Auto-tuning pour la détection automatique du meilleur format de compression pour matrice creuse

Ichrak Mehrez

NNT : 2018SACLV054

Auto-tuning pour la détection automatique du meilleur format de compression pour matrice creuse

Thèse de doctorat de la Faculté des Sciences de l'Université de Tunis El-Manar et de l'Université Paris-Saclay, préparée à l'Université de Versailles Saint-Quentin en Yvelines

Ecole doctorale n°580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Tunisie, le 27 septembre 2018, par

Ichrak Mehrez

Composition du Jury :

Riadh ROBBANA Professeur, Université de Carthage, Tunisie	Président
Yousef SAAD Professeur, Université de Minnesota, ETATS-UNIS	Rapporteur
Fabienne JÉZEQUEL Maître de Conférence, Université de Sorbonne	Rapporteur
Nabil ABDENNADHER Associate professor, University of Applied Sciences, Suisse	Examinateur
Nahid EMAD Professeur, UVSQ	Directrice de thèse
Zaher MAHJOUB Professeur, Université de Tunis elManar, Tunisie	Co-Directeur de thèse
Christophe CALVIN Chercheur HDR, CEA, Paris-Saclay	Invité

Thèse de doctorat

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre : Auto-tuning pour la détection automatique du meilleur format de compression pour matrice creuse

Mots clés : Auto-tuning, cluster, data parallel, format de compression, machine learning, matrice creuse

Résumé : De nombreuses applications en calcul scientifique traitent des matrices creuses de grande taille ayant des structures régulières ou irrégulières. Pour réduire à la fois la complexité spatiale et la complexité temporelle du traitement, ces matrices nécessitent l'utilisation d'une structure particulière de stockage (ou de compression) des données ainsi que l'utilisation d'architectures cibles parallèles/distribuées. Le choix du format de compression le plus adéquat dépend généralement de plusieurs facteurs dont, en particulier, la structure de la matrice creuse, l'architecture de la plateforme cible et la méthode numérique utilisée. Etant donné la diversité de ces facteurs, un choix optimisé pour un ensemble de données d'entrée peut induire de mauvaises performances pour un autre. D'où l'intérêt d'utiliser un système permettant la sélection automatique du meilleur format de compression (MFC) en prenant en compte ces différents facteurs. C'est dans ce cadre précis que s'inscrit cette thèse. Nous détaillons notre approche en présentant la modélisation d'un système automatique qui, étant donnée une matrice creuse, une méthode numérique, un modèle de programmation parallèle et une architecture, permet de sélectionner automatiquement le MFC. Dans une première étape, nous validons notre modélisation par une étude de cas impliquant (i) Horner, et par la suite le produit matrice-vecteur creux (PMVC), comme méthodes numériques, (ii) CSC, CSR, ELL, et COO comme formats de compression, (iii) le data parallel comme modèle de programmation et (iv) une plateforme multi-cœurs comme architecture cible. Cette étude nous permet d'extraire un ensemble de métriques et paramètres qui influent sur la sélection du MFC. Nous démontrons que les métriques extraites de l'analyse du modèle data parallel ne suffisent pas pour prendre une décision (sélection du MFC). Par conséquent, nous définissons de nouvelles métriques impliquant le nombre d'opérations effectuées par la méthode numérique et le nombre d'accès à la mémoire. Ainsi, nous proposons un processus de décision prenant en compte à la fois l'analyse du modèle data parallel et l'analyse de l'algorithme. Dans une deuxième étape, et en se basant sur les données que nous avons extrait précédemment, nous utilisons les algorithmes du Machine Learning pour prédire le MFC d'une matrice creuse donnée. Une étude expérimentale ciblant une plateforme parallèle multi-cœurs et traitant des matrices creuses aléatoires et/ou provenant de problèmes réels permet de valider notre approche et d'évaluer ses performances. Comme travail futur, nous visons la validation de notre approche en utilisant d'autres plateformes parallèles telles que les GPUs.

Title: Auto-tuning for the automatic sparse matrix optimal compression format detection

Keywords: Auto-tuning, cluster, data parallel, compression format, machine learning, sparse matrix

Abstract : Several applications in scientific computing deals with large sparse matrices having regular or irregular structures. In order to reduce required memory space and computing time, these matrices require the use of a particular data storage structure as well as the use of parallel/distributed target architectures. The choice of the most appropriate compression format generally depends on several factors, such as matrix structure, numerical method and target architecture. Given the diversity of these factors, an optimized choice for one input data set will likely have poor performances on another. Hence the interest of using a system allowing the automatic selection of the Optimal Compression Format (OCF) by taking into account these different factors. This thesis is written in this context. We detail our approach by presenting a design of an auto-tuner system for OCF selection. Given a sparse matrix, a numerical method, a parallel programming model and an architecture, our system can automatically select the OCF.

In a first step, we validate our modeling by a case study that concerns (i) Horner scheme, and then the sparse matrix vector product (SMVP), as numerical methods, (ii) CSC, CSR, ELL, and COO as compression formats; (iii) data parallel as a programming model; and (iv) a multicore platform as target architecture. This study allows us to extract a set of metrics and parameters that affect the OCF selection. We note that data parallel metrics are not sufficient to accurately choose the most suitable format. Therefore, we define new metrics involving the number of operations and the number of indirect data access. Thus, we proposed a new decision process taking into account data parallel model analysis and algorithm analysis.

In the second step, we propose to use machine learning algorithm to predict the OCF for a given sparse matrix. An experimental study using a multicore parallel platform and dealing with random and/or real-world random matrices validates our approach and evaluates its performances. As future work, we aim to validate our approach using other parallel platforms such as GPUs.

Dédicaces

A mes chers parents

Leila et Abdelhak

Que ce travail qui vous est particulièrement dédié et qui représente l'aboutissement de vos prières et de vos encouragements soit le modeste témoignage de ma grande affection et mon profond attachement.

Que Dieu vous prête santé et longue vie afin que je puisse vous combler à mon tour.

A mon cher mari Walid

Aucun mot ne saurait exprimer ma gratitude et ma profonde reconnaissance.

Je te remercie pour ton soutient, tes encouragements, ta patience et ton sacrifice afin que je puisse réaliser ce travail.

Que Dieu te garde et nous réunisse pour toujours.

A mon petit prince Omar

J'espère que tu seras fière de ta maman.

Que Dieu te protège et te prête longue vie.

A ma sœur Fatma

A mon frère Mohamed

Je vous présente mes remerciements pour vos efforts remarquables au cours de ce travail

Je vous souhaite une vie pleine de succès et de réussite

Ichrak

Remerciements

A l'issue de ce travail, je suis convaincue que cette thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu la réaliser sans le soutien d'un grand nombre de personnes dont la générosité, la bonne humeur et l'intérêt manifestés à l'égard de mes recherches m'ont permis de progresser dans cette phase délicate.

Je tiens tout d'abord à remercier M. Riadh ROBBANA, Professeur à l'Institut National des Sciences Appliquées et de Technologie (Tunisie), qui m'a fait l'honneur de présider le jury de ma soutenance.

Je voudrais aussi remercier Mme Fabienne JÉZEQUEL, Maître de Conférence HDR à l'Université de Sorbonne (France), pour avoir bien voulu accepter de rapporter mon travail ainsi pour ses remarques qui m'ont permis d'améliorer ce manuscrit.

J'exprime mes profonds remerciements à M. Yousef SAAD, 'CSE Distinguished Professor' à l'Université de Minneapolis (USA), qui m'a honoré en acceptant de rapporter ce travail. Je le remercie infiniment pour les remarques et les recommandations qu'il m'a fourni afin de pouvoir approfondir mes travaux futurs.

Je tiens à remercier M. Nabil ABDENNADHER, Professeur ordinaire à l'Université des Sciences appliquées de Suisse qui m'a honoré en acceptant d'être membre examinateur dans ce travail.

Je tiens à remercier exceptionnellement ma directrice de thèse Mme Nahid EMAD, Professeur à l'Université de Versailles Saint-Quentin-en-Yvelines, pour sa patience et ses encouragements pour mener à bien ce travail. Je lui exprime ma profonde reconnaissance pour l'aide précieuse qu'elle m'a apportée.

Je voudrais, particulièrement, exprimer mes plus profonds remerciements à M. Zaher MAHJOUB, professeur à la Faculté des Sciences de Tunis, d'avoir dirigé cette thèse, ainsi que pour m'avoir accueilli au sein de son unité de recherche. Les mots ne seront jamais assez explicites pour décrire ses qualités humaines, son sens de l'écoute et de compréhension.

Je tiens à remercier chaleureusement Mme Olfa HAMDI-LARBI, maître assistante à l'Institut Supérieur d'Informatique (Tunisie), pour son encadrement, sa disponibilité, sa sympathie, ainsi que pour son aide précieuse de tous les jours. Tout au long de ce travail de thèse, elle a su orienter mes recherches aux bons moments, en me donnant des conseils judicieux et des idées innovantes.

Je souhaite aussi exprimer ma gratitude à M. Thomas DUFAUD, maître de conférence à l'Université de Versailles Saint-Quentin-en-Yvelines pour sa collaboration tout au long de cette thèse. Je le remercie infiniment pour son aide, ses remarques et ses encouragements tout au long de la réalisation de ce travail. Je lui exprime ma gratitude pour ses qualités humaines et son aide lors de mes séjours en France.

Un grand Merci à tous membres de l'unité URAPOP pour leur bonne humeur et leur attitude à mon égard. Je remercie particulièrement Sana, Bchira, Hajer et khawla pour leurs soutiens et pour les bons moments passés ensemble. Je n'oublierai pas aussi de remercier Mme Yosr et Mme Anissa pour leurs encouragements. Je remercie également Hatem pour ses encouragements et je lui souhaite bonne continuation.

Je réserve un remerciement chaleureux à M. Zarouk pour l'ambiance qu'il maintien au sein du département informatique de la Faculté des Sciences de Tunis.

Je tiens à remercier Mme Isabelle JAUNY, Directrice du Soutien à la Recherche et aux études doctorales à l'UVSQ et Mme Bénédicte De FERRON, Gestionnaire Ecole Doctorale STIC, pour leurs aides et leurs patiences.

Un grand merci à tante Zohra, George, Atfa et Mohamed qui m'ont accueilli durant mes premiers séjours en France.

Mes dernières pensées iront vers ma famille, et surtout mes parents, qui m'auront permis de poursuivre mes études jusqu'à aujourd'hui.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

INTRODUCTION GENERALE

En calcul numérique, une matrice est qualifiée de $creuse$ ($sparse$ en anglais) si elle comporte une faible proportion d'éléments non nuls. Cependant, une matrice peut être qualifiée de creuse dès qu'on peut tirer parti de son grand nombre d'éléments nuls et de leurs emplacements afin d'appliquer des techniques spéciales permettant l'optimisation des performances des noyaux de calcul traitant ce type de matrices.
L'utilisation d'une structure adaptée de stockage de matrices creuses devient une nécessité afin d'exploiter leurs natures. Ces structures sont appelées formats de compression ou encore formats de stockage de matrices creuses. L'utilisation du bon format de compression permet de réduire considérablement l'espace mémoire et d'optimiser les performances de calcul. En effet, ces structures de données permettent de stocker que les éléments non nuls et par la suite éviter les opérations inutiles sur les zéros. Ainsi, le choix du meilleur format de compression (MFC) est une étape très importante afin d'avoir les meilleures performances. Suite à un ensemble de tests (différentes matrices creuses et différentes architectures) [MHD18-b], la perte moyenne des performances (PMP) dans le cas où on décide d'utiliser le même format (CSR, CSC, COO ou ELL) pour toutes les matrices, au lieu d'utiliser le MFC correspondant varie entre 27% et 84.96% [MHD18-b]. Ces tests ont été réalisés dans le cas du calcul du Produit Matrice Vecteur Creux (PMVC).

Le choix du format de compression le plus adéquat est un processus critique qui dépend de plusieurs facteurs. Un de ces facteurs est la structure de la matrice creuse qui peut être régulière ou non (selon la répartition de ses éléments non nuls). Une matrice creuse a une structure régulière si ses éléments non nuls sont regroupés dans la matrice selon un motif particulier. On cite comme exemple les matrices triangulaires et les matrices diagonales. Cependant, une matrice creuse a une structure irrégulière si ses éléments non nuls sont éparpillés d'une manière aléatoire. Pour chaque structure de matrice, un ou plusieurs formats de compression ont été proposés dans la littérature.

Le concept de matrice creuse est très utilisé dans de nombreuses applications en calcul scientifique telles que la simulation de la mécanique des structures, le traitement d'image ou l'étude de la dynamique des fluides [Ham10]. La plupart de ces applications reviennent à des

problèmes de l'algèbre linéaire creuse dont les deux problèmes fondamentaux sont la résolution des systèmes linéaires et le calcul des valeurs propres. Contrairement au problème de calcul de valeurs propres où on ne peut utiliser que des méthodes itératives, les méthodes de résolution des systèmes linéaires peuvent être directes ou itératives. Cependant, dans le cas creux il est plus intéressant d'utiliser les méthodes itératives puisqu'elles conservent bien le caractère creux des matrices traitées. Ces méthodes itératives reposent sur le noyau de calcul du produit matrice vecteur creux. Les méthodes itératives génèrent une suite de solutions approchées qui convergent sous certaines conditions, vers la solution exacte. Pendant le calcul, la densité de la matrice reste intacte. Par conséquent, la méthode numérique utilisée pour la résolution d'un système linéaire creux peut affecter le choix du format de compression à utiliser.

Le traitement de matrices creuses de grandes tailles par des méthodes itératives exige également l'utilisation de machines de haute performance qui, souvent, peuvent ne pas être disponibles. Dans le cas de certaines applications scientifiques, les systèmes distribués, tels que les grappes de calcul, peuvent constituer une bonne alternative. Plusieurs caractéristiques des architectures parallèles telles que l'espace mémoire, la hiérarchie de la mémoire, etc. peuvent affecter le choix du format de compression. En effet, l'ordre dans lequel les éléments non nuls sont stockés, puis accédés, est directement affecté par l'architecture utilisée.
L'expression d'un algorithme parallèle est aussi liée au modèle de programmation parallèle. Ce modèle décrit les composants/tâches de l'algorithme parallèle, leurs interactions et leurs effets sur l'état de calcul. Il existe deux principaux modèles de programmation parallèle : le parallélisme de tâche ou de contrôle et le parallélisme des données. Dans le premier cas, le parallélisme résulte de l'application de différentes tâches sur les données. Cependant, le parallélisme de données est défini par l'application d'un flot unique d'instructions sur un ensemble de données différentes.

Ainsi, le choix du format de compression le plus approprié est un processus critique qui dépend de plusieurs facteurs, à savoir, la matrice creuse, la méthode numérique, l'architecture et le modèle de programmation parallèle ciblés. Notre problématique consiste à prendre en compte tous ces facteurs afin de trouver le meilleur format de compression.

L'étude de différents formats de compression et de leurs impacts sur les calculs creux ont fait l'objet de nombreux projets de recherche. Certains travaux se concentrent généralement sur

l'optimisation des formats de compression existants ou sur la proposition de nouveaux formats [YzB11] [Bis04] [KGK08] [ShU11]. Ces études ont été réalisées pour une architecture bien définie et afin d'optimiser un noyau de calcul spécifique (PMVC dans la plupart des cas).
Des travaux plus récents ([LBE16] [GuL16] [NRR14] [BJW16-a] [SMP15]) se concentrent généralement sur l'étude de la sélection du MFC pour les architectures GPU. Ils utilisent des techniques de l'apprentissage automatique pour prédire le temps d'exécution du noyau de calcul, et par la suite choisir le MFC. Différents algorithmes d'apprentissage sont utilisés dans ces travaux : les arbres de décision ([SMP15] [LZT12]), KNN [LBE16], les machines à support de vecteurs [BJW16-a] et le perceptron multicouche [BJW16-a]. Ces travaux ne prennent pas en compte les caractéristiques de la méthode ou de l'algorithme numérique dans le processus de sélection.

Nous proposons dans cette thèse la conception et la mise en œuvre d'un système qui, étant donnés une matrice creuse, une méthode numérique, un modèle de programmation parallèle et une architecture, peut automatiquement prédire le meilleur format de compression à utiliser. Pour cela, nous adoptons une approche orientée composants capable de définir un système extensible. Ainsi, d'autres composantes (fonctionnalités) peuvent être ajoutées progressivement à notre système, sans affecter ses principes de base. On remarque que les méthodes actuelles reposent sur une analyse des données de la matrice ou du noyau de calcul uniquement. Nous étudions l'introduction de l'expertise de l'utilisateur pour guider le processus de décision en fournissant des informations sur le domaine d'application ainsi que sur les environnements de programmation et d'exécution [MHD16]. Nous proposons d'appliquer notre approche au schéma de Horner et au noyau de calcul le produit matrice-vecteur creux.

L'analyse statique d'un programme peut ne pas rendre compte des mouvements des données lors de l'utilisation des architectures parallèles et/ou distribuées. La prise en compte d'une analyse du modèle de programmation ciblé pour l'algorithme considéré peut remédier à ce problème. En effet, l'intégration de l'expertise de l'utilisateur consiste en grande partie à insérer ces paramètres, connus de celui-ci, dans ce système. Ainsi, nous avons étudié cette approche et présentons notre analyse qui permet de guider dynamiquement la décision de l'utilisateur lors de son choix pour le format de compression.

