fragment i
        EndIf
        NZrestant=NZrestant – NZfrag(i);
        If ((la moyenne de NZrestant < Seuil) Or (nombre de lignes non affectées ≤
        nombre de fragments restants)) Then
                excès=Faux
        Else    excès=Vrai
        EndIf
EndDo
```

Figure 5.2 Pseudo-code de la phase 1 de NEZ

Exemple 3 : fragmentation NEZ

Indice de ligne	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Nombre d'éléments non nuls par ligne	2	1	4	10	3	4	8	15	10	12	6	7	12	1	9

Après la phase 1, on obtient ce qui suit:

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (4 lignes)	1(2) ; 2(1) ; 3(4) ; 4(10)	17
2 (4 lignes)	5(3) ; 6(4) ; 7(8) ; 8(15)	30
3 (1 ligne)	9(10)	10
4 (2 lignes)	10(12) ; 11(6)	18
5 (2 lignes)	12(7) ; 13(12)	19
6 (2 lignes)	14(1) ; 15(9)	10

Notons ici que :

(i) FD=20, FDR=3 alors que dans NLE (voir ci-dessus), nous avions FD=26 et FDR=4.71,

(ii) Le résultat donné par la phase 1 de NEZ est meilleur que celui donné par la phase 1 de NLEG,

(iii) *fcmx* correspond ici au fragment #2 et *fcmn* au fragment #3. La charge du fragment #2 peut être diminuée en affectant la ligne 5 au fragment #1. C'est la seule amélioration qui peut être effectuée ici.

Après l'amélioration obtenue en phase 2, on obtient les nouvelles charges suivantes.

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (5 lignes)	1(2) ; 2(1) ; 3(4) ; 4(10) ; 5(3)	20
2 (2 lignes)	6(4) ; 7(8)	12
3 (2 lignes)	8(15) ; 9(10)	25
4 (2 lignes)	10(12) ; 11(6)	18
5 (2 lignes)	12(7) ; 13(12)	19
6 (2 lignes)	14(1) ; 15(9)	10

Maintenant, nous avons FD=15 et FDR=2.5. Ce n'est pas optimal. La fragmentation optimale est en fait la suivante.

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (4 lignes)	1(2) ; 2(1) ; 3(4) ; 4(10)	17
2 (3 lignes)	5(3) ; 6(4) ; 7(8)	15
3 (1 ligne)	8(15)	15
4 (2 lignes)	9(10) ; 10(12)	22
5 (2 lignes)	11(6) ;12(7)	13
6 (3 lignes)	13 (12) ; 14(1) ; 15(9)	22

Nous avons FD=9 et FDR=1.69.

4.3 Approche 3: fragmentations NEZG, NEZGT et NEZGTC

Dans NEZ, nous avons construit les fragments sous la contrainte que les lignes d'un fragment donné doivent être nécessairement contiguës dans la matrice A. Du fait que la distribution des éléments non nuls dans A est en général aléatoire, cette décomposition ne garantit pas un équilibrage de charges optimal au niveau des fragments. Une alternative consiste à relaxer cette contrainte i.e. permettre qu'un fragment puisse contenir des lignes non contiguës. Nous nous ramenons ainsi au problème général d'ordonnancement de tâches indépendantes non préemptives sur des processeurs homogènes. Diverses heuristiques (constructives) sont connues à ce propos. Les plus connues sont les suivantes : LS (List Scheduling) de complexité O(N), SPT (Shortest Processing Time) de complexité O(NlogN) et LPT (Largest Processing Time) de même complexité O(NlogN) [Che04]. Les deux premières heuristiques possèdent un ratio ρ de garantie égal à 2-1/*f* et la troisième possède un ratio de garantie égal à 4/3-1/(3*f*) [BEP07][Che04]. Précisons que le ratio ρ d'une heuristique dite garantie est défini par le rapport :

$$\rho = \text{Coût}(S) / \text{Coût}(S^*)$$

où Coût(S*) est la valeur de solution optimale et Coût(S) la valeur de celle fournie par l'algorithme. On dira que l'algorithme d'approximation (heuristique) est garanti(e) si on peut déterminer (ou du moins borner) un tel ratio.

Nous avons utilisé les trois heuristiques LS, SPT et LPT et les avons combinées avec une heuristique itérative d'amélioration. Ainsi, nous avons conçu trois algorithmes, à savoir NEZG (NEZ Généralisé) basé sur LS, NEZGT (NEZG avec Tri décroissant) basé sur SPT et NEZGTC (NEZGT, tri Croissant) basé sur LPT.

4.3.1 Fragmentation NEZG

C'est une heuristique à deux phases qui se détaille comme suit :

- Dans la première, nous utilisons l'heuristique LS qui consiste d'abord, à affecter la ligne i (i=1…*f*) au fragment i. Par la suite, la ligne suivante est affectée au fragment le moins chargé et ainsi de suite. En d'autres termes, la ligne courante non affectée est toujours attribuée au fragment le moins chargé jusqu'à épuisement des lignes.

- La seconde phase, qui est une phase d'amélioration, est une heuristique itérative. Elle permet, à travers des raffinements successifs, d'améliorer la fragmentation précédente, conduisant ainsi à un meilleur équilibrage. Le critère choisi étant FD (différence entre les deux charges extrêmes), la procédure conçue consiste à effectuer des échanges ou transferts successifs de lignes entre le fragment le plus chargé et le fragment le moins chargé.

Plus précisément, soit *fcmx* (resp. *fcmn*) le fragment le plus (resp. moins) chargé, la procédure que nous adoptons consiste, soit à transférer une ligne appropriée de *fcmx* à *fcmn* ou bien à échanger une ligne du premier fragment avec une ligne du second et ce, dans le but de réduire la différence entre les deux charges, notée *Diff*. L'opération de transfert consiste à choisir une ligne dont le nombre d'éléments non nuls, noté nzx est inférieur à *Diff*. L'opération d'échange consiste, quant à elle, à choisir deux lignes (la première appartenant à *fcmx* et la seconde à *fcmn*) telles que la différence entre le nombre des éléments non nuls du premier, noté nzx, et le nombre des éléments non nuls du second, noté nzn, soit inférieure à *Diff* i.e. nzx – nzn < *Diff*. Une meilleure alternative, toutefois plus coûteuse, consiste à optimiser le choix afin de minimiser le nouvel écart entre les charges, ce qui revient à minimiser la quantité $|Diff/2-nzx|$ en cas de transfert (resp. $|Diff/2-(nzx-nzn)|$ en cas d'échange). Cette procédure est itérée tant qu'il est possible de réduire le critère FD et/ou qu'on ne dépasse pas un nombre d'itérations fixé à l'avance.

Exemple 4: fragmentation NEZG (appliqué à la matrice de l'exemple 1)

Après la phase 1, nous obtenons la décomposition suivante :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (3 lignes)	1(2) ; 8(15); 13(12)	29
2 (3 lignes)	2(1) ; 7(8) ; 12(7)	16
3 (2 lignes)	3(4) ; 10(12)	16
4 (3 lignes)	4(10) ; 14(1) ; 15(9)	20
5 (2 lignes)	5(3) ; 9(10)	13
6 (2 lignes)	6(4) ; 11(6)	10

Notons qu'ici FD=29-10=19 et FDR=29/10=2.9.

Concernant la phase 2, elle fournit la fragmentation suivante :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (2 lignes)	1(2) ; 8(15)	17
2 (2 lignes)	4(10) ; 12(7)	17
3 (3 lignes)	2(1) ; 3(4) ; 10(12)	17
4 (3 lignes)	7(8) ; 14(1) ; 15(9)	18
5 (3 lignes)	5(3) ; 6(4) ; 9(10)	17
6 (2 lignes)	11(6) ; 13(12)	18

FD=18-17=1 et FDR=18/17=1.05 : fragmentation optimale. 4 itérations ont été nécessaires.

4.3.2 Fragmentations NEZGTC et NEZGT

NEZGTC (resp. NEZGT) est une heuristique à trois phases. Les phases 0 et 1 correspondent à l'heuristique SPT (resp. LPT). La phase 2 est une heuristique d'amélioration itérative. La phase 0 consiste à trier les lignes de A par nombre d'éléments non nuls croissant (resp. décroissant). Les phases 1 et 2 sont les mêmes que dans NEZG (voir section 4.3.1). Nous illustrons ces décompositions par les exemples suivants :

Exemple 5: fragmentation NEZGT (même matrice)

Indice de ligne	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Nombre d'éléments non nuls par ligne	2	1	4	10	3	4	8	15	10	12	6	7	12	1	9

En triant les lignes dans l'ordre décroissant du nombre d'éléments non nuls, on obtient :

Indice de ligne	8	4	10	4	9	15	7	12	11	3	6	5	1	2	14
Nombre d'éléments non nuls par ligne	15	12	12	10	10	9	8	7	6	4	4	3	2	1	1

La phase 1 de l'algorithme NEZGT fournit le résultat optimal suivant:

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (2 lignes)	8(15) ; 5(3)	18
2 (3 lignes)	13(12) ; 6(4) ; 1(2)	18
3 (3 lignes)	10(12) ; 3(4) ; 14(1)	17
4 (2 lignes)	9(10) ; 12(7) ;	17
5 (3 lignes)	4(10) ; 11(6) ; 2(1)	17
6 (2 lignes)	15(9) ; 7(8)	17

FD=1 et FDR=1.05 ce qui est optimal.

Exemple 6: fragmentation NEZGTC (même matrice)

Le tri des lignes de A par ordre croissant du nombre d'éléments non nuls, fournit ce qui suit :

Indice de ligne	2	14	1	5	3	6	11	12	7	15	4	9	10	13	8
Nombre d'éléments non nuls par ligne	1	1	2	3	4	4	6	7	8	9	10	10	12	12	15

Après la phase 1, on obtient la fragmentation suivante:

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (3 lignes)	2(1) ; 11(6) ; 10(12)	19
2 (3 lignes)	14(1) ; 12(7) ; 13(12)	20
3 (3 lignes)	1(2) ; 7(8) ; 8(15)	25
4 (2 lignes)	5(3) ; 15(9)	12
5 (2 lignes)	3(4) ; 4(10)	14
6 (2 lignes)	6(4) ; 9(10)	14

Notons ici que FD=13 et FDR=1.92 (meilleur que dans la phase 1 de NEZ). La phase 2 fournit la fragmentation suivante :

Fragment	*Indice de ligne (Nombre d'éléments non nuls par ligne)*	*Nombre d'éléments non nuls par fragment*
1 (2 lignes)	11(6) ; 10(12)	18
2 (2 lignes)	12(7) ; 4(10)	17
3 (2 lignes)	5(3) ; 8(15)	18
4 (2 lignes)	7(8) ; 15(9)	17
5 (3 lignes)	1(2) ; 3(4) ; 13(12)	18
6 (4 lignes)	2(1) ; 9(10) ; 14(1) ; 6(4)	16

Précisons que 7 itérations ont été nécessaires dans la phase 2.

Ici, FD=2 et FDR=1.12. Remarquons que NEZG et NEZGT ont donné de meilleurs résultats.

5. Analyse des performances théoriques

5.1 Approche 1 : NLE et NLEG

Il est clair que l'algorithme NLE peut être effectué en un temps O(*f*). Concernant NLEG, sa première phase est de complexité O(*f*) et il est facile de prouver que sa seconde phase (transfert de ligne) peut être effectuée (au maximum) en un temps O(N∗*nit*) où *nit* est le nombre (maximal) d'itérations. En conséquence, NLEG nécessite un temps O(N∗*nit*). Si on choisit *nit*=O(1) (resp. O(N)) on atteint une complexité de O(N) (resp. O(N^2)). Concernant l'heuristique NLEG, du fait qu'elle est de type Glouton, elle est garantie avec un ratio qui ne dépasse pas 2 [SHK95]. Autrement dit, si on note par FD* le facteur de déséquilibrage (voir section 4.1.2) optimal (i.e minimal), nous avons $1 \leq \rho = FD/FD^* \leq 2$.

D'un autre côté, on peut prouver qu'au pire des cas, FD est égal à (q+1)N–q (rappelons que $N = q*f+p$, $0 \leq p \leq f-1$) et que FDR est borné par 3N/2=O(N). En effet, le pire des cas a lieu quand le fragment le plus chargé (*fcmx*) contient q+1 lignes n'ayant aucun élément nul i.e. sa charge est égale à (q+1)N et le fragment de charge minimale (*fcmn*) contient q lignes où chaque ligne contient un seul élément non nul i.e. sa charge est égale à q. Ainsi, FD est dans ce cas égal à (q+1)N–q et FDR est borné par $N(q+1)/q = N+N/q \leq 3N/2$ puisque $q \geq 2$.

De plus, si nous considérons le pire des cas qui peut avoir lieu dans la phase 1 de NLEG, nous avons ce qui suit :

(i) *fcmn* se trouve entre deux fragments dont chacun contient q lignes

(ii) *fcmx* se trouve entre deux fragments dont chacun contient q+1 lignes

Par conséquent, aucune amélioration ne peut être obtenue dans la phase 2. Dans ce cas, la valeur de FD est (q+1)N-q. Remarquons que dans le meilleur des cas, on peut obtenir un FDR égal à 1. Ceci peut avoir lieu, par exemple, quand p=0 (i.e. N multiple de f) et les éléments non nuls sont uniformément distribués sur les lignes de la matrice.

5.2 Approche 2: NEZ

La phase 1 de NEZ peut être effectuée en un temps O(N) et la phase 2 (au pire) en un temps O(N∗*nit*) où *nit* est le nombre maximal d'itérations. Par conséquent, NEZ peut être effectuée en un temps O(N∗*nit*). Si on choisit *nit*=O(1) (resp. O(N)), on atteint une complexité de O(N) (resp. O(N^2)).

Concernant la qualité de l'heuristique, considérons la division Euclidienne $NZ = f*q_z + p_z$ avec $0 \leq p_z \leq f-1$ et $q_z = \lfloor NZ/f \rfloor$). Supposons que $f << N$. Au pire des cas nous avons les quatre propositions suivantes:

(i) *fcmn* (fragment le moins chargé) est un fragment qui n'a pas le droit (pour des raisons d'équilibrage, voir section 4.2) de contenir plus que q_z éléments non nuls

(ii) *fcmn* (resp. *fcmx*) contient q_z-(N-1) (resp. q_z -1+N) éléments non nuls,

(iii) *fcmn* contient q_z -N+1 éléments non nuls et la ligne qui suit la dernière ligne de *fcmn* est pleine (contient N éléments non nuls),

(iv) Nous admettons que *fcmx* (fragment le plus chargé) a le droit de dépasser q_z et que la ligne qui suit la dernière ligne de *fcmx* est pleine,

(v) *fcmx* contient q_z -1 éléments avant de dépasser q_z. Ainsi, nous obtenons un *fcmn* avec q_z-N+1 éléments non nuls et un *fcmx* avec qz-1+N éléments non nuls. Dans ce cas, nous avons FD = 2N-2 = O(N) et FDR = (qz-1+N)/(qz-N+1) = O(1).

Le meilleur des cas a lieu, en particulier, lorsque les lignes de A contiennent le même nombre d'éléments non nuls.

Ajoutons que, comme NLEG, NEZ est aussi de type glouton et est garanti avec un ratio égal à 2 [SHK95] i.e. nous avons $1 \leq \rho = FD/FD^* \leq 2$ (voir section 5.1).

5.3 Approche 3: NEZG, NEZGT et NEZGTC

Comme nous l'avons déjà vu, NEZGT ainsi que NEZGTC sont des algorithmes comportant chacun trois phases. La phase 0 (tri) peut être effectuée en un temps (N∗logN). La phase 1 est une phase constructive qui peut être effectuée en un temps O(N). Concernant la phase 2, elle peut être effectuée (au pire en un temps O(N∗*nit*) où *nit* est le nombre maximal d'itérations. En conséquence, NEZGT et NEZGTC sont effectués en temps O(N(logN+*nit*)). Si l'on choisit *nit*=O(1) (resp. O(N)), on obtient une complexité totale de O(NlogN) (resp. $O(N^2)$).

Quant à NEZG, elle requiert un temps O(N∗*nit*), soit un temps O(N) (resp. $O(N^2)$) si *nit*=O(1) (resp.O(N)).

Concernant la qualité des heuristiques NEZG, NEZGT et NEZGTC, au pire des cas, on obtient un FD (resp. FDR) égal à N-1 (resp. N). Ceci a lieu, par exemple, quand f = N i.e. chaque fragment contient une seule ligne et A contient une ligne ayant un seul élément non nul et une autre ligne ayant N éléments non nuls i.e. une ligne pleine.

Remarquons que NEZ, NEZG, NEZGT et NEZGTC sont aussi de type Glouton, garantis et nous avons $1 \leq \rho = FD/FD^* \leq 2$ (voir section 5.1).

Un tableau récapitulatif est donné ci-dessous.

Fragm.	Phase0	Phase1	Phase2	Total	Pire FD	Pire FDR
NLE	-	O(f)		O(f)	(N/f+1)$*$N-N/f	3N/2
NLEG	-	O(f)	O(N$*$nit)	O(N$*$nit)	(N/f+1)$*$N-N/f	3N/2
NEZ	-	O(N)	O(N$*$nit)	O(N$*$nit)	2N-2	(NZ/f +N)/ (NZ/f -N+1)
NEZG		O(N)	O(N$*$nit)	O(N$*$nit)	N-1	N
NEZGT	O(N$*$logN)	O(N)	O(N$*$nit)	O(N$*$(logN+nit))	N-1	N
NEZGTC	O(N$*$logN)	O(N)	O(N$*$nit)	O(N$*$(logN+nit))	N-1	N

Table 5.1 Récapitulatif de l'étude théorique

6. Etude expérimentale

6.1 Résultats numériques

Dans le but d'évaluer les performances pratiques des différentes approches de fragmentation, nous avons effectué une série d'expérimentations sur les trois ensembles de matrices suivants.

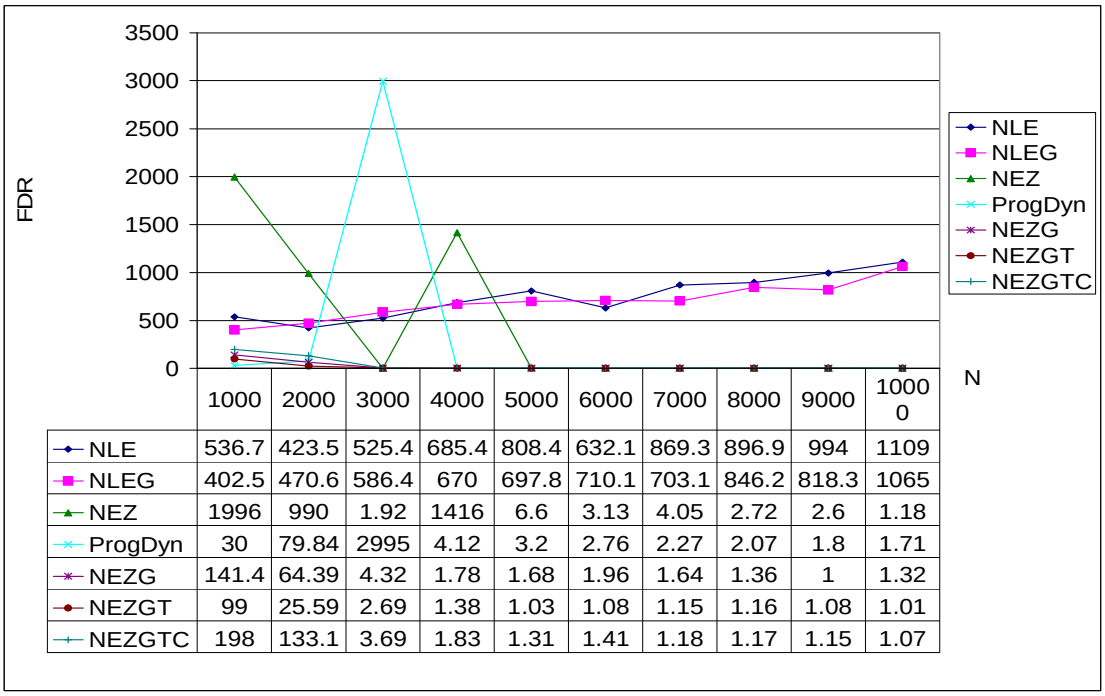
(i) matrices quelconques générées aléatoirement

(ii) matrices bandes générées aléatoirement

(iii) matrices téléchargées du site Web Matrix Market [Mat07].

Précisons que pour le premier (resp. second) ensemble, étant données une taille N et une densité d (resp. une demi-largeur de bande dlb et une densité de bande bd), nous avons conçu un générateur qui fournit en sortie une matrice générale (resp. matrice bande) où les éléments non nuls sont distribués aléatoirement. Concernant le troisième ensemble, les caractéristiques des matrices sont données dans la Table 5.3.

L'étude a consisté à comparer les différents algorithmes de fragmentation à travers les facteurs de déséquilibrage relatifs (FDR) i.e. le rapport charge_max/charge_min et ce, en fonction de la taille N de la matrice, de sa densité d, du nombre de fragments f et de la demi-largeur de bande pour les matrices bandes. Les expérimentations ont été effectuées sur un PC Pentium dual-core. Ajoutons que la taille N appartient à l'intervalle [1000 10000]. Nous avons testé plusieurs valeurs de f dans l'intervalle [2 512] i.e. $f=2^k$: k=1…9. Par mesure de concision, nous nous restreignons dans la description de certains résultats à f =512, valeur jugée représentative. Chaque matrice dans les trois ensembles d'entrées est donc décomposée en f fragments. Précisons que nous avons également expérimenté l'algorithme de programmation dynamique dans le cas de la contrainte de contiguïté des lignes par fragment.



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NLE	536.7	423.5	525.4	685.4	808.4	632.1	869.3	896.9	994	1109
NLEG	402.5	470.6	586.4	670	697.8	710.1	703.1	846.2	818.3	1065
NEZ	1996	990	1.92	1416	6.6	3.13	4.05	2.72	2.6	1.18
ProgDyn	30	79.84	2995	4.12	3.2	2.76	2.27	2.07	1.8	1.71
NEZG	141.4	64.39	4.32	1.78	1.68	1.96	1.64	1.36	1	1.32
NEZGT	99	25.59	2.69	1.38	1.03	1.08	1.15	1.16	1.08	1.01
NEZGTC	198	133.1	3.69	1.83	1.31	1.41	1.18	1.17	1.15	1.07

Figure 5.3 FDR pour matrices quelconques générées aléatoirement, d=5, *f*=512

N		1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NLE	nzx	1610	4235	7355	13023	20209	22124	38251	46639	54667	64310
	nzn	3	10	14	19	25	35	44	52	55	58
NLEG	nzx	1610	4235	8209	13399	16747	24143	30234	43157	46642	58549
	nzn	4	9	14	20	24	34	43	51	57	55
	NbIter	9	2	2	3	3	1	2	2	2	1
NEZ	nzx	990	1996	9428	5663	8805	11454	13384	18352	23275	84955
	nzn	1	1	4899	4	1334	3661	3308	6752	8949	72001
	NbIter	1	18	16	87	56	49	128	101	37	47
PROG. DYN.	nzx	990	1996	2995	4261	6591	9147	12013	15187	18781	23100
	nzn	33	25	1	1035	2062	3318	5288	7324	10453	13481
NEZG	nzx	990	1996	2995	3993	6406	10243	12352	14579	189847	22548
	nzn	7	31	693	2238	3816	5221	7550	10716	189841	17065
	NbIter	4	21	136	493	450	85	400	937	3915	701
NEZGT	nzx	990	1996	2995	3993	5021	7495	10394	13607	16782	19713
	nzn	10	78	1115	2887	4855	6945	9077	11735	15484	19514
	NbIter	0	0	0	0	0	4	20	79	0	0
NEZGTC	nzx	990	1996	2995	3993	5511	8212	10408	13525	16966	20220
	nzn	5	15	812	2181	4211	5804	8818	11522	14787	18855
	NbIter	3	12	41	102	118	116	213	702	277	617

Table 5.2 Charges extrêmes obtenues pour les matrices quelconques aléatoires

Légendes : nzx : nombre d'éléments non nuls dans le fragment le plus chargé ; nzn : nombre d'éléments non nuls dans le fragment le moins chargé

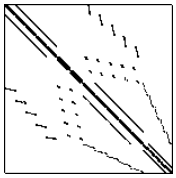
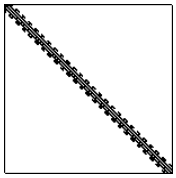
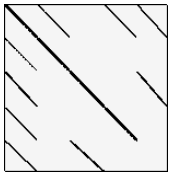
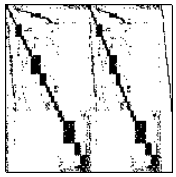
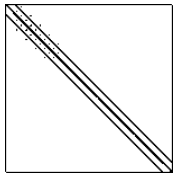
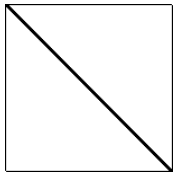
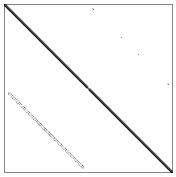
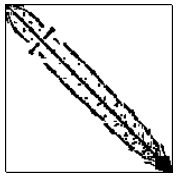
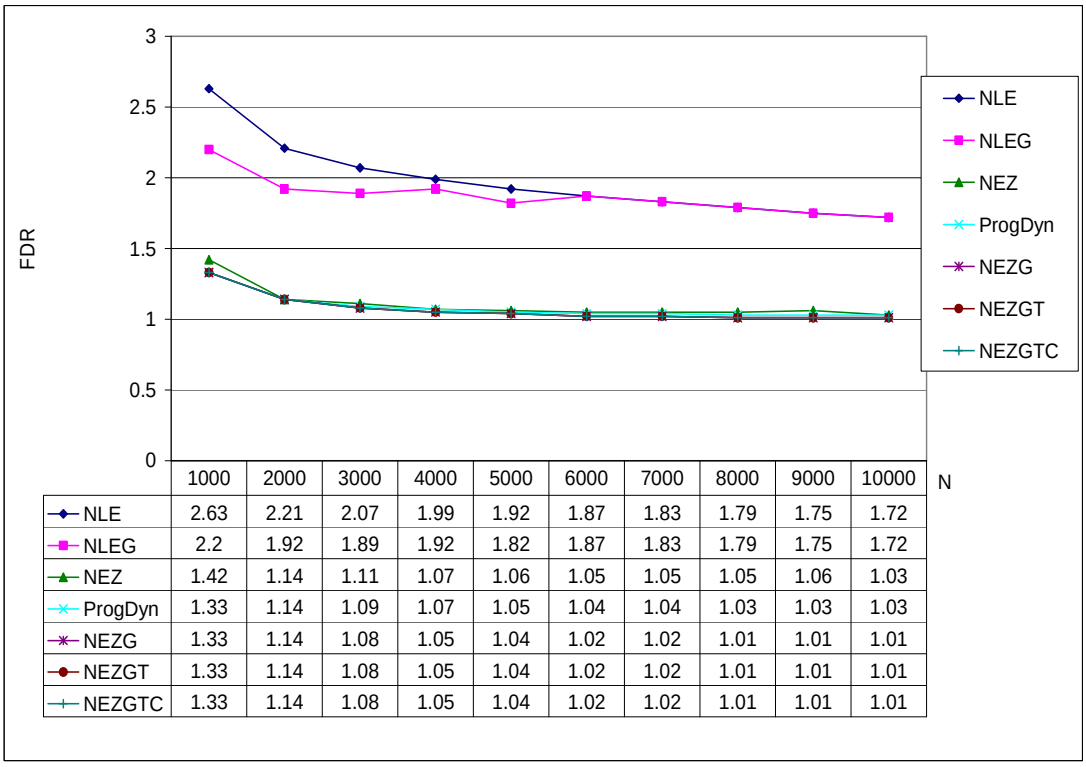
Structure	Nom	Taille	Densité
	Orsirr_1	1030	0.65
	Fidap008	3096	1.11
	Lns_3937	3937	0.16
	Gemat12	4929	0.14
	Utm5940	5940	0.24
	Fidap015	6867	0.2
	Dw8192	8192	0.06
	Fidapm37	9152	0.91

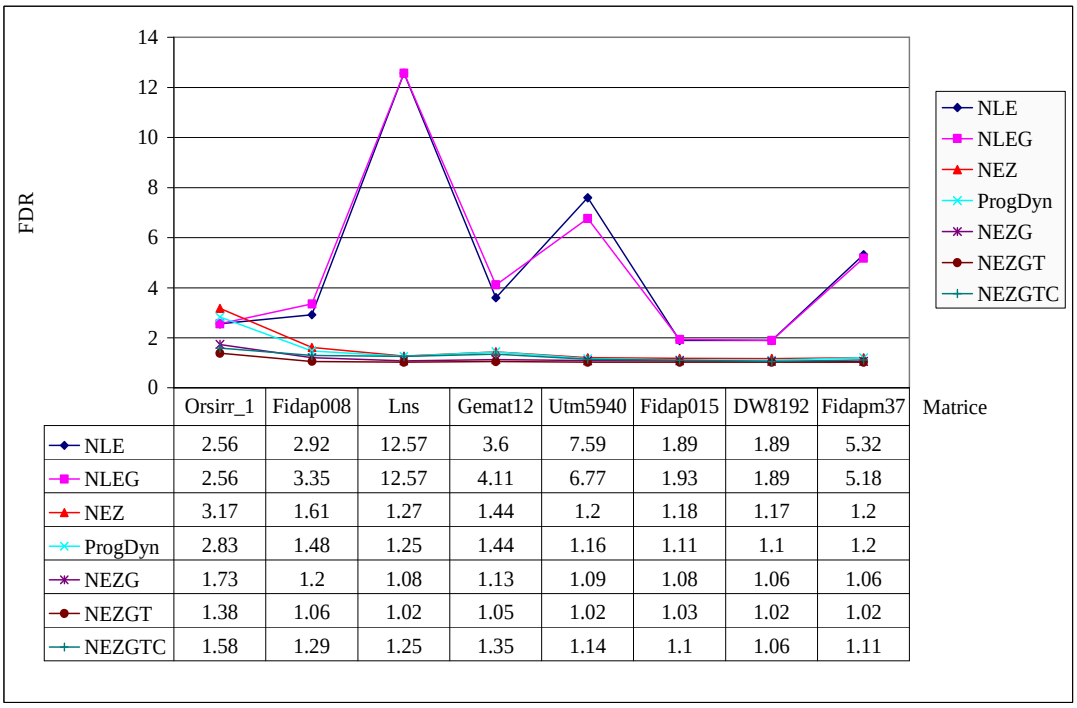
Table 5.3 Matrices téléchargées du site Web Matrix Market



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NLE	2.63	2.21	2.07	1.99	1.92	1.87	1.83	1.79	1.75	1.72
NLEG	2.2	1.92	1.89	1.92	1.82	1.87	1.83	1.79	1.75	1.72
NEZ	1.42	1.14	1.11	1.07	1.06	1.05	1.05	1.05	1.06	1.03
ProgDyn	1.33	1.14	1.09	1.07	1.05	1.04	1.04	1.03	1.03	1.03
NEZG	1.33	1.14	1.08	1.05	1.04	1.02	1.02	1.01	1.01	1.01
NEZGT	1.33	1.14	1.08	1.05	1.04	1.02	1.02	1.01	1.01	1.01
NEZGTC	1.33	1.14	1.08	1.05	1.04	1.02	1.02	1.01	1.01	1.01

N

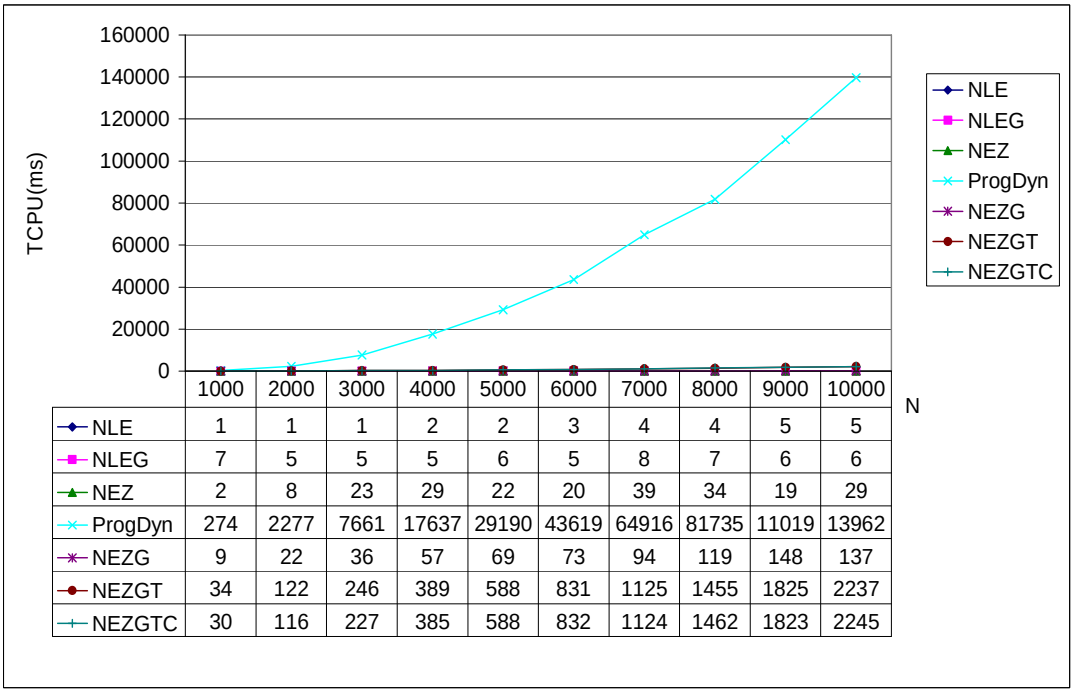
Figure 5.4 FDR pour matrices bande, dlb=100



	Orsirr_1	Fidap008	Lns	Gemat12	Utm5940	Fidap015	DW8192	Fidapm37
NLE	2.56	2.92	12.57	3.6	7.59	1.89	1.89	5.32
NLEG	2.56	3.35	12.57	4.11	6.77	1.93	1.89	5.18
NEZ	3.17	1.61	1.27	1.44	1.2	1.18	1.17	1.2
ProgDyn	2.83	1.48	1.25	1.44	1.16	1.11	1.1	1.2
NEZG	1.73	1.2	1.08	1.13	1.09	1.08	1.06	1.06
NEZGT	1.38	1.06	1.02	1.05	1.02	1.03	1.02	1.02
NEZGTC	1.58	1.29	1.25	1.35	1.14	1.1	1.06	1.11

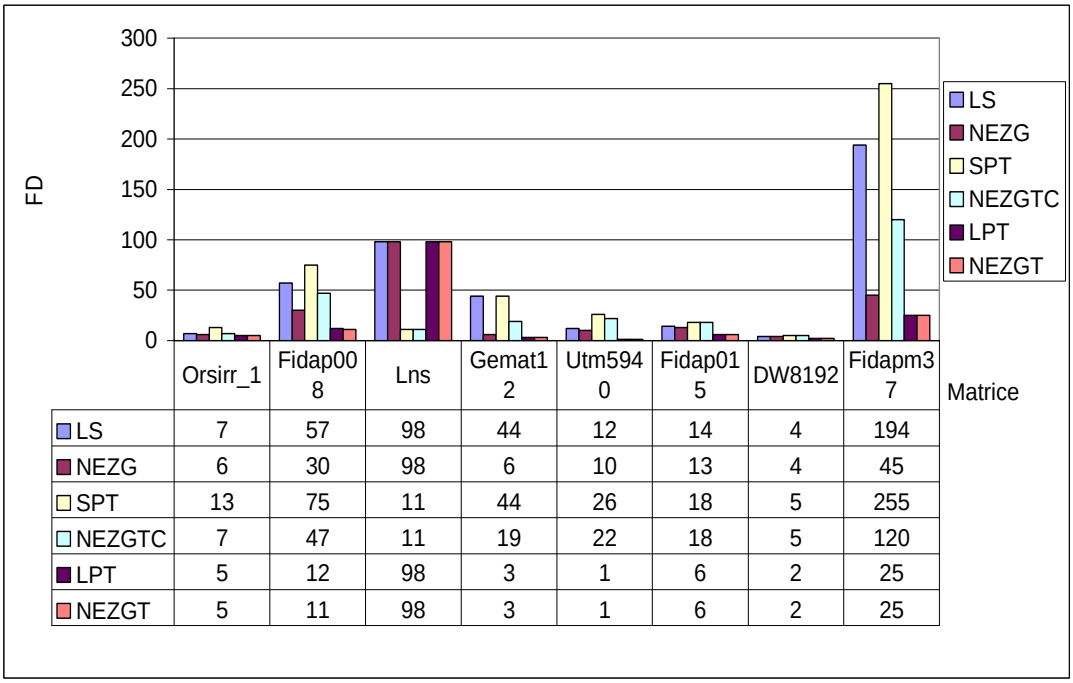
Matrice

Figure 5.5 FDR pour les matrices téléchargées de Matrix Market



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
N										
NLE	1	1	1	2	2	3	4	4	5	5
NLEG	7	5	5	5	6	5	8	7	6	6
NEZ	2	8	23	29	22	20	39	34	19	29
ProgDyn	274	2277	7661	17637	29190	43619	64916	81735	11019	13962
NEZG	9	22	36	57	69	73	94	119	148	137
NEZGT	34	122	246	389	588	831	1125	1455	1825	2237
NEZGTC	30	116	227	385	588	832	1124	1462	1823	2245

Figure 5.6 Temps CPU pour matrices quelconques générées aléatoirement, d=5%, *f*=512



	Orsirr_1	Fidap008	Lns	Gemat12	Utm5940	Fidap015	DW8192	Fidapm37
Matrice								
LS	7	57	98	44	12	14	4	194
NEZG	6	30	98	6	10	13	4	45
SPT	13	75	11	44	26	18	5	255
NEZGTC	7	47	11	19	22	18	5	120
LPT	5	12	98	3	1	6	2	25
NEZGT	5	11	98	3	1	6	2	25

Figure 5.7 Comparaison de FD pour les matrices de Matrix Market

Précisons que dans la phase 2 des algorithmes NLEG et NEZ, nous avons choisi, comme nombre maximal d'itérations, *nit*=*f*/2 alors que pour NEZG, NEZGT et NEZGTC, nous avons

choisi *nit*=N/2. Ces valeurs ont été déduites à partir des expérimentations. Il est à souligner que dans les expérimentations effectuées, le nombre d'itérations effectives dans la phase 2 n'a jamais atteint la valeur seuil de *nit* (voir Table 5.2). Nous présentons dans ce qui suit nos principaux commentaires.

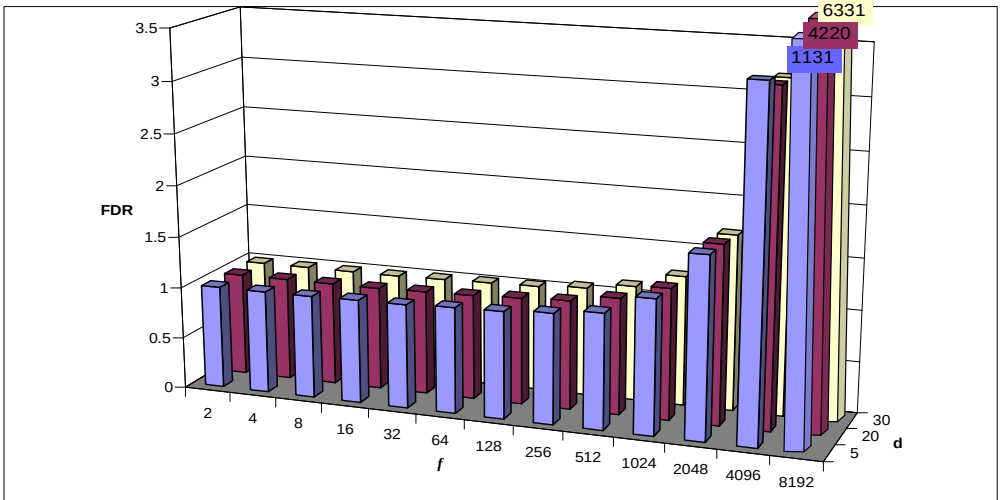
6.2 Analyse des résultats

- L'équilibrage des charges dépend fortement du nombre de fragments *f* i.e. plus *f* est élevé, plus le déséquilibrage (FD et FDR) est important. Plus précisément, c'est le rapport N/*f* qui semble avoir un impact direct sur l'équilibrage des charges. Nous avons remarqué que le déséquilibrage des charges croît lorsque ce rapport décroît. Cette constatation est vraie pour tous les algorithmes de fragmentation. A titre d'exemple, quand N/*f* est élevé, FDR peut atteindre la valeur 1. Le déséquilibrage devient très grand quand *f* dépasse N/2 i.e. quand le nombre de lignes par fragment est égal à 1 ou 2. En effet, dans ce cas et pour certains algorithmes, nous pouvons avoir un fragment contenant un élément unique et un autre fragment contenant 2N éléments. Ainsi, il devient difficile d'obtenir une amélioration.
- Le comportement des algorithmes de fragmentation en fonction de la densité de la matrice d semble aléatoire. En effet, la distribution des éléments non nuls (qui peut être aléatoire) a un impact direct sur les charges maximale et minimale. En outre, NZ (nombre d'éléments non nuls) ne semble pas avoir d'impact particulier.

Dans ce qui suit, nous présentons un histogramme (voir Figure 5.8) relatif à l'approche NLE (équilibrage du nombre de lignes contiguës par fragment) représentant les variations du facteur FDR en fonction du nombre de fragments (2, …, 8192) et de la densité (5, 20 et 30) et ce, pour N = 10000. Le même comportement a été remarqué pour des matrices bandes (voir Figure 5.9).

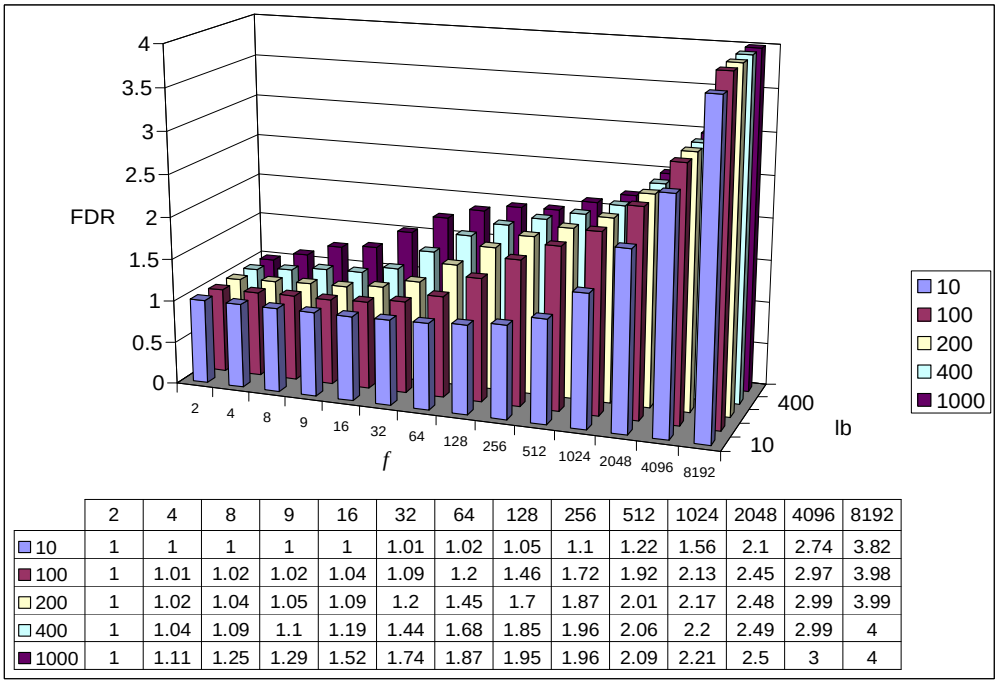
- Les temps totaux d'exécution des algorithmes de fragmentation, le temps d'exécution pour la phase 2 et le temps nécessaire pour la construction du vecteur *y* (*y* = A*x*) sont élevés lorsque *f* est grand. La densité de la matrice n'a pas d'influence sur ces résultats.
- Dans les Figures 5.3-5.6, nous représentons les FDR obtenus par les différentes approches. Notons que ProgDyn correspond à l'algorithme de programmation dynamique pour la résolution du problème de partitionnement linéaire [Ski97] qui fournit une solution optimale en un temps $O(f*N^2)$.
- Nous remarquons que les approches de construction de fragments formés par des lignes **non contiguës** fournissent toujours des FDR meilleurs. L'algorithme NEZGT basé sur le LPT

et qui comporte une phase de tri (par coût décroissant) fournit des résultats meilleurs que NEZGTC (basé sur SPT) qui comporte une phase de tri (par coût croissant).



	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
5	1	1	1	1.01	1.01	1.03	1.04	1.07	1.12	1.3	1.75	3.33	1131
20	1	1	1	1	1.01	1.02	1.04	1.06	1.13	1.27	1.73	3.21	4220
30	1	1	1	1	1.01	1.02	1.03	1.06	1.12	1.26	1.7	3.19	6331

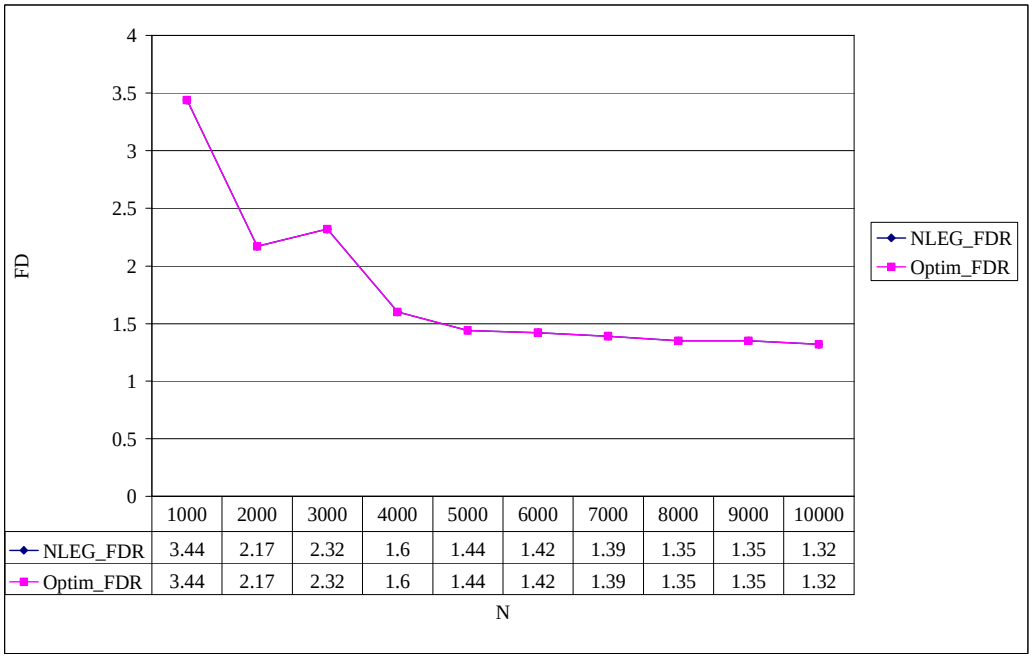
Figure 5.8 FDR pour l'approche NLE en fonction de *f* et d, N=10000 (matrice quelconque).



	2	4	8	9	16	32	64	128	256	512	1024	2048	4096	8192
10	1	1	1	1	1	1.01	1.02	1.05	1.1	1.22	1.56	2.1	2.74	3.82
100	1	1.01	1.02	1.02	1.04	1.09	1.2	1.46	1.72	1.92	2.13	2.45	2.97	3.98
200	1	1.02	1.04	1.05	1.09	1.2	1.45	1.7	1.87	2.01	2.17	2.48	2.99	3.99
400	1	1.04	1.09	1.1	1.19	1.44	1.68	1.85	1.96	2.06	2.2	2.49	2.99	4
1000	1	1.11	1.25	1.29	1.52	1.74	1.87	1.95	1.96	2.09	2.21	2.5	3	4

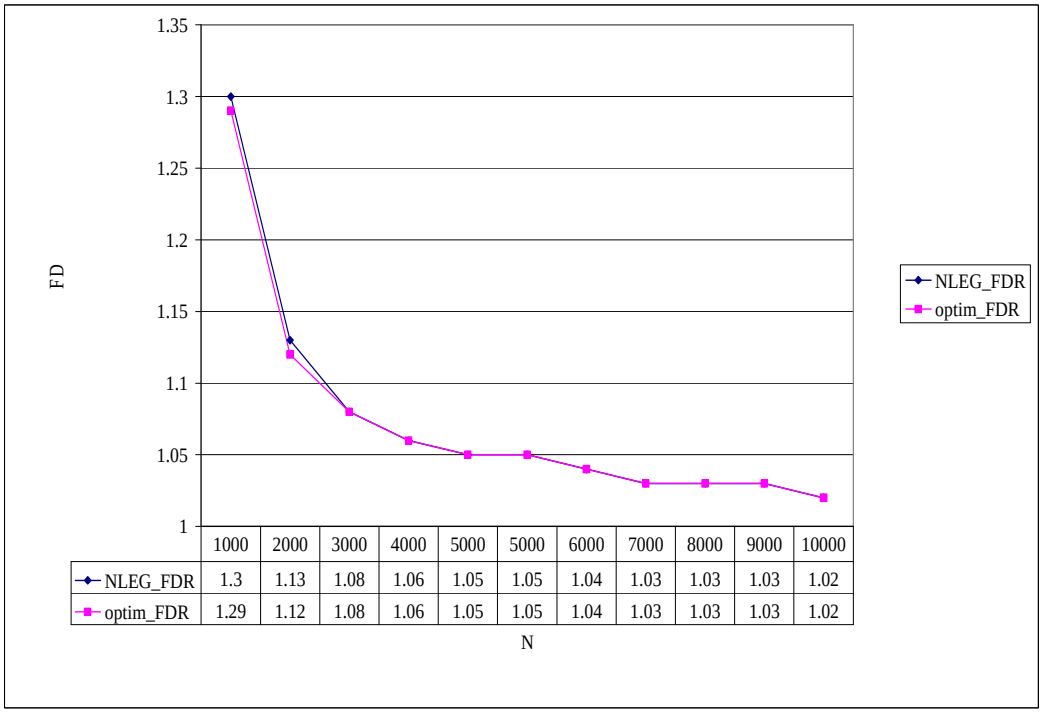
Figure 5.9 FDR pour l'approche NLE en fonction de *f* et dlb, N=10000 (matrice bande)

- Le temps d'exécution de NEZGT est plus élevé que celui de NEZG.
- Dans la Figure 5.3, les matrices générées aléatoirement sont de tailles différentes mais de même densité (5%), alors que dans la figure 5.5 les matrices de Matrix Market sont de tailles et de densités différentes.
- Les FDR fournis par ProgDyn et NEZ sont, dans la plupart des cas, très proches. Toutefois, le temps CPU pour ProgDyn est beaucoup plus élevé.
- Dans la Figure 5.7, nous comparons les heuristiques NEZG, NEZGT et NEZGTC respectivement aux algorithmes LS, LPT et SPT. Nous remarquons ainsi que dans la plupart des cas, les heuristiques que nous avons conçues améliorent l'équilibrage des charges, excepté pour NEZGT, où dans certains cas, la phase d'amélioration n'a aucun effet (pas d'amélioration).
- Dans le but de comparer nos résultats approchés aux solutions optimales, nous nous sommes restreints à l'algorithme NLEG (voir section 4.1.2 ci-dessus). Dans ce but, nous avons effectué une série de tests sur des matrices générées aléatoirement ayant des tailles N entre 1000 et 10000. Etant donné un nombre *f* de fragments et p le reste de la division Euclidienne de N par *f* i.e. N = *f*∗q + p, nous avons généré les C_N^p fragmentations possibles (voir section 4.1.2) et retenu celles qui étaient optimales. Pour toutes les valeurs testées de *f*, NLEG s'est révélée optimale, que ce soit pour les matrices aléatoires ou pour les matrices bandes. Dans les Figures 5.10 et 5.11 nous présentons les résultats pour *f* = 9.



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NLEG_FDR	3.44	2.17	2.32	1.6	1.44	1.42	1.39	1.35	1.35	1.32
Optim_FDR	3.44	2.17	2.32	1.6	1.44	1.42	1.39	1.35	1.35	1.32

Figure 5.10 FDR de NLEG - Matrices quelconques aléatoires, *f*=9



	1000	2000	3000	4000	5000	5000	6000	7000	8000	9000	10000
NLEG_FDR	1.3	1.13	1.08	1.06	1.05	1.05	1.04	1.03	1.03	1.03	1.02
optim_FDR	1.29	1.12	1.08	1.06	1.05	1.05	1.04	1.03	1.03	1.03	1.02

Figure 5.11 FDR de NLEG - Matrices bandes aléatoires, *f*=9

7. Conclusion

Dans ce chapitre, nous avons proposé six heuristiques de fragmentation par lignes pour matrice creuse, dénommés NLE, NLEG, NEZ, NEZG, NEZGT et NEZGTC. Elles consistent toutes à décomposer la matrice, de taille N, en *f* blocs de lignes. Nous pouvons classer ces 5 algorithmes en deux groupes G1 et G2. G1 (resp. G2) est constitué par les trois premiers (resp. les trois derniers) algorithmes. Les algorithmes de G1 (resp. G2) construisent des blocs contenant des lignes contiguës (resp. non nécessairement contiguës). Notre objectif était d'obtenir des fragments équilibrés i.e. des fragments ayant (presque) le même nombre d'éléments non nuls. Les algorithmes conçus, de complexités différentes induisent des solutions également différentes. En fait, pour les algorithmes de G1, nous avons une complexité en O(*f*) pour NLE et dans le pire des cas, une complexité en $O(N^2)$ pour NLEG et NEZ. Chaque algorithme du second groupe possède une complexité dans le pire des cas égale à $O(N^2)$. Concernant l'équilibrage des charges, il a été mesuré par le facteur de déséquilibrage relatif FDR (charge maximale divisée par charge minimale). Dans la plupart des cas, ces facteurs dans G2 sont inférieurs donc meilleurs que dans G1.

Les expérimentations que nous avons réalisées montrent que pour les matrices aléatoires (resp. matrices bandes), la densité (resp. demi-largeur de bande) n'a pas d'impact sur le facteur de

déséquilibrage. Nous avons aussi remarqué que ce dernier croît avec *f*. Ceci est vrai aussi bien pour les matrices quelconques que pour les matrices bandes. De plus, lorsqu'on utilise l'algorithme NLEG (groupe G1), le FDR devient très élevé lorsque *f* dépasse N/2. En effet, dans ce cas, chaque fragment contient une à deux lignes. Du fait que les éléments non nuls sont distribués (en général) non uniformément sur les lignes de la matrice, il y a une forte probabilité d'avoir un fragment ayant un très faible nombre d'éléments non nuls (<<N) et un autre ayant un grand nombre d'éléments non nuls (≥ N). Lorsque N est assez élevé, la différence entre le nombre d'éléments non nuls des deux fragments est aussi élevée, ainsi le déséquilibrage de charge peut être très important. En conséquence, pour N grand et lorsque nous utilisons des nœuds homogènes, ce déséquilibrage peut engendrer un déséquilibrage au niveau du temps d'exécution sur chaque nœud, donc de mauvaises performances globales de calcul.

Par conséquent, le traitement de matrices de grande taille sur un SDGE n'est intéressant que pour des tailles (N) nettement supérieures au nombre de nœuds du système (*f*) i.e. N >> *f*. En d'autres termes, il convient d'ajuster taille et nombre de nœuds à cibler pour la distribution du PMVC. Ainsi, l'utilisation de tous les noeuds disponibles d'un SDGE, donc de sa puissance totale ne permet pas forcément d'obtenir les meilleures performances globales de l'application. Utiliser un peu moins de ressources et mieux répartir les charges est plus judicieux. Ceci soulève la question intéressante de la détermination du nombre optimal de fragments pour une taille de matrice donnée.

Dans le chapitre suivant, nous tenons compte des résultats présentés dans ce chapitre, pour étudier le traitement et le post-traitement du PMVC sur un système distribué à grande échelle.

CHAPITRE 6.

DISTRIBUTION DU PRODUIT MATRICE-VECTEUR CREUX SUR UN SYSTÈME DE CALCUL À GRANDE ÉCHELLE

1. Introduction

Dans ce chapitre, nous nous intéressons à l'étude de la distribution du produit matrice-vecteur creux (PMVC) sur un système volontaire, à savoir XtremWeb-CH. Dans la section 2, nous définissons une géométrie virtuelle pour un système distribué à grande échelle (SDGE) et précisons les hypothèses de travail. Pour ce faire, nous utilisons une analogie avec le modèle parallèle introduit dans [PeE96]. Dans la section 3, nous détaillons notre approche pour la distribution du PMVC sur un SDGE. La section 4 est consacrée à l'évaluation théorique des performances. La section 5 est dévolue à la présentation de la plate-forme cible ainsi que des résultats expérimentaux. Finalement, dans la section 6, nous concluons le chapitre.