Notre démarche consiste à l'analyse du modèle de programmation data parallel et l'analyse de l'algorithme. Cette analyse nous permet d'extraire un ensemble de métriques permettant de limiter les paramètres affectants le choix du meilleur format de compression.

Notre approche est illustrée par la suite sur des cas de tests proches de l'étude théorique afin de valider nos choix. Ces cas de tests correspondent aux cas extrêmes détectés lors de l'étude théorique du modèle data parallel pour un format donné [MHD16] [MHD17] [MHD18-a].

D'un autre côté, nous remarquons que le problème de la sélection du meilleur format de compression de matrice creuse peut être formulé comme un problème de classification où chaque format représente une classe. Les algorithmes du Machine Learning peuvent donc être employés pour servir à la résolution de ce problème. Afin de pouvoir tirer profit de l'étude théorique et de l'expertise utilisateur, les métriques extraites sont alors utilisées pour définir un ensemble d'attributs qui serviront comme entrée à l'algorithme d'apprentissage. L'évaluation des performances de notre classificateur montre que notre système peut atteindre une valeur de précision de l'ordre de 95.65% [MHD18-b].

Le présent manuscrit est structuré en cinq chapitres organisés comme suit :

Le premier chapitre est consacré aux définitions des concepts de base du calcul creux. Il s'agit d'une présentation des matrices creuses, leurs structures et, les formats de compression existants dans la littérature. Nous présentons aussi certaines applications réelles traitant ce type de matrices. Ces dernières reviennent généralement au calcul du produit matrice-vecteur creux. La deuxième partie de ce chapitre est consacrée à la présentation des architectures parallèles et distribuées ainsi qu'aux modèles de programmation associés.

Le deuxième chapitre est dédié à la présentation d'un état de l'art sur la détection du meilleur format de compression pour matrice creuse. Nous exposons par la suite notre solution pour la résolution de ce problème. Une conception détaillée du système intelligent proposé est ensuite présentée à la fin de ce chapitre.

Dans le troisième chapitre, nous nous intéressons à l'étude du choix du meilleur format de compression en prenant en compte le modèle de programmation data parallel. Nous définissons ce modèle de programmation et les différentes métriques d'évaluation associées.

Nous présentons par la suite notre approche qui consiste, pour chaque format de compression considéré, à étudier la méthode de Horner parallélisée selon le modèle de programmation *data parallel*. Dans la dernière partie de ce chapitre, une analyse des algorithmes est effectuée afin de compléter la liste des paramètres qui peuvent influer sur le processus de sélection du meilleur format de compression.

Le quatrième chapitre est consacré à la validation de l'approche proposée dans le cas des architectures multiprocesseurs. Deux études détaillées sur des matrices pathologiques permettent de valider nos choix pour les métriques de performance.

Dans le cinquième chapitre, nous présentons un état de l'art sur les algorithmes de machine learning. Les différentes étapes nécessaires pour la mise en œuvre de notre système d'apprentissage sont alors présentées. La dernière partie de ce chapitre est dévolue à la présentation des résultats expérimentaux concernant l'évaluation des performances de notre système.

Nous clôturons le présent manuscrit par une conclusion résumant nos principaux résultats et présentant quelques perspectives de notre travail.

CONCEPTS DE BASE ET MOTIVATIONS

Introduction

Beaucoup d'applications en calcul scientifique telles que la simulation numérique se ramènent à des problèmes de l'algèbre linéaire où le calcul de valeurs propres et la résolution de systèmes linéaires constituent deux problèmes fondamentaux. Ces algorithmes traitent des matrices creuses de grandes tailles. Dans ce cas, on peut tirer profit de la nature de ces matrices en stockant seulement ses éléments non nuls. Ainsi, nous obtenons un gain en espace mémoire et en temps de calcul.

Les performances d'un algorithme creux peuvent dépendre fortement du format de stockage choisi pour la matrice creuse. Plusieurs formats de stockage ont été proposés dans la littérature. Le choix du format le plus adéquat est un processus critique qui dépend de plusieurs facteurs.

Nous commençons ce chapitre par la présentation des matrices creuses et de leur structure. Nous exposons alors quelques formats représentatifs de compression proposés dans la littérature pour le stockage de ce type de matrices. Après avoir décrit quelques applications qui traitent des matrices creuses, nous passons en revue quelques systèmes parallèles et distribués cibles. Nous détaillons par la suite les modèles ainsi que les bibliothèques de programmation parallèles.

Calcul creux

Dans cette section nous présentons les matrices creuses et quelques formats de compression proposés dans la littérature. Nous détaillons par la suite les principaux algorithmes traitant ce type de matrice à savoir le produit matrice-vecteur creux et le schéma de Horner.

Matrices creuses et formats de compression

Définition de matrice creuse

Soit A une matrice réelle, $A \in \mathbb{R}^{n \times m}$, où n correspond au nombre de lignes et m au nombre de colonnes. Soit nnz le nombre d'éléments non nuls de A. La matrice A est dite $creuse$ ($sparse$ en anglais) si nnz est petit ($nnz = O(n)$). Dans le cas où nnz est grand ($O(n^2)$), A est dite $dense$[HME07].

Nous visons les matrices creuses de grande taille ayant un nombre faible d'éléments non nuls par rapport à celui des éléments nuls [EHM06-a]. On définit d la densité d'une matrice creuse avec :

$$d = \frac{n \times m}{nnz}$$

Structures de matrices creuses

La structure d'une matrice creuse dépend de la distribution de ses éléments non nuls. On distingue deux structures particulières : structures régulières et structures irrégulières.

Structures régulières

Une matrice creuse possède une structure régulière si tous les éléments non nuls sont organisés dans la matrice de telle manière qu'on peut y accéder facilement [EHM06-a]. On cite en particulier :

Matrices triangulaires : Soit *A* une matrice carrée ($A \in \mathbb{R}^{n\times n}$). *A* est dite triangulaire supérieure si :

$$a_{ij} = 0 \; \forall \; i > j$$ (*Figure* ***Error! No text of specified style in document.****-1*)

Matrices bande : une matrice creuse *A* est dite bande si ses éléments non nuls sont regroupés autour de la diagonale [ALR] (*Figure* ***Error! No text of specified style in document.****-2*).

Matrices Hessenberg: une matrice creuse $A \in \mathbb{R}^{n\times n}$ est dite de Hessenberg supérieure si tous les éléments en dessous de sa première sous-diagonale sont nuls (Figure Error*!* ***No text of specified style in document.****-3*) :

$$a_{ij} = 0 \; \forall \; i > j + 1$$

Matrices par bloc : une matrice est dite par blocs ses éléments peuvent être regroupés en un ensemble de sous matrices de dimensions inférieures. (Figure Error*!* ***No text of specified style in document.****-4*).

Structures irrégulières

Une matrice creuse possède une structure irrégulière si ses éléments non nuls sont distribués dans la matrice d'une manière irrégulière [EHM06-a]. On cite comme exemple :

(i) Matrice bande variable: Soit *A* une matrice carrée ($A \in \mathbb{R}^{n\times n}$). Soient l_i et u_j pour $1 \leq i \leq n, 1 \leq j \leq n$ définis par :

$$l_i = i - \; min\{\frac{j}{a_{ij}} \neq 0\}$$

$$u_i = j - \; min\{\frac{i}{a_{ij}} \neq 0\}$$

Chaque l_i et u_j indique respectivement la largeur de demi-bande inférieure et supérieure dans la ligne i et la colonne j :

$$a_{ij} \neq 0 \; ; (-u_j \leq \; i - j \; \leq l_i)$$

La forme bande variable d'une matrice est définie par les vecteurs $u = (u_i)\; et\; l = (l_i)$, avec u et $l \in \; \mathbb{N}^n$ (*Figure* ***Error! No text of specified style in document.****-5*).

Matrice à structure aléatoire : une matrice creuse est dite à structure aléatoire si ses éléments non nuls sont éparpillés de manière quelconque et ne forment pas une structure particulière (*Figure* ***Error! No text of specified style in document.****-6*).

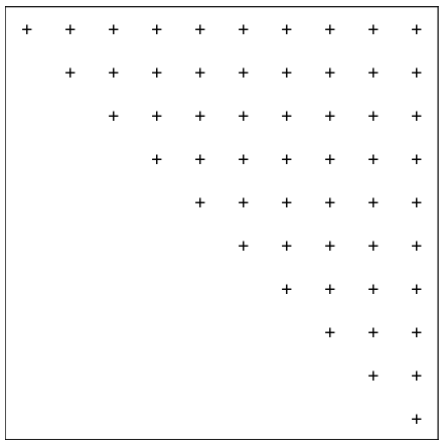

Figure ***Error! No text of specified style in document.****-1 Matrice triangulaire supérieure*

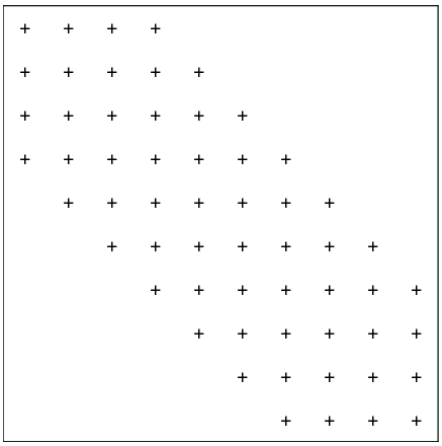

Figure ***Error! No text of specified style in document.****-2 Matrice bande*

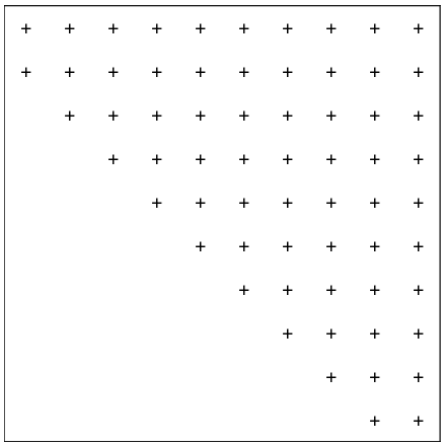

Figure ***Error! No text of specified style in document.****-3 Matrice Hessenberg supérieure*

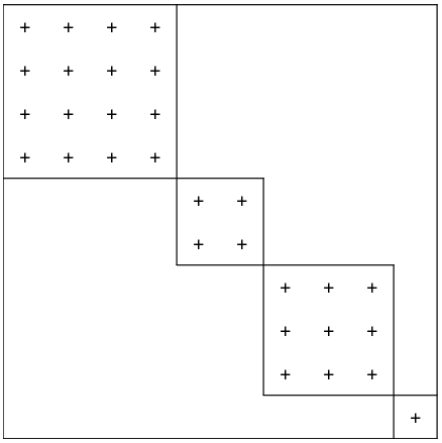

Figure ***Error! No text of specified style in document.****-4 Matrice par bloc*

Formats de compression de base

Pour des raisons d'espace mémoire et de performances de calcul, l'utilisateur est obligé d'exploiter la nature creuse de ces matrices et d'utiliser un format de stockage des éléments non

nuls uniquement. Les performances d'un algorithme creux peuvent dépendre fortement du choix du format de stockage pour la matrice creuse [ShU07]. Ainsi, l'étude de différents formats de stockage compressés et leurs impacts sur le calcul parallèle creux, en particulier le produit

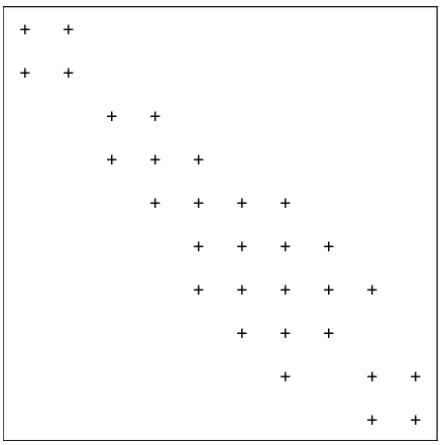

*Figure **Error! No text of specified style in document.**-5 Matrice bande variable*

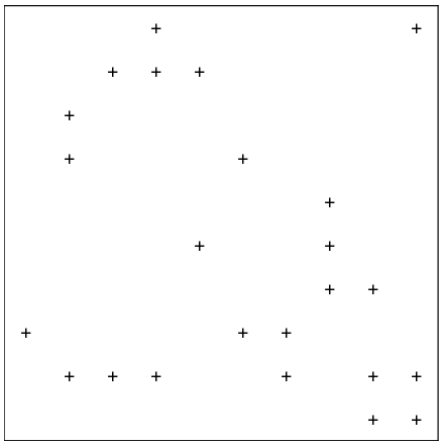

*Figure **Error! No text of specified style in document.**-6 Matrice quelconque*

matrice-vecteur creux (PMVC), est le sujet de plusieurs travaux de recherche. Il existe une grande variété de formats de stockage de matrices creuses. Nous présentons ici quelques formats représentatifs proposés dans la littérature.

a) <u>COOrdinate</u>

Le format COO (Coordinate) est le plus simple mode de stockage pour matrices creuses [Saa03]. Il est constitué de trois tableaux (*Figure **Error! No text of specified style in document.**-7*) :

- Un tableau de réel val, de taille nnz, contenant les éléments non nuls de la matrice A,
- Un tableau d'entiers ind_ligne, de taille nnz, contenant les indices de ligne des éléments de A stockés dans val,
- Un troisième tableau d'entiers ind_col, de taille nnz, contenant les indices de colonne des éléments de A stockés dans val.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 9 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 2 \\ 0 & 7 & 3 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \end{pmatrix}$$

val	1	9	4	6	5	2	7	3	8
ind_ligne	1	1	2	3	4	4	5	5	6
ind_col	1	5	2	4	1	6	2	3	2

*Figure **Error! No text of specified style in document.**-7: Format de compression COO*

Compressed Sparse Row

Le format CSR (Compressed Sparse Row) est constitué de trois tableaux (*Figure* ***Error! No text of specified style in document.****-8*) :

- Un tableau de réels val de taille nnz. Ce tableau contient les éléments non nuls de la matrice A stockés ligne par ligne,
- Un tableau d'entiers ind_col de taille nnz. Il contient les indices de colonne des éléments stockés dans le tableau val.
- Un tableau d'entiers $ligne\_ptr$ de taille $(n+1)$. Il contient les pointeurs sur le début de chaque ligne dans les tableaux val et ind_col. Ainsi, $ligne\_ptr(i)$ donne la position dans val et ind_col là où commence la ligne i de A [Saa03].
  Par convention, $ligne\_ptr[n+1] = ligne\_ptr[1] + nnz$.

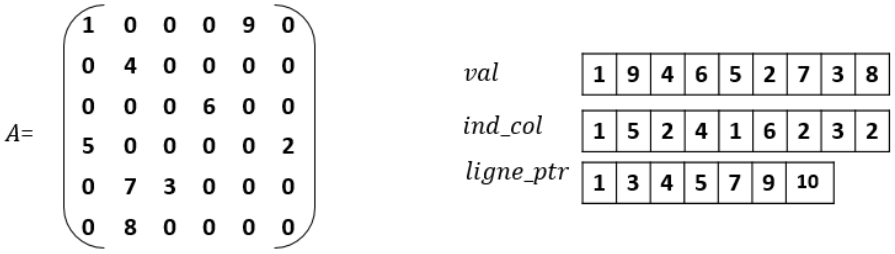

Figure ***Error! No text of specified style in document.****-8: Format de compression CSR*

Compressed Sparse Column

Le format CSC (Compressed Sparse Column) est similaire au format CSR. Néanmoins, dans ce cas, on stocke des colonnes [Saa03]. Le format CSC utilise trois tableaux (*Figure* ***Error! No text of specified style in document.****-9*) :

- Un tableau de réels val de taille nnz. Ce tableau contient les éléments non nuls de la matrice A stockés colonne par colonne,
- Un tableau d'entiers ind_ligne de taille nnz. Il contient les indices de ligne des éléments stockés dans le tableau val.
- Un tableau d'entiers col_ptr de taille $(n+1)$. Il contient les pointeurs sur le début de chaque colonne dans les tableaux val et ind_ligne. Ainsi, $col\_ptr(i)$ donne la position dans val et ind_col là où commence la colonne i de A.
  Par convention, $col\_ptr[n+1] = col\_ptr[1] + nnz$.

ELLPACK/ITPACK

Le format ELL (ELLPACK/ITPACK) est utilisé pour stocker une matrice creuse avec compression de ligne ou de colonnes. Nous présentons dans ce qui suit le format ELL avec compression de lignes. ELL est constitué de deux matrices (*Figure* ***Error! No text of specified style in document.****-10*) :

- Une matrice de réels val de dimension $n \times nz_{maxR}$, avec nz_{maxR} est le nombre maximal d'éléments non nul par ligne. La chaque $i^{ème}$ ligne de val contient la $i^{ème}$ ligne de A compressée. Les éléments supplémentaires sont complétés par des zéros.
- Une matrice ind, de dimension $n \times nz_{maxR}$, contenant les indices des colonnes de chaque élément dans val.