2. Définition d'une géométrie virtuelle d'un SDGE

Afin d'étudier la distribution du PMVC sur un système de calcul à grande échelle, nous adoptons un modèle de programmation inspiré de celui de la programmation parallèle de données, généralement associé au calcul massivement parallèle [PeE96]. Ainsi, les processeurs sont remplacés par les pairs (nœuds) et les opérations par les tâches.

Nous définissons un flux de contrôle unique pour les tâches telles que chacune effectue le traitement d'un ensemble de données structurées sur un nœud virtuel. Dans ce chapitre, nous considérons uniquement des géométries virtuelles de dimension un ou deux.

Pour des raisons de simplicité, nous supposons que tous les nœuds virtuels possèdent les mêmes caractéristiques ainsi que les mêmes performances.

Dans un algorithme distribué donné, les tâches n'ont pas nécessairement le même coût. Des échanges de données entre les nœuds virtuels peuvent être effectués explicitement.

La matrice en entrée étant décomposée en fragments (voir chapitre 5, section 4), nous supposons que le nombre P de nœuds virtuels est, au moins, du même ordre que le nombre de fragments de la matrice. Du fait que dans les systèmes volontaires à grande échelle actuels, la taille des données est souvent plus grande que le nombre de nœuds physiques [AoP04][Gri04], nous devons simuler plusieurs nœuds virtuels sur un même nœud physique.

Nous supposons également, pour des raisons de simplicité, que les communications inter-nœuds sont directes, bien que dans les systèmes volontaires à grande échelle actuels, les communications soient centralisées autour d'une machine dédiée [AKK03][AoP04][Lod04].

3. Distribution du PMVC sur un SDGE

3.1 Présentation du problème

Soient A une matrice creuse N×N ayant *NZ* éléments non nuls et *x* un vecteur de taille N. Notre objectif est de calculer le produit matrice vecteur creux (PMVC) $y = Ax$. Ce noyau de calcul sur un système distribué volontaire est illustré dans la Figure 6.1.

L'approche que nous concevons pour la distribution de ce calcul consiste à utiliser une géométrie virtuelle à une dimension pour les nœuds. La matrice A sera décomposée en *f* fragments. En se basant sur l'étude de la fragmentation effectuée dans le chapitre 5, nous considérons uniquement la fragmentation en blocs de lignes i.e. chaque fragment est constitué d'un ensemble de lignes de A stockées dans le format CSR (voir chapitre 2, section 3.2.2).

Un fragment (bloc de lignes), noté A_i (i= 1...*f*), est envoyé au nœud virtuel *i*. Concernant le vecteur *x*, il est diffusé à tous les nœuds. Par la suite, en parallèle, chaque nœud multiplie un fragment A_i de la matrice compressée A par le vecteur *x*. Nous obtenons ainsi un fragment vecteur y_i du vecteur résultat *y*.

Nous supposons que la mémoire locale de chaque nœud participant à l'application est capable de stocker un fragment de la matrice A en plus des vecteurs *x* et *y*.

Précisons, de plus, que nous négligeons l'espace nécessaire pour le stockage des structures de compression IA et JA vu qu'elles correspondent à des vecteurs d'entiers [HME07]. Une fois cette phase de calcul terminée, une phase de communication est alors nécessaire pour rassembler les fragments du vecteur résultat *y* sur un nœud unique. Cette phase correspond à une opération de réduction.

Rappelons que si nous utilisons la fragmentation en blocs de colonnes, nous aurons besoin, après le calcul du produit fragment-vecteur, de communiquer les résultats partiels $y^{(j)}$ (j=1...*f*, $y^{(j)}$ étant de dimension N) afin de réduire le vecteur résultat final *y*. Ajoutons aussi que sur les SDGE, nous devons minimiser les communications entre les nœuds puisqu'elles sont effectuées via le réseau Internet qui reste encore assez lent.

Nous utilisons l'algorithme de la Figure 6.2 pour la phase de calcul creux du produit fragment-vecteur, où le fragment est compressé dans le format CSR [EHM05]. Précisons qu'ici, n est le nombre de lignes du fragment, alors que les vecteurs $\mathcal{A}$, IA et JA sont ceux définis par le format CSR vu au chapitre 2.

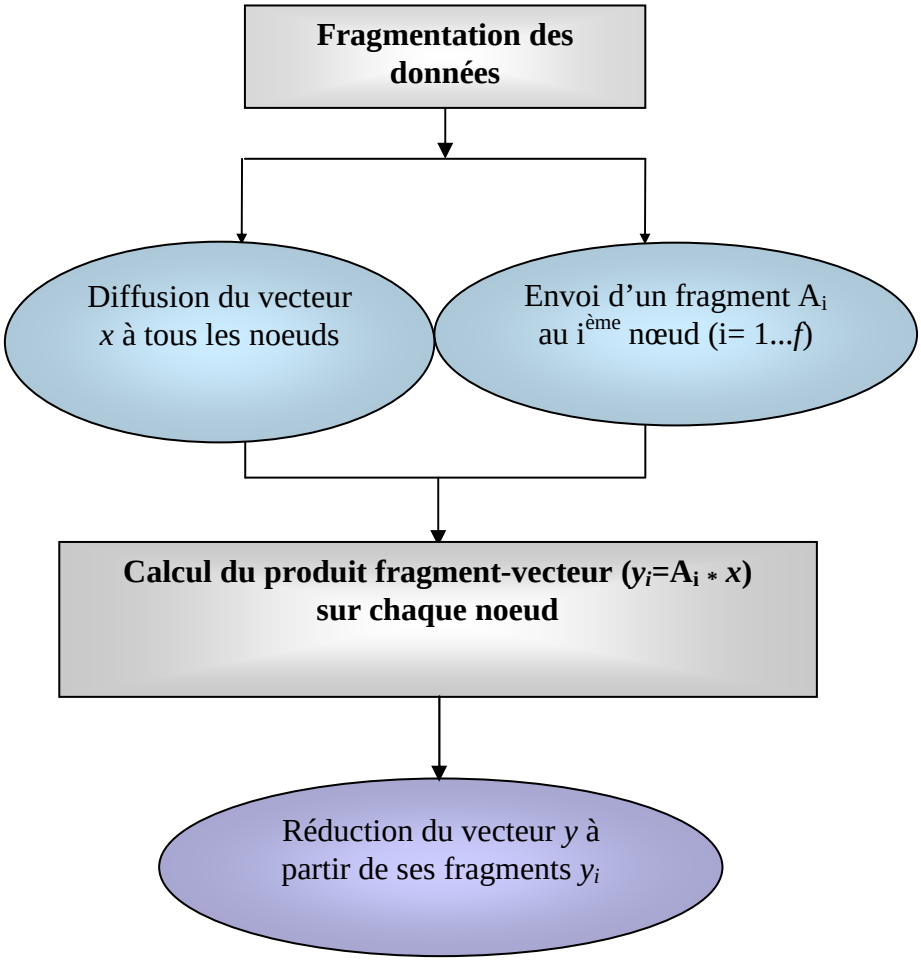


Figure 6.1 Calcul du PMVC sur un système distribué à grande échelle

```
Do  k=1,n
    var=0 ; pt=IA[k+1]
    Do  j= IA[k] , pt
        var=var+ A[j]*x[JA[j]]
    Enddo
    yi[k]=var
Enddo
```

Figure 6.2 Produit fragment-vecteur creux

L'algorithme du produit-matrice vecteur creux (PMVC) de la Figure 6.2 correspond à une version optimisée en se basant sur l'étude effectuée dans le chapitre 4 [EHM05][EHM09].

3.2 Fragmentation des données

Dans le chapitre 5, nous avons présenté trois approches pour décomposer une matrice creuse, notée A, en *f* fragments (blocs) [HME07].

Dans ce chapitre, nous nous intéressons uniquement à l'algorithme NLE qui consiste à décomposer la matrice A en blocs (fragments) de lignes ayant le même nombre de lignes. Notre étude restreinte à cette approche de fragmentation peut être facilement généralisée pour le reste des approches vues dans le chapitre 5. Rappelons que dans la décomposition NLE, nous ne nous intéressons pas à l'information concernant la distribution des éléments non nuls sur les lignes de la matrice. En fait, nous effectuons la fragmentation sur une matrice non compressée. L'exemple de la Figure 6.3 illustre la décomposition NLE en deux blocs.

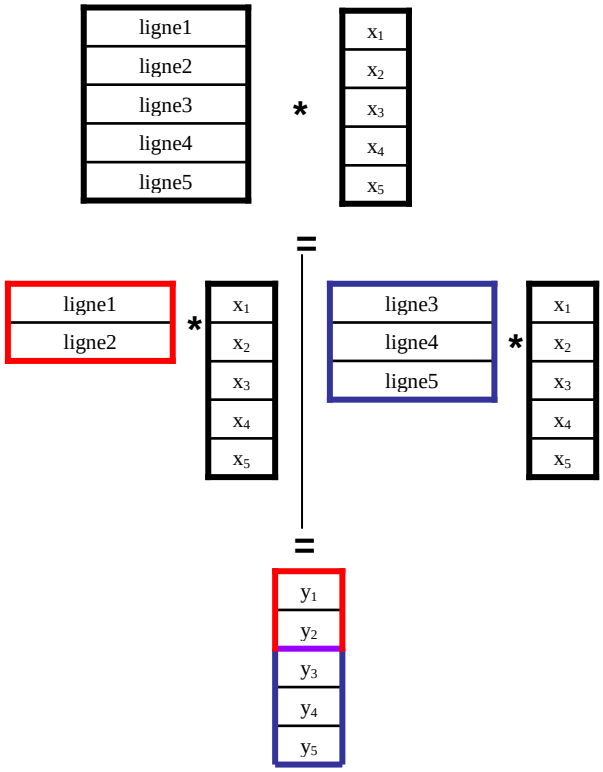


Figure 6.3 Décomposition NLE

3.3 Construction du vecteur résultat *y*

Supposons que $f = P = s^{\alpha}$, (s, $\alpha \in$ N). Une fois les fragments y_i (i=1.. f) de y sont calculés par les différents nœuds, ils sont rassemblés par ensembles de s fragments jusqu'à obtenir finalement le vecteur y au niveau d'un seul nœud. Le paramètre s est appelé *facteur de*

réduction (FR). Ainsi, dans ce cas, le graphe de tâches a une structure d'arbre complet *s*-aire comme illustré dans la Figure 6.4 (s=3).

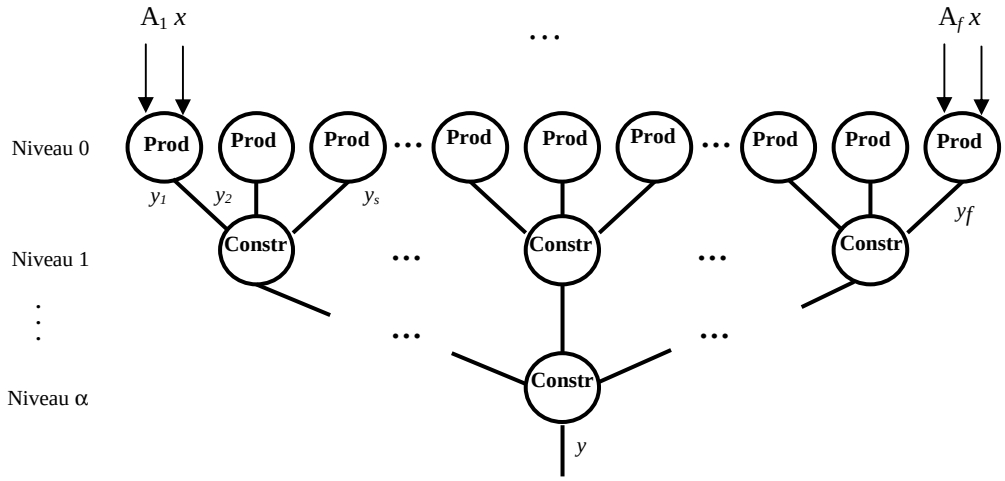


Figure 6.4 Construction du vecteur résultat

4. Evaluation théorique des performances

4.1 Complexité de calcul

La matrice A étant subdivisée en f blocs (fragments), nous avons donc f fragments à distribuer sur P (=f) nœuds virtuels. Notons que cette distribution est effectuée une seule fois au début de la distribution du calcul du PMVC. Le produit fragment-vecteur creux (PFVC) d'un fragment compressé A_i (i=1...f) par le vecteur x est effectué en utilisant l'algorithme de la Figure 6.2. Sa complexité pour un fragment A_i contenant NZ_i éléments non nuls est $2*NZ_i$. Soit Fc_i la complexité pour un fragment A_i. Nous avons donc:

$$Fc_i = 2*NZ_i = O(NZ_i)$$

Soit $NZMAX = \max(NZ_k)$, k=1…P. Remarquons que $NZMAX$ dépend de P ainsi que de la distribution des éléments non nuls sur les lignes de la matrice. La complexité de calcul de la distribution du PMVC $y = Ax$ est donc:

$$C_{NLE} = 2*NZMAX = O(NZMAX) = O(NZ/f)$$

4.2 Complexité de communication

Comme illustré dans la Figure 6.1, nous devons distribuer la matrice A sur P nœuds et diffuser le vecteur x à tous les nœuds. Chaque nœud i reçoit alors (nombre de réels) :

$$DonnéesReçues1 = N+NZ_i$$

Notons que la valeur de NZ_i est aléatoire mais bornée par N/*P* (i.e. il y au moins 1 élément non nul par ligne) et N^2/P (i.e. il y a au plus N éléments non nuls par ligne). La quantité moyenne des données reçue est alors :

$$DonnéesReçues1 = N(1+(1+N)/2P)=O(N^2/P)$$

Nous avons besoin, par la suite, de communications pour construire le vecteur *y*. Rappelons que le graphe de tâches représentant la distribution du PMVC sur un système de calcul volontaire possède une structure d'arbre (voir Figure 6.4). L'arbre est composé de $\log_s(P)+1 = \alpha+1$ niveaux numérotés de 0 à α. Après l'exécution du module *Prod*, les nœuds du niveau 0 doivent envoyer *P* fragments de *y*, chacun de taille N/*P*, aux nœuds du niveau 1. Par la suite, les nœuds du niveau 1 envoient *P/s* vecteurs, chacun de taille $s*N/P$, au niveau 2…jusqu'à ce que le niveau α-1 envoie deux vecteurs, chacun de taille N/2, au niveau α. Ainsi, chaque niveau envoie un volume de données de taille N au niveau suivant. Du fait que le nombre de niveaux est égal à α+1, la quantité de données envoyées pour construire *y* est :

$$ENVOI = N*\alpha = N*\log_s(P)$$

Concernant les données reçues, un nœud reçoit *s* fragments de *y* à partir de *s* nœuds différents. Un nœud du niveau k (k=1...α) reçoit *s* fragments de taille $s^{k-1}*N/P$ chacun. Le volume de données reçues par un nœud du niveau k est alors s^k*N/P, k=1…α. Ainsi, le niveau k reçoit $(P/s^k)*(s^k*N/P)=N$ données du niveau k-1.

La quantité de données reçue par un nœud interne est alors :

$$DonnéesReçues2 = N*\alpha=N*\log_s(P)$$

Pour conclure, la quantité totale de données reçue quand nous utilisons la décomposition NLE est la suivante :

$$\begin{aligned} RECEPTION &= DonnéesReçues1 + DonnéesReçues2 \\ &= N(1+(1+N)/2P)+N*\log_s(P) \\ &= O(N^2/P+ N*\log_s(P)) \end{aligned}$$

Notons qu'on a $ENVOI \geq N$. En fait, *ENVOI* peut être égal à N lorsque $\log_s(P)=1$. Ceci a lieu quand le facteur de réduction *s* est égal à *P*, autrement dit, quand le graphe de tâches du PMVC contient une seule tâche *Constr* qui rassemble tous les vecteurs partiels y_i pour construire *y* (c'est-à-dire utilisant une communication all-to-one).

5. Expérimentations

Dans le but d'évaluer les performances de la distribution du PMVC sur un système distribué à grande échelle, nous avons installé une plate-forme expérimentale XtremWeb-CH. Nous

avons analysé les performances en fonction de la taille de la matrice creuse, de sa densité, du nombre de fragments obtenus après la décomposition, du facteur de réduction pour la construction du vecteur résultat final, etc.

5.1 Plate-forme XtremWeb-CH

XtremWeb-CH est une plate-forme de calcul distribué à grande échelle dite aussi de calcul volontaire. C'est une version améliorée de XtremWeb. Cet environnement permet l'exécution d'applications parallèles/distribuées composées de modules communicants [AbB07][Abd08].
L'application est décrite par un graphe de flot de données où les nœuds sont les modules de traitement et les liens inter-nœuds représentent les échanges de données entre ces modules.
Le graphe de flot de données est représenté par un fichier XML qui décrit les modules et les données échangées.
Un module n'est affecté à un worker (pair, nœud) que si ses données d'entrée sont disponibles. Une donnée est dite disponible si elle a été générée par un module précédent. Initialement, la seule donnée d'entrée disponible est celle qui est fournie par l'utilisateur pour exécuter le (ou les) premier(s) module(s). Les différents modules de l'application (à l'exception du premier) sont initialement "bloqués" i.e. ils ne peuvent être exécutés puisque leurs données d'entrée ne sont pas disponibles.
A la fin de l'exécution d'un module par un worker, celui-ci informe le serveur. Un processus particulier de XtremWeb-CH, appelé "espion", parcourt alors tous les modules bloqués pour débloquer ceux dont les données d'entrée sont maintenant disponibles. Le processus ordonnanceur de XtremWeb-CH se charge alors de les affecter à des workers libres. La communication entre les modules se fait directement entre les workers exécutant ces modules (sans passer par le serveur).
Cette nouvelle fonctionnalité est une extension de XtremWeb-CH par rapport à XtremWeb. En effet, dans XtremWeb les communications entre workers ne peuvent avoir lieu que via le serveur.
Cette fonctionnalité a été rajoutée dans le but de s'approcher davantage du concept du P2P, à savoir que tous les nœuds doivent avoir les mêmes fonctions, le même "pouvoir". L'architecture de XtremWeb-CH est donc partiellement décentralisée par rapport à celle de XtremWeb. Lorsqu'un worker finit son exécution, il stocke ses résultats dans un fichier temporaire et envoie au serveur XtremWeb-CH un signal lui indiquant la fin normale de son exécution ainsi que l'emplacement du fichier résultat (adresse IP et répertoire).

Lorsque le serveur XtremWeb-CH affecte les modules nouvellement débloqués à des workers libres, il leur envoie aussi l'emplacement de leurs données d'entrée. Le transfert des données se fait donc directement entre workers [DiE06].

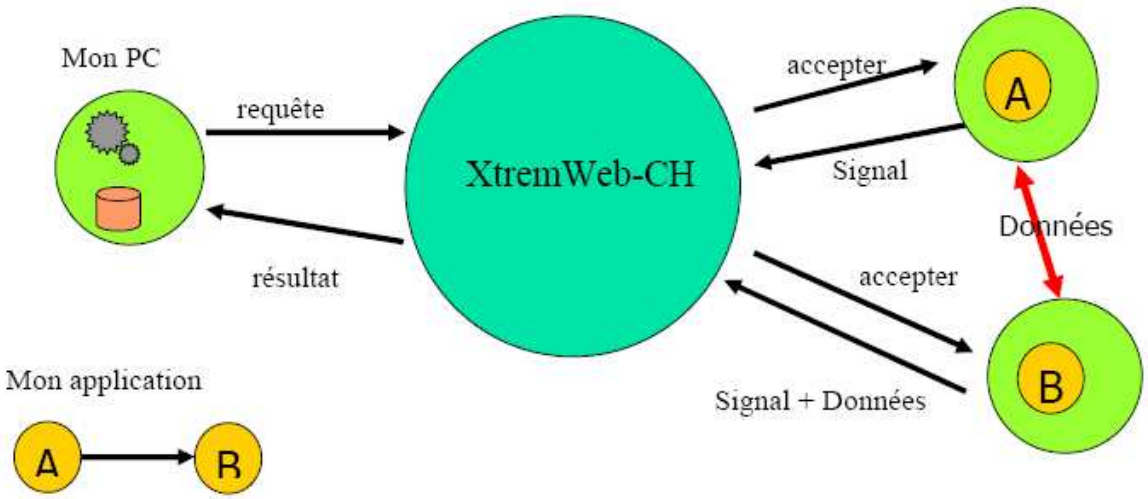


Figure 6.5 Architecture de XtremWeb-CH

Déploiement de XtremWeb-CH:

Le serveur de XtremWeb-CH tourne sous Linux. Les workers tournent sous Linux et Windows. La version Windows du module worker permettra à terme de disposer d'un nombre important de machines workers.

5.2 Plate-forme matérielle

Nous avons installé une plate-forme XtremWeb-CH à l'Institut Supérieur d'Informatique (Tunis-Ariana). Le module coordinateur a été installé sur un PC Pentium dual-core (fréquence : 2GHz, RAM : 2Go, mémoire cache : 2Mo et disque dur : 120 Go). Concernant les noeuds, nous avons utilisé 9 PC UNIX ayant des performances similaires : Pentium dual-core de fréquence 3 GHz. Toutes les machines ont été connectées par un réseau local ayant une bande passante de 100Mo/sec.

5.3 Paramètres testés

Afin d'évaluer les performances du PMVC sur la plate-forme XtremWeb-CH, nous avons utilisé un échantillon de matrices de tailles et de densités différentes que nous avons soit générées aléatoirement, soit téléchargées du site Matrix Market [Mat07]. Précisons que pour le premier ensemble, étant donné une taille N et une densité d, nous avons conçu un générateur qui fournit en sortie une matrice quelconque où les éléments non nuls sont distribués aléatoirement sur les lignes de la matrice. Concernant le second ensemble, il est en fait restreint à une seule matrice dont les caractéristiques sont données dans la Table 6.1.

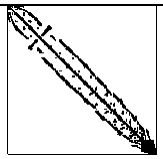
Structure	Nom	Taille	Densité
	Fidapm37	9152	0.91

Table 6.1 Caractéristiques de la matrice Fidapm37 de Matrix Market.

Notons qu'en se basant sur l'étude de l'optimisation de l'algorithme du PMVC (voir chapitre 4) [EHM09], nous utilisons l'option de compilation –O2 pour compiler l'algorithme du PMVC présenté dans la Figure 6.2 et utilisé par les nœuds pour calculer le produit fragment-vecteur.

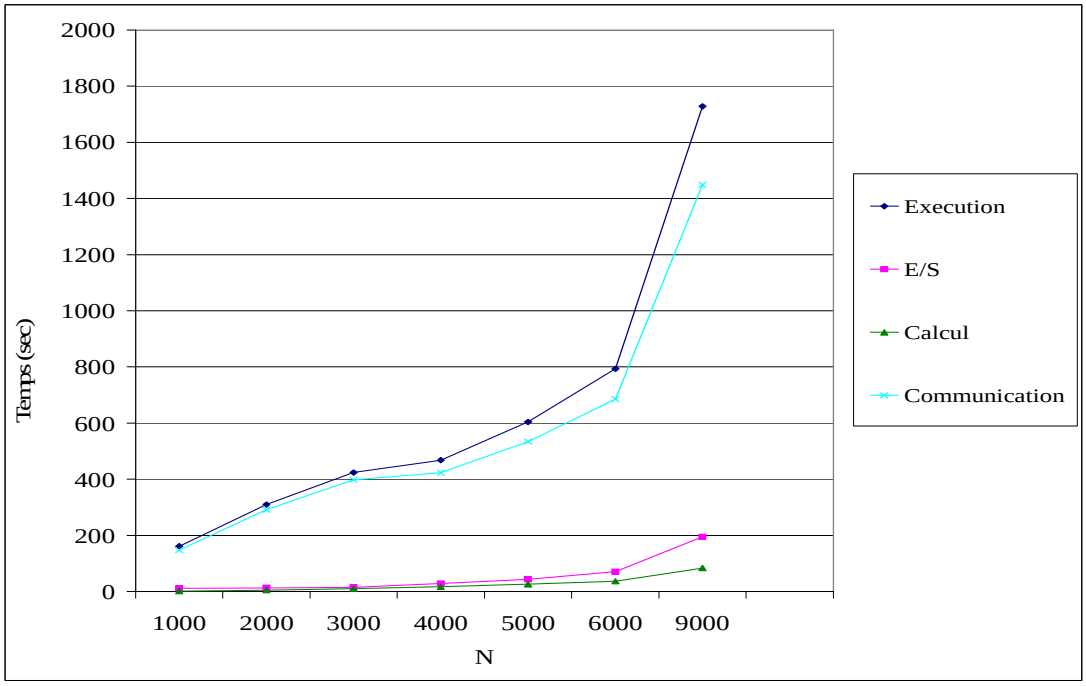
L'évaluation des performances du PMVC sur XtremWeb-CH a consisté à mesurer 4 paramètres : (i) la durée totale de l'exécution de l'application, (ii) la durée totale de calcul, (iii) la durée totale des E/S et (iv) la durée totale de communication. Nous en déduisons ensuite le ratio d'E/S (rapport temps total d'E/S sur temps total d'exécution), le ratio de calcul (rapport temps total de calcul sur temps total d'exécution) et le ratio de communication (temps total de communication sur temps total d'exécution). Ces trois derniers paramètres permettront de raffiner l'analyse des résultats.

Précisons qu'à cause de contraintes techniques liées au nombre de nœuds de la plate-forme et leur disponibilité, nous nous sommes restreints à 9 nœuds et n'avons effectué qu'une seule exécution par test.

5.4 Résultats et interprétations

5.4.1 Evaluation des performances en fonction de la taille de la matrice

Les valeurs des paramètres sont les suivants : densité de la matrice creuse d=10, nombre de fragments *f*=64, facteur de réduction FR=2. La taille N de la matrice varie dans l'intervalle [1000 9000]. Les résultats sont présentés dans les Figures 6.6 et 6.7.



N	1000	2000	3000	4000	5000	6000	9000
Temps total d'exécution (s)	161	310	424	468	604	793	1728
Temps total d'E/S (s)	11.48	13.45	15.73	28.28	43.98	70.7	194.87
Ratio d'E/S (%)	7	4	4	6	7	9	11
Temps total de calcul (s)	2.06	4.99	9.94	16.81	25.98	36.85	84.21
Ratio de calcul (%)	1	2	2	3	4	5	5
Temps total de communication (s)	147.46	291.56	398.33	422.91	534.04	685.45	1448.92
Ratio de communication (%)	91	94	94	90	88	86	84

Figure 6.6 Performances du PMVC en fonction de la taille de la matrice, d=10, FR=2 et *f*=64

Nous remarquons ainsi que lorsque la taille N de la matrice augmente, la durée totale d'exécution ainsi que la durée de communication augmentent. Ce résultat peut être justifié par le fait que lorsque N croît (pour une densité fixée à d=10), le nombre d'éléments non nuls augmente, donc la granularité des tâches augmente. En conséquence, la quantité de données

traitée par les calculs augmente, d'où la durée totale d'exécution augmente. De plus, la quantité de données circulant sur le réseau durant l'exécution de l'application augmente donc la durée de communication augmente.