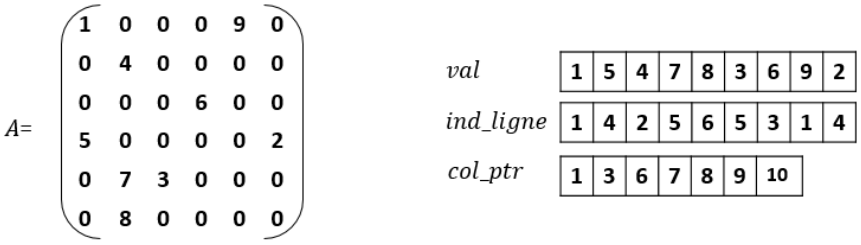

Figure ***Error! No text of specified style in document.****-9 Format de compression CSC*

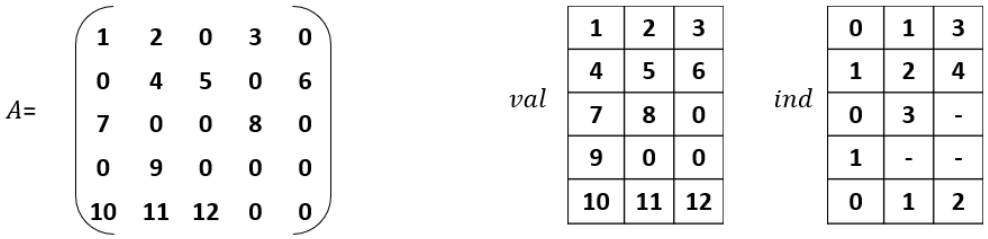

Figure ***Error! No text of specified style in document.****-10 Format de compression ELL*

DIA

Le format *DIA* est utilisé pour compresser des matrices creuses dont les éléments non nuls sont situés sur un petit nombre de diagonales Nd. Le format *DIA* utilise deux matrices (*Figure* ***Error! No text of specified style in document.****-11*):

- La matrice $DIAG$, de taille $n \times Nd$, pour stocker les diagonales de la matrice,
- La matrice $IOFF$, de taille Nd, pour stocker l'offset de chaque diagonale dans $DIAG$ (par rapport à la diagonale principale). Ainsi, un élément $a_{ii} + IOFF(j)$ de la matrice creuse A est stocké dans la position (i, j) de $DIAG$.

$$DIAG(i,j) = a_{ii} + IOFF(j)$$

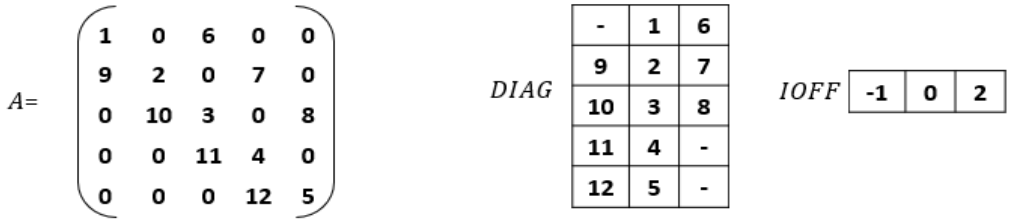

Figure ***Error! No text of specified style in document.****-11 Format de compression DIA*

Jagged Diagonals Storage

Le format JDS (Jagged Diagonals Storage) utilise quatre tableaux pour stocker une version modifiée d'une matrice creuse (*Figure* ***Error! No text of specified style in document.****-12*). En première étape les lignes et les colonnes de la matrice sont permutées de telle sorte que les lignes de la matrice soient triées par ordre décroissant de nombre d'éléments non nuls. Puis, dans chaque ligne, les éléments non nuls sont décalés à gauche. Les colonnes résultats, avec un nombre décroissant d'éléments non nuls, sont appelées *'jagged diagonals'* [SGF09] [EkM03].

- Un tableau de réels val, de taille nnz, pour stocker les éléments non nuls (colonne par colonne) de la nouvelle version de la matrice creuse.
- Un tableau d'entiers ind_col , de taille nnz, contenant les indices de colonne de chaque élément non nul correspondant dans val.
- Un troisième tableau d'entiers jd_ptr , de taille $nj + 1$ contenant les indices des débuts des *'jagged diagonals'* dans val et ind_col. nj est le nombre de '*jagged diagonals*'. Par convention, $jd\_ptr[nj + 1] = nnz$,
- Un tableau $permu$ contenant la position d'origine de chaque ligne dans la matrice triée.

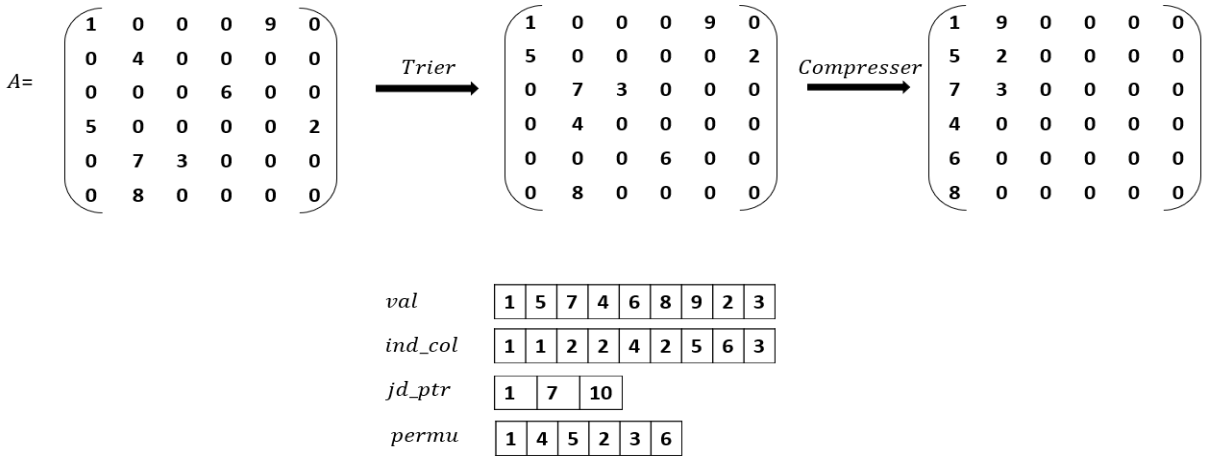

Figure ***Error! No text of specified style in document.****-12: Format de compression JDS*

Modified Sparse Row

Le format de compression MSR (Modified Sparse Row) utilise deux tableaux (*Figure* ***Error! No text of specified style in document.****-13*) :

- Un tableau de réels val, de taille $nnz + 1$, contenant les éléments non nuls de la matrice. Nous commençons par stocker les éléments de la diagonale de la matrice A dans les n premières positions de val. Le reste des éléments non nuls de la matrice sont stockés dans val, ligne par ligne, à partir de la position $n + 2$. La position $n + 1$ de val n'est pas utilisée.
- Un tableau d'entiers JA, de taille $nnz + 1$. Les $n + 1$ premiers éléments de JA contiennent les indices de début de chaque ligne dans val [FKA09]. A partir de la position $n + 2$, JA contient les indices des colonnes des éléments correspondants dans val.

  Notons que, dans l'exemple présenté dans la Figure Error! ***No text of specified style in document.****-13*, $JA(n) = JA(n + 1) = 14$. Ce ci indique que, une fois l'élément diagonal supprimé, tous les autres éléments de la dernière ligne la matrice sont nuls ($a_{ni} = 0 \ ; \forall \ i \neq n$).

La version colonne du format MSR et appelé MSC (Modified Sparse Column).

$A=$

1	0	0	2	0
0	3	4	5	6
0	0	7	8	9
0	0	0	11	10
0	0	0	0	12

val	1	3	7	11	12	-	2	4	5	6	8	9	10
JA	7	8	11	13	14	14	4	3	4	5	4	5	5

Figure ***Error! No text of specified style in document.****-13: Format de compression MSR*

Formats de compression dérivés

Nous présentons dans cette section quelques formats de compression dérivés existant dans la littérature. Ces formats sont soit le résultat de l'optimisation de l'un des formats de base ou encore la combinaison de plusieurs d'entre eux.

b) CSR-ZZ

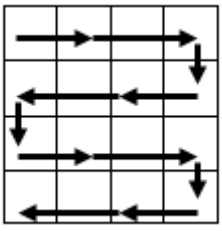

Figure ***Error! No text of specified style in document.****-14: Format de compression CSR-ZZ*

Grace à sa simplicité et son efficacité, le format CSR est le format le plus utilisé dans la plupart des travaux de parallélisation de PMVC [AKA12] [SrS11] [Jai09] [LeE08] [WOV07]. De plus, il a été prouvé que le format CSR est très adapté aux processeurs multi-cœurs modernes [SFH11]. Cependant, plusieurs variantes de ce format ont été proposées. Parmi ces variantes on trouve le *zig-zag* CSR (noté CSR-ZZ ou ZZCSR) proposé pour la minimisation des défauts de cache (*cache misses*) (voir Figure Error! ***No text of specified style in document.****-14*). Dans CSR-ZZ les éléments non nuls des lignes paires sont stockés dans l'ordre croissant de leurs indices de colonne alors que les éléments non nuls des lignes impaires sont stockés dans l'ordre décroissant de leurs indices de colonne [AKA12] [YzB09] [YzB11].

CSRI

Une autre variante du format CSR est le CSRI (CSR Incrémentale) [AKA12][KoJ02][YzB09] [YzB11]. Dans CSRI les éléments non nuls d'une ligne sont stockés dans l'ordre croissant de leurs indices de colonne dans un tableau val. La position $(i + j)$ d'un élément a_{ij} est enregistrée dans un tableau d'indices $indices$. La différence de chaque indice de ce tableau et l'élément non nul précédent, appelée $incrément$, est stockée dans un tableau inc.

CSR-DU

CSR-DU (CSR Delta Unit) est une variante du format CSR. Dans ce cas, les structures de données correspondant aux indices seront compressées. Le format CSR-DU se base sur le codage du tableau ind_col qui contient les indices de colonne du format CSR. Pour cela, on utilise le codage $delta$. Un $delta$ est défini comme étant la différence entre l'indice courant et l'indice précédent dans ind_col. Il est positif et inférieur ou égal à la valeur de l'indice courant. Une fois codé, le tableau ind_col 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 [KGK08].

CSR-VI

Une autre variante du format CSR, est le format CSR-VI (CSR Value Indexed). Le principe de ce dernier consiste à compresser le vecteur val du format CSR en éliminant les redondances des éléments non nuls. Ce vecteur est donc remplacé par deux autres, à savoir, val_unique et val_ind. Le premier tableau stocke les éléments non nuls de la matrice creuse sans redondance

et le deuxième contient l'indice de chaque élément non nul dans le tableau val_unique [KGK08].

BCSR

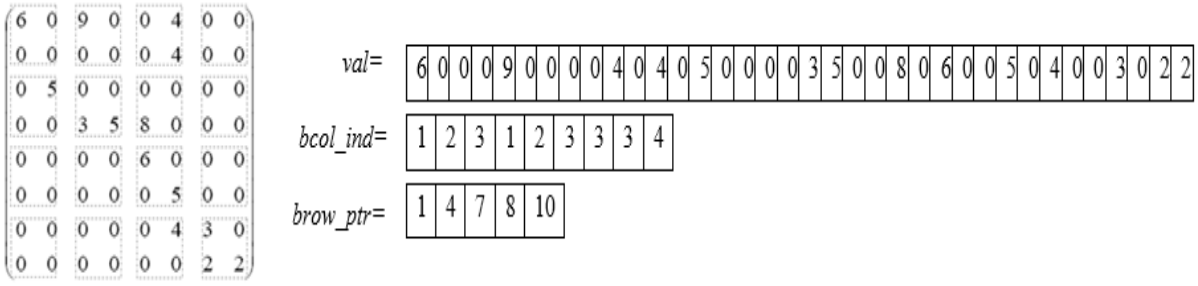

*Figure **Error! No text of specified style in document.**-15 Format BCSR avec des blocs de tailles (2×2)*

Dans le format Blocked CSR (BCSR), la matrice est divisée en sous-matrices appelés blocs de tailles $br \times bc$, où br et bc sont données [Saa94]. BCRS est représenté par trois tableaux val, $bcol\_ind$ et $brow\_ptr$ (Figure Error! ***No text of specified style in document.**-15*). Le tableau val, de longueur $bnze \times br \times bc$, contient les éléments des blocs non nuls, avec $bnze$ est le nombre des blocs non nuls dans la matrice. $bcol\_ind$, de longueur $bnze$, contient les indices des colonnes des blocs non nuls. Le tableau $brow\_ptr$, de longeur $brows + 1$, contient des pointeurs sur les indices des débuts de chaque bloc dans $bcol\_ind$ ($brows = \frac{n}{br}$).

SBCRS

Un autre format dénommé SBCRS (Sparse Block CSR) est présenté dans [ShU11] pour améliorer les localités temporelles et spatiales du format CSR. Dans SBCRS la matrice est divisée à des blocs supposés non nuls de taille ($S \times S$), quatre tableaux sont utilisés pour le stockage de la matrice. $Block\_array$ contient les éléments non nuls de la matrice ainsi que leurs positions (position ligne et position colonne de chaque élément dans le bloc auquel il appartient). Un deuxième tableau $Block\_ptr$ contient des pointeurs vers les blocs non nuls. Les indices des colonnes de ces blocs sont stockés dans $Bcol\_ind$, les pointeurs vers le début de chaque bloc dans $Bcol\_ind$ se trouvent dans le tableau $Brow\_ptr$. Figure Error! ***No text of specified style in document.**-16* illustre un exemple de compression d'une matrice creuse sous format SBCRS avec des blocs de tailles (2×2).

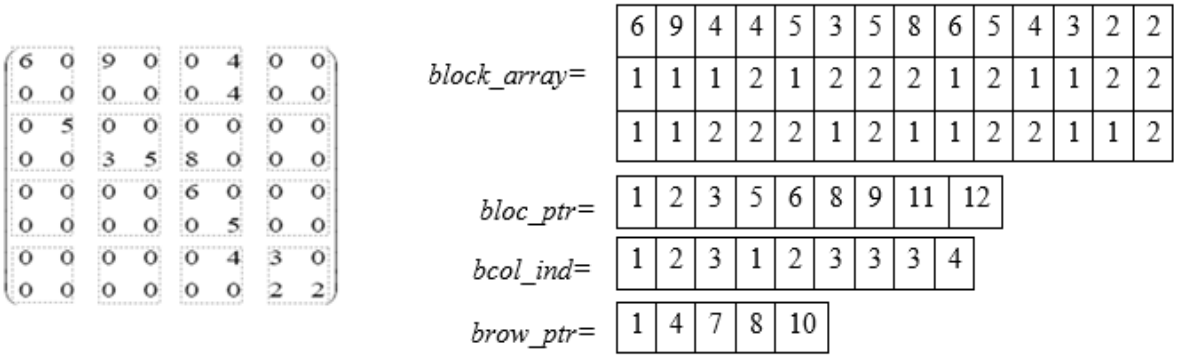

*Figure **Error! No text of specified style in document.**-16: Format de compression SBCSR avec des blocs de tailles (2×2)*

HYB

Le format HYB (HYBrid) est la combinaison des deux formats ELL et COO. Il est conçu afin de bénéficier des avantages de ces deux formats. Bien que le format ELL est bien adapté aux architectures vectorielles et SIMD, son efficacité se dégrade rapidement lorsque le nombre des éléments non nuls par ligne varie. En revanche, l'efficacité de stockage du format COO est invariante à la distribution des éléments non nuls [BeG09].

Dans HYB, la matrice est partitionnée en deux parties : une partie régulière dont les éléments non nuls sont stockés sous ELL et une deuxième partie irrégulière dont les éléments non nuls sont compressés en utilisant COO (Figure Error! ***No text of specified style in document.**-17*).

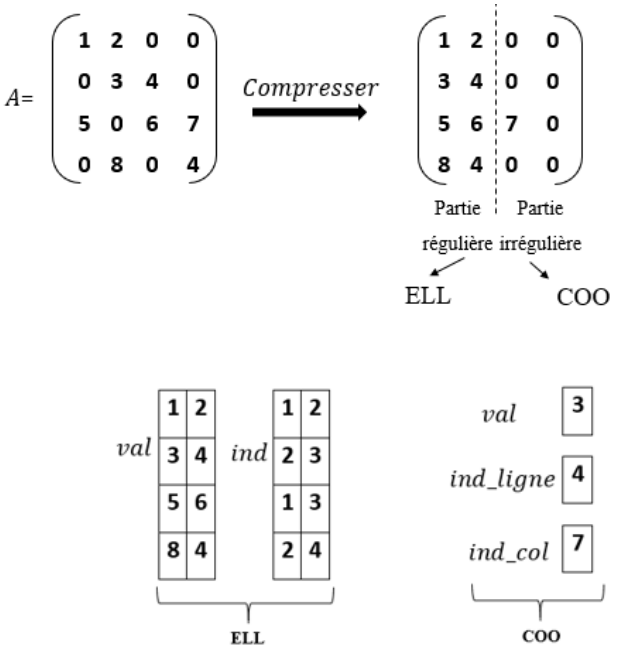

*Figure **Error! No text of specified style in document.**-17 Format de compression HYB*

Sliced ELL

Le format *Sliced ELL* est proposé dans [MLA10] pour améliorer les performances du PMVC sur des architectures GPU. *Sliced ELL* est un format paramétré utilisant comme paramètre S représentant la taille des tranches (*slice*). La matrice en entrée est partitionnée en un ensemble de blocs chacun formé par S lignes contiguës. Chaque bloc est stocké sous le format ELL. Le nombre d'éléments non nul peut être différent d'un bloc à un autre. Un autre tableau $slice\_start$ est utilisé pour stocker les indices du premier élément de chaque bloc. Le dernier élément de ce tableau contient le nombre total des éléments non nuls.