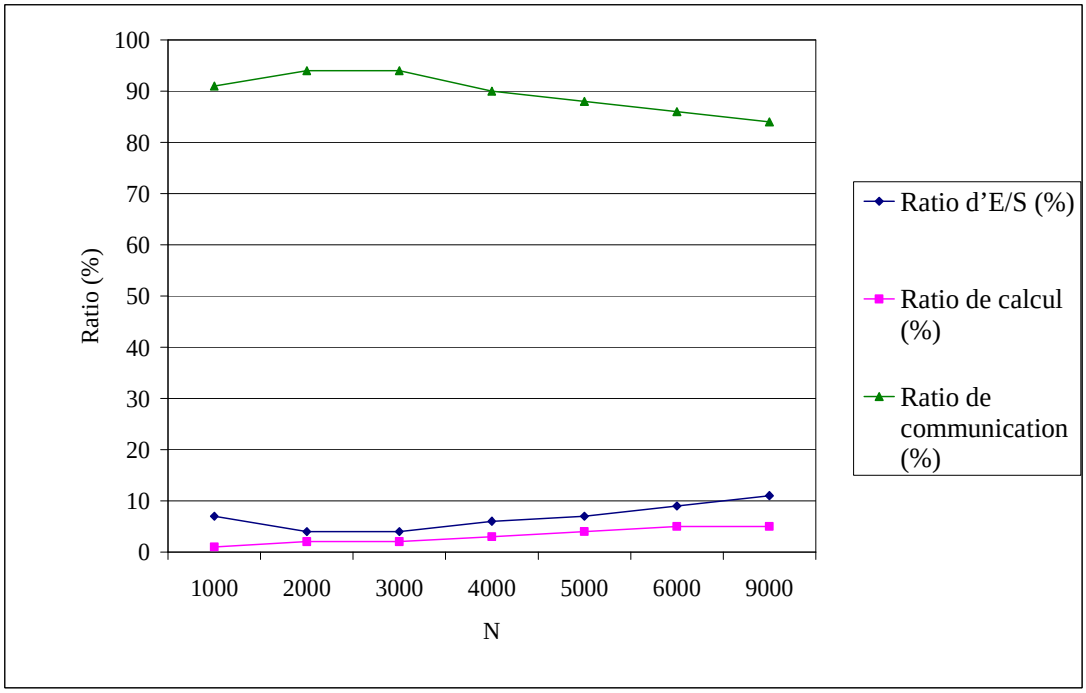


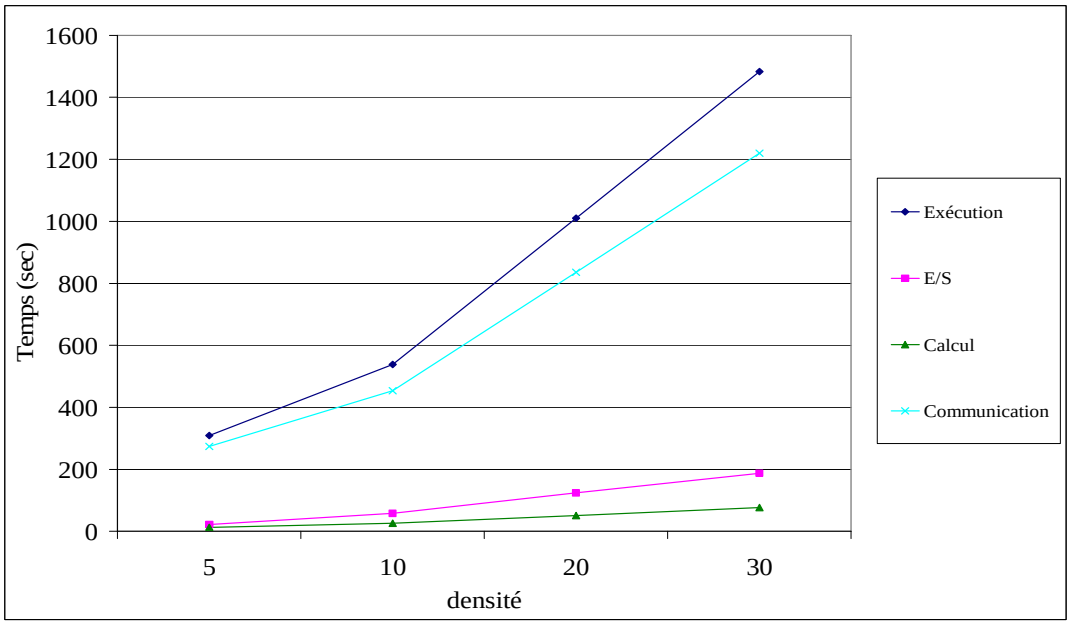
Figure 6.7 Ratios en fonction de la taille de la matrice, d=10, FR=2 et *f*=64

Notons que, par rapport au temps total d'exécution, le ratio de communication décroît lorsque N croît (à partir de N=2000) et ce, en faveur du ratio de calcul qui croît quant à lui. Ainsi, quand N augmente, l'efficacité de la distribution du PMVC s'améliore. D'un autre côté, le ratio d'E/S croît aussi avec N (à partir de N=2000).

5.4.2 Evaluation des performances en fonction de la densité

Valeurs des paramètres : taille de la matrice N=5000, nombre de fragments *f*=64, densité de la matrice variant dans l'intervalle [5 30].

Dans la Figure 6.8, nous remarquons que comme dans le cas précédent, lorsque la densité de la matrice croît, le nombre d'éléments non nuls croît aussi, ce qui influe sur la durée totale d'exécution, la durée totale de calcul, la durée totale d'E/S et la durée totale de communication qui tous croissent. Concernant les ratios, nous constatons comme auparavant (d fixé et N variable) que le ratio de communication décroît lorsque d croît alors que les ratios d'E/S et de calcul croissent.



Densité	5	10	20	30
Temps total d'exécution (s)	309	538	1010	1483
Temps total d'E/S (s)	21.76	58.06	123.73	187.28
Ratio d'E/S (%)	7	11	12	13
Temps total de calcul (s)	13.29	26.03	51.09	76.44
Ratio de calcul (%)	4	5	5	5
Temps total de communication (s)	273.95	453.91	835.18	1219.28
Ratio de communication (%)	89	84	83	82

Figure 6.8 Performances du PMVC en fonction de la densité de la matrice, N=5000 et *f*=64

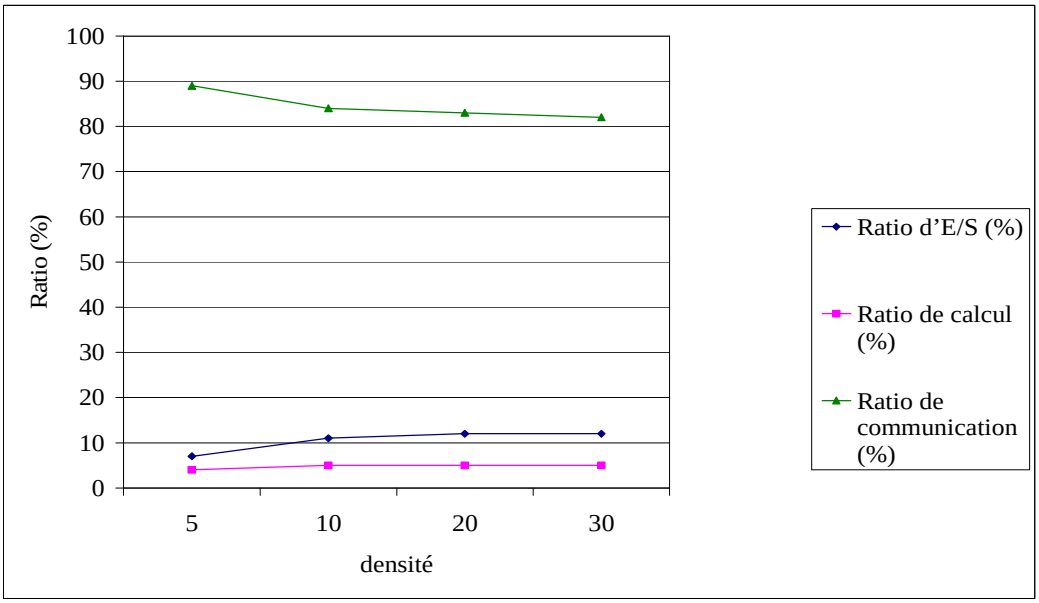
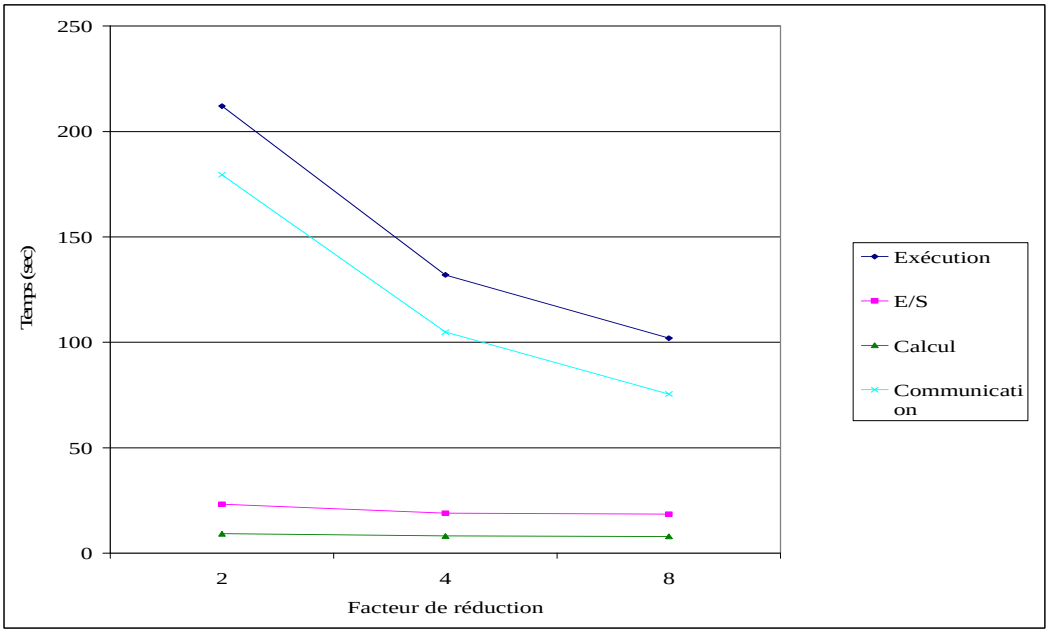


Figure 6.9 Ratios en fonction de la densité de la matrice, N=5000 et *f*=64

5.4.3 Evaluation des performances en fonction du facteur de réduction

Nous avons téléchargé la matrice FIDAPM37 du site Matrix Market [Mat07]. Les caractéristiques de la matrice sont données dans la Table 6.1. Nous avons décomposé la matrice en 64 fragments et avons fait varier le facteur de réduction. Nous avons obtenu les résultats représentés par les Figures 6.10 et 6.11. Nous constatons que lorsque nous augmentons le facteur de réduction (FR), la durée de communication décroît. En d'autres termes, lorsque le nombre de tâches dans le graphe de tâches diminue, le nombre ainsi que la quantité de données échangées entre les tâches diminuent. En conséquence, nous concluons que la structure arbre du graphe des tâches n'est pas efficace dans le cas de la distribution du PMVC sur un système volontaire. Il est plus intéressant, dans ce cas, d'utiliser une seule tâche pour effectuer la construction du vecteur résultat final en rassemblant tous les résultats partiels, que d'utiliser plusieurs tâches pour l'effectuer. Ajoutons, comme ce fut le cas auparavant, que le ratio de communication décroît lorsque FR augmente, alors que les ratios d'E/S et de calcul croissent.



Facteur de réduction	2	4	8
Temps total d'exécution (s)	212	132	102
Temps total d'E/S (s)	23.23	18.94	18.52
Ratio d'E/S (%)	11	14	18
Temps total de calcul (s)	9.27	8.18	7.97
Ratio de calcul (%)	4	6	8
Temps total de communication (s)	179.5	104.88	75.51
Ratio de communication (%)	85	79	74

Figure 6.10 Performances du PMVC pour la matrice FIDAPM37, *f*=64 et FR variable

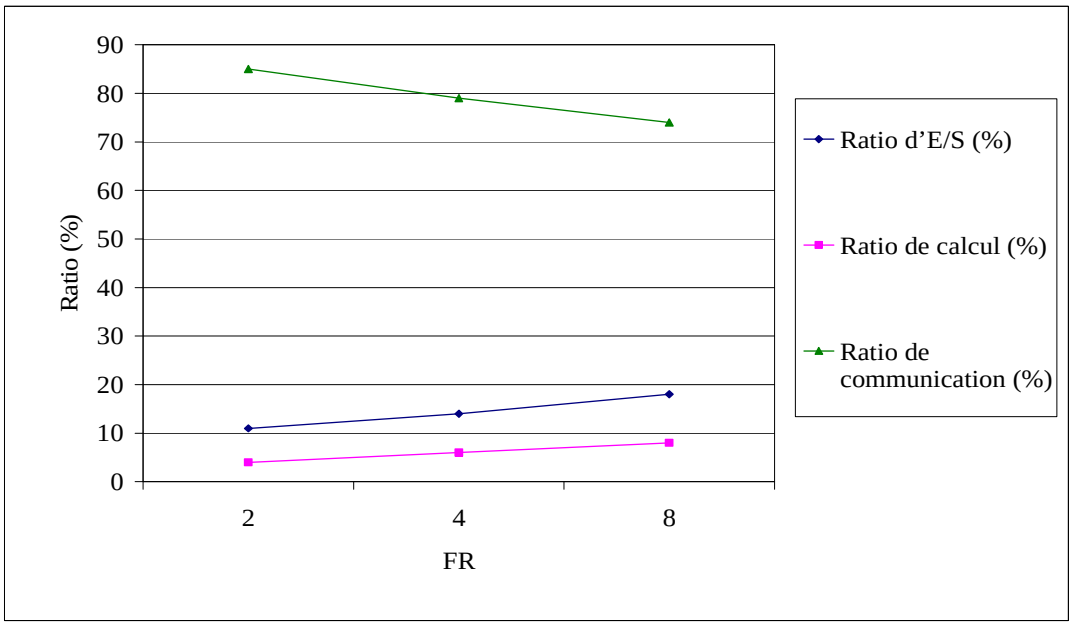
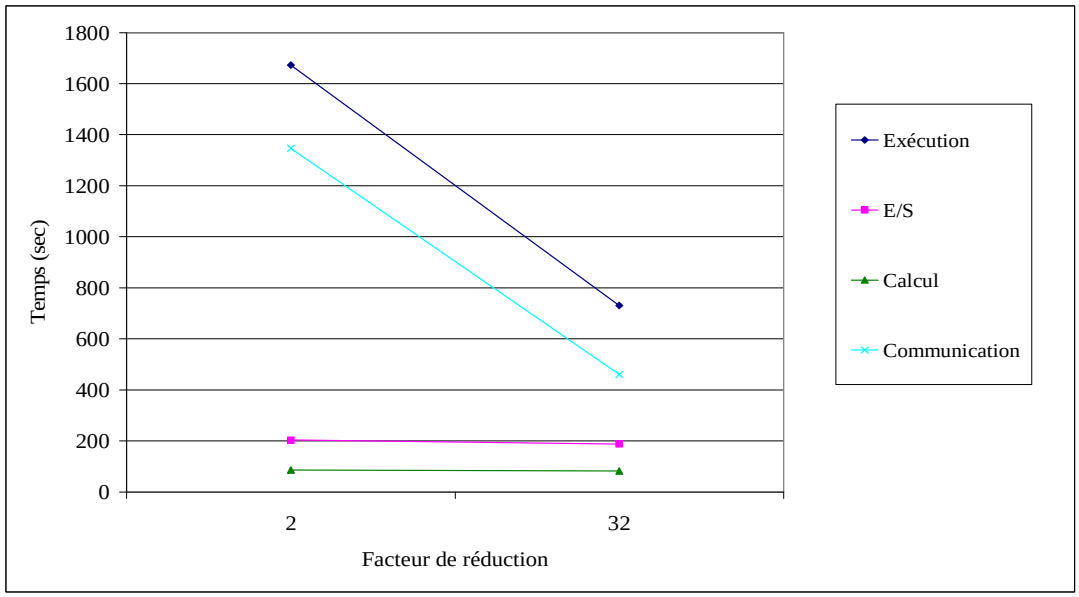


Figure 6.11 Ratios pour la matrice FIDAPM37, *f*=64 et FR variable



Facteur de reduction	**2**	**32**
Temps total d'exécution (s)	1673	731
Temps total d'E/S (s)	203.59	187.87
Ratio d'E/S (%)	12	26
Temps total de calcul (s)	86.51	82.35
Ratio de calcul (%)	5	11
Temps total de communication (s)	1346.9	460.78
Ratio de communication (%)	8	63

Figure 6.12 Performances du PMVC : N=9000, d=10 (matrice aléatoire), *f*=32, RF=2 et 32

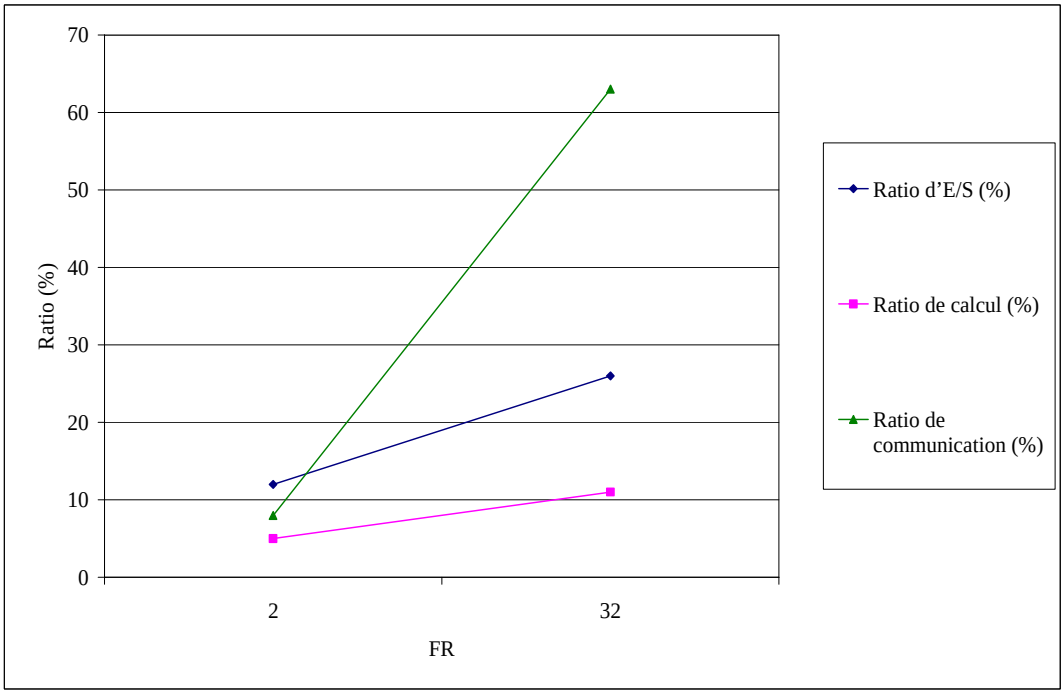


Figure 6.13 Ratios pour N=9000, d=10 (matrice aléatoire), *f*=32, RF=2 et 32

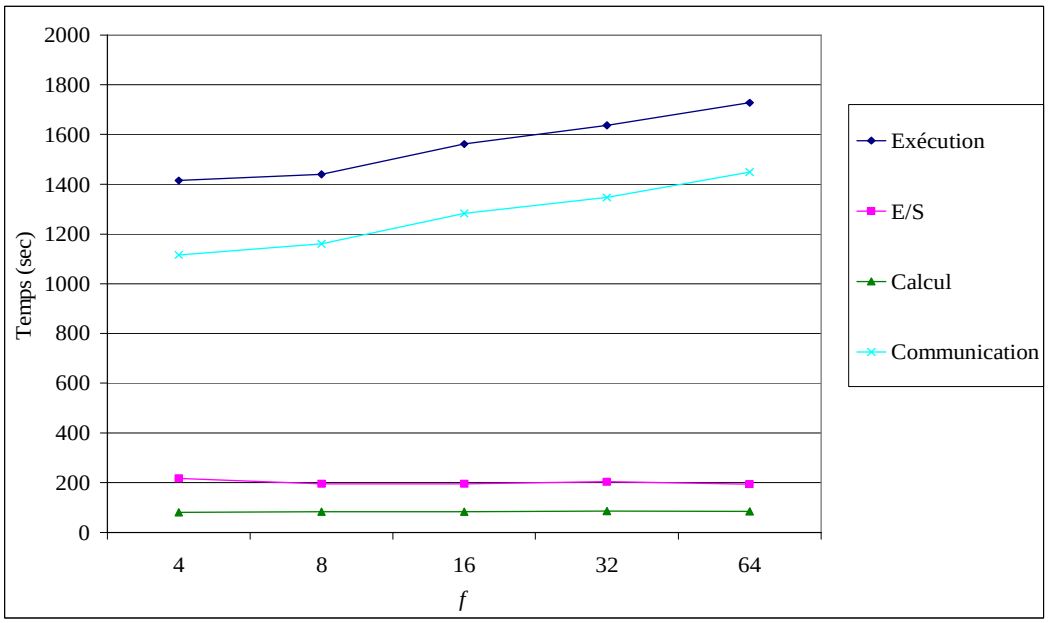
En conclusion, les expérimentations confirment les résultats théoriques (section 4.2) qui montrent qu'il est plus intéressant d'utiliser un facteur de réduction égal au nombre de fragments (une communication de type all-to-one). Les Figures 6.12 et 6.13 confirment cette conclusion.

5.4.4 Evaluation des performances en fonction du nombre de fragments

Pour une matrice de densité donnée, si nous fixons le facteur de réduction (2 dans la Figure 6.14) à une valeur inférieure au nombre de fragments et faisons varier le nombre de fragments, nous remarquons que la durée totale de communication, donc la durée totale d'exécution augmente. Ceci est dû aux communications générées par l'utilisation d'un graphe de tâches ayant une structure arbre pour la réduction du vecteur résultat final. En effet, en augmentant le nombre de fragments, le nombre de communications augmente aussi. Par conséquent, la durée totale de communication augmente.

Notons que le ratio de communication (resp. d'E/S) croît (resp. décroît) alors que celui de calcul est (presque) stable.

Ajoutons que le ratio de communication croît lorsque *f* augmente alors que les ratios de calcul ainsi que d'E/S décroissent. Ainsi, une grande valeur de *f* génère d'importantes opérations de communication.



f	**4**	**8**	**16**	**32**	**64**
Temps total d'exécution (s)	1415	1440	1562	1637	1728
Temps total d'E/S (s)	217.59	195.95	195.58	203.59	194.87
Ratio d'E/S (%)	15	13	12	12	11
Temps total de calcul (s)	81.2	83.03	83.29	86.51	84.21
Ratio de calcul (%)	6	6	5	5	5
Temps total de communication (s)	1116.21	1161.02	1283.13	1346.9	1448.92
Ratio de communication (%)	79	80	82	82	84

Figure 6.14 Performances du PMVC pour N=9000 en fonction du nombre de fragments, FR=2

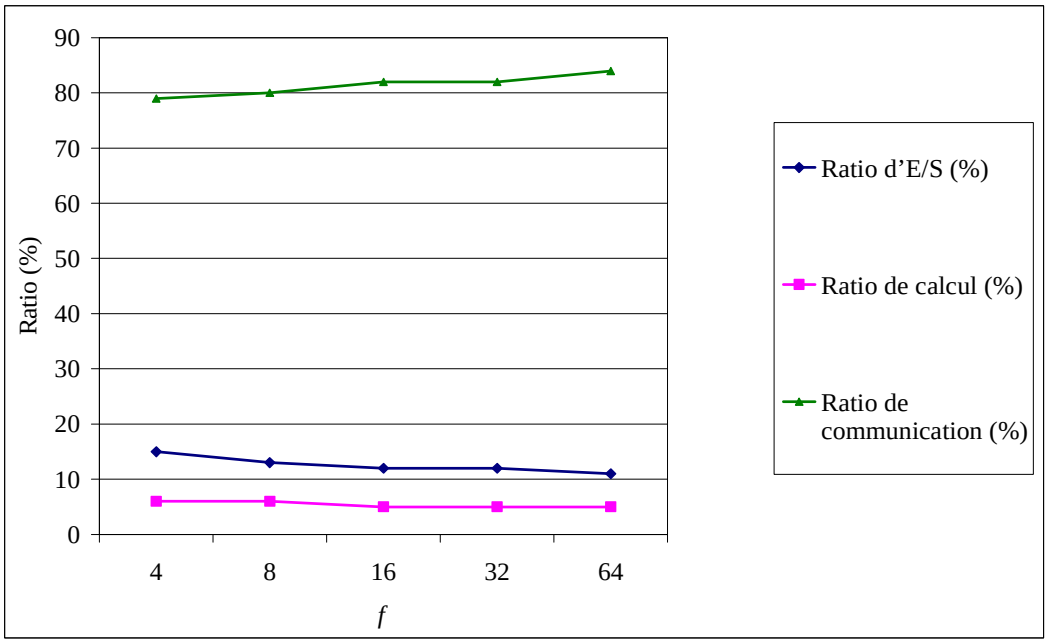
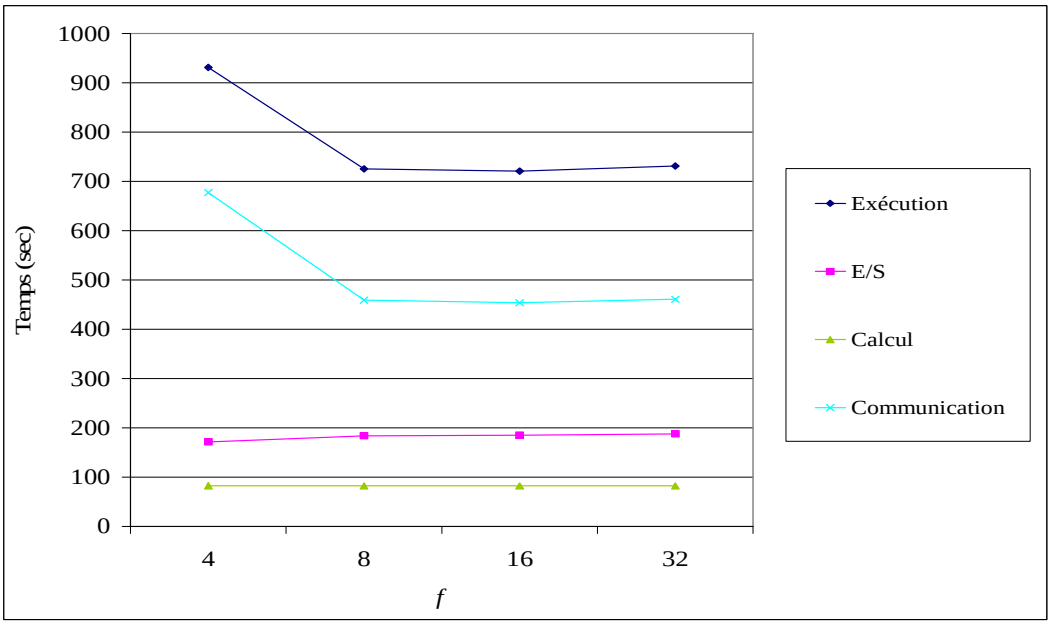


Figure 6.15 Ratios pour N=9000 en fonction du nombre de fragments, FR=2



f	**4**	**8**	**16**	**32**
Temps total d'exécution (s)	931	725	721	731
Temps total d'E/S (s)	171.26	183.68	184.83	187.87
Ratio d'E/S (%)	18	25	26	26
Temps total de calcul (s)	82.73	82.35	82.41	82.35
Ratio de calcul (%)	9	11	11	11
Temps total de communication (s)	677.01	458.97	453.76	460.78
Ratio de communication (%)	73	63	63	63

Figure 6.16 Performances du PMVC pour N=9000 en fonction de *f* avec *f*=FR

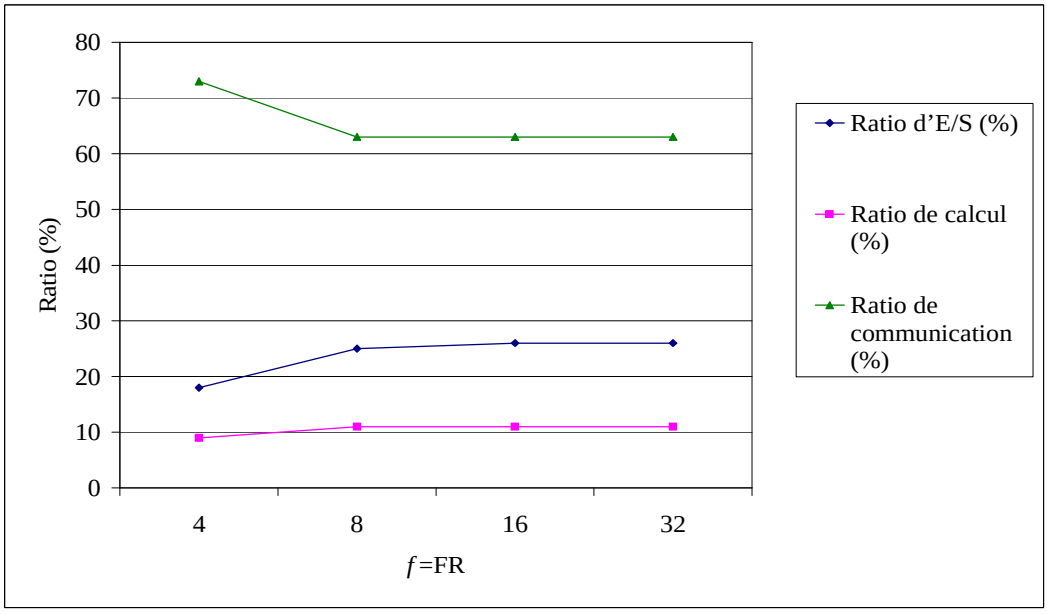


Figure 6.17 Ratios pour N=9000 en fonction de *f* tel que *f*=FR

Enfin, comme le montre la Figure 6.16, quand f dépasse le nombre de nœuds de la plate-forme, les performances se stabilisent.