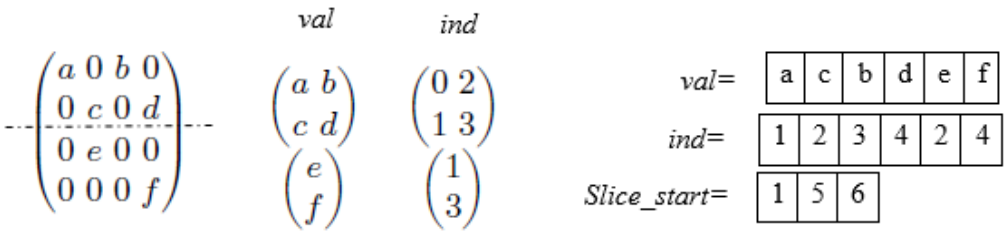

Figure ***Error! No text of specified style in document.****-18 Format Sliced ELL*

SELL-C- σ

SELL-C- σ est une variante du format *sliced ELL* utilisant deux paramètres C et σ. C indique la taille des blocs (slice) et σ indique l'étendu du tri ('*sorting scope*') [KHW14]. Cette étape de tri permet de minimiser le nombre des zéros ajoutés pour avoir des lignes équilibrées dans chaque bloc (Figure Error! ***No text of specified style in document.****-19*).

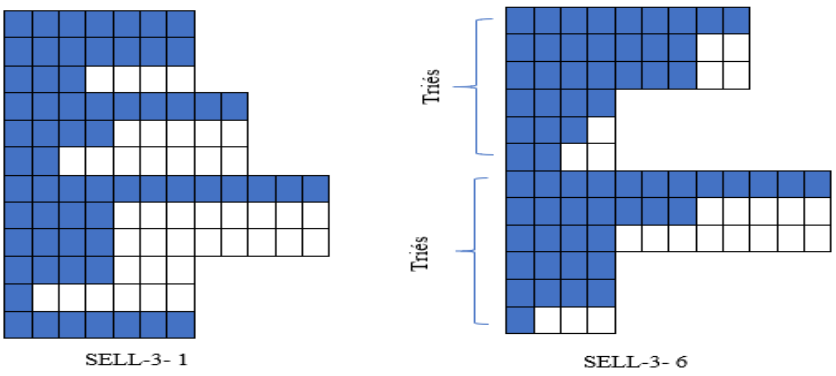

Figure ***Error! No text of specified style in document.****-19 format SELL-C- σ*

ELLPACK-R

ELLPACK-R est une autre variante du format ELLPACK. Ce format utilise un troisième tableau RL de taille n pour stocker le nombre d'éléments non nuls dans chaque ligne de la matrice ELL val [VGM09].

ELLPACK-RP

ELLPACK-RP est une combinaison des deux formats ELLPACK-R et JDS [CYL10]. Dans une première étape, on stocke la matrice d'entrée sous le format ELLPACK-R. Dans l'étape suivante, comme dans JDS, nous permutons les lignes en déplaçant les lignes les plus longues vers le haut et les plus courtes vers le bas dans l'ordre décroissant, et permutons également le tableau RL dans l'ordre décroissant. Le tableau $permu$ est utilisé pour sauvegarder la position d'origine de chaque ligne.

BELL

Le format Blocked ELLPACK (BELL) combine les avantages des deux formats BCSR et ELLPACK. Dans une première étape, la matrice en entrée est réorganisée sous la forme d'un ensemble de sous blocs denses de tailles $r \times c$. Par la suite, les lignes-blocks de la nouvelle matrice sont triées dans un ordre décroissant de nombre de blocs par ligne. L'étape de tri est suivie par une compression des blocs par ligne comme dans ELL. La matrice obtenue à cette étape est partitionnée en sous matrices de tailles $R \times (n/c)$. Chaque sous matrice est stockée sous ELL ou $(r \times c)$ BELL [CSV10].

TJDS

Le format TJDS (Transpose Jagged Diagonals Storage) est une variante de JDS. En première étape les colonnes sont triées par ordre décroissant de nombre d'éléments non nuls. Puis, dans chaque colonne, les éléments non nuls sont décalés en haut [EkM03].

pJDS

pJDS (padded JDS) est une autre variante du format JDS. Après modification des lignes et des colonnes (matrice obtenue triée), Des blocs de br lignes consécutives sont complétés par des zéros selon la ligne la plus longue [KHW12].

Figure ***Error! No text of specified style in document.****-20* représente les compositions des formats dérivés.

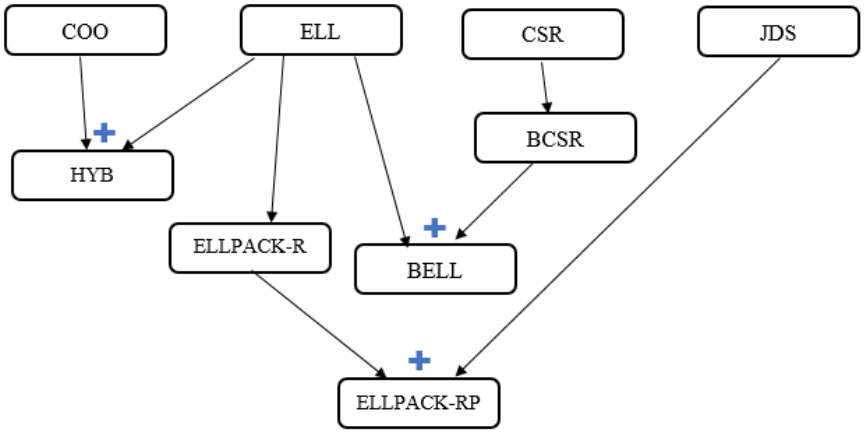

*Figure **Error! No text of specified style in document.**-20 Formats combinés*

Synthèse des formats

Le Tableau 1 présente l'ensemble des formats de compression et leurs complexités en termes de stockage. Les éléments de la matrice creuse sont, généralement, stockés selon un ordre de ligne (L), de colonne (C) ou de diagonale (D). Par exemple le format CSR utilise un stockage ligne (L) puisque les éléments dans le tableau val sont stockés ligne par ligne. Nous observons aussi que dans un certain nombre de formats de compression la structure initiale de la matrice creuse peut être modifiée suite à des opérations de tri ou de permutation des lignes/colonnes par exemple. Comme par exemple le cas du format JDS où les lignes de la matrice en entrée sont triées, et puis une opération de compression de colonnes est effectuée.

Tableau 1 Récapitulatif des formats de compression

Format	Stockage			Structure de la matrice Modifiée	Complexité spatiale		Plateforme	Références
	L	C	D		Nbr réels	Nbr entiers		
BELL				Oui	$n.c$	$n.c$		[CSV10]
COO	✔	✔		Non	nnz	$2 \times nnz$		[Saa94]
CSC		✔		Non	nnz	$nnz + n + 1$		[Saa94]
CSR	✔			Non	nnz	$nnz + n + 1$		[Saa94]
CSR-DU				Non	$n.c$	$n.c$		[KGK08]
CSRI				Non	$n.c$	$n.c$		[AKA12][KoJ02][YzB09] [YzB11]
CSR-VI					$n.c$	$n.c$		[KGK08]
CSR-ZZ				Non	$n.c$	$n.c$		[AKA12][YzB09][YzB11]
DIA			✔	Non	$n \times Nd$	Nd		[Saa94]
ELL	✔	✔		Oui	$n \times nz_{maxR}$	$n \times nz_{maxR}$		
ELLPACK-R	✔			Oui	$n.c$	$n.c$	GPU	[VGM09][VFG11]
ELLPACK-RP				Oui	$n.c$	$n.c$	GPU	[CYL10]
HYB	✔			Oui	nnz	$nnz_{EH} + nnz_{CH}$	GPU	[BeG09]
JDS				Oui	nnz	$n + nj$		[SGF09][EkM03]
MSC		✔		Non	$nnz + 1$	$nnz + 1$		[FKA09]
MSR	✔			Non	$nnz + 1$	$nnz + 1$		[Saa94]

pJDS				Oui	$n.c$	$n.c$		[KHW12]
BCSR				Oui	$n.c$	$n.c$		[Saa94]
SBCRS				Oui	$n.c$	$n.c$		[ShU11]
SELL-C- σ				Oui	$n.c$	$n.c$	GPU	[KHW14]
Sliced ELL				Oui	$n.c$	$n.c$	GPU	[MLA10][KHW14]
TJDS				Oui	nnz	$n + ntj$		[EkM03][MoE04]

$n.c$: valeur non connue à l'avance.
Nd : Nombre éléments non nuls sur la diagonale,
nz_{maxR} : Nombre maximal d'éléments non nuls par ligne,
nj : Nombre de 'jagged diagonals',
ntj : Nombre de 'jagged diagonals' transposé,
L: stockage ligne
C : stockage colonne
D : stockage diagonale

Produit matrice-vecteur creux (PMVC)

Les méthodes itératives génèrent une séquence de solutions approximées $\{x^{(k)}\}$ qui converge sous certaines conditions, vers la solution exacte $\tilde{x}$ du système à résoudre. Classiquement les schémas itératifs sont de la forme

$$x^{k+1} = \mathcal{L}\, x^k + c$$

Avec $\mathcal{L}$ matrice de l'itération (ne dépend que de la matrice A) et c un vecteur constant. Généralement, les méthodes itératives utilisent donc des boucles ayant comme noyau le produit matrice-vecteur (PMV). Tout au long des itérations, la matrice en entrée reste intacte. Dans le cas où la matrice en question est dense, on parle de produit matrice-vecteur dense (PMV). Soient A une matrice carrée ($A \in \mathbb{R}^{n\times n}$), x et y deux vecteurs denses ($x, y \in \mathbb{R}^n$). L'algorithme du PMV, $y = A \times x$, est illustré par **Algorithme Error!** *No text of specified style in document.**-1***. La complexité de calcul de cet algorithme est de $O(n^2)$.

Algorithme *Error! No text of specified style in document.**-1** Algorithme PMV*

```
POUR i = 1, n
    y[i] = 0;
    POUR j = 1, n
        y[i] = y[i] + A[i][j] × x[j]
    FIN POUR
FIN POUR
```

Dans le cas où A est creuse, la structure de l'algorithme du PMV dépend du format de stockage utilisé. L'utilisation du format de stockage CSR conduit à une version du produit matrice-vecteur creux (PMVC) représentée par **Algorithme Error!** *No text of specified style in document.**-2***. La complexité de calcul de cet algorithme est de $O(nnz)$.

Algorithme *Error! No text of specified style in document.**-2***

Algorithme PMVC pour le format CSR

```
POUR i = 1, n
    y[i] = 0;
    POUR j = ligne_ptr[i], ligne_ptr[i + 1] − 1
        y[i] = y[i] + val[j] × x[ind_col[j]]
    FIN POUR
FIN POUR
```

Schéma de Horner

La méthode de Horner, connue aussi sous les noms de schéma de Horner, méthode de Ruffini-Horner, algorithme de Ruffini-Horner ou règle de Ruffini, permet d'évaluer un polynôme en un point x_0. Cette méthode fut systématisée par ces deux mathématiciens au début du XIX^{e} siècle, dans le but de rechercher des valeurs approchées des racines d'un polynôme. D'abord intéressante pour le gain de performance qu'elle produit par rapport à une approche naïve, elle comporte de nombreuses applications numériques dont la division euclidienne d'un polynôme par un polynôme unitaire, le changement de variable d'un polynôme, la recherche de racine d'un polynôme, le calcul des dérivées successives d'un polynôme en un point.

Etant donné un polynôme P_n tels que :

$$P_n(x) = \sum_{i=0}^{n} a_i \, x^i$$

Hörner propose de factoriser $P_n(x)$ sous la forme :

$$P_n(x) \; = \; a_0 \; + \; x(a_1 + x(a_2 + \; \ldots x(a_{n-1} \; + \; xa_n) \; \ldots))$$

Suivant ce schéma, le calcul de $P_n(x_0)$ n'implique que n multiplications. On passe donc, étant donné que le coût des additions est négligeable devant celui des multiplications, d'une complexité quadratique $(O(n^2))$ à une complexité linéaire $(O(n))$ [FFS11].

Applications creuses

Les matrices creuses se trouvent au cœur d'un ensemble varié d'applications dans de nombreux domaines, tels que le calcul scientifique, l'ingénierie, la modélisation économique, la recherche de l'information, etc. Nous présentons dans cette section quelques applications utilisant ce type de matrice. Nous commençons par la présentation de la collection de l'université de Floride contenant des matrices creuses provenant de problèmes réels. Nous présentons par la suite deux exemples : une application correspondant à l'édition des images et la matrice des liens dans Google.

Collection de Tim Davis

La collection des matrices creuses de l'université de Floride, ou encore appelée collection Tim Davis (qu'on note T.D), est un grand ensemble de matrices creuses qui sont tirées des applications réelles de différents domaines [DaY11]. *Figure* ***Error! No text of specified style in document.****-21* illustre la répartition des matrices T.D selon les différents domaines d'applications. Les données présentées dans cette figure sont collectées à partir des

informations qui existent en ligne sur le site de la collection [DaY11]. Cette collection de matrices est largement utilisée par la communauté de l'algèbre linéaire numérique pour le développement et l'évaluation des performances des algorithmes creux. Les densités de ces matrices varient entre 4.17×10^{-6} % jusqu'à 100% (voir *Figure* ***Error! No text of specified style in document.****-22*).

Tableau 2 Caractéristiques des matrices creuses de la collection Tim Davis

n	***nnz***	**Densité (%)**	**Structures**		**Types**		**Binaire**
			Cr	**Rg**	**R**	**C**	
[1 : 118,142,155]	[1 : 1,949,412,601]	[4.17E-06 : 100]	081	76	708	9	563
Cr : Carrée **Rg** : Rectangulaire **R** : Réel **C** : Complexe							

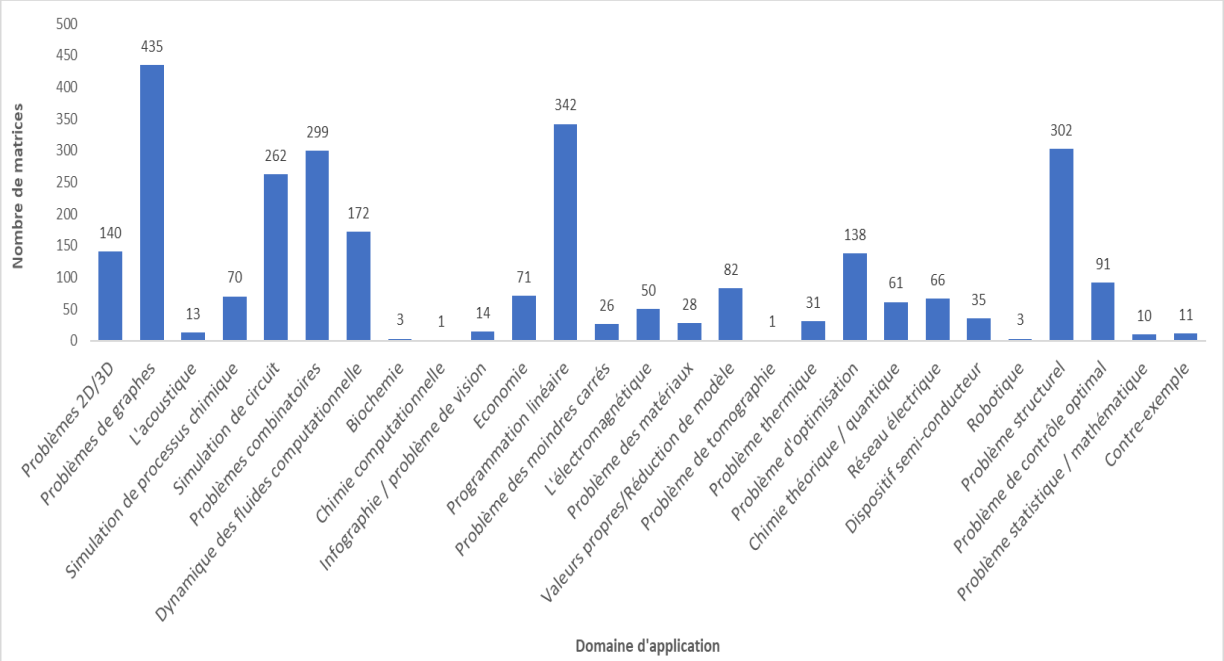

Figure ***Error! No text of specified style in document.****-21 : domaines d'application des matrices creuses de la collection de Tim Davis*

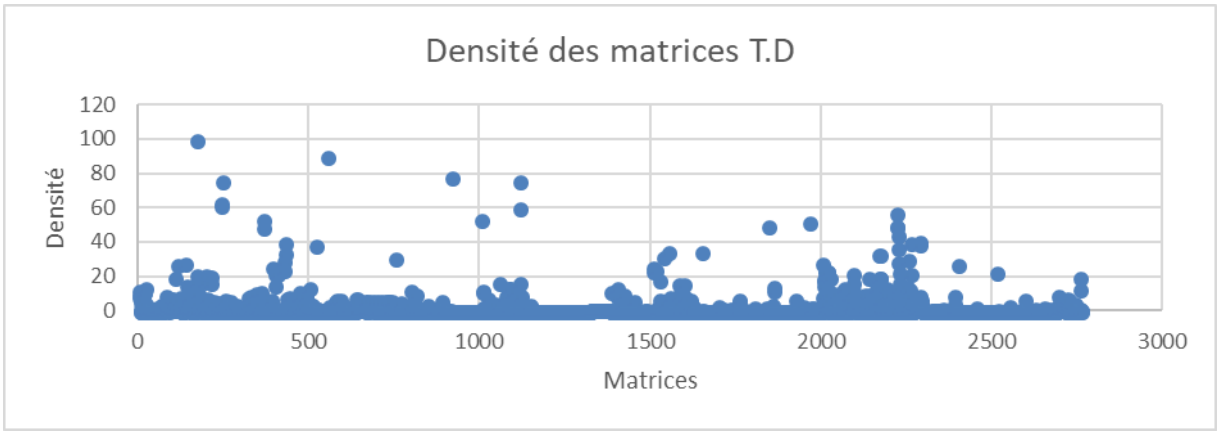

*Figure **Error! No text of specified style in document.**-22 Densité des matrices de la collection de Tim Davis*

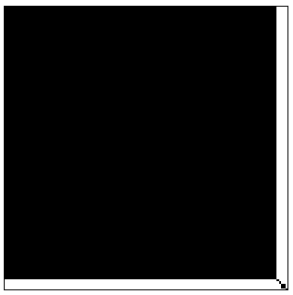

*Figure **Error! No text of specified style in document.**-23 : matrice webbase-2001, la matrice la plus grande (nombre de lignes) dans la collection Tim Davis des matrices creuses [DaY11]*

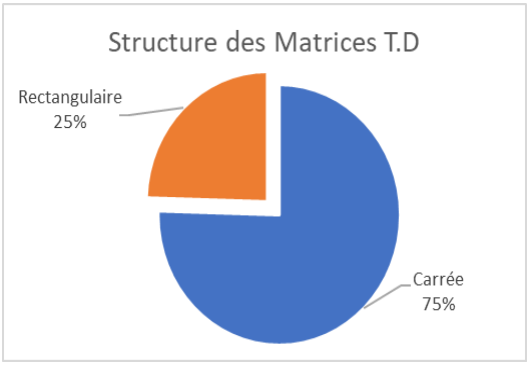

*Figure **Error! No text of specified style in document.**-24 structures des matrices de T.D*

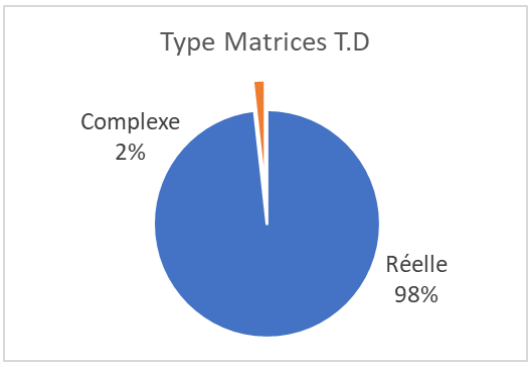

*Figure **Error! No text of specified style in document.**-25 Types des matrices de T.D*

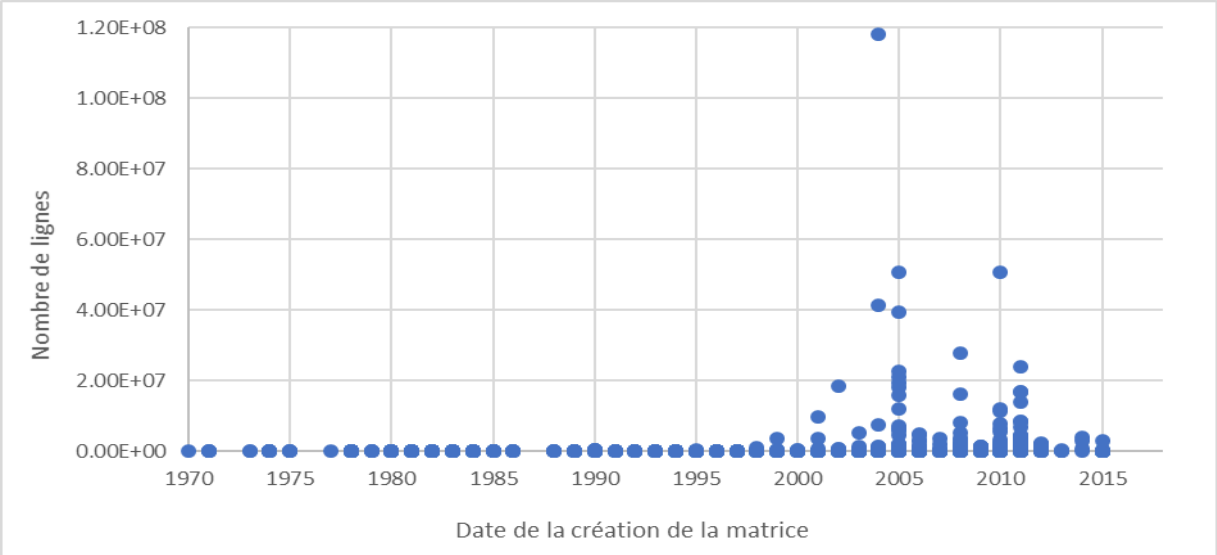

Figure ***Error! No text of specified style in document.****-26 Dimensions des matrices de la collection de Tim Davis (en fonction de l'année de création)*

A nos jours, la collection contient 2757 matrices avec des tailles allant jusqu'à $n = 118\,142\,155$ (la matrice $webbase - 2001$, illustrée dans *Figure* ***Error! No text of specified style in document.****-23*, provenant d'un problème de graphe). Les matrices de cette collection sont généralement des matrices carrées (*Figure* ***Error! No text of specified style in document.****-24*) avec des éléments non nuls de type complexe ou réel (*Figure* ***Error! No text of specified style in document.****-25*).

Edition d'images

Dans [WPY11], les auteurs présentent un modèle pour l'édition d'images en utilisant le PMVC. Le principe est de transformer le problème d'édition d'images en un problème de minimisation d'énergie linéaire et de le traiter par la suite par la résolution d'un système linéaire creux. Ceci permet d'avoir une solution optimale du problème.

*Figure **Error! No text of specified style in document.**-27 exemple de clonage d'image en utilisant différentes méthodes (a) image source (b) image cible (c) clonage Poisson (d) clonage avec la méthode proposée dans [WPY11]*

Tout d'abord, trois opérations classiques d'édition d'image, y compris le filtrage linéaire, le redimensionnement et la sélection, sont présentées sous la forme du PMVC. Cette reformulation aide à mettre en place un mécanisme simple pour combiner de manière flexible et naturelle diverses caractéristiques de l'image (caractéristiques visuelles de bas niveau ou caractéristiques géométriques) et des contraintes en une fonction intégrée de minimisation de l'énergie sous la norme L_2. Ensuite, ce modèle est appliqué pour implémenter les tâches de pan-sharpening, de clonage d'image, d'édition d'images mixtes et de transfert de texture, qui sont couramment utilisées dans le domaine de l'art numérique. *Figure **Error! No text of specified style in document.**-27* illustre un exemple de clonage, l'image de gauche est l'image source dans laquelle la ligne rouge indique les limites. Nous remarquons que le modèle proposé permet non seulement d'effectuer un bon clonage, mais aussi de conserver le ton de couleur de l'image source.

Matrice de Google

Depuis plus d'une décennie Google domine le marché des moteurs de recherche sur internet. Son point fort est qu'il trie intelligemment ses résultats par ordre de pertinence. Nous expliquons ici brièvement l'algorithme PageRank qui est à la base de ce classement. L'idée principale est une judicieuse modélisation mathématique qui permet d'estimer la pertinence ou plutôt la popularité des pages web [INF].

En effet, la plupart des pages Web incluent des liens hypertextes vers d'autres pages, c'est pourquoi lors d'une requête le moteur de recherche sélectionne les pages qui sont en adéquation avec la requête en analysant le contenu de la page Web et, d'autre part, propose un classement

des pages en fonction de leur indice de popularité. Ce dernier ne dépend pas de la consultation de la page par les internautes mais seulement du nombre de liens qui pointent sur cette page à partir d'autres pages Web. C'est cet indice qui distingua Google de ses concurrents dès le départ. Nous nous intéressons ici au calcul de cet indice et nous allons voir que celui-ci fait intervenir le calcul des vecteurs propres et des valeurs propres d'une grande matrice. Pour cela, nous considérons qu'il y a n pages Web au total, la structure du Web peut être représentée simplement par une matrice $C \in \mathbb{R}^{+}_{n,n}$ telle que :

$$C_{ij} = \begin{cases} 1 \text{ si la page j pointe sur la page i,} \\ 0 \text{ sinon} \end{cases}$$

Les liens d'une page sur elle-même ne sont pas pris en compte, nous posons alors que $C_{ij} = 0$.

Prenons l'exemple illustré dans *Figure* ***Error! No text of specified style in document.****-28* où le web comporte quatre pages :

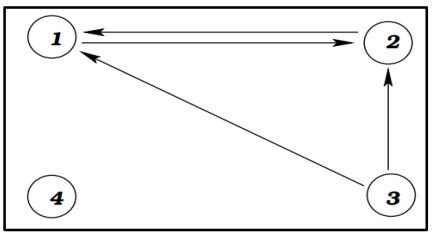

Figure ***Error! No text of specified style in document.****-28: Exemple de graphe utilisé par l'algorithme PageRank de Google*

Il est possible de construire une matrice $C \in \mathbb{R}^{+}_{4\times4}$ ne contenant que des 1 ou des 0 correspondant à ce graphe :

$$C = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Nous souhaitons attribuer à chaque page Web i un indice de pertinence $x_i \in \mathbb{R}^{+}$ de façon à pouvoir classer l'ensemble des pages par score décroissant et présenter à l'utilisateur une liste classée des pages relatives à sa requête. Il s'agit donc de calculer l'indice x_i correspondant à la page i, en partant du principe qu'un lien de la page j pointant sur la page i contribue au score de cette dernière, avec une pondération par x_j (une page ayant un indice élevé a plus de poids qu'une page n'ayant qu'un indice médiocre) et par le nombre total de liens présents sur ladite page n_j (une page ayant beaucoup de liens a moins d'influence).

$$n_j = \sum_{k=1}^{n} \mathrm{C}_{k,j}$$

La composante x_i vérifie ainsi l'équation suivante :

$$x_i = \sum_{j=1}^{n} (\mathrm{C}_{i,j} \times \frac{x_j}{n_j})$$

Le problème du classement des pages du Web se trouve ainsi ramené à la recherche d'un vecteur propre associé à la valeur propre 1 de la matrice $\check{\mathrm{C}}$ dont les composantes sont :

$$\check{\mathrm{C}}_{i,j} = \frac{c_{i,j}}{n_j} \qquad \forall\ 1 \leq i,j \leq n.$$

Mise en œuvre

Architectures parallèles et distribuées

Nous-nous intéressons dans cette partie à la description détaillée des différentes architectures cibles que nous aurons la possibilité d'utiliser au moment de l'implémentation de notre noyau de calcul. Pour cela, après la présentation d'une classification des architectures parallèles, nous détaillons les différents modèles de programmation. Quelques API de programmation parallèle sont présentées dans la dernière partie de cette section.

Classification de Flynn

Les ordinateurs parallèles standards sont des machines ayant une architecture constituée de plusieurs processeurs identiques. Différents types de machines parallèles peuvent être rencontrés [Ham10].

Plusieurs classifications ont été proposées dans la littérature. La plus populaire, celle de Flynn (voir *Tableau 3*), est basée sur le type d'organisation des flots de données (les entrées) et des flots d'instructions (les algorithmes) [CoT93].

Cette classification catégorise les ordinateurs suivant le type d'organisation du flot de données et du flot d'instructions.

- SISD (Simple instruction Simple Data) : correspond à une architecture séquentielle (modèle de Von Neumann), un seul flot d'instructions, un seul flot de données.
- SIMD (Single Instruction Multiple Data) : un seul flot d'instructions agissant sur des flots de données, utilisé dans les processeurs vectoriels et les GPU : un seul flot d'instructions et plusieurs flots de données.

- MISD (Multiple Instruction Simple Data) : un seul flot de données traité par plusieurs unités de traitement associées à plusieurs flots d'instructions.
- MIMD (Multiple Instruction Multiple Data) : flot d'instructions et de données multiples. Dans cette classe, on trouve les machines à mémoire partagée, les machines à mémoire distribuée, et les machines à mémoire hybride (voir *Figure* ***Error! No text of specified style in document.****-29*).

Tableau 3 Taxonomie de Flynn

	Flot d'instructions unique	**Flot d'instructions multiple**
Flot de données unique	SISD	MISD
Flot de données multiple	SIMD	MIMD
S: Single **M:** Multiple **I:** Instruction **D:** Data		

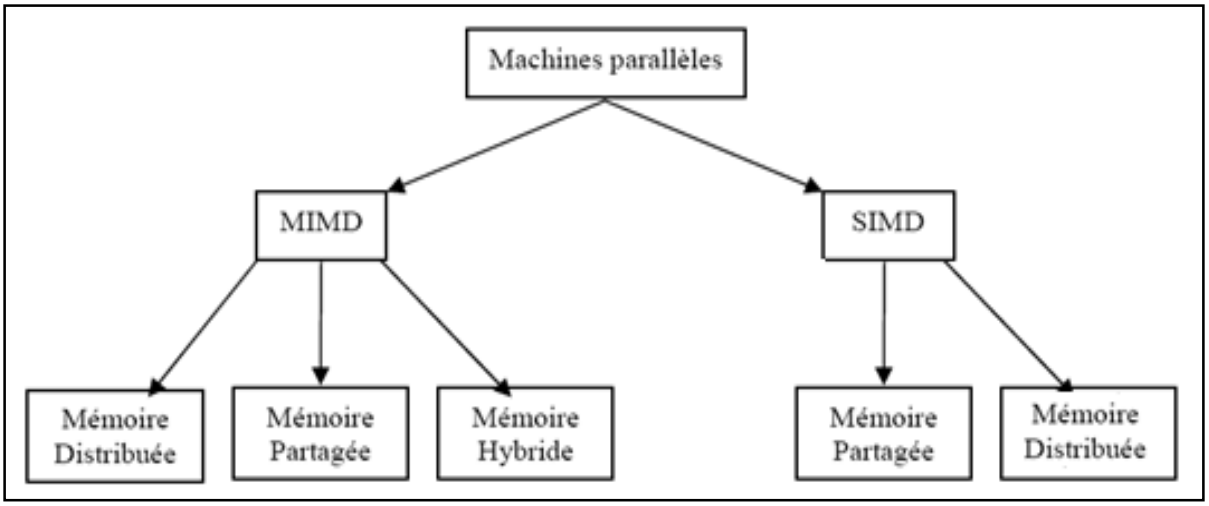

Figure ***Error! No text of specified style in document.****-29: Taxonomie de haut niveau des architectures parallèles*

c) <u>Machines à mémoire partagée</u>

Un système à mémoire partagée mettant en jeu plusieurs ressources de calcul se partageant une mémoire commune et ce, physiquement ou de manière logicielle. Ainsi, un processus p placé sur un processeur P a la possibilité physique d'accéder aux données d'un autre processus p' placé sur un autre processeur P'. Dans une machine à mémoire partagée, tous les processeurs accèdent à la mémoire globale, mais chacun travaille indépendamment de l'autre. Chaque processeur a sa propre mémoire locale qui est la mémoire cache.

Il est à noter qu'un processeur peut comporter un ou plusieurs threads (un 'thread' ou 'processus léger' correspondant à l'exécution simultanée de plusieurs fonctions/procédures d'un programme, on parle alors de programme multithread, au sein d'un seul processus).