6. Conclusion

Dans ce chapitre, nous nous sommes intéressés à la distribution et à l'évaluation de performances du produit matrice-vecteur creux y=Ax sur un système à grande échelle. Considérant une géométrie virtuelle à une dimension pour les nœuds, nous avons décomposé la matrice A en blocs contenant le même nombre de lignes. Pour nos expérimentations, nous avons installé une plate-forme de calcul volontaire en utilisant XtremWeb-CH comme middleware.

Nous avons remarqué qu'en augmentant le nombre d'éléments non nuls dans la matrice (soit en augmentant sa taille tout en fixant sa densité, soit en fixant sa taille et en augmentant sa densité), la durée totale d'exécution, la durée totale de communication, ainsi que la durée totale d'E/S augmentent.

Nous avons également noté que la structure arbre du graphe des tâches n'est pas adéquate dans le cas de la distribution du PMVC sur un système distribué à grande échelle. En effet, les expérimentations confirment les résultats théoriques qui montrent qu'il est plus efficace qu'une seule tâche effectue la réduction du vecteur résultat. Il serait intéressant de voir comment s'extrapolent ces résultats pour un système réel et en adoptant d'autres algorithmes de fragmentation.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

L'étude bibliographique par laquelle nous avons entamé notre travail a permis de constater que plusieurs applications scientifiques effectuent des calculs sur des matrices creuses de grandes tailles. En examinant deux applications spécifiques types, la première se situant dans le domaine de la recherche d'information et la seconde dans celui de la simulation 3D, nous avons pu relever que les calculs scientifiques creux se ramènent aux deux problèmes fondamentaux d'algèbre linéaire i.e. la résolution de systèmes linéaires et le calcul d'éléments (valeurs/vecteurs) propres. Nous avons vu également que ces problèmes de grande taille nécessitent un espace de stockage des données ainsi qu'un volume de calcul importants. Ainsi, l'utilisation d'environnements cibles parallèles et distribués offrant des performances élevées s'avère incontournable.

En conséquence, nous nous sommes intéressés dans ce mémoire à l'étude de la distribution, au sein d'un système distribué à grande échelle (SDGE), des calculs dans des méthodes itératives pour résolution de systèmes linéaires et calcul d'éléments propres et ce, dans le cas creux. Il s'avère que ces calculs reposent essentiellement sur le noyau produit matrice-vecteur (PMV).

Afin d'avoir une idée précise sur les matrices creuses et leurs différents modes de stockage, nous avons effectué un passage en revue que nous avons voulu assez exhaustif sur les différentes structures régulières et irrégulières de matrices creuses. Des statistiques basées sur des informations au sein du site Matrix Market [Mat07] ont été effectuées afin de déterminer les structures creuses les plus fréquemment rencontrées dans les applications scientifiques. Cet ensemble comporte les matrices les plus référencées dans la littérature et constitue un riche banc d'essai.

Nous avons ainsi constaté que la structure la plus fréquente est la structure (irrégulière) quelconque, suivie par deux structures régulières i.e. la structure diagonale et la structure bande. Pour les deux classes de structures i.e. régulières et irrégulières, nous avons décrit une série de modes de stockage compressé possibles. Cela a conduit à classer les formats de compression en deux catégories : formats généralistes et formats spécifiques.

Les formats généralistes tels que le CSR, le CSC, le COO, le LNK, … peuvent être utilisés pour n'importe quelle structure de matrice. Les formats spécifiques tels que le BND, le MSR, le DIA, le SSK, … ne peuvent l'être que pour des matrices de structure particulière.

Notre problématique étant l'étude de la distribution du PMVC sur un SDGE, suite à l'étude bibliographique approfondie que nous avons effectuée sur les concepts de base la constituant, notre contribution s'est basée sur l'adoption d'une démarche à trois étapes.

La première étape est l'étape de *prétraitement* qui consiste en la préparation des données, l'optimisation d'algorithme, la prédiction des performances, etc. La seconde est l'étape de *traitement* et correspond à la mise en œuvre des algorithmes étudiés sur les architectures ciblées. Concernant la troisième étape, dite de *post-traitement*, elle consiste à analyser les résultats de l'application.

A l'étape de prétraitement, nous avons étudié les performances de l'algorithme du produit matrice-vecteur creux (PMVC) sur des plates-formes séquentielles. Dans cette étude nous nous sommes intéressés à plusieurs versions de l'algorithme relatives à différents formats de compression pour la matrice creuse.

Nous avons adopté une approche générique consistant à appliquer au PMVC des techniques d'optimisation de haut niveau. Nous avons classé ces techniques en deux catégories : (i) celles pouvant être appliquées manuellement sur le code (telles que le remplacement scalaire, la technique d'unrolling, etc.), et (ii) celles pouvant être sélectionnées comme options lors de la compilation (telles que les options –On, funroll-loop, funroll-all-loops, etc.).

Les résultats obtenus ont montré que les critères d'optimisation des performances du PMVC varient d'une machine à l'autre. Ils doivent donc être bien ajustés aux caractéristiques matérielles et logicielles des nœuds du SDGE cible et ce, afin d'atteindre des performances globales intéressantes.

Après avoir étudié le code du PMVC, nous sommes passés à l'analyse de la distribution d'une matrice creuse utilisée comme entrée par le PMVC sur un SDGE.

Nous avons proposé six heuristiques de fragmentation par lignes pour matrice creuse, dénommées NLE, NLEG, NEZ, NEZG, NEZGT et NEZGTC. Elles consistent toutes à décomposer la matrice, de taille N, en *f* blocs de lignes. Nous avons classé ces six algorithmes en deux groupes G1 et G2. G1 (resp. G2) est constitué par les trois premiers (resp. les deux derniers) algorithmes. Les algorithmes de G1 (resp. G2) construisent des blocs contenant des lignes contiguës (resp. non nécessairement contiguës). Notre objectif était d'obtenir des fragments équilibrés au niveau du nombre d'éléments non nuls. Les algorithmes conçus, de complexités différentes, ont induit des solutions également différentes.

Cela nous a permis de constater que le traitement de matrices de grande taille sur un SDGE n'est intéressant que pour des tailles (N) nettement supérieures au nombre de nœuds du système (*f*) i.e. N >> *f*. En d'autres termes, il convient d'ajuster taille et nombre de nœuds à cibler pour la distribution du PMVC. Ainsi, l'utilisation de tous les noeuds disponibles d'un SDGE, donc de sa puissance totale ne permet pas forcément d'obtenir les meilleures

performances globales de l'application. Utiliser un peu moins de ressources et mieux répartir les charges est plus judicieux.
Nous avons tenu compte des résultats obtenus lors de l'optimisation du PMVC, puis de la distribution/fragmentation de matrice creuse pour étudier les phases de traitement ainsi que de post-traitement du PMVC sur un système distribué à grande échelle.
Ainsi, considérant une géométrie virtuelle à une dimension pour les nœuds, nous avons décomposé la matrice en blocs en se basant sur l'approche NLE vue précédemment. Pour nos expérimentations, nous avons installé une petite plate-forme de calcul volontaire en utilisant XtremWeb-CH comme middleware.
Nous avons ainsi remarqué qu'en augmentant le nombre d'éléments non nuls dans la matrice (soit en augmentant sa taille tout en fixant sa densité, soit en fixant sa taille et en augmentant sa densité), la durée totale d'exécution, la durée totale de communication, ainsi que la durée totale d'E/S du PMVC distribué sur XtremWeb-CH augmentent.
Nous avons également noté que la structure d'arbre que nous avons adoptée pour le graphe des tâches n'est pas adéquate dans le cas de la distribution du PMVC sur un système distribué à grande échelle. En effet, les expérimentations ont confirmé les résultats de notre étude théorique qui ont montré qu'il est plus efficace qu'une seule tâche effectue la réduction du vecteur résultat.

L'étude de la distribution du PMVC sur un SDGE pourrait encore être approfondie et avoir diverses suites dans le cadre d'une architecture séquentielle, parallèle ou distribuée.

En effet, dans le cas séquentiel, une suite peut se situer dans le cadre de la *programmation générative*, approche qui consiste à générer des composants logiciels ou des systèmes paramétrés à la demande et ce, afin d'améliorer leur réutilisabilité.
En d'autres termes, on pourrait concevoir un système d'aide à la décision qui, étant données une matrice creuse et une machine cible séquentielle, génère le meilleur algorithme du PMVC i.e. conduisant aux meilleures performances possibles.
La génération du meilleur algorithme du PMVC sous entend le choix du meilleur format de compression pour la matrice creuse donnée ainsi que les meilleures optimisations à appliquer pour l'algorithme du PMVC. D'après notre étude, les choix dépendront certainement des caractéristiques de la matrice creuse ainsi que de la machine cible.
Les résultats auxquels nous avons abouti peuvent ainsi conduire à déduire des critères spécifiques de choix.

D’un autre côté, il est possible d’enrichir l’étude de l’optimisation du PMVC en considérant et en intégrant d’autres techniques d’optimisation parmi celles connues dans la littérature.
Les structures de matrices creuses qui ont concerné notre étude sont la structure quelconque et la structure bande, vu qu’elles constituent les structures les plus répandues dans les applications réelles. Pour ces deux structures, notre échantillon de formats de compression a été limitée essentiellement à quatre, à savoir, le DNS, le CSR, le COO et le BND. Ainsi, une extension pourrait être facilement effectuée à d’autres structures ainsi qu’à d’autres formats de compression.
Concernant l’aspect parallèle, l’étude effectuée dans le cas séquentiel pourrait servir de base, en particulier, en ce qui concerne l’optimisation de l’algorithme du PMVC.
La distribution de la matrice creuse, stockée dans un format compressé, sur des processeurs aussi bien homogènes qu’hétérogènes [BiM05][Chr98] a été étudiée essentiellement dans le cas du format CSR. L’extension à d’autres formats de compression serait intéressante. En conséquence, il s’agira d’étudier des approches différentes de distribution pour la matrice compressée et qui dépendront nécessairement du format de compression adopté.

Dans le cas d’un SDGE, il est possible de tirer profit de l’étude de l’optimisation du PMVC séquentiel sur des architectures séquentielles différentes afin de générer automatiquement pour chaque nœud du système le meilleur algorithme du PMVC.
Au niveau des données, on pourrait étendre notre étude effectuée sur la distribution/ fragmentation d’une matrice creuse stockée dans le format CSR sur des ressources homogènes, pour traiter le cas de ressources hétérogènes, tout en considérant d’autres structures de matrices creuses ainsi que d’autres formats de compression.
Se situer au sein d’un SDGE, permettrait aussi d’envisager la prise en compte de la volatilité des nœuds et l’adoption d’approches de tolérance aux fautes pour la distribution du PMVC.

BIBLIOGRAPHIE

[AbB07] N. Abdennadher & R. Boesh, Towards a peer-to-peer platform for high performance computing, GPC, Paris, France, 2007.

[Abd08] N. Abdennadher & al., Initializing a national grid infrastructure - Lessons learned from the Swiss National Grid Association Seed Project, CCGRID, Lyon, France, 2008.

[ACK02] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky & D. Werthimer, SETI@home: an experiment in public-resource computing, *Communications of the ACM*, 45(11), pp. 56-61, 2002.

[Alo05] K. Aloui, Optimisation du produit matrice vecteur creux séquentiel, Mémoire de Maîtrise en Informatique, ESSTT, Tunisie, 2005.

[Amo07] O. Amoignon, Moving mesh adaptation scheme for aerodynamic shape optimization, *Thèse de Doctorat*, Uppsala University, Sweden, 2007

[AoP04] L. Aouad & S. Petiton, Experimentations and programming paradigms for matrix computing on peer to peer grid, Com. GRID2004, Pittsburgh, Pennsylvania, USA, 2004.

[Ase96] R. Asenjo & al., On the automatic parallelization of sparse and irregular Fortran codes, Technical Report UMA-DAC-96/34, University of Illinois, USA, 1996.

[Ban93] U. Banerjee, *Loop transformations for restructuring compilers: the foundations*, Kluwer, 1993.

[Bau02] F. Baude, *Calcul réparti à grande échelle*, Hermès Science Publications, 2002.

[BBL02] M. Baker, R. Buyya & D. Laforenza, Grids and Grid technologies for wide area distributed computing, *Software: Practice and Experience*, 32(15), pp.1437-1466, 2002.

[BeA04] B. Belhadj Mohamed & M. Akil, Optimisation du produit matrice vecteur creux, Mémoire de Maîtrise en Informatique, FST-DSI, Tunisie, 2004.

[BEP07] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt & J. Weglarz, *Handbook on scheduling : from theory to applications*, Springer-Verlag, 2007.

[Bik96] A.J.C. Bik, Compiler support for sparse matrix computations, *Ph-D Thesis*, University of Leiden, The Netherlands, 1996.

[BiM05] R. H. Bisseling & W. Meesen, Communication balancing in parallel sparse matrix-vector multiplication, *Electronic Transactions on Numerical Analysis, Special Issue on Combinatorial Scientific Computing*, 21, pp. 47-65, 2005.

[Bou02] P. Boulet, Contributions aux environnements de programmation pour le calcul intensif, Thèse *HDR*, Université des Sciences et Technologies de Lille, France, 2002.

[BrL06] K. Bryan & T. Leise, The $25,000,000,000 eigenvector: the linear algebra behind Google, *SIAM Review*, 48(3), pp. 569–581, 2006.

[Bur98] K. Burrage & al., A deflation technique for linear systems of equations, *SIAM Journal on Scientific Computing*, 19(4), pp. 1245-1260, 1998.

[BUY99] R. Buyya, *Cluster computing*, Prentice-Hall Inc, 1999.

[Cag00] E. Cagniot, Algorithmes data-parallèles irréguliers appliqués à la simulation électromagnétique tridimensionnelle, *Thèse de Doctorat*, Université des Sciences et Technologies de Lille, France, 2000.

[Che04] B. Chen, *Parallel scheduling for early completion, Handbook on scheduling: algorithms, models and performance analysis*, (ed J.Leung), CRC Press, 2004.

[Chr98] P. Christen, A parallel iterative linear system solver with dynamic load balancing, The 12th International conference on Supercomputing, Melbourne, Australia, 1998.

[Cia82] P. G. Ciarlet, *Introduction à l'analyse numérique matricielle et à l'optimisation*, Masson, 1982.

[CuS99] D. E. Culler & J. P. Singh, *Parallel computer architecture, a hardware / software approach*, Morgan Kaufmann Publishers, 1999.

[DER92] I. S. Duff, A. M. Erisman & J. K. Reid, *Direct methods for sparse matrices*, Oxford Science Publications, 1992.

[DiE06] A. Dijoux & S. Emma, Peer-to-Peer, Mémoire de Master Professionnel Système informatique et réseaux, Université Claude Bernard Lyon 1, France, 2006.

[DoH79] J. Dongarra and A. R. Hinds, Unrolling Loops in Fortran, *Software–Practice and Experience*, 9(3), pp. 219-226, 1979.

[EES83] S. C. Eisenstat, H. C. Elman, & M. H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM Journal of Numerical Analysis*, 20, pp. 345-357, 1983.

[EHM05] N. Emad, O. Hamdi & Z. Mahjoub, On sparse matrix-vector product optimization, ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05), Cairo, Egypt, 2005.

[EHM06] N. Emad, O. Hamdi-Larbi & Z. Mahjoub, Autour des calculs creux et des calculs creux parallèles, Rapport Technique, PRiSM-UVSQ France & URAPAD-FST Tunisie, 2006, http://www.prism.uvsq.fr/rapports/2006/document_2006_96.pdf.

[EHM09] N. Emad, O. Hamdi-Larbi & Z. Mahjoub, On sparse matrix-vector product performance evaluation for efficient distribution on large scale systems, The 9th Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA'09), Athens, Greece, 2009.

[Ema01] N. Emad, Méthodes hybrides et métacomputing pour le calcul scientifique intensif, Thèse *HDR*, Université de Versailles Saint-Quentin-en-Yvelines, France, 2001.

[EPE05] N. Emad, S. Petiton, G. Edjlali, Multiple explicitly restarted Arnoldi method for solving large eigenproblems, *SIAM Journal on Scientific Computing*, p. 19, 2005.

[GKA08] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, & N. Koziris, Understanding the performance of sparse matrix-vector multiplication, 16th Euromicro International Conference on Parallel, Distributed and network-based Processing, Toulouse, France, 2008.

[GoV83] G. H. Golub & C. F. Van Loan, *Matrix computations*, The Johns Hopkins University Press, 1983.

[Gue06] H. Guesmi, Parallélisation d'algorithmes de résolution de l'EDP de Poisson 2D modélisant une décharge électrique dans un gaz, Mémoire de Mastère, FST-DSI, Tunisie, Novembre 2007.

[Haz02] M. Hazewinkel, *Encyclopaedia of mathematics*, Springer-Verlag, 2002. http://eom.springer.de/L/l059050.htm.

[HME07] O. Hamdi-Larbi, Z. Mahjoub & N. Emad, Load balancing in pre-processing of large-scale distributed sparse computation, 8th LCI Conference on High Performance Clustered Computing, South Lake Tahoe, California, USA, 2007.

[Hoc97] D.S. Hochbaum, A*pproximation algorithms for NP-Hard problems*, PWS Publishing Co., 1997.

[HOD98] A.W. Halderen, B.J. Overeinder, P.M.A. Sloot, R. Dantzig, D.H.J. Epema & M. Liyny, Hierarchical resource management in the Polder metacomputing Initiative, *Parallel Computing*, 24, pp. 1807-1825, 1998.

[HSS03] K. Hormann, S. Spinello, and P. Schröder, *C*1-Continuous Terrain Reconstruction from Sparse Contours, In Proceedings of Vision, Modeling, and Visualization, 2003.

[ImY01] E. Im & K. A. Yelick, Optimizing sparse matrix-vector multiplication for register reuse, International Conference on Computational Science, California, USA, 2001.

[Jak99] K. Jamsa & L. Klander, *C/C++, la bible du programmeur*, Editions Reynald Goulet inc, 1999.

[Jen80] A. Jennings, *Matrix computation for engineers and scientists*, John Wiley & Sons, 1980.

[JeS06] P. Jetzer & N. Straumann, Josephson junctions and dark energy, *Physics letters. Section B*, 639(2), pp. 57-58, 2006. http://cat.inist.fr/?aModele=afficheN&cpsidt=17962099.

[KGK08] K. Kourtis, G. Goumas & N. Koziris, Optimizing sparse matrix-vector multiplication using index and value compression, Proceedings of the 2008 Conference on computing frontiers, Ischia, Italy, 2008.

[LaM06] A. Langville & C. Meyer, Updating Markov chains with an eye on Google's PageRank, *SIAM J. Matrix Analysis*, 27(4), 968-987, 2006.

[LaT00] P. Lascaux & R. Théodor, *Analyse numérique matricielle appliquée à l'art de l'ingénieur, Méthodes itératives*, Dunod, 2000.

[MeG03] J. Mellor-Crummey & J. Garvin, Optimizing sparse matrix vector multiply using unroll-and-jam, *International Journal of High Performance Computing Applications*, 18(2), pp. 225-236, 2004.

[MKJ03] B. Mandhani, K. Kummamuru & S. Joshi, A matrix density based algorithm to hierarchically co-cluster documents and word**s,** The 12th International World Wide Web Conference, Budapest, Hungary, 2003.

[Mor94] H. S. Morse, *Practical parallel computing*, Academic Press, 1994.

[OsZ83] O. Osterby & Z. Zlatev, *Direct methods for sparse matrices*, Springer-Verlag, 1983.

[PeE96] S.G. Petiton & N. Emad, *A data parallel scientific computing introduction in data parallelism*, Springer Verlag, 1996.

[PiC05] J. C. Pichel, D. B. Heras, J. C. Cabaleiro & F. F. Rivera, Performance optimization of irregular codes based on the combination of reordering and blocking techniques, *Parallel Computing*, 31, pp. 858 – 876, 2005.

[PiH99] A. Pinar & M. T. Heath, Improving performance of sparse matrix-vector multiplication, ACM/IEEE SC 1999 Conference (SC '99), Portland, USA, 1999.

[Pot97] W. M. Pottenger, Theory, techniques, and experiments in solving recurrences in computer programs, *Ph-D Thesis*, University of Illinois, USA, 1997.

[Reg59] V. Regemorter, Recherches sur les problèmes théoriques de classification: I. Les courbes de croissance dans la théorie du transfert radiatif, *Annales d'Astrophysique*, 22, p. 249, 1959. http://adsabs.harvard.edu/full/1959AnAp...22..249V.

[SaS86] Y. Saad & M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Computing*, 7, pp. 856-869, 1986.

[SaY99] S.M. Sait & H. Youssef, *Iterative computer algorithms with applications in engineering*, IEEE C.S ed., 1999.

[Sha05] S.A. Shahzadeh, Stratégies de redémarrage des méthodes itératives d'algèbre linéaire pour le calcul global, *Thèse de Doctorat*, Université de Versailles Saint Quentin-en-Yvelines, France, 2005.

[SHK95] B.A. Shirazi, A.R. Hurson & K.M. Kavi, *Scheduling and load balancing in parallel and distributed systems*, IEEE Computer Society Press & The Institute of Electrical and Electronics Engineers, INC, 1995.

[Sla06] Y. Slama, Génération automatique de code à partir de placement pour machines parallèles à mémoire partagée, *Thèse de Doctorat*, Faculté des Sciences de Tunis, Tunisie & Université de Versailles Saint-Quentin-en-Yvelines, France, 2006.

[Son89] P. Sonneveld, CGS: a fast Lanczos-type solver for nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput*., 10, pp. 36-52, 1989.

[Sor92] D. C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, *SIAM J. Matrix Anal. Appl.*, 13 (1), pp. 357-385, 1992.

[Tol97] S. Toledo, Improving the memory-system performance of sparse-matrix vector multiplication, *IBM Journal of Re-search and Development*, 41(6), 711–725, 1997.

[Tom03] J. A. Tomlin, A new paradigm for ranking pages on the World Wide Web, The Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, 2003.
http://www2003.org/cdrom/papers/refereed/p042/paper42_html/p42-tomlin.htm#eqn-Cons3.

[VaB05] B. Vastenhouw & R.H. Bisseling, A two dimensional data distribution method for parallel sparse matrix-vector multiplication, *SIAM Review*, 47(1), pp. 67-95, 2005.

[Van92] H. A. Van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems, *SIAM J. Sci. Statist. Comput*., 13, pp. 631-644, 1992.

[Vud02] R. Vuduc, J. Demmel, K. A. Yelick, S. Kamil, R. Nishtala & B. C. Lee, Performance optimizations and bounds for sparse matrix-vector multiply, SC International Conference for High Performance Computing, Storage and Analysis, Los Alamitos, California, 2002.

[VuM05] R. W. Vuduc & H. J. Moon, Fast sparse matrix-vector multiplication by exploiting variable block structure, HPCC'2005, Sorrento, Italy, 2005.

[WhB97] J. B. White & P. Sadayappan, On improving the performance of sparse matrix vector multiplication, The Fourth International Conference on High-Performance Computing, Bangalore, India, 1997.

[WiL06] J. Willcock & A. Lumsdaine, Accelerating sparse matrix computations via data compression, ICS'06 : Proceedings of the 20th Annual International Conference on Supercomputing, ACM Press, New York, USA, 2006.

[Wil07] S. Williams & al., Optimization of sparse matrix-vector multiplication on emerging multicore platforms, SC07 International Conference for High Performance Computing, Storage and Analysis, Reno, Nevada, USA , 2007.

[YIE09] N. Yigitbasi, A. Iosup, D. Epema & S. Ostermann, C-Meter: A framework for performance analysis of computing clouds, International Workshop on Cloud Computing (Cloud 2009*)*, Shanghai, Chine, May 2009.

[You71] D.M. Young, *Iterative solution of large linear systems*, Academic Press, 1971.

NETOGRAPHIE

[AKK03] M. Arenas, I. Kiringa & T. Kementsietsidis
http://www.cs.toronto.edu/db/hyperion/bibliography.html, 2003.

[AlS98] G. W. Althaus & E. Speedicato, *Algorithms for large scale linear algebraic systems: Applications in science and engineering*, Kluwer Academic Publisher, 1998,
http://books.google.fr/books?id=zQCpgSvjWwcC&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[Bar94] R. Barrett & al., *Templates for the solution of linear systems: Building blocks for iterative methods*, SIAM, 1994,
http://books.google.fr/books?id=zMv8_9W0a60C&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[BBG01] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, http://www.mcs.anl.gov/petsc, 2001.

[Gre97] A. Greenbaum, *Iterative methods for solving linear systems*, SIAM, 1997,
http://books.google.fr/books?id=WwMDNLxrwocC&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[Gri04] http://www.grid.org/, 2004.

[HaB98] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing, 1998.

[Hoj92] H. Adeli, *Supercomputing in engineering analysis*, Marcell Dekker Inc., 1992,
http://books.google.fr/books?id=3GqB6DknJl0C&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[INR02] Inria, Aladin : algorithmes adaptés au calcul numérique intensif (projet)
http://www.inria.fr/recherche/equipes/aladin.fr.html, 2002.

[ISE07] Ingenierie des structures et des énergies
http://www.inse.fr/presentation.php, 2007.

[ISG00] http://math.nist.gov/MatrixMarket/data/misc, 2000.

[JeC08] P. Jeannin & J. Carpentier, Réseaux de puissance - Méthodes de résolution des équations,
http://209.85.135.104/search?q=cache:xjgEIaXN0DoJ:www.techniques-ingenieur.fr/affichage/noeud.asp%3FnID%3D429305+mod%C3%A8le+r%C3%A9seau+de+puissance&hl=fr&ct=clnk&cd=3&gl=fr, 2008.

[LEV08] T. Levy-Abegnoli, Cloud computing : vers la dématérialisation des salles informatiques, ZDNet France, Juin 2008.
http://www.zdnet.fr/actualites/it-management/0,3800005311,39381489,00.htm.

[Lod04] O. Lodygensky, Xtremweb : Une plate-forme de calcul global et pair à pair, http://www.lal.in2p3.fr/~lodygens/ - TALKS, 2004.

[Mat07] http://math.nist.gov/MatrixMarket/, 2007.

[Mes06] Définition du PageRank, http://www.mes-partenaires.com/article/11-definition-pagerank.html, 2006.

[NEP98] http://math.nist.gov/MatrixMarket/data/NEP, 1998.

[Saa94] Y. Saad, SPARSKIT: a Basic tool kit for sparse matrix computation, http://www.cs.umn.edu/Research/arpa/SPARSKIT/paper.ps, 1994.

[Saa03] Y. Saad, *Iterative methods for sparse linear systems*, Society of Industrial and Applied Mathematics, 2003, http://books.google.fr/books?id=ZdLeBlqYeF8C&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[Ski97] S. Skiena, Lecture 11: Introduction to dynamic programming, http://www.cs.sunysb.edu/~skiena, 1997.

[SpK00] http://math.nist.gov/MatrixMarket/data/SPARSKIT, 2000.

[Van94] H. A. Van der Vorst, Recent developments in CG methods, Proceedings of the International Conference and Exhibition on HPCN, Munich, Germany, 1994, http://books.google.com/books?id=j0iNpFChAaAC&printsec=frontcover&hl=fr&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[Van03] H. A. Van der Vorst, *Iterative Krylov methods for large linear systems*, Cambridge University Press, 2003, http://books.google.fr/books?id=_WJoLfl19e8C&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false.

[VFS93] H. A. Van der Vorst, D. R. Fokkema & G. L. Sleijper, Further improvements in nonsymmetric hybrid iterative methods, *Simulation of Semiconductor Devices And Processes*, 5, 1993, http://in4.iue.tuwien.ac.at/pdfs/sisdep1993/pdfs/VanderVorstHA_21.pdf.

[Wyr01] R. Wyrzykowsky, Parallelizing sparse matrix computations; http://k2.pcz.pl/~roman/badania/prz_rown/parallelizing.html, 2001.