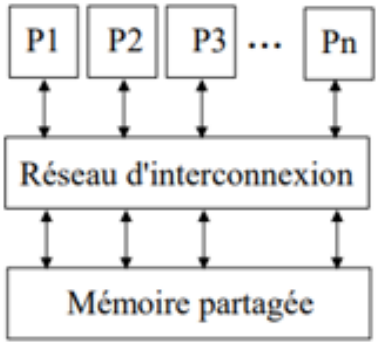

*Figure **Error! No text of specified style in document.**-30 Systèmes à mémoire partagée*

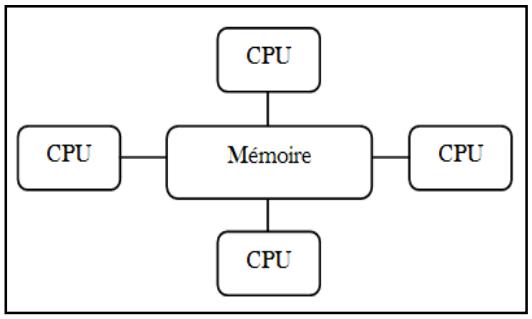

*Figure **Error! No text of specified style in document.**-31: Architecture SMP*

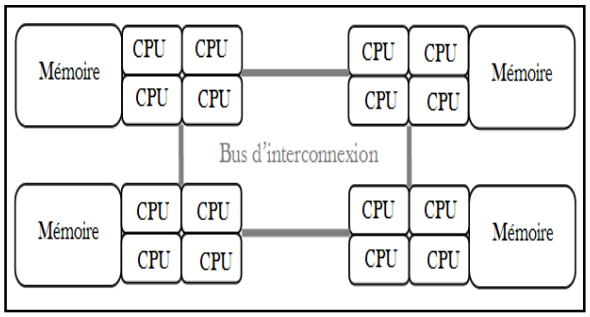

*Figure **Error! No text of specified style in document.**-32: Architecture NUMA*

Le plus grand ensemble de processeurs partageant matériellement de la mémoire est appelé ***nœud***. On parle de nœud SMP ou de nœud NUMA [AMP]. La différence majeure entre une machine SMP et une machine NUMA est la hiérarchie de la mémoire.

Dans tout programme, il est possible de mettre en évidence un phénomène de localité des accès mémoire, exprimé en termes de :

- *Localité temporelle* : plus l'accès à une zone mémoire est récent, plus sa probabilité d'être accédé à nouveau est élevée ;
- *Localité spatiale* : plus une zone mémoire est proche de la dernière zone mémoire accédée, plus la probabilité qu'elle soit à son tour accédée est élevée.

(ii) Architectures SMP

Dans les architectures SMP (**S**ymmetrical **M**ulti**P**rocessor) tous les processeurs accèdent à n'importe quel point de la mémoire en un temps uniforme (i.e. de façon symétrique). Cette architecture est appelée aussi UMA (**U**niform **M**emory **A**cces).

Architectures NUMA

Une machine NUMA (Non Uniform Memory Acces) est constituée de plusieurs processeurs connectés à plusieurs mémoires distinctes (on parle de bancs mémoires) reliées entre elles par des mécanismes matériels. D'un point de vue du système d'opération, il existe une mémoire unique dont la taille est la somme des mémoires physiques distinctes [AMP]. Dans une machine NUMA, le temps d'accès à la mémoire dépend du placement des données dans celle-ci, d'où le nom de la machine. Ainsi, l'accès est :

- rapide si les données sont dans le banc mémoire local au processeur,
- lent (d'un facteur appelé ***facteur NUMA***) si c'est dans un autre banc mémoire

Machines à mémoire distribuée

Un système à mémoire distribuée est un système mettant en jeu plusieurs ressources de calcul qui n'ont pas de mémoire partagée, que ce soit de manière physique ou logicielle [AMP]. Chaque processeur possède alors sa propre mémoire locale privée et n'a pas la possibilité d'accéder aux données d'un autre processus. C'est pourquoi il est obligatoire d'ajouter un réseau externe d'intercommunication entre les processeurs (*Figure* ***Error! No text of specified style in document.-33***).

d) Machines à mémoire hybride.

Aujourd'hui, plusieurs ordinateurs parallèles sont équipés de mémoire hybride, i.e. partagée et distribuée. La brique de base (nœud) est un multiprocesseur à mémoire partagée. Ces briques sont interconnectées par un réseau (type Ethernet, Myrinet, Infiniband).

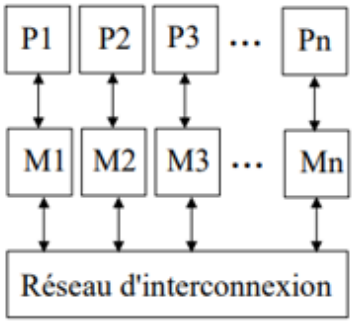

*Figure **Error! No text of specified style in document.**-33 Système à mémoire distribuée*

Architectures multi-cœurs

L'augmentation de la puissance des processeurs se traduit principalement par un nombre croissant de cœurs de calcul et les puces multi-cœurs sont aujourd'hui à la base de la plupart des architectures multiprocesseurs. Le multi-cœur est devenu, aujourd'hui, la solution courante pour augmenter les performances tout en conservant une basse consommation électrique [NRI]. Un processeur est dit multi-cœurs (multi-core en anglais) quand il est doté d'un microprocesseur qui intègre au moins deux unités de calcul ou cœurs indépendants (*Figure **Error! No text of specified style in document.**-34* et *Figure **Error! No text of specified style in document.**-35*) [PCB].

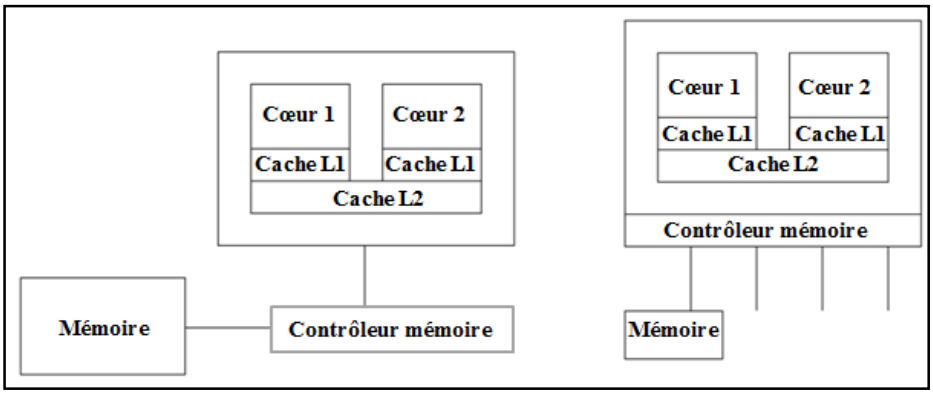

*Figure **Error! No text of specified style in document.**-34: Deux approches différentes d'architectures multi-cœurs (le contrôleur mémoire peut être présenté comme entité à part ou bien intégré au sein du processeur)*

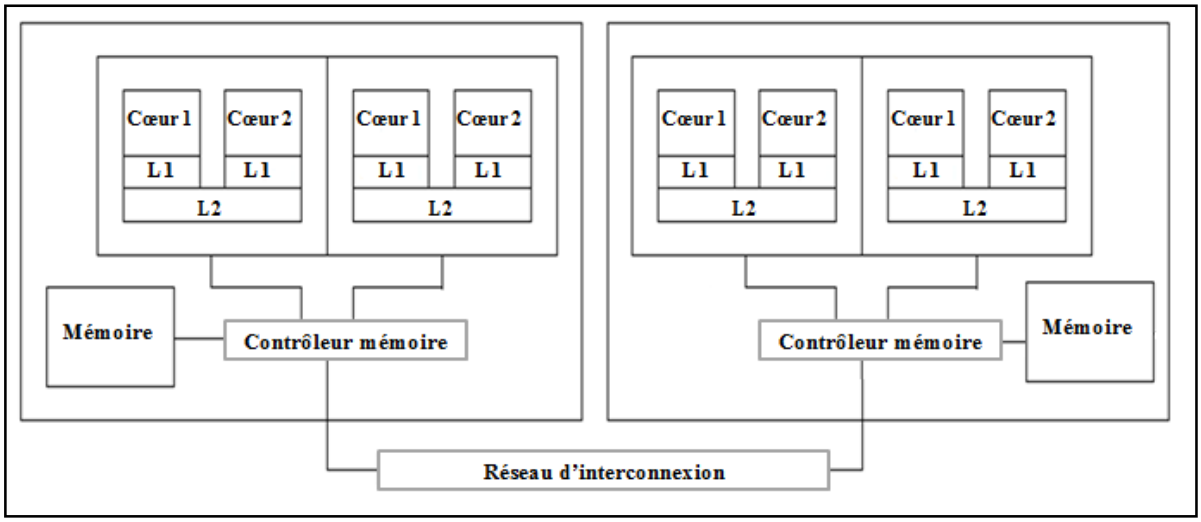

*Figure **Error! No text of specified style in document.**-35: Une architecture de machine NUMA hiérarchique à 4 processeurs bi-cœurs*

Architectures GPU

Le GPU (Graphical Processing Unit) est un processeur utilisé principalement pour le traitement des données graphiques. La première compagnie à développer des GPU est NVIDIA avec son GeForce 256. Il est capable d'afficher 10 millions de polygones par seconde et possède pas moins de 22 millions de transistors, comparé aux 9 millions que possède le Pentium III.
De nos jours, les GPU peuvent être utilisés pour effectuer des calculs haute performance. Physiquement, un GPU peut être exister sur le même circuit avec un CPU. Il est aussi possible qu'il soit sur une carte graphique ou sur la carte.

Grappes de calcul (Cluster)

Un cluster de calcul est une machine parallèle, regroupant de plusieurs ordinateurs, dédiée au calcul intensif (*Figure **Error! No text of specified style in document.**-36*)[MIG].

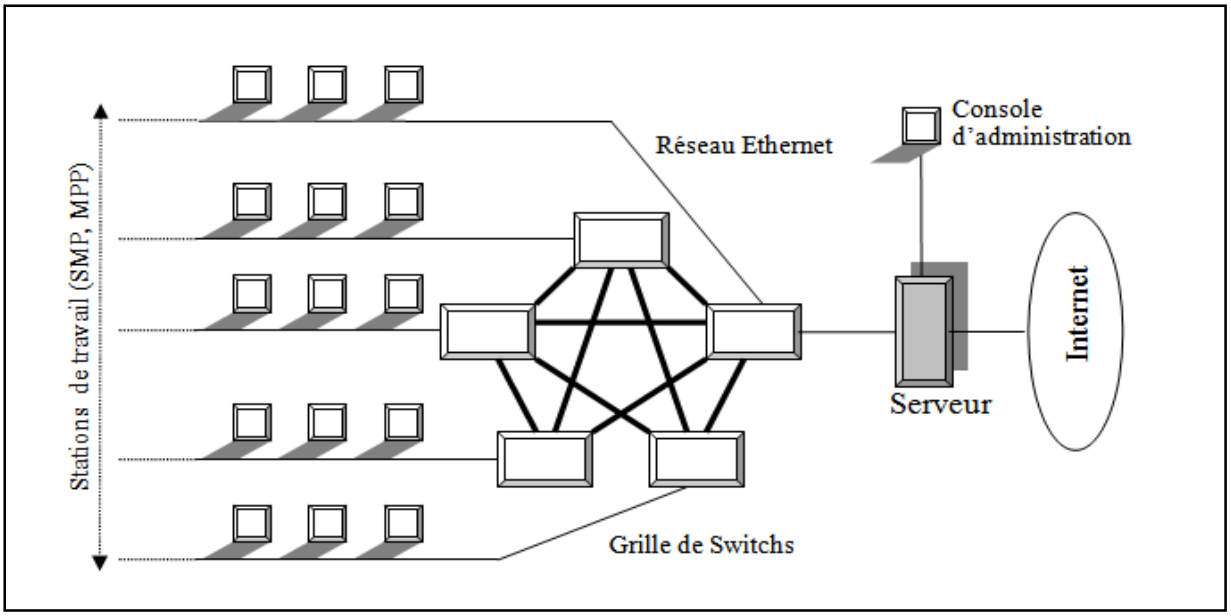

*Figure **Error! No text of specified style in document.**-36: Grappe de PC*

Les machines d'un cluster, appelées nœuds, peuvent travailler de manière collective ou individuelle.

Grilles de calcul

Une grille de calcul est un ensemble de nœuds de calcul distribués géographiquement et connecté par une combinaison de réseaux. Les grilles sont conçues pour surmonter le problème d'utilisation des supercalculateurs qui sont très coûteuses. Elles fournissent une infrastructure multi-utilisateur qui prend en charge les demandes discontinues d'un traitement d'informations volumineux.

Modèles de programmation

La mise en œuvre de ces architectures passe par la définition de modèles de programmation associés. On distingue généralement trois approches de parallélisation, à savoir, le parallélisme de données, de contrôle et de flux.

Le parallélisme de données (' data parallelism')

Le principe consiste à décomposer les données en parties approximativement de même taille qui seront affectées aux différents processus. Chaque processus sera affecté à un processeur différent mais dans la généralité, plusieurs processus peuvent être affectés à un même processeur.

Le parallélisme de contrôle ('control parallelism')

Appelé aussi parallélisme de tâche, consiste à découper le problème (une tâche) en sous-problèmes (des sous-tâches) dans le but d'exécuter le maximum de sous-tâches dans un même laps de temps. La mise en œuvre consiste alors à exécuter des tâches différentes (attribuées aux unités de calculs disponibles) en même temps afin de générer du parallélisme.

Le parallélisme de flux ('flow parallelism')

Correspond à la technique du travail à la chaîne. Chaque donnée subit une séquence de traitements. C'est cette séquence qui est utilisée comme source de parallélisme et le parallélisme de flux réalise la séquence des traitements en mode pipeline en exploitant une régularité des données.

API pour programmation parallèle

Bibliothèque de communication MPI

MPI (*Message Passing Interface*) est l'environnement de programmation le plus connu pour les architectures à mémoire distribuée [HCE09]. Dans un modèle de programmation parallèle par échange de messages (MPI), le programme est exécuté par plusieurs processus. Chaque processus exécute un exemplaire du programme et a accès à sa propre mémoire. De ce fait, les variables du programme deviennent des variables locales au niveau de chaque processus. De plus, un processus ne peut pas accéder à la mémoire des processus voisins. Il peut toutefois envoyer des informations à d'autres processus à condition que ces derniers (processus récepteurs) soient au courant qu'ils devaient recevoir ces informations du processus émetteur [DeS08].

La communication entre deux processus peut être de type point à point (envoi et réception d'une donnée d'un émetteur vers un destinataire), de type collectif (diffusion d'un message à un groupe de processus, opération de réduction, distribution ou redistribution des données envoyées) (voir Figure 4.6).

Dans un programme MPI, un commutateur permet de connaître l'ensemble des processus actifs. MPI définit des fonctions qui, à tout moment, permettent de connaître le nombre de processus gérés dans un commutateur et le rang d'un processus dans le commutateur. La bibliothèque met de nombreuses fonctions à la disposition du programmeur, lui permettant ainsi d'effectuer différents types d'envoi et de réception de messages (point à point bloquant (synchrone), point à point non bloquant (asynchrone), collectifs et des opérations globales (barrière, réduction, diffusion)).

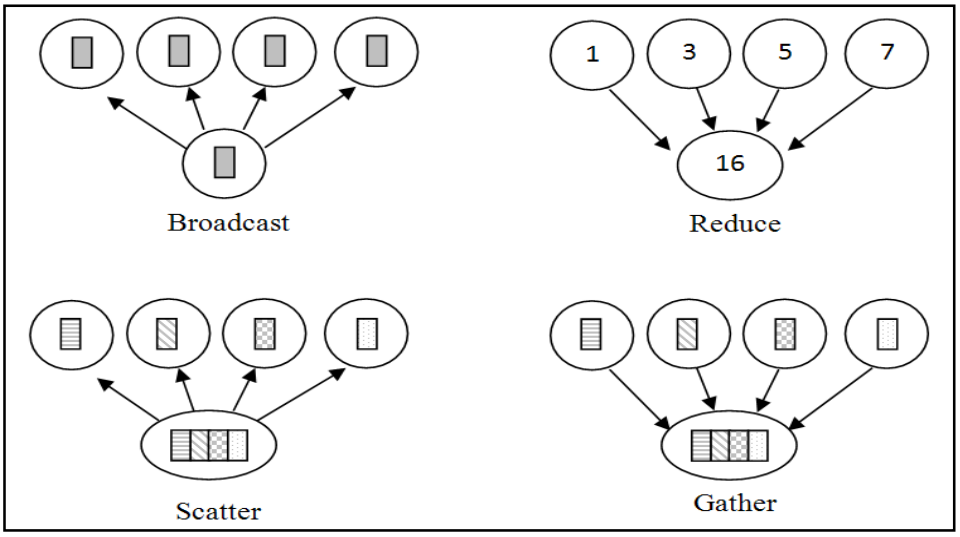

*Figure **Error! No text of specified style in document.**-37: Opérations de type 'collectif' de MPI*

API OpenMP

OpenMP est une API de programmation à base de directives pour l'implémentation des programmes sur des systèmes parallèles à mémoire partagée en utilisant le concept de multithreading. L'approche la plus simple pour paralléliser un code séquentiel avec OpenMP consiste à paralléliser les boucles. Pour le faire, il s'agit de lancer plusieurs threads, chacun exécute une ou plusieurs itérations d'une boucle [HCE09]. OpenMP possède une interface de programmation pour les langages de programmation C, C++ et Fortran, et il est supporté par de nombreux environnements (e.g. Linux ou Windows).

Programmation hybride

La programmation hybride parallèle consiste à mélanger plusieurs paradigmes de programmation parallèle dans le but de tirer parti des avantages des différentes approches. Avec les évolutions récentes des architectures HPC, la programmation hybride MPI-OpenMP se développe et semble devenir l'approche privilégiée pour les supercalculateurs [LaW10]. L'approche hybride MPI-OpenMP consiste à mixer au sein d'un même code une parallélisation à plusieurs niveaux : un niveau OpenMP au sein d'un nœud à mémoire partagée (parallélisation entre cœurs) et un niveau MPI entre les différents nœuds de calcul [HCE09] [RHJ09]. Des résultats récents ont montré qu'une telle réorganisation peut améliorer les performances sur des clusters dont les nœuds sont à mémoire partagée [RHJ09].

CUDA

CUDA (Compute Unified Device Architecture) est l'un des premiers langages proposé pour la programmation des architectures GPU. CUDA a été développé en 2007 par NVIDIA pour l'utilisation des architectures Geforce 8. Il supporte plusieurs langages, à savoir, C, C++ et Fortran. Actuellement, CUDA est l'environnement le plus utilisé pour la conception des applications GPU bien que certaines alternatives sont disponibles, telles que OpenCL (*Open Computing Language*) [Cou14].

Dans un programme CUDA, le CPU (appelé *hôte*) demande au GPU (appelé *périphérique*) d'effectuer un certain traitement parallèle. Chaque portion du code parallèle (appelée *kernel*) est exécutée sur le *périphérique* par des *threads*. Les *threads* sont organisés comme suit : un ensemble de threads constitue un *bloc*. Un ensemble de ses blocs constitue une *grille*. En pratique, les *threads* sont regroupés en groupes de 32 *threads* appelés *warps* [Cou14].

Conclusion

Les méthodes de résolution des systèmes linéaires peuvent être directes ou itératives. Cependant, dans le cas creux il est plus intéressant d'utiliser les méthodes itératives puisqu'elles conservent bien le caractère creux des matrices traitées. Ces méthodes itératives reposent sur le noyau de calcul du produit matrice vecteur creux. Généralement, les matrices traitées sont des matrices creuses de grandes tailles nécessitant un espace mémoire ainsi qu'un taux de calcul important. Ainsi, nous avons présenté dans ce chapitre les matrices creuses, leur structure régulière ou irrégulière ainsi que les formats de compression correspondants. Nous avons également présenté dans ce chapitre des applications scientifiques traitant des matrices creuses telles que l'application de Google. Dans la dernière partie de ce chapitre nous avons passé en revue les principales architectures parallèles et distribuées, ainsi que les modèles de programmation parallèles.

Nous avons observé qu'il existe un grand nombre de paramètres à prendre en considération lors du processus de décision pour le choix du meilleur format de compression. Un choix optimisé pour un paramètre peut dégrader les performances d'un autre. Ceci nous motive à l'introduction de l'expertise utilisateur dans le processus de décision.

ETUDE DE LA DETECTION AUTOMATIQUE DU MEILLEUR FORMAT DE COMPRESSION DE MATRICES CREUSES

Introduction

Plusieurs applications numériques traitent des matrices creuses ayant une structure régulière ou irrégulière. La taille élevée de ces matrices exige d'utiliser des formats de compression et des architectures parallèles/distribuées afin de réduire à la fois la complexité spatiale et temporelle de traitement. Nous avons montré dans le chapitre précèdent que plusieurs paramètres peuvent intervenir lors du choix du format de compression. En effet, ce choix dépend de la matrice creuse, de la méthode numérique, de l'architecture matérielle cible et du modèle de programmation. Etant donnée la diversité de ces paramètres, le processus de décision devient plus complexe. Ainsi, nous proposons la conception et la mise en œuvre d'un système intelligent pour la détection automatique du meilleur format, où l'expertise utilisateur est prise en compte.

Nous présentons dans la première section de ce chapitre notre problématique. Nous exposons dans la section 0 un état de l'art sur les différentes approches et stratégies connues dans la littérature pour le choix du format de compression. Nous présentons dans la section Chapitre 1.1 notre approche pour la sélection automatique du meilleur format de compression. La dernière partie de ce chapitre est dévolue à la présentation d'une conception détaillée du système intelligent que nous proposons.

Présentation de la problématique

De nombreuses applications en calcul scientifique telles que la dynamique des fluides, le traitement d'images ou l'analyse de données, aboutissent à des problèmes d'algèbre linéaire creux. Dans la plupart de ces applications, les matrices creuses traitées sont de grandes tailles et nécessitent donc l'utilisation d'une structure particulière de stockage des données.

Le choix du format de stockage le plus adéquat dépend généralement de plusieurs facteurs dont les principaux sont : la structure de la matrice creuse (répartition régulière ou irrégulière de ses éléments non nuls), l'architecture de la plateforme ciblée, la méthode numérique utilisée et le modèle de programmation parallèle. Ainsi, le choix du format de compression peut avoir un impact direct sur les performances de la méthode numérique. L'objectif de notre travail est donc de choisir le format de compression le plus adéquat en prenant en compte tous ces paramètres.

Etat de l'art sur la détection du meilleur format de compression de matrices creuses

Les performances d'un algorithme creux peuvent dépendre fortement du choix du format de compression de la matrice creuse [ShU07]. Plusieurs formats ont été proposés dans la littérature pour optimiser le stockage des matrices creuses. La plupart de ces formats de stockage ont opté pour l'élimination des éléments nuls inutiles dans le calcul. Les performances de ces formats varient selon les structures des matrices creuses (distribution de ses éléments non nuls). Ainsi, il a été démontré que le format qui donne des bons résultats pour une structure donnée de matrice creuse peut être utilisé pour les matrices qui ont des structures similaires [KhU09]. En outre, la même optimisation et la même mise en œuvre peuvent se comporter différemment sur des plateformes différentes.

Nous avons essayé de classer des travaux existant dans la littérature en deux groupes :

- **Groupe 1** : Les travaux qui s'intéressent en premier lieu à l'optimisation des noyaux de calcul. Ainsi, une adaptation automatique des paramètres des structures de données est effectuée lors de l'optimisation des opérations de calcul.
- **Groupe 2** : les travaux qui s'intéressent directement au choix automatique du meilleur format de compression. Ces travaux sont destinés aux calculs creux effectués sur différents types d'architectures, à savoir, les architectures multi-cœurs à mémoire partagée et les GPU.
    - Pour les travaux s'intéressant aux architectures multi-cœurs, le choix du meilleur format de compression est basé sur l'utilisation des algorithmes de l'apprentissage automatique.
    - Dans les cas des architectures GPU, différentes méthodes ont été proposées. Nous classons ces méthodes selon trois modèles, à savoir, le modèle statique, le modèle probabiliste et le modèle d'apprentissage automatique.

Optimisation automatique des noyaux de calcul à l'exécution

e) SPARSITY

Le système SPARSITY est conçu pour aider les utilisateurs à obtenir des noyaux de calculs de matrices creuses optimisés. L'utilisateur n'a pas besoin de connaître ni les

détails de la hiérarchie de la mémoire de la machine ni la manière avec laquelle les structures des matrices seront portées sur cette hiérarchie. SPARSITY effectue de nombreuses optimisations qui concernent à la fois le code et les structures des données [ImY01]. Le blocage des registres est l'une des optimisations proposées.

Le blocage des registres permet de modifier la structure de stockage de la matrice creuse afin d'améliorer la localité des données lors du calcul. Soit A une matrice creuse compressée selon le format CSR, et x et y deux vecteurs denses. Le calcul du produit matrice vecteur creux $y = A \times x$ revient à effectuer une suite de produits scalaires impliquant les lignes de A et le vecteur x. Ainsi, les lignes de A et les éléments de y ne sont accédés qu'une seule fois. Néanmoins, les éléments de x sont accédés plusieurs fois. Pour cela, les auteurs proposent le stockage de la matrice sous formes d'un ensemble de blocs denses de taille $r \times c$. Dans ce cas le calcul du PMVC sera effectué bloc par bloc, ce qui permet de garder les éléments de x le plus possible dans le registre. La taille de ces blocs est à déterminer automatiquement lors de la compilation [IYV04].

Les performances de calcul dépendent de la taille des blocs, ainsi le choix de cette taille est assez critique. Ce choix dépend à la fois de la structure de la matrice et de l'architecture utilisée. Pour cela, les auteurs proposent un modèle permettant la prédiction des performances de calcul pour différentes tailles des blocs. Le modèle proposé contient deux composantes principales : la première, permet de faire une approximation en Mflops dans le cas d'une matrice ayant des blocs de tailles données. La deuxième composante sert à avoir une approximation sur le taux de calcul non nécessaire. En effet, la création des blocs nécessite l'ajout des zéros qui seront impliqués dans le calcul [IYV04] [ImY01].

OSKI

OSKI (Optimized Sparse Kernel Interface) est une collection de primitives de bas niveau fournissant des noyaux de calcul creux. Ces primitives peuvent être intégrées dans d'autres bibliothèques [VDY05].

Les noyaux proposés dans OSKI comprennent, entre autres, le calcul du produit matrice-vecteur creux (PMVC) et la résolution de systèmes triangulaires creux (RSTC). OSKI permet la sélection automatique de la structure de données et du code

à utiliser. Etant données une machine et une matrice creuse, cette sélection aide à la mise en œuvre rapide d'un noyau de calcul.

L'approche proposée consiste à l'adaptation automatique des paramètres d'un format de compression d'une matrice creuse donnée [VDY05].

Sélection automatique du format de compression

Sélection automatique du format de compression sur des architectures multi-cœurs

Dans [LZT12], les auteurs proposent une solution pour remédier à ce problème : un système qui permet de choisir dynamiquement le meilleur format de stockage ce qui offre des meilleures performances, pour le PMVC. Ce système, appelé SMAT (Sparse Matrix-vector product Auto-Tuner), fournit aux programmeurs une interface unique prenant en entrée une matrice creuse stocké selon le format CSR (Compressed Sparse Row). SMAT permet à l'utilisateur de choisir implicitement, quel que soit la matrice creuse en entrée, le meilleur format de stockage et l'implémentation la plus rapide. Ce processus de choix est effectué pendant l'exécution. Les auteurs s'intéressent en particulier aux méthodes multi-niveaux. Dans ce type de problème, les matrices traitées dans chaque niveau ne sont pas les mêmes. Elles sont créées par l'algorithme pendant le calcul. Ces nouvelles matrices sont de plus petites tailles et n'ont pas forcément la même structure que la matrice de départ. Ainsi, les auteurs proposent un système qui fait le choix du format pendant le calcul de ce qui permet d'avoir différentes combinaisons de formats. SMAT tire parti d'un modèle de *data mining*, qui est formulé sur la base d'un ensemble de paramètres de performance extrait de 2373 matrices, sélectionnées parmi la collection des matrices creuses de T. Davis, université de Florida [CTD], et ce dans le but de permettre une recherche rapide de la meilleure combinaison. SMAT est destiné pour des architectures multi-cœurs à mémoire partagée.

De même, dans [ArP10], les auteurs s'intéressent à l'étude de sélection du meilleur format de compression sur des architectures multi-cœurs non distribuées. Ainsi, ils décrivent un algorithme d'apprentissage permettant la détection automatique du format. Ils considèrent une matrice, ou un bloc de matrice, dont le PMVC sera exécuté sur des architectures différentes au cours du calcul. Pendant le calcul, le même noyau sera exécuté sur des architectures différentes en utilisant des formats de compression différents. Les auteurs proposent une méthode pour pouvoir choisir le format optimal suite à ces tests. La méthode proposée est basée sur l'utilisation des algorithmes d'apprentissage par renforcement qui apprennent suite à des expérimentations.

Le système proposé a la particularité de supporter 65 formats de compression, à savoir CSR, CSC, et BCSR et leurs dérivés. Dans le cas de BCSR, différentes versions avec des tailles de blocs différentes sont étudiées. Le système possède trois types d'entrée : une matrice creuse, un vecteur et le nombre des multiplications à effectuer. Une analyse de la matrice permet d'extraire un nombre de caractéristiques à utiliser dans l'étape de l'apprentissage. Ces attributs sont le nombre de lignes, le nombre de colonnes, le nombre des éléments non nuls, le nombre moyen de voisins et la déviation npr. Le nombre voisin d'un élément non nul correspond au nombre des éléments adjacents non nuls (entre 0 et 8). La déviation npr correspond à l'écart-type du nombre des éléments non nuls dans toutes les lignes de la matrice en entrée [ArP10].
Un simulateur a été utilisé pour l'évaluation de l'algorithme proposé afin d'étudier la sélection du meilleur format de compression. Un ensemble de tests de calcul du produit matrice vecteur creux, sur un ensemble de matrices creuse, est effectué en utilisant les 65 formats. Le temps de calcul de chaque multiplication est enregistré et il est fourni par la suite avec le simulateur. Les tests sont effectués sur deux architectures différentes [ArP10].

Sélection automatique du format de compression sur des architectures GPU

D'autres travaux se sont concentrés sur le choix du meilleur format de compression pour des architectures GPU.

f) *Modèles statiques*

Dans [NRR14], les auteurs proposent un modèle statique pour la prédiction du format de compression optimal, et ce dans le cas de calcul du PMVC. Le modèle de prédiction proposé est basé sur les caractéristiques de la matrice en entrée et les communications CPU-GPU.

Le calcul du PMVC sur une architecture GPU passe par trois étapes. La première étape est une étape de pré-traitement qui se fait au niveau CPU et qui correspond au stockage de la matrice en entrée selon le format désiré (conversion entre formats). La deuxième étape est l'étape de transfert des données de calcul (une matrice et un vecteur) vers le GPU. La dernière étape est le calcul du PMVC. Les auteurs cherchent alors à avoir un système qui prend en considération la matrice en entrée ainsi que le '$overhead$' provoqué par les trois étapes de calcul.

Ainsi, pour la prédiction du format de compression, les auteurs commencent par une analyse de la matrice où un nombre d'attributs sont définis, à savoir, le nombre de

lignes, le nombre de colonnes, le nombre des éléments non nuls, le nombre maximal des éléments non nuls par ligne et la moyenne des éléments non nuls par ligne. Selon les valeurs de ces attributs, une affinité de la matrice à un format est définie. Ainsi, une liste de format est proposée dans un ordre décroissant de priorité. Les formats étudiés sont COO, CSR, ELL et HYB.

D'un autre côté, le temps de communication CPU-GPU est calculé d'une manière statique, et ce en fonction des tailles des données associées à chaque format et la bande passante du bus PCI-E[1]. Par la suite, un format est choisi à partir de la liste proposée à l'étape précédente, et ce en comparant leurs temps de communication.

Dans le cas où les deux métriques définies précédemment ne permettent pas la décision, le temps de transformation de la matrice en entrée sera considéré comme critère de choix [NRR14].

Modèles probabilistes

Un modèle probabiliste pour la prédiction des performances du PMVC sur GPU, et par la suite le format de compression, est présenté dans [LYL15]. Ce modèle est basé sur la fonction de masse de probabilité qui reflète la distribution des éléments non nuls dans la matrice. En effet, une fonction de masse de probabilité (FMP) permet de donner la probabilité pour qu'une variable discrète soit égale à une variable donnée. Dans [LYL15], cette variable discrète correspond au nombre des éléments non nuls par ligne.

Etant donnée une matrice creuse en entrée, la première étape consiste à déterminer la FMP de cette matrice. Dans l'étape suivante, une formule d'estimation des performances est déterminée pour chaque format (CSR, COO, ELL et HYB), et ce en se basant sur la FMP et les informations sur la structure de stockage.

La dernière étape consiste à la prédiction des performances du calcul du PMVC pour chaque format. Etant donnée un format de compression, les performances du PMVC y associées dépendent des paramètres de l'architecture GPU et de la formule d'estimation des performances du format.

Lors de la définition des paramètres GPU, deux métriques sont à prendre en considération : le temps des communications et le temps de calcul. Le temps des communications correspond au transfert des données entre le CPU et le GPU. Le

[1] PCI-E (*'Peripheral Component Interconnect-Express'*) est le bus qui interconnecte le CPU au GPU.

temps de calcul implique le temps nécessaire pour effectuer les opération arithmétiques (multiplication et addition) et le temps des accès mémoire en lecture ou en écriture [LYL15].

De même, dans [GuL16], les auteurs présentent un framework pour la prédiction des performances du PMVC, pour une architecture GPU, où un modèle probabiliste est utilisé pour l'analyse de la matrice cible. Les auteurs proposent une approche à modélisation des performances à deux étapes : une première étape hors ligne et une deuxième en ligne.

Dans la première étape, un benchmark de matrices est généré pour chaque architecture GPU. Les tailles des matrices dans ce benchmark sont choisies en fonction de la taille mémoire globale du GPU. Par la suite, une mesure des performances du PMVC est effectuée pour chaque matrice appartenant au benchmark. Les matrices sont stockées selon les formats de compression (CSR, COO, ELL et HYB). A ce niveau, les auteurs proposent d'établir un ensemble de relations linéaires permettant la prédiction des performances de calcul du PMVC. Ces relations sont établies en fonction des caractéristiques statistiques des matrices du benchmark et des performances mesurées du PMVC.

Etant donnée une matrice creuse cible, les auteurs proposent que la prédiction du format soit faite en ligne [GuL16]. Tout d'abord, un ensemble de caractéristiques de la matrice cible est extraite. Pour cela une méthode probabiliste est utilisée. Dans un deuxième lieu, une estimation des performances de calcul du PMVC, pour chaque format, est effectuée à l'aide de l'ensemble des relations [GuL16].