ANNEXES

(au chapitre 4)

Annexe A : Résultats pour le PMVC-DNS

1. Sur la machine 1

1.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
so	1039498.81	526227.38	243975.8	217410	1039554.19	1039032.62
su_rs	755203	203842.59	204595.2	205577	755994	755260
u2	617857	204216.6	204082.8	204530.2	616881	617282.81
u4	511312.81	203170.8	202034.8	201755.8	511584.19	511499.41
u8	445305	205196.4	204208.2	204216.4	445372.41	445253.19
u16	415680.59	210184.8	208452	207929.2	415594	415898.41
u32	404277.59	223087.6	222211.6	221444.4	403987.81	404059.41
u64	427411.81	248957.4	247318.2	246204.6	423288.19	423813.59
u128	432627.41	238551.2	322537.6	322408.8	431288.59	431520.19
u256	412522.59	213333.4	303920.2	303350.4	410710.59	411126

Figure 1. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.98	4.26	4.78
su_rs	3.70	3.69	3.67
u2	3.03	3.03	3.02
u4	2.52	2.53	2.53
u8	2.17	2.18	2.18
u16	1.98	1.99	2.00
u32	1.81	1.82	1.83
u64	1.72	1.73	1.74
u128	1.81	1.34	1.34
u256	1.93	1.36	1.36

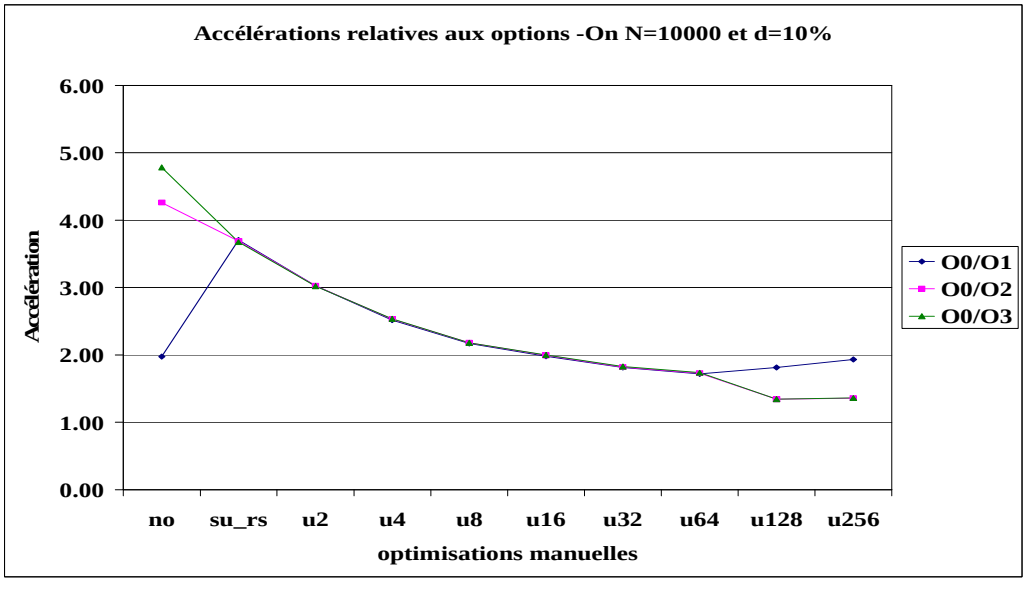


Figure 1. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.22	1.00	1.00	1.01	1.23	1.22
u4	1.48	1.00	1.01	1.02	1.48	1.48
u8	1.70	0.99	1.00	1.01	1.70	1.70
u16	1.82	0.97	0.98	0.99	1.82	1.82
u32	1.87	0.91	0.92	0.93	1.87	1.87
u64	1.77	0.82	0.83	0.83	1.79	1.78
u128	1.75	0.85	0.63	0.64	1.75	1.75
u256	1.83	0.96	0.67	0.68	1.84	1.84

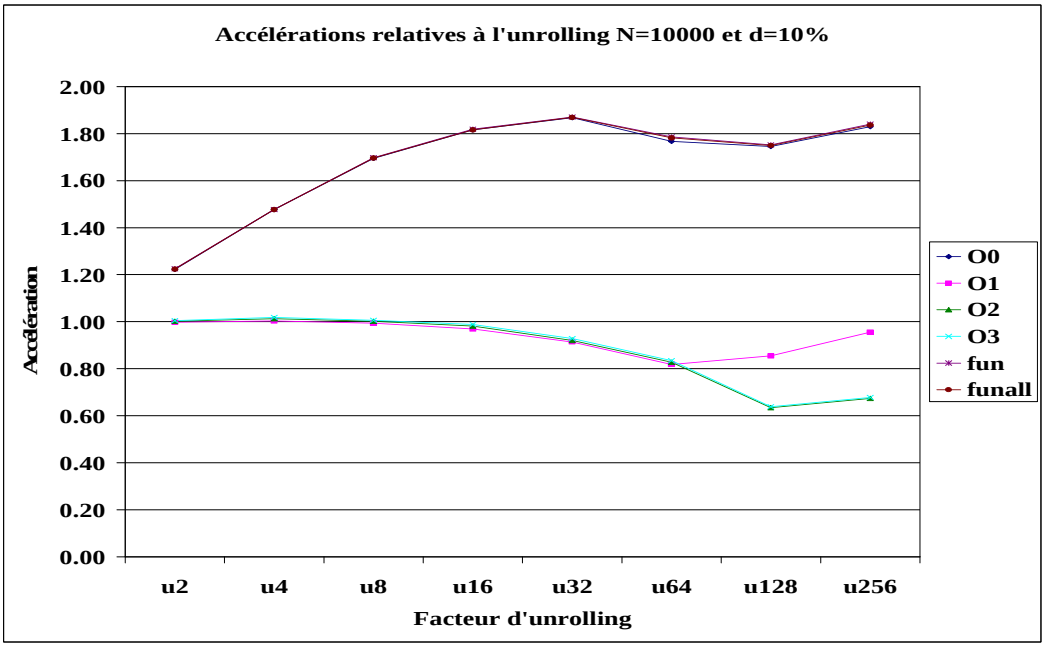


Figure 1. (c) Accélération pour l'unrolling(su_rs/u_n)

N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
so	1037119.81	524003.19	244502.8	217717.41	1037998.19	1038203
su_rs	754294.62	204274.8	204715.41	204511.59	755625.19	756312.81
u2	616365.81	203693.6	202991.6	203843.2	617111.81	617542.38
u4	510805.41	203275.8	201841.4	202165.6	511328.81	511599
u8	444579.41	204968.4	203819.2	203678	445556	445549.81
u16	415019.41	209651.6	208239.4	207977.4	416215.41	416065.41
u32	403734.19	222410.2	222343.8	221924.2	404078.59	404166.41
u64	423091.59	247740.8	246357	245677.8	423714.41	423866.81
u128	430586.59	238350.6	322682.8	322354.2	431325	431639.81
u256	410166.19	213017.2	303312.4	303493.2	411080.81	411007.81

Figure 2. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.98	4.24	4.76
su_rs	3.69	3.68	3.69
u2	3.03	3.04	3.02
u4	2.51	2.53	2.53
u8	2.17	2.18	2.18
u16	1.98	1.99	2.00
u32	1.82	1.82	1.82
u64	1.71	1.72	1.72
u128	1.81	1.33	1.34
u256	1.93	1.35	1.35

Figure 2. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.22	1.00	1.01	1.00	1.22	1.22
u4	1.48	1.00	1.01	1.01	1.48	1.48
u8	1.70	1.00	1.00	1.00	1.70	1.70
u16	1.82	0.97	0.98	0.98	1.82	1.82
u32	1.87	0.92	0.92	0.92	1.87	1.87
u64	1.78	0.82	0.83	0.83	1.78	1.78
u128	1.75	0.86	0.63	0.63	1.75	1.75
u256	1.84	0.96	0.67	0.67	1.84	1.84

Figure 2. (c) Accélération pour l'unrolling(su_rs/u_n)

1.3 Sherman3, N=5005 et d=0.08%

	O0	O1	O2	O3	fun	funall
so	262489.41	133790	64148.4	58178	262561.41	260522.2
nu_sr	194179.2	53983	55023.2	55489.2	193883.8	191025.41
u2	156868.2	53754	54989.6	54937	156989.2	154799.2
u4	129446	53801.8	54767.2	54946.4	129550.2	127421.2
u8	114544	53813	54664.6	55028.2	114909.4	113008.4
u16	107940.8	55419.8	56203.2	56469.6	107497	106103.2
u32	104471	58275.2	60080.4	60978.8	104602	102727.6
u64	108013.8	63125.2	63968.4	64356.4	107832.2	106407
u128	110808.2	63884.2	84984.4	84837.2	110590.8	109031.8
u256	108507.6	56114.4	80411.2	80337.2	108722.6	106566.6

Figure 3. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.96	4.09	4.51
su_rs	3.60	3.53	3.50
u2	2.92	2.85	2.86
u4	2.41	2.36	2.36
u8	2.13	2.10	2.08
u16	1.95	1.92	1.91
u32	1.79	1.74	1.71
u64	1.71	1.69	1.68
u128	1.73	1.30	1.31
u256	1.93	1.35	1.35

Figure 3. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.24	1.00	1.00	1.01	1.24	1.23
u4	1.50	1.00	1.00	1.01	1.50	1.50
u8	1.70	1.00	1.01	1.01	1.69	1.69
u16	1.80	0.97	0.98	0.98	1.80	1.80
u32	1.86	0.93	0.92	0.91	1.85	1.86
u64	1.80	0.86	0.86	0.86	1.80	1.80
u128	1.75	0.85	0.65	0.65	1.75	1.75
u256	1.79	0.96	0.68	0.69	1.78	1.79

Figure 3. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.38	2.58	1.19	1.06	1.38	1.38
N=10000 et d=50%	1.37	2.57	1.19	1.06	1.37	1.37
Sherman3	1.35	2.48	1.17	1.05	1.35	1.36

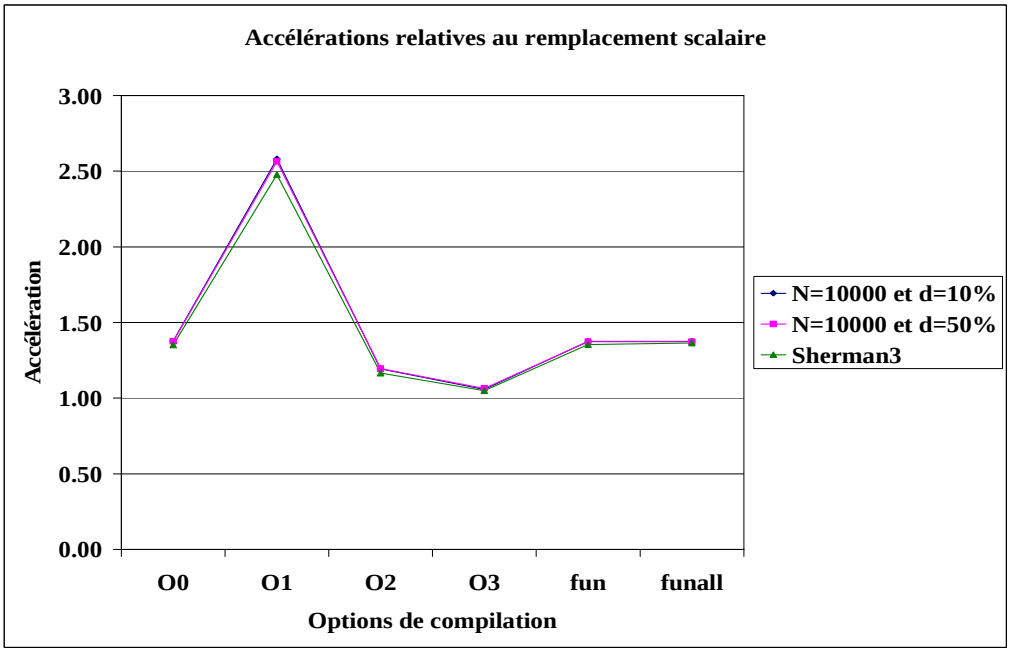


Figure 4. Accélération pour le remplacement scalaire (no/su_rs)

2. Sur la machine 2

2.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	Funall
no	1154011.62	753252.19	434412	435765.19	1101678	1102059.62
su_rs	1276152.62	434074.41	408805.41	410196.81	1276654.75	1275611
u2	921342	393757.81	364855.41	365191.19	943443.38	943148.38
u4	710614.62	367062.81	415833.81	417502.19	692682.81	693971.62
u8	613036.81	355791.19	346245.81	347648.59	586632	585682.81
u16	536910.38	353726	347579.19	348461.19	524403.19	523905.59
u32	512115.41	340503.59	353901.81	354790	501422.19	501427
u64	503856	351215.19	350918.19	352535.81	495075	495650.19
u128	496570	343547.59	342032	343142.41	490598.81	490412.19
u256	494854	343495	360306.81	361938.41	488273.59	488524.41

Figure 5. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
No	1.49	2.57	2.57
su_rs	3.01	3.13	3.14
u2	2.29	2.40	2.40
u4	1.89	1.75	1.76
u8	1.64	1.64	1.64
u16	1.50	1.50	1.51
u32	1.46	1.43	1.43
u64	1.43	1.41	1.42
u128	1.41	1.41	1.41
u256	1.45	1.39	1.40

Figure 5. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.46	1.11	1.12	1.11	1.43	1.43
u4	1.86	1.17	1.04	1.04	1.82	1.82
u8	2.20	1.20	1.15	1.15	2.13	2.13
u16	2.43	1.21	1.16	1.16	2.36	2.36
u32	2.54	1.24	1.16	1.16	2.48	2.48
u64	2.58	1.23	1.17	1.17	2.52	2.52
u128	2.61	1.22	1.17	1.17	2.55	2.55
u256	2.52	1.22	1.12	1.12	2.46	2.47

Figure 5. (c) Accélération pour l'unrolling(su_rs/u_n)

2.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	1150453.62	751369	434693.41	435985.41	1155325.62	1106575.25
su_rs	1276297.75	431773.19	408642.41	410424.19	1277010.38	1296519.25
u2	919633.62	391892.81	364222.41	365590.19	922349	933867
u4	710398.81	365001.81	416077	417729.59	710614.38	712551.38
u8	612040.19	354009.19	346079	347818.19	612848	585709.62
u16	535632	352032.19	347543	348808	537461.81	529935.38
u32	510497.81	338627.41	353442.19	354921.19	512690.59	508820.41
u64	502340.81	349268	350873	352626.41	504231	502012.41
u128	495442	341545.41	342069.41	343384.19	496967.81	496610.41
u256	493353.41	341623.59	360362	362112.59	495152	494043.59

Figure 6. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.53	2.66	2.65
su_rs	2.94	3.12	3.11
u2	2.34	2.53	2.52
u4	1.94	1.71	1.70
u8	1.72	1.77	1.76
u16	1.52	1.54	1.54
u32	1.50	1.45	1.44
u64	1.43	1.44	1.43
u128	1.45	1.45	1.45
u256	1.44	1.37	1.37

Figure 6. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.39	1.10	1.12	1.12	1.35	1.35
u4	1.80	1.18	0.98	0.98	1.84	1.84
u8	2.08	1.22	1.18	1.18	2.18	2.18
u16	2.38	1.23	1.18	1.18	2.43	2.43
u32	2.49	1.27	1.16	1.16	2.55	2.54
u64	2.53	1.24	1.16	1.16	2.58	2.57
u128	2.57	1.26	1.20	1.20	2.60	2.60
u256	2.58	1.26	1.13	1.13	2.61	2.61

Figure 6. (c) Accélération pour l'unrolling(su_rs/u_n)

2.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	fun	funall
no	278040.59	187166.8	108281.8	108010.2	286172.59	285860.81
su_rs	319456.19	106242.4	102066.6	101761.6	317765.59	317423.81
u2	219234.41	95787.8	91478.4	91270.4	222556.8	222541
u4	171993.8	90796	98218.8	97964.4	174888.2	174642.41
u8	145302.2	88648.8	88604.8	88369.2	149068.2	148855.41
u16	131671	87690.2	87613.2	87382	134420	134246.41
u32	125599.2	85960.2	87870.4	87630.4	128186.2	128056.4
u64	123634.4	86477.4	87454.2	87226	125911.2	125741.4
u128	122450.8	86735.2	87054.4	86872.2	124512.2	124339.4
u256	126868.6	87420.8	91052	90720.2	128923.8	128749.6

Figure 7. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.53	2.65	2.64
su_rs	2.96	3.12	3.11
u2	2.35	2.52	2.52
u4	1.95	1.71	1.70
u8	1.73	1.77	1.76
u16	1.52	1.54	1.54
u32	1.51	1.44	1.44
u64	1.44	1.43	1.42
u128	1.45	1.45	1.44
u256	1.44	1.37	1.36

Figure 7. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.39	1.10	1.12	1.12	1.38	1.39
u4	1.80	1.18	0.98	0.98	1.80	1.82
u8	2.09	1.22	1.18	1.18	2.08	2.21
u16	2.38	1.23	1.18	1.18	2.38	2.45
u32	2.50	1.28	1.16	1.16	2.49	2.55
u64	2.54	1.24	1.16	1.16	2.53	2.58
u128	2.58	1.26	1.19	1.20	2.57	2.61
u256	2.59	1.26	1.13	1.13	2.58	2.62

Figure 7. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	0.90	1.74	1.06	1.06	0.86	0.86
N=10000 et d=50%	0.90	1.10	1.12	1.12	1.35	1.35
Sherman3	0.87	1.07	0.88	0.87	1.36	1.36

Figure 8 . Accélération pour le remplacement scalaire (no/su_rs)

3. Sur la machine 3

3.1 N=10000 et d=10%

	O0	O1	O2	O3	funroll	funall
so	4626360	3677060.75	3144334.5	3145120.75	4626261.5	4625145.5
su_rs	1894922.75	3145269.5	1846878.25	1846696.75	1896800.75	1897810.75
u2	1747483.25	2418860	1220588.62	1220397.25	1751854	1748095
u4	1582102	2054952.62	1221064.75	1226019.25	1582818.38	1581457.38
u8	1517006.25	1283584.25	1217880.25	1220594	1516050	1518135.25
u16	1500435	1282290.75	1735417.38	1735297.62	1501672.38	1502165
u32	1480366.75	1254389.38	2449948.25	2451286.5	1479935.75	1485049.75
u64	1471248.25	1236006	2668157.5	2666216	1471695	1472483.25
u128	1469533	1229576.62	2942559.5	2941547.75	1468160.62	1468631.62
u256	1469451	1227454.25	3037191.75	3035614.25	1470147.75	1465837.75

Figure 9. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.26	1.47	1.47
su_rs	0.60	1.03	1.03
u2	0.72	1.43	1.43
u4	0.77	1.30	1.29
u8	1.18	1.25	1.24
u16	1.17	0.86	0.86
u32	1.18	0.60	0.60
u64	1.19	0.55	0.55
u128	1.20	0.50	0.50
u256	1.20	0.48	0.48

Figure 9. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.08	1.30	1.51	1.51	1.08	1.09
u4	1.20	1.53	1.51	1.51	1.20	1.20
u8	1.25	2.45	1.52	1.51	1.25	1.25
u16	1.26	2.45	1.06	1.06	1.26	1.26
u32	1.28	2.51	0.75	0.75	1.28	1.28
u64	1.29	2.54	0.69	0.69	1.29	1.29
u128	1.29	2.56	0.63	0.63	1.29	1.29
u256	1.29	2.56	0.61	0.61	1.29	1.29

Figure 9. (c) Accélération pour l'unrolling(su_rs/u_n)

3.2 N=10000 et d=50%

	O0	O1	O2	O3	funroll	funall
so	4625402.5	3798963.25	3266474.25	3266285.25	4624745	4625933
su_rs	1892579	3265616.5	1906037.62	1905402.38	1892400.25	1891215.62
u2	1761647.38	2540786.75	1218746.38	1219033.38	1760440.62	1757969.62
u4	1582527.62	2174329	1220504.62	1219790	1581851.62	1582870.75
u8	1527757.62	1280479.62	1218221	1221191	1528041.38	1528413
u16	1517648.62	1282746.75	1737682.25	1736448.38	1518971.62	1516859.25
u32	1494076.38	1254039.25	2458107	2459587.5	1497875.25	1498709.62
u64	1493055.62	1235795.25	2668462.5	2668283.5	1490557.25	1489870.62
u128	1487132.38	1230005	2943160	2943838.5	1489603.38	1490785.38
u256	1487788.62	1228370.38	3040969.25	3037567.5	1485644.75	1486257

Figure 10. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.22	1.42	1.42
su_rs	0.58	0.99	0.99
u2	0.69	1.45	1.45
u4	0.73	1.30	1.30
u8	1.19	1.25	1.25
u16	1.18	0.87	0.87
u32	1.19	0.61	0.61
u64	1.21	0.56	0.56
u128	1.21	0.51	0.51
u256	1.21	0.49	0.49

Figure 10. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.07	1.29	1.56	1.56	1.07	1.08
u4	1.20	1.50	1.56	1.56	1.20	1.19
u8	1.24	2.55	1.56	1.56	1.24	1.24
u16	1.25	2.55	1.10	1.10	1.25	1.25
u32	1.27	2.60	0.78	0.77	1.26	1.26
u64	1.27	2.64	0.71	0.71	1.27	1.27
u128	1.27	2.65	0.65	0.65	1.27	1.27
u256	1.27	2.66	0.63	0.63	1.27	1.27

Figure 10. (c) Accélération pour l'unrolling(su_rs/u_n)

3.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	funroll	funall
no	1139002.75	902780.81	770002.38	769854.19	1140125.38	1139036.62
su_rs	468048.81	769504.38	449759.59	449982.59	469718.19	468564.81
u2	421002.41	588525.81	296890.59	297198.81	425087.19	421228.19
u4	390158.41	497138.19	296277.41	296972	391737.59	394252.19
u8	372336.59	316102	296617	297703.19	374364.81	381083
u16	370215.81	311936	425591.59	425787.81	372226	374944
u32	364375.19	306088.41	590213	591212.81	366270	367635.19
u64	360159.81	301865.59	672182.81	673516.19	362717.41	360945.19
u128	358657.59	300544.19	718242.19	717337.38	359923.59	358502.19
u256	361800.59	311376.19	735603.19	735655.19	363417.81	362097.59

Figure 11. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.26	1.48	1.48
su_rs	0.61	1.04	1.04
u2	0.72	1.42	1.42
u4	0.78	1.32	1.31
u8	1.18	1.26	1.25
u16	1.19	0.87	0.87
u32	1.19	0.62	0.62
u64	1.19	0.54	0.53
u128	1.19	0.50	0.50
u256	1.16	0.49	0.49

Figure 11. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.11	1.31	1.51	1.51	1.10	1.11
u4	1.20	1.55	1.52	1.52	1.20	1.19
u8	1.26	2.43	1.52	1.51	1.25	1.23
u16	1.26	2.47	1.06	1.06	1.26	1.25
u32	1.28	2.51	0.76	0.76	1.28	1.27
u64	1.30	2.55	0.67	0.67	1.29	1.30
u128	1.31	2.56	0.63	0.63	1.31	1.31
u256	1.29	2.47	0.61	0.61	1.29	1.29

Figure 11. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	2.44	1.17	1.70	1.70	2.44	2.44
N=10000 et d=50%	1.08	1.30	1.51	1.51	1.08	1.09
Sherman3	1.10	1.18	1.00	1.00	1.11	1.11

Figure 12. Accélération apportée pour le remplacement scalaire (no/su_rs)

Annexe B : Résultats pour le PMVC-CSR

1. Sur la machine 1

1.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	148172	226747.8	76874.2	126183.8	172015.2	404438.6
su_rs	96872	41741.4	42372	38995.8	96245.4	96093.8
u2	78541	39178.6	40625.4	39669.4	79600	79911.6
u4	71973.4	38483.6	41816.8	39225	72463.4	73598.8
u8	68131.6	38147.4	39536.8	38834.8	68928.4	69713
u16	66657.2	39413.8	40458.4	40732	66776.2	68042.4
u32	66286.2	43037.6	44367	43283.8	66293.8	68112.8
u64	69408.4	44084.2	43827	43554.6	69354.8	71378.8
u128	67551.2	42689.6	50575.6	50166.2	68196.4	69772
u256	70501.4	40675.8	49280	50320.2	70859	72607

Figure 1. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.65	1.93	1.17
su_rs	2.32	2.29	2.48
u2	2.00	1.93	1.98
u4	1.87	1.72	1.83
u8	1.79	1.72	1.75
u16	1.69	1.65	1.64
u32	1.54	1.49	1.53
u64	1.57	1.58	1.59
u128	1.58	1.34	1.35
u256	1.73	1.43	1.40

Figure 1. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.23	1.07	1.04	0.98	1.21	1.20
u4	1.35	1.08	1.01	0.99	1.33	1.31
u8	1.42	1.09	1.07	1.00	1.40	1.38
u16	1.45	1.06	1.05	0.96	1.44	1.41
u32	1.46	0.97	0.96	0.90	1.45	1.41
u64	1.40	0.95	0.97	0.90	1.39	1.35
u128	1.43	0.98	0.84	0.78	1.41	1.38
u256	1.37	1.03	0.86	0.77	1.36	1.32

Figure 1. (c) Accélération pour l'unrolling (su_rs/u_n)

1.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	6281428.4	5619624.8	5429941	6562077.4	4309401.8	5240520
su_rs	429963	173649.6	176867	154173.4	429483.4	427763
u2	347306.8	162567.4	174420.8	161777.4	347082.8	345279.2
u4	314548.4	158092	168150	158128.4	315207.6	312778.2
u8	297168.2	154828.8	162866.4	157055	298090.4	295533.6
u16	289149.8	164423.4	166082.8	165981.4	290915.4	287945.2
u32	287943.6	174096.6	174532.8	176899.8	288596.8	287547.8
u64	301259	177446	179597	178241.6	303177.4	299774.2
u128	288523	189142.8	221546.4	220788	290319.8	286136.2
u256	289567.8	162747.2	205175	204221.6	290343.6	287694.6

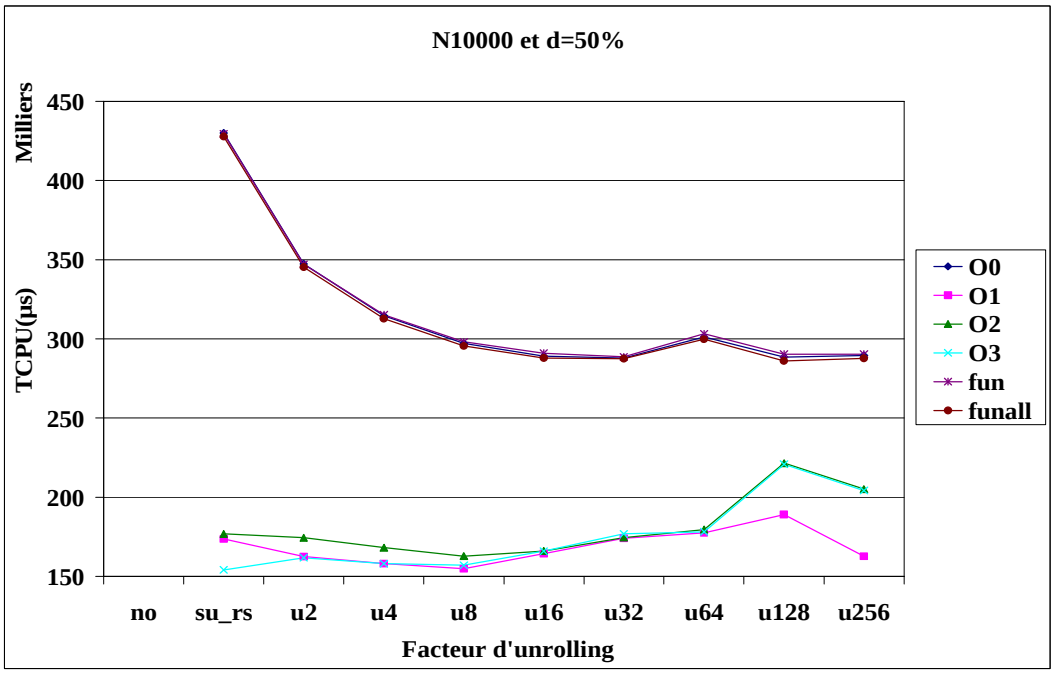


Figure 2. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.12	1.16	0.96
su_rs	2.48	2.43	2.79
u2	2.14	1.99	2.15
u4	1.99	1.87	1.99
u8	1.92	1.82	1.89
u16	1.76	1.74	1.74
u32	1.65	1.65	1.63
u64	1.70	1.68	1.69
u128	1.53	1.30	1.31
u256	1.78	1.41	1.42

Figure 2. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.24	1.07	1.01	0.95	1.24	1.24
u4	1.37	1.10	1.05	0.97	1.36	1.37
u8	1.45	1.12	1.09	0.98	1.44	1.45
u16	1.49	1.06	1.06	0.93	1.48	1.49
u32	1.49	1.00	1.01	0.87	1.49	1.49
u64	1.43	0.98	0.98	0.86	1.42	1.43
u128	1.49	0.92	0.80	0.70	1.48	1.49
u256	1.48	1.07	0.86	0.75	1.48	1.49

Figure 2. (c) Accélération pour l'unrolling(su_rs/u_n)

1.3 Sherman3, N=5005 et d=0.08%

	O0	O1	O2	O3	fun	funall
no	321.4	196	168.6	215	321.8	325.4
su_rs	255.2	159.8	160.4	176.6	257.2	254
u2	296.2	158.8	163.8	176.4	281.4	375.4
u4	268.4	158.2	161.8	160.2	268	275.4
u8	302.2	164.6	164.6	162.6	304	302.8
u16	305	166.2	207.2	166.2	313.2	305.6
u32	300	160.8	182.4	159.4	310	300.8
u64	309.6	239.8	163.4	160	300.4	300.8
u128	302.8	192.8	167.4	177.6	303.4	315.2
u256	300.2	186.4	170.2	619.2	302.4	301.8

Figure 3. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.64	1.91	1.49
su_rs	1.60	1.59	1.45
u2	1.87	1.81	1.68
u4	1.70	1.66	1.68
u8	1.84	1.84	1.86
u16	1.84	1.47	1.84
u32	1.87	1.64	1.88
u64	1.29	1.89	1.94
u128	1.57	1.81	1.70
u256	1.61	1.76	0.48

Figure 3. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	0.86	1.01	0.98	1.00	0.91	0.68
u4	0.95	1.01	0.99	1.10	0.96	0.92
u8	0.84	0.97	0.97	1.09	0.85	0.84
u16	0.84	0.96	0.77	1.06	0.82	0.83
u32	0.85	0.99	0.88	1.11	0.83	0.84
u64	0.82	0.67	0.98	1.10	0.86	0.84
u128	0.84	0.83	0.96	0.99	0.85	0.81
u256	0.85	0.86	0.94	0.29	0.85	0.84

Figure 3. (c) Accélération pour l'unrolling (su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.53	5.43	1.81	3.24	1.79	4.21
N=10000 et d=50%	14.61	32.36	30.70	42.56	10.03	12.25
Sherman3	1.26	1.23	1.05	1.22	1.25	1.28

Figure 4. Accélération pour le remplacement scalaire (no/su_rs)

2. Sur la machine 2

2.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	142605	84942	56394	55621.8	142645.4	142868.4
nu_sr	162868.6	52806.8	50622.2	50352.2	162928	162943.6
u2	108828.2	50933.2	49884.4	49615.6	108892	108991.6
u4	91924.2	49930.6	49702.8	49436.6	91960.8	92086
u8	85114.2	50181	50071.6	49856.8	85039.8	85301.4
u16	79494.4	51349.8	52041.8	51881.6	79305.2	79676.6
u32	78616.8	55324.6	55995.2	55664.6	78444.6	78824.2
u64	79847	55583.2	56252.4	55995	79707.4	80082.4
u128	81663.6	55037.6	55823	55777.2	81549.4	81896.2
u256	90547.8	55671.4	56646.6	56023.6	90509.8	90823.4

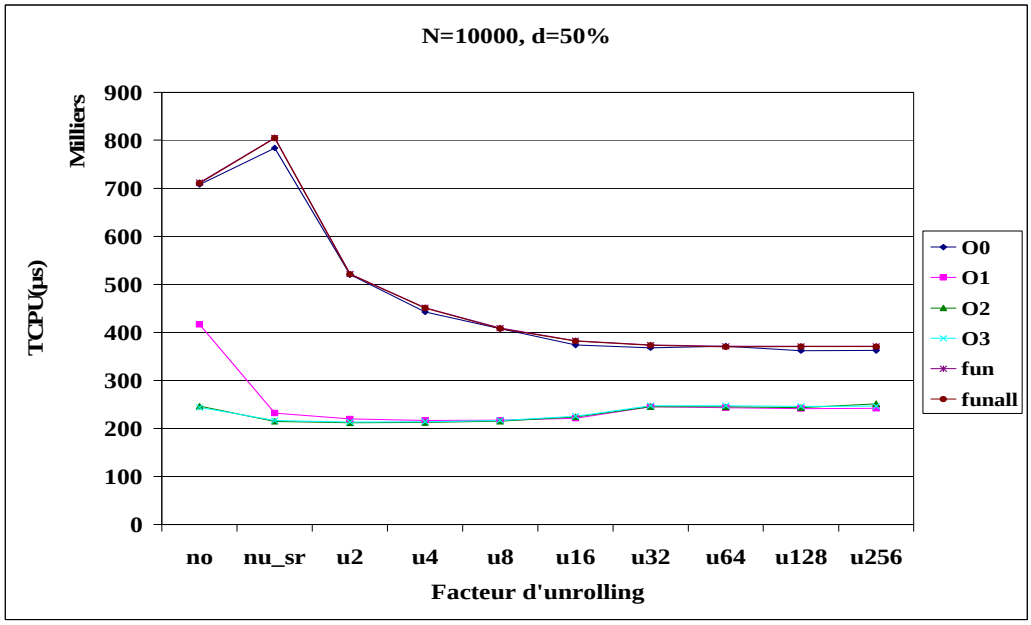


Figure 5. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.70	2.87	2.90
su_rs	3.38	3.65	3.62
u2	2.37	2.46	2.44
u4	2.04	2.09	2.07
u8	1.88	1.90	1.88
u16	1.69	1.67	1.66
u32	1.50	1.50	1.49
u64	1.53	1.52	1.50
u128	1.50	1.49	1.47
u256	1.49	1.44	1.47

Figure 5. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.51	1.05	1.01	1.01	1.54	1.54
u4	1.77	1.07	1.01	1.01	1.78	1.78
u8	1.92	1.07	1.00	1.00	1.97	1.97
u16	2.10	1.05	0.96	0.96	2.10	2.11
u32	2.13	0.95	0.88	0.88	2.15	2.16
u64	2.11	0.95	0.88	0.88	2.17	2.18
u128	2.17	0.96	0.88	0.88	2.17	2.17
u256	2.16	0.96	0.85	0.88	2.17	2.17

Figure 5. (c) Accélération pour l'unrolling(su_rs/u_n)

2.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	707763.4	416553.4	246382.8	243892.2	712112.6	710642.6
nu_sr	783879.8	231945.2	214738.6	216353.2	804879.6	804715
u2	520335.2	219982.4	211899.4	213488.4	521659	521049
u4	442807.6	216617.2	212372.8	213940	451435.8	450874
u8	407935.2	217115.6	215154.6	216676.8	408995.6	408158.4
u16	373783.2	221015.6	224111	225634	382586.4	381756.6
u32	367857	245019.4	245315.8	247216.8	373679.4	372829.8
u64	371250	243251.6	244779.8	247159.6	370935.8	369850.8
u128	361835	241417.8	243528.2	246066.6	371135.4	370414.8
u256	362176.8	242299.4	251362.8	246092.8	371090.4	370150.4

Figure 6. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.70	2.87	2.90
su_rs	3.38	3.65	3.62
u2	2.37	2.46	2.44
u4	2.04	2.09	2.07
u8	1.88	1.90	1.88
u16	1.69	1.67	1.66
u32	1.50	1.50	1.49
u64	1.53	1.52	1.50
u128	1.50	1.49	1.47
u256	1.49	1.44	1.47

Figure 6. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.51	1.05	1.01	1.01	1.54	1.54
u4	1.77	1.07	1.01	1.01	1.78	1.78
u8	1.92	1.07	1.00	1.00	1.97	1.97
u16	2.10	1.05	0.96	0.96	2.10	2.11
u32	2.13	0.95	0.88	0.88	2.15	2.16
u64	2.11	0.95	0.88	0.88	2.17	2.18
u128	2.17	0.96	0.88	0.88	2.17	2.17
u256	2.16	0.96	0.85	0.88	2.17	2.17

Figure 6. (c) Accélération pour l'unrolling(su_rs/u_n)

2.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	fun	funall
no	373.8	146.8	92	98	372.2	373.6
nu_sr	337	84.8	78	78.2	330.8	331.4
u2	328.6	96.6	92.8	91.4	330	339.6
u4	323.8	94.8	92.4	87.6	326	330.2
u8	408.4	100	94.4	88	410.4	402.8
u16	422.8	114.8	113.6	104.8	422	420.6
u32	410.4	114.6	109.2	109.2	412.6	411
u64	400.2	118.4	103.2	109.8	403.8	414.6
u128	400.8	113.8	105.4	108	402.8	415
u256	416.2	114.2	109.8	108.4	413.8	417.6

Figure 7. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.55	4.06	3.81
su_rs	3.97	4.32	4.31
u2	3.40	3.54	3.60
u4	3.42	3.50	3.70
u8	4.08	4.33	4.64
u16	3.68	3.72	4.03
u32	3.58	3.76	3.76
u64	3.38	3.88	3.64
u128	3.52	3.80	3.71
u256	3.64	3.79	3.84

Figure 7. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.03	0.88	0.84	0.86	1.00	0.98
u4	1.04	0.89	0.84	0.89	1.01	1.00
u8	0.83	0.85	0.83	0.89	0.81	0.82
u16	0.80	0.74	0.69	0.75	0.78	0.79
u32	0.82	0.74	0.71	0.72	0.80	0.81
u64	0.84	0.72	0.76	0.71	0.82	0.80
u128	0.84	0.75	0.74	0.72	0.82	0.80
u256	0.81	0.74	0.71	0.72	0.80	0.79

Figure 7. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	0.88	1.61	1.11	1.10	0.88	0.88
N=10000 et d=50%	0.90	1.80	1.15	1.13	0.88	0.88
Sherman3	1.11	1.73	1.18	1.25	1.13	1.13

Figure 8 . Accélération pour le remplacement scalaire (no/su_rs)

3. Sur la machine 3

3.1 N=10000 et d=10%

	O0	O1	O2	O3	funroll	funall
no	294306	411611.2	410738	410304.2	292795.2	292938.4
su_rs	214389.8	296278.6	298580.6	298509.8	212410.6	212506.4
u2	195781.8	273367.8	197049.4	196959	193136.8	193259.4
u4	186001.8	258304.6	192122.6	192273.4	183418.4	183603.2
u8	185037.8	250539.6	186646	187235.8	183912.4	184408
u16	184408.6	246733.4	231554	232602.8	184118.2	184522.4
u32	185358.6	245436	245026	244981.4	184573.4	184538.8
u64	186217.6	243500.2	265705	265745.6	184962	185053.8
u128	187065.6	242050.4	258516.4	257779	185887.4	186092
u256	191490.4	244571.2	271704.8	273202.8	189685.2	190060.2

Figure 9. (a) TCPU (µs)

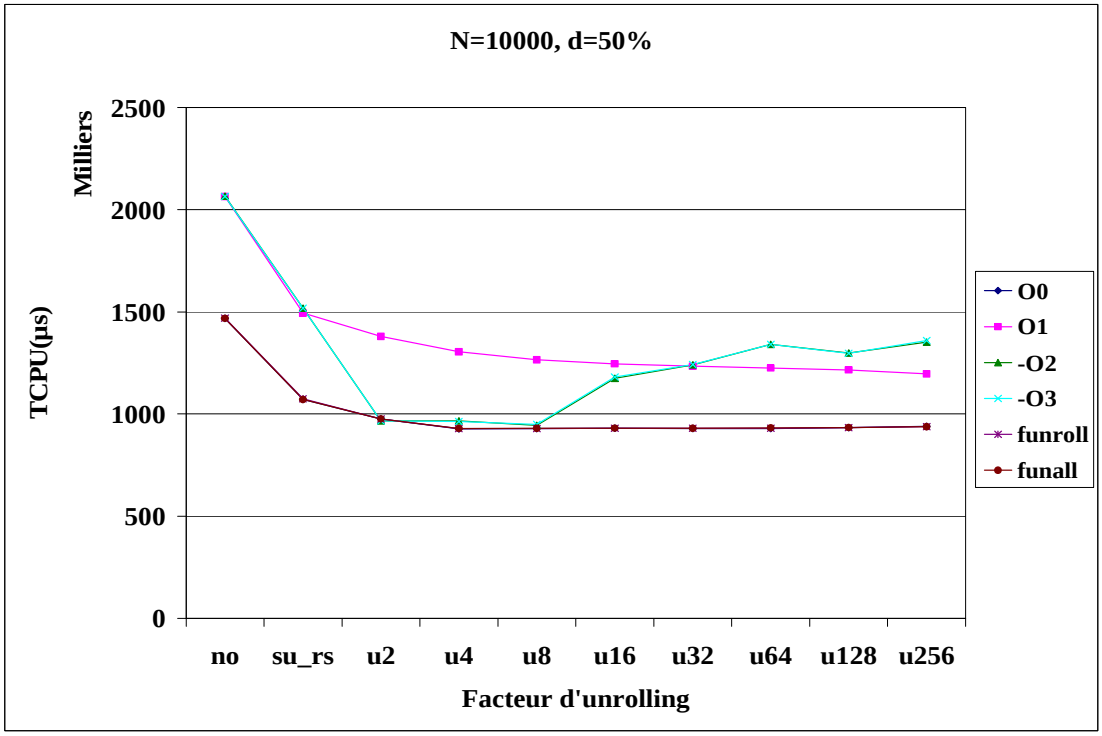
	O0/O1	O0/O2	O0/O3
no	0.72	0.72	0.72
su_rs	0.72	0.72	0.72
u2	0.72	0.99	0.99
u4	0.72	0.97	0.97
u8	0.74	0.99	0.99
u16	0.75	0.80	0.79
u32	0.76	0.76	0.76
u64	0.76	0.70	0.70
u128	0.77	0.72	0.73
u256	0.78	0.70	0.70

Figure 9. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.10	1.08	1.52	1.52	1.10	1.10
u4	1.15	1.15	1.55	1.55	1.16	1.16
u8	1.16	1.18	1.60	1.59	1.15	1.15
u16	1.16	1.20	1.29	1.28	1.15	1.15
u32	1.16	1.21	1.22	1.22	1.15	1.15
u64	1.15	1.22	1.12	1.12	1.15	1.15
u128	1.15	1.22	1.15	1.16	1.14	1.14
u256	1.12	1.21	1.10	1.09	1.12	1.12

Figure 9. (c) Accélération pour l'unrolling(su_rs/u_n)

3.2 N=10000 et d=50%



	O0	O1	O2	O3	funroll	funall
no	1468272.8	2064460.8	2065580.2	2065423.4	1470076.6	1468817.4
su_rs	1071640.4	1493732.2	1517482.4	1518491.4	1073914.8	1070758.2
u2	975543.2	1379661.6	966163	966430.8	975487	976256.4
u4	927792.4	1304430.2	966869	963520.8	928835	929410
u8	928668.6	1265643.2	945160.4	948327.4	929326	929763.6
u16	929749.8	1245535	1174866.4	1181182	931104.8	930468.8
u32	929605	1233845.2	1241205.6	1241742.8	930849	930354
u64	928991.8	1225292	1339892.4	1341354.8	930806.8	932560
u128	933445.2	1215520.6	1298838	1296512.8	932760.4	933706.6
u256	938927	1196190.4	1351070.4	1360691.8	937997.8	937538.2

Figure 10. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.71	0.71	0.71
su_rs	0.72	0.71	0.71
u2	0.71	1.01	1.01
u4	0.71	0.96	0.96
u8	0.73	0.98	0.98
u16	0.75	0.79	0.79
u32	0.75	0.75	0.75
u64	0.76	0.69	0.69
u128	0.77	0.72	0.72
u256	0.78	0.69	0.69

Figure 10. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.10	1.08	1.57	1.57	1.10	1.10
u4	1.16	1.15	1.57	1.58	1.16	1.15
u8	1.15	1.18	1.61	1.60	1.16	1.15
u16	1.15	1.20	1.29	1.29	1.15	1.15
u32	1.15	1.21	1.22	1.22	1.15	1.15
u64	1.15	1.22	1.13	1.13	1.15	1.15
u128	1.15	1.23	1.17	1.17	1.15	1.15
u256	1.14	1.25	1.12	1.12	1.14	1.14

Figure 10. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	funroll	funall
no	857.2	898.4	893.8	893.8	857.2	857.4
su_rs	823.6	606.4	615.4	617.4	811.8	939.6
u2	952.4	620.4	552.6	552.8	952.4	953.6
u4	931.8	583	511.6	511.2	933	1100
u8	918.6	635.8	643	644.2	918.4	927.4
u16	930.6	637.8	644.2	636.6	930.6	937.6
u32	931.4	633.2	636.4	640.2	931.4	935.2
u64	928.6	632.6	640	643.2	928.4	964.6
u128	931.8	633.8	643	636.4	931.8	934.6
u256	1057.2	633.2	642.6	636	930	1154.4

Figure 11. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.95	0.96	0.96
su_rs	1.36	1.34	1.33
u2	1.54	1.72	1.72
u4	1.60	1.82	1.82
u8	1.44	1.43	1.43
u16	1.46	1.44	1.46
u32	1.47	1.46	1.45
u64	1.47	1.45	1.44
u128	1.47	1.45	1.46
u256	1.67	1.65	1.66

Figure 11. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	0.86	0.98	1.11	1.12	0.85	0.99
u4	0.88	1.04	1.20	1.21	0.87	0.85
u8	0.90	0.95	0.96	0.96	0.88	1.01
u16	0.89	0.95	0.96	0.97	0.87	1.00
u32	0.88	0.96	0.97	0.96	0.87	1.00
u64	0.89	0.96	0.96	0.96	0.87	0.97
u128	0.88	0.96	0.96	0.97	0.87	1.01
u256	0.78	0.96	0.96	0.97	0.87	0.81

Figure 11. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.38	2.58	1.19	1.06	1.38	1.38
N=10000 et d=50%	1.37	2.57	1.19	1.06	1.37	1.37
Sherman3	1.35	2.48	1.17	1.05	1.35	1.36

Figure 12. Accélération pour le remplacement scalaire (no/su_rs)

Annexe C : Résultats pour le PMVC-COO

1. Sur la machine 1

1.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	420736.4	782005.8	447288.2	348292	516557.6	306352.6
su_rs	153719	74852.8	76899.2	76310.2	153247.2	152257.8
u2	135838.4	75802.2	74128.6	72837.6	136875.8	135607.8
u4	124102.2	80125.2	80895.4	80499.6	125847.8	123850.8
u8	121649.8	82026.6	69102.2	68341.2	123869.2	120583
u16	144963	84804.8	66717.6	65728.8	145082.8	143345.2
u32	121749.8	88860.2	67781	66991.2	123367	121442.2
u64	123660.6	86874.2	70750.6	71455.8	124175.4	123403.8
u128	123121.6	108898.8	94102.6	94536.8	123285	122512
u256	123895.2	111003.8	94918.8	95147.4	123975.8	124383.6

Figure 1. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.54	0.94	1.21
su_rs	2.05	2.00	2.01
u2	1.79	1.83	1.86
u4	1.55	1.53	1.54
u8	1.48	1.76	1.78
u16	1.71	2.17	2.21
u32	1.37	1.80	1.82
u64	1.42	1.75	1.73
u128	1.13	1.31	1.30
u256	1.12	1.31	1.30

Figure 1. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.13	0.99	1.04	1.05	1.12	1.12
u4	1.24	0.93	0.95	0.95	1.22	1.23
u8	1.26	0.91	1.11	1.12	1.24	1.26
u16	1.06	0.88	1.15	1.16	1.06	1.06
u32	1.26	0.84	1.13	1.14	1.24	1.25
u64	1.24	0.86	1.09	1.07	1.23	1.23
u128	1.25	0.69	0.82	0.81	1.24	1.24
u256	1.24	0.67	0.81	0.80	1.24	1.22

Figure 1. (c) Accélération pour l'unrolling (su_rs/u_n)

1.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	12387363.6	11753997.2	16287175.4	8950263.8	10266455.4	20034658.2
su_rs	681579.2	336761.6	354629.8	338266.4	680571.4	680655
u2	619480.6	335236.2	340864	326751	618530.4	618086.8
u4	571322.8	361152.8	371912	361334.4	570148.6	570939.2
u8	561924.8	375221.4	327479.6	308135.2	560776.2	561539.8
u16	698098.4	390268.2	313582	297598.4	697072.8	700073.2
u32	567703.4	409629.2	309735	304502.6	566657.4	568771.6
u64	574511.4	397883.8	329229.2	323612	575009.4	577930.6
u128	573353.2	521436.4	437452.8	428016.2	571616	571974.4
u256	575859.8	538424.2	444459	429522.4	576146.6	575945.6

Figure 2. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.05	0.76	1.38
su_rs	2.02	1.92	2.01
u2	1.85	1.82	1.90
u4	1.58	1.54	1.58
u8	1.50	1.72	1.82
u16	1.79	2.23	2.35
u32	1.39	1.83	1.86
u64	1.44	1.75	1.78
u128	1.10	1.31	1.34
u256	1.07	1.30	1.34

Figure 2. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.10	1.00	1.04	1.04	1.10	1.10
u4	1.19	0.93	0.95	0.94	1.19	1.19
u8	1.21	0.90	1.08	1.10	1.21	1.21
u16	0.98	0.86	1.13	1.14	0.98	0.97
u32	1.20	0.82	1.14	1.11	1.20	1.20
u64	1.19	0.85	1.08	1.05	1.18	1.18
u128	1.19	0.65	0.81	0.79	1.19	1.19
u256	1.18	0.63	0.80	0.79	1.18	1.18

Figure 2. (c) Accélération pour l'unrolling(su_rs/u_n)

1.3 Sherman3, N=5005 et d=0.08%

	O0	O1	O2	O3	fun	funall
no	262.8	150.2	149	173	260.4	260.8
su_rs	397.8	147	179.8	154.2	270	269.2
u2	240.4	147.6	166.6	153.6	239	346
u4	221	149.6	148.6	165	221.6	226
u8	219.4	157.6	138.4	151.4	216.8	236
u16	296.4	157.4	134.8	153	293.4	293.2
u32	216	163	138.8	250.8	220.8	313
u64	234.8	188.4	136.2	253.2	229.4	222.6
u128	221.4	214.8	277.4	210.4	219.6	237
u256	237	241	310.2	213.6	233.4	511

Figure 3. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.75	1.76	1.52
su_rs	2.71	2.21	2.58
u2	1.63	1.44	1.57
u4	1.48	1.49	1.34
u8	1.39	1.59	1.45
u16	1.88	2.20	1.94
u32	1.33	1.56	0.86
u64	1.25	1.72	0.93
u128	1.03	0.80	1.05
u256	0.98	0.76	1.11

Figure 3. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.65	1.00	1.08	1.00	1.13	0.78
u4	1.80	0.98	1.21	0.93	1.22	1.19
u8	1.81	0.93	1.30	1.02	1.25	1.14
u16	1.34	0.93	1.33	1.01	0.92	0.92
u32	1.84	0.90	1.30	0.61	1.22	0.86
u64	1.69	0.78	1.32	0.61	1.18	1.21
u128	1.80	0.68	0.65	0.73	1.23	1.14
u256	1.68	0.61	0.58	0.72	1.16	0.53

Figure 3. (c) Accélération pour l'unrolling (su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	2.74	10.45	5.82	4.56	3.37	2.01
N=10000 et d=50%	18.17	34.90	45.93	26.46	15.09	29.43
Sherman3	0.66	1.02	0.83	1.12	0.96	0.97

Figure 4. Accélération pour le remplacement scalaire (no/su_rs)

2. Sur la machine 2

2.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	187528	87388	87516.2	88103.6	187693.4	187848.4
su_rs	201388.2	88319	87465.4	88071.2	199441.6	201615.4
u2	179071.6	88226	87559	87001	178269	176512.4
u4	159723.4	91435.8	87001	86996.6	156671.8	155321.2
u8	155229.2	97448.2	86418	86608.8	150974.2	149904.4
u16	150220.2	108326.4	90496.2	90793	148018.8	147589.4
u32	152484.2	114124	98809	98558.2	150492.6	149065
u64	166051.4	129321.6	112457.2	112117.4	162795.8	162418
u128	161726.6	126331.4	120649.6	120526	157794.6	157846.6
u256	153120.6	119582.6	115126.6	114970.4	150670.2	149967.2

Figure 5. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.15	2.14	2.13
su_rs	2.28	2.30	2.29
u2	2.03	2.05	2.06
u4	1.75	1.84	1.84
u8	1.59	1.80	1.79
u16	1.39	1.66	1.65
u32	1.34	1.54	1.55
u64	1.28	1.48	1.48
u128	1.28	1.34	1.34
u256	1.28	1.33	1.33

Figure 5. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.12	1.00	1.00	1.01	1.12	1.14
u4	1.26	0.97	1.01	1.01	1.27	1.30
u8	1.30	0.91	1.01	1.02	1.32	1.34
u16	1.34	0.82	0.97	0.97	1.35	1.37
u32	1.32	0.77	0.89	0.89	1.33	1.35
u64	1.21	0.68	0.78	0.79	1.23	1.24
u128	1.25	0.70	0.72	0.73	1.26	1.28
u256	1.32	0.74	0.76	0.77	1.32	1.34

Figure 5. (c) Accélération pour l'unrolling(su_rs/u_n)

2.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	913055	423687	425623.4	426470.6	917446	948254.4
su_rs	974751.8	427304.4	425543.8	426443.8	971874.2	981368
u2	878399.2	428273	425198.2	423565.2	870802.2	887267.8
u4	764988	442025.2	421642.6	422516.6	756207.4	771239.4
u8	735109	458764.4	417497.2	418547.2	734701.2	733455.8
u16	718722	518966.8	436009.8	435977.2	713906	717043
u32	712378	525700.6	460220.8	463339.4	712594.2	712901.8
u64	704808	539176.2	480457	479860.4	703864.8	707665.2
u128	701121.6	550162.2	525378	526968	704381.8	703053
u256	701584	550613.6	532957.2	535042.8	706524.8	701994.6

Figure 6. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.16	2.15	2.14
su_rs	2.28	2.29	2.29
u2	2.05	2.07	2.07
u4	1.73	1.81	1.81
u8	1.60	1.76	1.76
u16	1.38	1.65	1.65
u32	1.36	1.55	1.54
u64	1.31	1.47	1.47
u128	1.27	1.33	1.33
u256	1.27	1.32	1.31

Figure 6. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.11	1.00	1.00	1.01	1.12	1.11
u4	1.27	0.97	1.01	1.01	1.29	1.27
u8	1.33	0.93	1.02	1.02	1.32	1.34
u16	1.36	0.82	0.98	0.98	1.36	1.37
u32	1.37	0.81	0.92	0.92	1.36	1.38
u64	1.38	0.79	0.89	0.89	1.38	1.39
u128	1.39	0.78	0.81	0.81	1.38	1.40
u256	1.39	0.78	0.80	0.80	1.38	1.40

Figure 6. (c) Accélération pour l'unrolling(su_rs/u_n)

2.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	fun	funall
no	389.2	163	161.4	160.6	393	380.6
su_rs	388.6	136.2	143.8	148.6	383.4	380.8
u2	336.2	140	137.4	135	332.2	333.6
u4	284	143.8	134.8	131.8	284.4	281.6
u8	272	158	135.8	130.8	271.6	273.8
u16	264.8	165	139.8	140.2	265.4	261.8
u32	265	176.8	144.6	144.2	256.8	262.6
u64	261.4	173.4	155.8	148.2	263	263.6
u128	263.6	184	167.8	162.6	264	268.2
u256	272.2	183	162.6	162.4	270.8	272.4

Figure 7. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.39	2.41	2.42
su_rs	2.85	2.70	2.62
u2	2.40	2.45	2.49
u4	1.97	2.11	2.15
u8	1.72	2.00	2.08
u16	1.60	1.89	1.89
u32	1.50	1.83	1.84
u64	1.51	1.68	1.76
u128	1.43	1.57	1.62
u256	1.49	1.67	1.68

Figure 7. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.16	0.97	1.05	1.10	1.15	1.14
u4	1.37	0.95	1.07	1.13	1.35	1.35
u8	1.43	0.86	1.06	1.14	1.41	1.39
u16	1.47	0.83	1.03	1.06	1.44	1.45
u32	1.47	0.77	0.99	1.03	1.49	1.45
u64	1.49	0.79	0.92	1.00	1.46	1.44
u128	1.47	0.74	0.86	0.91	1.45	1.42
u256	1.43	0.74	0.88	0.92	1.42	1.40

Figure 7. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	0.93	0.99	1.00	1.00	0.94	0.93
N=10000 et d=50%	0.94	0.99	1.00	1.00	0.94	0.97
Sherman3	1.00	1.20	1.12	1.08	1.03	1.00

Figure 8. Accélération pour le remplacement scalaire (no/su_rs)

3. Sur la machine 3

3.1 N=10000 et d=10%

	O0	O1	O2	O3	funroll	funall
no	357911.6	406032.2	399669.4	400909	358556.8	357745.8
su_rs	315166.8	404813.2	399127.4	400109.2	314829.6	315154
u2	269964.8	386728.6	352002	352622.4	269324.4	271545.8
u4	253166	369385.8	350105.8	350710	252855.2	253288.2
u8	248982.4	358882.2	350819.6	351448.4	248400.6	248636.8
u16	248426	355175.8	349783.6	350858	248263.6	248171.6
u32	282109.6	400564.4	386164	387328.2	281525.8	281850
u64	283694.2	401847.8	392426.8	393013.2	283625.2	283430.6
u128	285991.4	393639.4	400849.8	400879.6	285588	285568.2
u256	288076.6	389198.8	403343.4	404347.4	291005.4	290027.2

Figure 9. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.88	0.90	0.89
su_rs	0.78	0.79	0.79
u2	0.70	0.77	0.77
u4	0.69	0.72	0.72
u8	0.69	0.71	0.71
u16	0.70	0.71	0.71
u32	0.70	0.73	0.73
u64	0.71	0.72	0.72
u128	0.73	0.71	0.71
u256	0.74	0.71	0.71

Figure 9. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.17	1.05	1.13	1.13	1.17	1.16
u4	1.24	1.10	1.14	1.14	1.25	1.24
u8	1.27	1.13	1.14	1.14	1.27	1.27
u16	1.27	1.14	1.14	1.14	1.27	1.27
u32	1.12	1.01	1.03	1.03	1.12	1.12
u64	1.11	1.01	1.02	1.02	1.11	1.11
u128	1.10	1.03	1.00	1.00	1.10	1.10
u256	1.09	1.04	0.99	0.99	1.08	1.09

Figure 9. (c) Accélération pour l'unrolling (su_rs/u_n)

3.2 N=10000 et d=50%

	O0	O1	O2	O3	funroll	funall
no	1809267.2	2049577.8	2017784	2021130	1806591.2	1808008
su_rs	1593477	2047529.2	2017341	2021241.4	1592573.6	1593352.2
u2	1369654.8	1960761.4	1786809.2	1785175	1370642.4	1368892
u4	1284555.2	1871437.8	1778700.8	1777755	1285911.6	1282778.2
u8	1266977	1817094.6	1782094.8	1780621	1266700	1266009.4
u16	1263060.4	1796677.6	1774136.8	1776077.4	1263194.2	1261036.6
u32	1428037.6	2005084.6	1956408.4	1956275.8	1430411.8	1429844.2
u64	1436639	2009703.8	1984009.4	1980124.6	1441178.8	1436510.6
u128	1445257.4	1972069.2	2019727.2	2020718.4	1446297.2	1445535.2
u256	1467980.6	1960257	2034355.2	2035338.2	1458344.6	1459970.4

Figure 10. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.88	0.90	0.90
su_rs	0.78	0.79	0.79
u2	0.70	0.77	0.77
u4	0.69	0.72	0.72
u8	0.70	0.71	0.71
u16	0.70	0.71	0.71
u32	0.71	0.73	0.73
u64	0.71	0.72	0.73
u128	0.73	0.72	0.72
u256	0.75	0.72	0.72

Figure 10. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.16	1.04	1.13	1.13	1.16	1.16
u4	1.24	1.09	1.13	1.14	1.24	1.24
u8	1.26	1.13	1.13	1.14	1.26	1.26
u16	1.26	1.14	1.14	1.14	1.26	1.26
u32	1.12	1.02	1.03	1.03	1.11	1.11
u64	1.11	1.02	1.02	1.02	1.11	1.11
u128	1.10	1.04	1.00	1.00	1.10	1.10
u256	1.09	1.04	0.99	0.99	1.09	1.09

Figure 10. (c) Accélération pour l'unrolling(su_rs/u_n)

3.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	funroll	funall
no	699.4	802.6	786.8	786.6	699.8	699.4
su_rs	594.2	785	768.8	768.4	593.8	594.2
u2	499.2	748	676	676.4	498.8	499
u4	456.4	713.4	672.2	672	456.2	456.2
u8	451.6	697.4	680.2	675.4	451.8	452
u16	453	726.4	673.4	673.4	452.8	452.6
u32	448	701.4	679.6	679.8	448	448.2
u64	451.2	725.6	696.4	694.2	451.4	455
u128	453.8	708	704	702.8	454.2	453.8
u256	461.8	705.2	708.6	711	459	465.6

Figure 11. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	0.87	0.89	0.89
su_rs	0.76	0.77	0.77
u2	0.67	0.74	0.74
u4	0.64	0.68	0.68
u8	0.65	0.66	0.67
u16	0.62	0.67	0.67
u32	0.64	0.66	0.66
u64	0.62	0.65	0.65
u128	0.64	0.64	0.65
u256	0.65	0.65	0.65

Figure 11. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.19	1.05	1.14	1.14	1.19	1.19
u4	1.30	1.10	1.14	1.14	1.30	1.30
u8	1.32	1.13	1.13	1.14	1.31	1.31
u16	1.31	1.08	1.14	1.14	1.31	1.31
u32	1.33	1.12	1.13	1.13	1.33	1.33
u64	1.32	1.08	1.10	1.11	1.32	1.31
u128	1.31	1.11	1.09	1.09	1.31	1.31
u256	1.29	1.11	1.08	1.08	1.29	1.28

Figure 11. (c) Accélération pour l'unrolling(su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.14	1.00	1.00	1.00	1.14	1.14
N=10000 et d=50%	1.14	1.00	1.00	1.00	1.13	1.13
Sherman3	1.18	1.02	1.02	1.02	1.18	1.18

Figure 12. Accélération apportée par le remplacement scalaire (no/su_rs)

Annexe D : Résultats pour le PMVC-BND

1. Sur la machine 1

1.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	186511.59	103416.4	122014.6	120094	186688	186771.8
su_rs	155634.2	87430.2	101600.6	101323.6	155104.2	154813
u2	135475.59	93483.6	97586.4	97887.6	134934.8	134772.59
u4	126286	89616.8	93969	94120.8	125695.2	125543.6
u8	122228.8	85296.6	100930.6	100755	121723.2	121623.8
u16	122713.2	89881.2	102237	102340.2	121365	121570.6
u32	120581.4	89747.4	101815.8	101958.2	119045.8	120354.6
u64	121576.2	88547.6	102406.4	102998.2	119898.6	121068.6
u128	121817.6	93573	106385.4	106636.4	119972.4	120400
u256	123481	93708.2	108292.4	108496.8	122152.8	122603.6

Figure 1. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.80	1.53	1.55
su_rs	1.78	1.53	1.54
u2	1.45	1.39	1.38
u4	1.41	1.34	1.34
u8	1.43	1.21	1.21
u16	1.37	1.20	1.20
u32	1.34	1.18	1.18
u64	1.37	1.19	1.18
u128	1.30	1.15	1.14
u256	1.32	1.14	1.14

Figure 1. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.15	0.94	1.04	1.04	1.15	1.15
u4	1.23	0.98	1.08	1.08	1.23	1.23
u8	1.27	1.03	1.01	1.01	1.27	1.27
u16	1.27	0.97	0.99	0.99	1.28	1.27
u32	1.29	0.97	1.00	0.99	1.30	1.29
u64	1.28	0.99	0.99	0.98	1.29	1.28
u128	1.28	0.93	0.96	0.95	1.29	1.29
u256	1.26	0.93	0.94	0.93	1.27	1.26

Figure 1. (c) Accélération pour l'unrolling (su_rs/u_n)

1.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	Funall
no	186805	102833.6	122239.8	118014.6	186393.2	186310.41
su_rs	155060.8	87089.8	101456	102913.6	154980.2	155124
u2	135177.2	93360.6	97696.4	92403.6	134937	134933.2
u4	125578.2	89395.2	93779.6	99725.2	125703.2	125852.2
u8	121831.8	85091.2	100862.2	100878.8	121602	121811.8
u16	121813.2	89717.2	101796.2	102418.8	121653.6	122122.2
u32	120169.4	89890.4	101651.4	102346	119056.2	120474.4
u64	121212.6	88583.4	102478	102877.8	119981.2	121149.2
u128	120405.8	93470.8	106499.4	106617.8	120320.8	120912.2
u256	122420	93529.2	108577.8	108438.6	121904.8	123092.6

Figure 2. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.82	1.53	1.58
su_rs	1.78	1.53	1.51
u2	1.45	1.38	1.46
u4	1.40	1.34	1.26
u8	1.43	1.21	1.21
u16	1.36	1.20	1.19
u32	1.34	1.18	1.17
u64	1.37	1.18	1.18
u128	1.29	1.13	1.13
u256	1.31	1.13	1.13

Figure 2. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.15	0.93	1.04	1.11	1.15	1.15
u4	1.23	0.97	1.08	1.03	1.23	1.23
u8	1.27	1.02	1.01	1.02	1.27	1.27
u16	1.27	0.97	1.00	1.00	1.27	1.27
u32	1.29	0.97	1.00	1.01	1.30	1.29
u64	1.28	0.98	0.99	1.00	1.29	1.28
u128	1.29	0.93	0.95	0.97	1.29	1.28
u256	1.27	0.93	0.93	0.95	1.27	1.26

Figure 2. (c) Accélération pour l'unrolling(su_rs/u_n)

1.3 Sherman3, N=5005 et d=0.08%

	O0	O1	O2	O3	fun	Funall
no	90919.8	50524.8	59433	56181.8	90915.2	89952.2
su_rs	75181.2	41883.2	50031.2	48096.6	75046.2	75377.4
u2	65225.8	45061	45006.2	47628.6	65329.4	65251.6
u4	60768.8	41531	48772.4	48122.6	60702.6	60776.8
u8	58939.6	43320.6	48821.4	48753.8	58740.2	59068.2
u16	58816.4	41570.8	50033.8	49606.8	58518.6	59052.6
u32	57991	41328	49274.2	49170.2	57583.4	58026.6
u64	57707.8	41402.8	49425.2	49590.8	57318.2	58367
u128	57749.4	42239.8	51195.8	51528.6	57230.8	58052.2
u256	58593.4	42498.6	52277.6	52483.2	58192.2	58636

Figure 3. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.80	1.53	1.62
su_rs	1.80	1.50	1.56
u2	1.45	1.45	1.37
u4	1.46	1.25	1.26
u8	1.36	1.21	1.21
u16	1.41	1.18	1.19
u32	1.40	1.18	1.18
u64	1.39	1.17	1.16
u128	1.37	1.13	1.12
u256	1.38	1.12	1.12

Figure 3. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.15	0.93	1.11	1.01	1.15	1.16
u4	1.24	1.01	1.03	1.00	1.24	1.24
u8	1.28	0.97	1.02	0.99	1.28	1.28
u16	1.28	1.01	1.00	0.97	1.28	1.28
u32	1.30	1.01	1.02	0.98	1.30	1.30
u64	1.30	1.01	1.01	0.97	1.31	1.29
u128	1.30	0.99	0.98	0.93	1.31	1.30
u256	1.28	0.99	0.96	0.92	1.29	1.29

Figure 3. (c) Accélération pour l'unrolling (su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.20	1.18	1.20	1.19	1.20	1.21
N=10000 et d=50%	1.20	1.18	1.20	1.15	1.20	1.20
Sherman3	1.21	1.21	1.19	1.17	1.21	1.19

Figure 4. Accélération pour le remplacement scalaire (no/su_rs)

2. Sur la machine 2

2.1 N=10000 et d=10%

	O0	O1	O2	O3	fun	funall
no	243868.8	106650	83217.8	83524	243586.8	243863.41
su_rs	225020.41	69677.6	82764.2	80033.4	225089.2	224868.8
u2	190226.59	71344.6	71133	70657.8	188875.8	190194.59
u4	176637	68433.8	68600.2	69548	176290.8	176648.2
u8	162877.2	68088.8	68645	69026	165113.8	162886.41
u16	158009.8	74519.4	71284.8	69167.2	159981	158226.8
u32	156143	68636	69958.4	69690.6	157888.8	156273.8
u64	157926	67539.6	71258.8	70863.4	159789.2	157940.2
u128	157585	68163	70210.2	69838.8	159546.41	157632.41
u256	158190.2	68260.6	71487.2	71220.2	160339.59	158277.8

Figure 5. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.29	2.93	2.92
su_rs	3.23	2.72	2.81
u2	2.67	2.67	2.69
u4	2.58	2.57	2.54
u8	2.39	2.37	2.36
u16	2.12	2.22	2.28
u32	2.27	2.23	2.24
u64	2.34	2.22	2.23
u128	2.31	2.24	2.26
u256	2.32	2.21	2.22

Figure 5. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.18	0.98	1.16	1.13	1.19	1.18
u4	1.27	1.02	1.21	1.15	1.28	1.27
u8	1.38	1.02	1.21	1.16	1.36	1.38
u16	1.42	0.94	1.16	1.16	1.41	1.42
u32	1.44	1.02	1.18	1.15	1.43	1.44
u64	1.42	1.03	1.16	1.13	1.41	1.42
u128	1.43	1.02	1.18	1.15	1.41	1.43
u256	1.42	1.02	1.16	1.12	1.40	1.42

Figure 5. (c) Accélération pour l'unrolling (su_rs/u_n)

2.2 N=10000 et d=50%

	O0	O1	O2	O3	fun	funall
no	244004.8	106637.6	83571.6	83367.2	244030.2	243549.59
su_rs	230016	69590.2	82786	80493.8	230114.59	227386.41
u2	188681.2	71395	71213.4	71836.4	188805.59	189828.8
u4	176280.41	68108.8	68679.2	69744	176217.2	176242.8
u8	163056.2	68515	68732.4	68878	163087	162496.59
u16	158861.2	70345.8	69587.6	69142.6	158801	158065
u32	157045.59	68338.2	69662.4	69837.4	157051.41	156151.41
u64	158698.41	68403.6	71303	71337.2	158691.2	157789.41
u128	158372.8	68232.6	70226.2	70260.2	158379.2	157468.8
u256	158843.41	68222.6	71385.2	71237.2	158651.41	158087.59

Figure 6. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.29	2.92	2.93
su_rs	3.31	2.78	2.86
u2	2.64	2.65	2.63
u4	2.59	2.57	2.53
u8	2.38	2.37	2.37
u16	2.26	2.28	2.30
u32	2.30	2.25	2.25
u64	2.32	2.23	2.22
u128	2.32	2.26	2.25
u256	2.33	2.23	2.23

Figure 6. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.22	0.97	1.16	1.12	1.22	1.20
u4	1.30	1.02	1.21	1.15	1.31	1.29
u8	1.41	1.02	1.20	1.17	1.41	1.40
u16	1.45	0.99	1.19	1.16	1.45	1.44
u32	1.46	1.02	1.19	1.15	1.47	1.46
u64	1.45	1.02	1.16	1.13	1.45	1.44
u128	1.45	1.02	1.18	1.15	1.45	1.44
u256	1.45	1.02	1.16	1.13	1.45	1.44

Figure 6. (c) Accélération pour l'unrolling (su_rs/u_n)

2.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	fun	funall
no	117258.2	51963	40559.4	40387	117398.2	117370.2
su_rs	108423.4	33509.8	40234.2	38476	108409.6	109456.4
u2	90644.2	34629.6	34386.2	34511.8	90646.6	92448.6
u4	84814.2	32778	33207.2	33572.4	84858.8	85401.2
u8	78635.8	33115.2	33373	33436	78637.8	78711.6
u16	76467	33160.4	34703.8	33421.4	76459.2	76569.4
u32	75522.4	33007.2	33556.6	33706.6	75502.6	75344.4
u64	74928	32698.2	34448.6	34202.2	74935.8	74811.4
u128	74836.8	32722	33305.6	33704.4	74792	74629
u256	75021	32540.8	34195.2	34173	75087.4	74943

Figure 7. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	2.26	2.89	2.90
su_rs	3.24	2.69	2.82
u2	2.62	2.64	2.63
u4	2.59	2.55	2.53
u8	2.37	2.36	2.35
u16	2.31	2.20	2.29
u32	2.29	2.25	2.24
u64	2.29	2.18	2.19
u128	2.29	2.25	2.22
u256	2.31	2.19	2.20

Figure 7. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.20	0.97	1.17	1.11	1.20	1.18
u4	1.28	1.02	1.21	1.15	1.28	1.28
u8	1.38	1.01	1.21	1.15	1.38	1.39
u16	1.42	1.01	1.16	1.15	1.42	1.43
u32	1.44	1.02	1.20	1.14	1.44	1.45
u64	1.45	1.02	1.17	1.12	1.45	1.46
u128	1.45	1.02	1.21	1.14	1.45	1.47
u256	1.45	1.03	1.18	1.13	1.44	1.46

Figure 7. (c) Accélération pour l'unrolling (su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.08	1.53	1.01	1.04	1.08	1.08
N=10000 et d=50%	1.06	1.53	1.01	1.04	1.06	1.07
Sherman3	1.08	1.55	1.01	1.05	1.08	1.07

Figure 8 . Accélération pour le remplacement scalaire (no/su_rs)

3. Sur la machine 3

3.1 N=10000 et d=10%

	O0	O1	O2	O3	funroll	funall
no	710735.19	426154.41	379856.41	381249.81	731597	723917
su_rs	476344.59	352138.41	356979.19	302503.19	492522.41	485997.59
u2	426940.59	374594	292395.81	289037.41	452852.19	442168
u4	431396.19	366862.41	283079	282145	449495.19	438727.81
u8	424783	331208	332031.41	339461.41	444792.81	432680.59
u16	476161.59	338750.81	316785.59	325073	497372.41	483544
u32	478407.59	342074.81	363845.41	370274.19	494120.59	485624.81
u64	478783.59	343546.59	405802.41	404652.41	491075.59	488672
u128	480546.81	335636	429762.81	431853.81	491244	489836.81
u256	476368	340571.59	433664	433354.41	491378.19	494759.19

Figure 9. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.67	1.87	1.86
su_rs	1.35	1.33	1.57
u2	1.14	1.46	1.48
u4	1.18	1.52	1.53
u8	1.28	1.28	1.25
u16	1.41	1.50	1.46
u32	1.40	1.31	1.29
u64	1.39	1.18	1.18
u128	1.43	1.12	1.11
u256	1.40	1.10	1.10

Figure 9. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.12	0.94	1.22	1.05	1.09	1.10
u4	1.10	0.96	1.26	1.07	1.10	1.11
u8	1.12	1.06	1.08	0.89	1.11	1.12
u16	1.00	1.04	1.13	0.93	0.99	1.01
u32	1.00	1.03	0.98	0.82	1.00	1.00
u64	0.99	1.03	0.88	0.75	1.00	0.99
u128	0.99	1.05	0.83	0.70	1.00	0.99
u256	1.00	1.03	0.82	0.70	1.00	0.98

Figure 9. (c) Accélération pour l'unrolling (su_rs/u_n)

3.2 N=10000 et d=50%

	O0	O1	O2	O3	funroll	funall
no	745487.81	433090	386951.59	387786.81	711262.19	729557.38
su_rs	503821	354320.81	360673.59	301951.81	470967	488442.19
u2	467826.59	360542.81	295863.19	291537.59	427657.41	450664
u4	457794.41	347963	320061.81	288034.59	432631.59	442687.59
u8	454583.41	342438	328399.19	331845.19	425707	438002.41
u16	508794.59	344351	318405.59	328557.59	477466.59	495419.19
u32	508861.81	356691.41	370829.41	368021.41	479456.59	494347
u64	506834.59	350830	408239.41	410316.41	478497.81	495337.19
u128	507598	344151.59	431588.19	431510.81	476946	496608.19
U256	509299	352203	433352.19	432778.19	476587.41	497155.41

Figure 10. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.72	1.93	1.92
su_rs	1.42	1.40	1.67
u2	1.30	1.58	1.60
u4	1.32	1.43	1.59
u8	1.33	1.38	1.37
u16	1.48	1.60	1.55
u32	1.43	1.37	1.38
u64	1.44	1.24	1.24
u128	1.47	1.18	1.18
u256	1.45	1.18	1.18

Figure 10. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.08	0.98	1.22	1.04	1.10	1.08
u4	1.10	1.02	1.13	1.05	1.09	1.10
u8	1.11	1.03	1.10	0.91	1.11	1.12
u16	0.99	1.03	1.13	0.92	0.99	0.99
u32	0.99	0.99	0.97	0.82	0.98	0.99
u64	0.99	1.01	0.88	0.74	0.98	0.99
u128	0.99	1.03	0.84	0.70	0.99	0.98
u256	0.99	1.01	0.83	0.70	0.99	0.98

Figure 10. (c) Accélération pour l'unrolling (su_rs/u_n)

3.3 Sherman3, N=5005 et D=0.08%

	O0	O1	O2	O3	Funroll	funall
no	343891.59	205252.59	182784.8	183258.59	353629.41	343929
su_rs	224819	170078.41	172020	146182.8	236734	229318.41
u2	207151.41	163321.41	141062.8	139053	216323.8	207144.8
u4	208476.8	174041.2	135915.59	137774.8	213716	208314.8
u8	205255.41	160496.2	155048	154660.2	209679.41	205121.59
u16	230868.8	165395	154162.59	157538	235662	230632.8
u32	231748.2	164548	175982.59	176760.2	235740	231623.8
u64	232425.2	165167.41	197745.59	197070.59	236146.59	232268.59
u128	231353	164001.2	211414.8	210756	235590.59	231189.2
u256	230907.2	162301.59	212027	211400.2	235197.59	230384.59

Figure 11. (a) TCPU (µs)

	O0/O1	O0/O2	O0/O3
no	1.68	1.88	1.88
su_rs	1.32	1.31	1.54
u2	1.27	1.47	1.49
u4	1.20	1.53	1.51
u8	1.28	1.32	1.33
u16	1.40	1.50	1.47
u32	1.41	1.32	1.31
u64	1.41	1.18	1.18
u128	1.41	1.09	1.10
u256	1.42	1.09	1.09

Figure 11. (b) Accélération pour les options -On (O0/On)

	O0	O1	O2	O3	fun	funall
u2	1.09	1.04	1.22	1.05	1.09	1.11
u4	1.08	0.98	1.27	1.06	1.11	1.10
u8	1.10	1.06	1.11	0.95	1.13	1.12
u16	0.97	1.03	1.12	0.93	1.00	0.99
u32	0.97	1.03	0.98	0.83	1.00	0.99
u64	0.97	1.03	0.87	0.74	1.00	0.99
u128	0.97	1.04	0.81	0.69	1.00	0.99
u256	0.97	1.05	0.81	0.69	1.01	1.00

Figure 11. (c) Accélération pour l'unrolling su_rs/u_n)

	O0	O1	O2	O3	fun	funall
N=10000 et d=10%	1.49	1.21	1.06	1.26	1.49	1.49
N=10000 et d=50%	1.48	1.22	1.07	1.28	1.51	1.49
Sherman3	1.53	1.21	1.06	1.25	1.49	1.50

Figure 12. Accélération pour le remplacement scalaire (no/su_rs)

Etude de la distribution, sur système à grande échelle, de calcul numérique traitant des matrices creuses compressées

Résumé. Plusieurs applications scientifiques effectuent des calculs sur des matrices creuses de grandes tailles. Pour des raisons d'efficacité en temps et en espace lors du traitement de ces matrices, elles sont stockées selon des formats compressés adéquats. D'un autre coté, la plupart des calculs scientifiques creux se ramènent aux deux problèmes fondamentaux d'algèbre linéaire i.e. la résolution de systèmes linéaires et le calcul d'éléments (valeurs/vecteurs) propres de matrices. Nous étudions dans ce mémoire la distribution, au sein d'un Système Distribué à Grande Echelle (SDGE), des calculs dans des méthodes itératives de résolution de systèmes linéaires et de calcul d'éléments propres et ce, dans le cas creux. Le produit matrice-vecteur creux (PMVC) constitue le noyau de base pour la plupart de ces méthodes. Notre problématique se ramène en fait à l'étude de la distribution du PMVC sur un SDGE. Généralement, trois étapes sont nécessaires pour accomplir cette tâche, à savoir, (i) le prétraitement, (ii) le traitement et (iii) le post-traitement. Dans la première étape, nous procédons d'abord à l'optimisation de quatre versions de l'algorithme du PMVC correspondant à quatre formats de compression spécifiques de la matrice, puis étudions leurs performances sur des machines cibles séquentielles. Nous nous focalisons de plus sur l'étude de l'équilibrage des charges pour la distribution des données traitées (se ramenant en fait aux lignes de la matrice creuse) sur un SDGE. Concernant l'étape de traitement, elle a consisté à valider l'étude précédente par une série d'expérimentations réalisées sur une plate-forme gérée par l'intergiciel XtremWeb-CH. L'étape de post-traitement, quant à elle, a consisté à analyser et interpréter les résultats expérimentaux obtenus au niveau de l'étape précédente et ce, afin d'en tirer des conclusions adéquates.

Mots clés: équilibrage de charge, format de compression, matrice creuse, optimisation d'algorithme, produit matrice-vecteur creux, système distribué à grande échelle.

Study of the distribution on a large scale system of numerical computing processing compressed sparse matrices

Abstract. The treatment of sparse numerical problems on large scale systems is often reduced to that of their kernels. For reasons of efficiency in time and space, specific compressing formats are used for storing the matrices of these problems. Most of sparse scientific computations are led to linear algebra problems. Here two fundamental problems are often considered: linear systems resolution and eigenvalue computation. In this thesis, we address the study of distribution of computations performed in iterative methods to solve such problems. The sparse matrix-vector product (SMVP) constitutes a basic kernel in such iterative methods. Thus, our problem reduces to the study of SMVP distribution on large scale distributed systems. Three phases are required for achieving our objectives (i)- pre-processing, (ii)- processing and (iii)- post-processing. In phase 1, we first process the optimization of four versions of the SMVP algorithm corresponding to four specific matrix compressing formats; we study then their performances on sequential target machines. In addition, we focus on the study of load balancing in the procedure of data distribution (i.e. the sparse matrix rows) on a large scale distributed system. Concerning the processing phase, it consists in validating our study by a series of experimentations achieved on a volunteer distributed system that we installed through using XtremWeb-CH middleware. As to the post-processing phase, it consists in interpreting the experimental results previously obtained in order to deduce adequate conclusions.

Keywords: algorithm optimization, compressing format, large scale distributed system, load balancing, sparse matrix, sparse matrix-vector product.

دراسة توزيع، في إطار نظام واسع النطاق، لحساب رقميّ يعالج مصفوفات جوفاء مضغوطة

الملخص. تعتمد العديد من التطبيقات العلمية على حسابات تَستعمل مصفوفات جوفاء كبيرة الحجم. وسعيا وراء نجاعة معالجة تلك المصفوفات يقع اتباع أشكال مضغوطة مناسبة لخزنها. هذا وتُختصر معظم الحسابات العلمية الجوفاء في مشكلين جوهريين في الجبر الخطي و هما حل النظم الخطية و حساب العناصر (القيم/المتجهات) الذاتية للمصفوفات. نهتم في هذه الأطروحة بدراسة توزيع - داخل نظام واسع النطاق- لحسابٍ يُستعمل في طرق حل النظم الخطية و تحديد العناصر الذاتية و ذلك في إطار أجوف. تمثل عملية ضرب مصفوفة جوفاء في متجه نواة أساسية لمعظم هذه الطرق. يتمثل موضوع بحثنا إذن في دراسة توزيع عملية ضرب مصفوفة جوفاء في متجه على نظام موزع واسع النطاق. نعتمد في إنجاز ذلك على طريقة تشتمل على ثلاث مراحل وهي المعالجة الأولية، ثم المعا لجة ثم ما بعد المعالجة. نقوم في المرحلة الأولى بتحقيق أمثلية أربع صيغ لخوارزمية ضرب مصفوفة جوفاء في متجه ثم ندرس أداءها على حواسيب متسلسلة. من ناحية أخرى، نركز على دراسة توازن الحمولة عند توزيع المعطيات أي المصفوفة الجوفاء و ذلك على نظام موزع واسع النطاق. وفيما يتعلق بمرحلة المعالجة، تمثل عملنا في القيام بمجموعة من التجارب على نظام موزع واسع النطاق جهزناه باستعمال نظام التطوع XtremWeb-CH. أما فيما يخص مرحلة ما بعد المعالجة، فنقوم بتحليل النتائج التجريبية التي حصلنا عليها أثناء المرحلة السالفة و ذلك لاستخلاص استنتاجات مناسبة لتحقيق نجاعة الطريقة.

الكلمات المفاتيح : تحقيق أمثلية خوارزمية ، توازن الحمولة ، شكل مضغوط ، ضرب مصفوفة جوفاء في متجه ، مصفوفة جوفاء ، نظام مُوَزَّع واسع النطاق.