Modèles d'apprentissage automatique

D'autres travaux se sont concentrés sur la prédiction du format de compression creux sur des architectures GPU en utilisant différents algorithmes du *machine learning* tels que les arbres de décision [SMP15], SVM [BJW16] [BJW16-a], KNN [LBE16] et PMC [BJW16].

Le *Tableau 4* représente une synthèse des travaux existants dans la littérature pour le choix automatique du meilleur format de compression. Les travaux existants utilisent généralement le calcul du PMVC sous différents formats de compression. Les formats les plus utilisés sont CSR, COO, ELL et HYB (dans le cas des GPU). Le processus de choix du format le plus

adéquat est basé généralement sur l'utilisation des méthodes probabilistes ou les algorithmes d'apprentissage.

1. Système intelligent de détection automatique du meilleur format de compression

Solution proposée

Le choix du format de compression permettant de donner la meilleure performance est un processus assez critique et qui dépend de plusieurs paramètres. Etant donnée la complexité de cette problématique, nous avons besoin de nouvelles techniques permettant de sélectionner, pour une matrice particulière, le format de stockage qui lui convient le mieux. L'objectif est de détecter automatiquement le meilleur format de compression d'une matrice creuse pour une architecture, une méthode numérique et un modèle de programmation données et ce, en utilisant des techniques d'*autotuning* ou encore du *smart-tuning*.

Conformément aux travaux présentés dans la section 0, les systèmes proposés dans la littérature prennent en compte l'architecture utilisée et les caractéristiques de la matrice en entrée. Nous proposons d'ajouter l'expertise utilisateur pour pouvoir ajouter d'autres facteurs pouvant impacter les performances du noyau de calcul tels que le modèle de programmation (*Figure **Error! No text of specified style in document.**-38*).

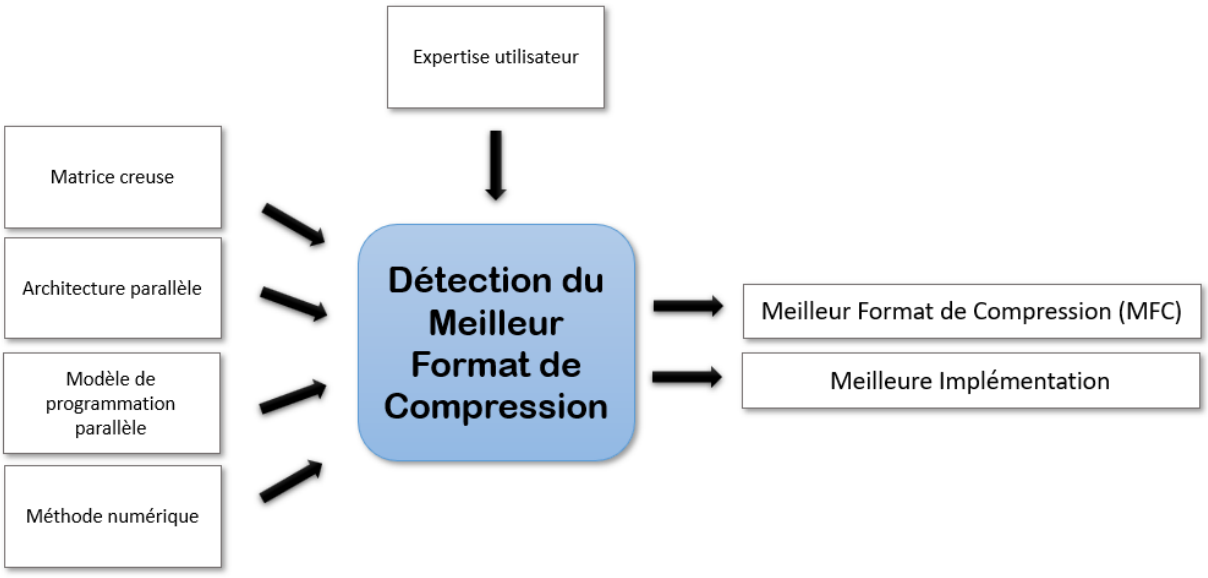

*Figure **Error! No text of specified style in document.**-38 Modélisation simplifiée du processus proposé pour la selection du meilleur format de compression*

Tableau 4 Tableau synthétique des travaux existants pour la selection du format de compression optimal

			Travaux	Noyau	Formats					Algorithme d'apprentissage
					CSR	COO	ELL	HYB	AUTRES	
Optimisation automatique des noyaux de calcul à l'exécution			[ImY04] [IYV01]	PMVC	*					
			[VDY05]	PMVC	*					
Sélection automatique du format de compression	Architectures multi-cœurs	Modèles d'apprentissage automatique	[LZT12]	PMVC	*	*	*		DIA	Arbre de décision
			[ArP10]	PMVC	*	*			BCSR	Algorithmes d'apprentissage par renforcement
	Architectures GPU	Modèles statiques	[NRR14]	PMVC	*	*	*	*		Modèle statique pour la prédiction
		Modèles probabili	[GuL16]	PMVC	*	*	*	*		

			[LYL15]	PMVC	*	*	*	*		
		Modèles d'apprentissage automatique	[SMP15]	PMVC	*	*	*	*	Hyb-MU	Arbre de décision
			[BJW16]	PMVC	*	*	*	*		SVM PMC
			[LBE16]	PMVC	*	*	*		ELL-BRO SELL-C- σ	KNN
			[BJW16-a]	PMVC	*	*	*	*		SVM

Conception du système intelligent pour la détection du meilleur format de compression

Nous présentons dans cette partie une description détaillée de notre système intelligent proposée pour la sélection du meilleur format de compression creux. Nous commençons d'abord par la présentation des principales composantes du système. Nous détaillons par la suite ses différents modules et leurs fonctionnalités.

Les composantes de notre système

Notre système de sélection du Meilleur Format de Compression (MFC) est composé d'un ensemble de bases que nous détaillons dans ce qui suit.

g) Base de tests

Une version initiale est fournie avec notre système. Initialement, cette base contient l'ensemble des tests réalisés sur différentes architectures. Ce sont les résultats d'application de différents noyaux de calcul sur des matrices creuses compressées selon différents formats de compression. Une mise à jour de cette base est possible si l'utilisateur du système le souhaite.

Base de règles de décision

Une version initiale est fournie avec notre système. Initialement, cette base contient un ensemble de règles de décision permettant la sélection du meilleur format de compression, et ce, suite à une analyse faite sur la base de tests initiales. Plusieurs méthodes de l'intelligence artificielle peuvent être utilisées pour la génération de cette base. Nous présentons dans le dernier chapitre de ce manuscrit une étude de cas où nous utilisons les algorithmes du *machine learning* pour la génération de la base de règles de décision.

Base de caractéristiques

La base de caractéristiques est utilisée pour sauvegarder les informations sur les données à utiliser lors des nouveaux tests (caractéristiques de la matrice, caractéristiques de l'architecture cible, etc.). Lors de l'installation de notre système, il est possible d'enrichir cette base par des caractéristiques spécifiques de l'architecture.

Base de matrices

La base de matrices contient l'ensemble des matrices de tests. Ces matrices proviennent de plusieurs domaines d'application. La base contient des matrices ayant différentes tailles, densités, structures, etc. Nous présentons dans le dernier chapitre de ce manuscrit l'ensemble des matrices fournies initialement.

Base de formats

La base de formats contient un ensemble de formats de compression de matrices creuses. Cet ensemble représente les formats supportés par notre système.

Base d'algorithmes

Cette base contient une liste d'algorithmes relatifs à différents noyaux de calcul. Pour chaque noyau sont fournies différentes versions de l'algorithme (avec différentes optimisations) associées à différents formats de stockage creux (appartenant à la base des formats).

Modules de notre système

Nous présentons dans ce qui suit les modules de notre système de détection automatique du meilleur format de compression creux. Le système est composé de trois modules principaux : module « installation », module « utilisation » et un module « utilisateur expert ».

h) Module « Installation »

C'est le module de l'installation du système (Figure Error! ***No text of specified style in document.-39***). Un ensemble de bases est fourni avec notre système : base de tests, base de règles de décision, base de matrices, base de formats et base d'algorithmes.

Lors de l'installation, il y aura création de la base des caractéristiques. Cette dernière est extraite des paramètres architecturaux de la plateforme utilisée.

En outre, un ensemble de tests est réalisé et ce en utilisant l'ensemble des algorithmes et des matrices creuses fourni avec notre système. La mise à jour de la base des règles de décision est donc effectuée.

Module « Utilisation » :

Le module « utilisation » est le module principal de notre système, c'est le module utilisé pour la sélection du meilleur format de compression associé à une matrice donnée (Figure Error! ***No text of specified style in document.-40***).

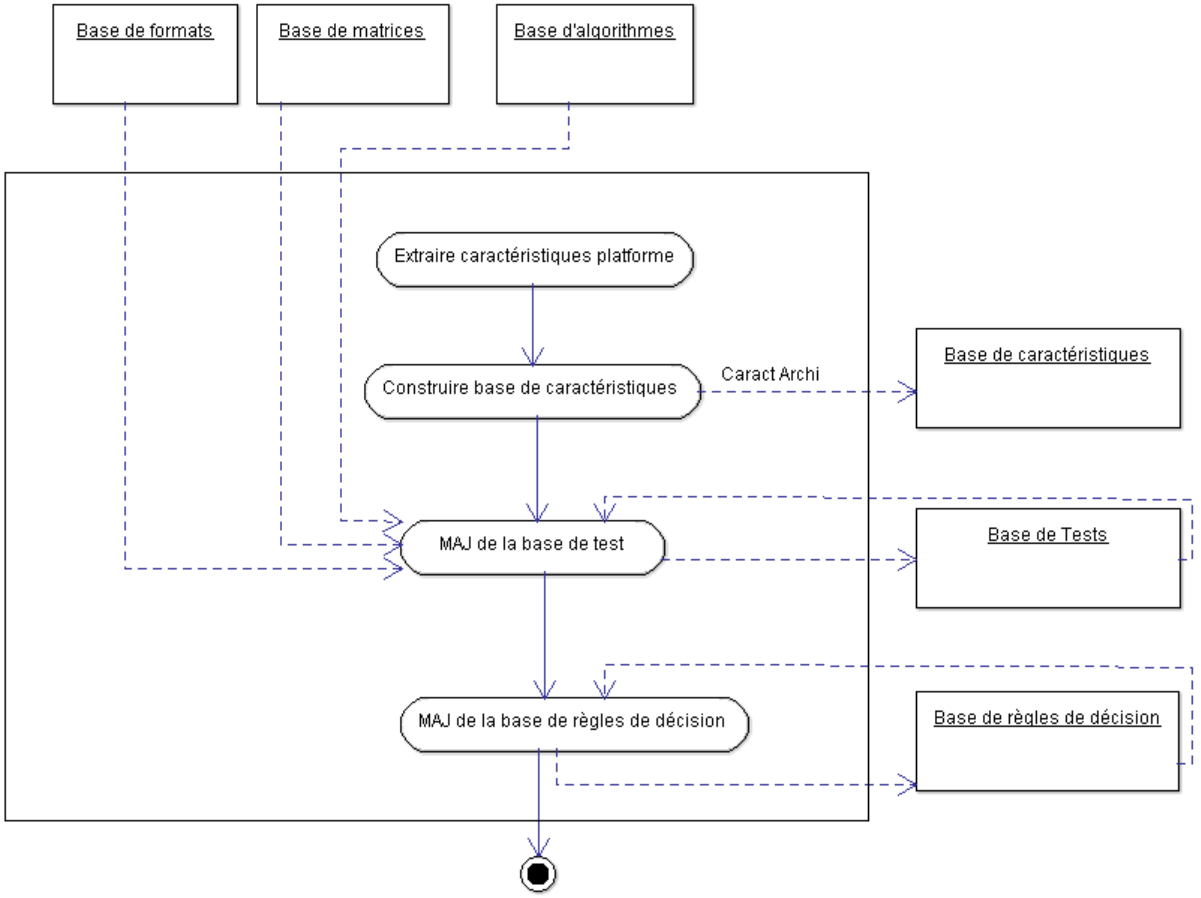

*Figure **Error! No text of specified style in document.**-39: module « installation »*

Etant donnée une matrice creuse, une méthode numérique, une architecture et un modèle de programmation parallèle, notre système doit être capable de déterminer le meilleur format de compression (MFC) associé. Le choix du MFC passe par les étapes suivantes :

- Etape 0 : Définir les entrées du système : une matrice creuse, une méthode numérique, une architecture et un modèle de programmation parallèles.
- Etape 1 : Mise à jour de la base des caractéristiques. Pour cela, il faut :
  - Analyser la matrice creuse et extraire un ensemble de caractéristiques (structure, taille, densité, …). L'ensemble des caractéristiques à extraire est à déterminer par le système en fonction des entrées et de la base des règles de décision.
  - Déterminer les caractéristiques de la plateforme cible (mémoire, fréquence, nombre processeurs, etc.). Si le calcul est à effectuer sur la même machine sur laquelle le système est installé, ces informations se trouvent déjà dans la base des caractéristiques. Sinon, ils peuvent être

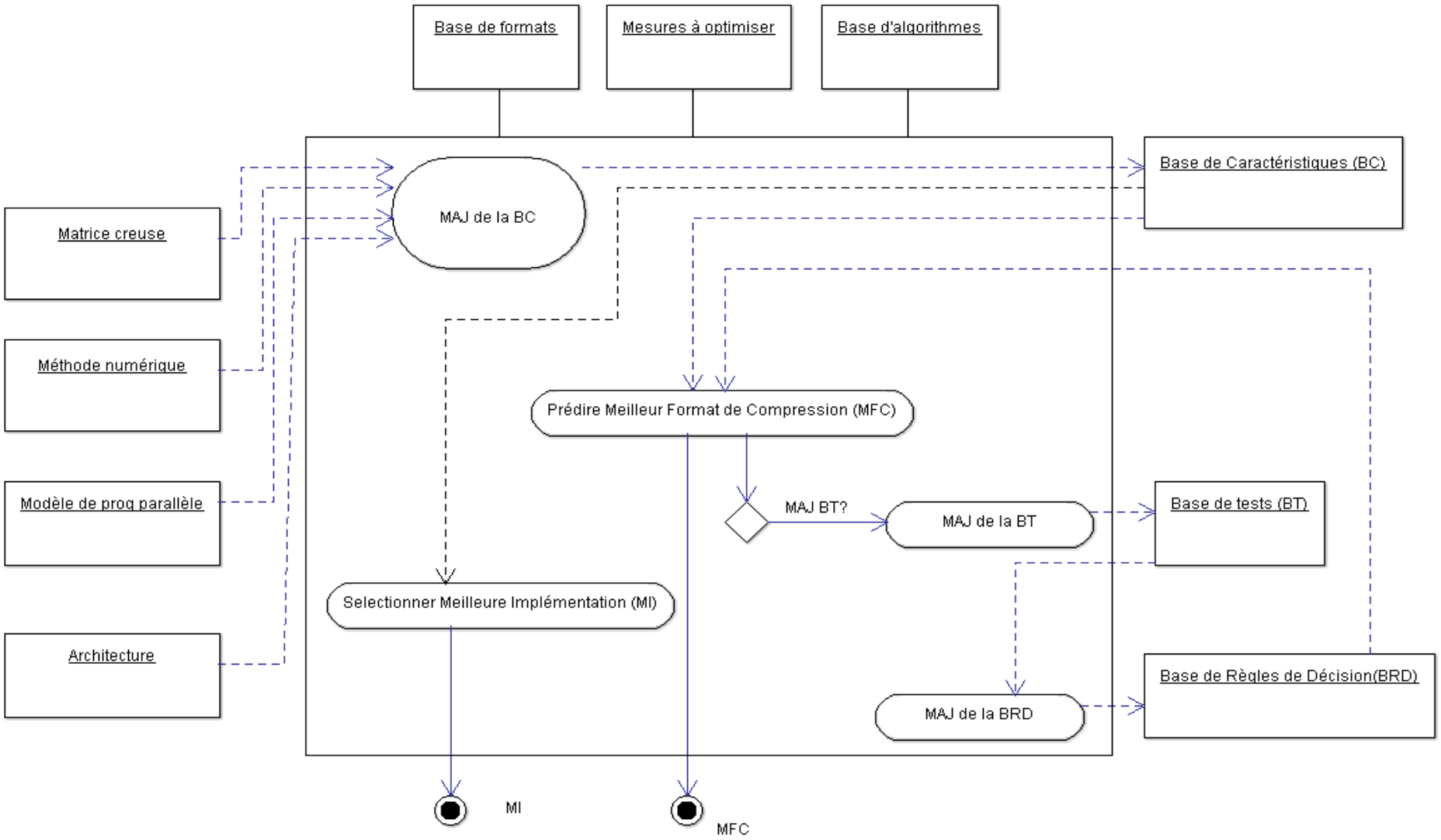

Figure ***Error! No text of specified style in document.****-40 Module Utilisation*

fournies directement par l'utilisateur ou en effectuant une analyse de la plateforme cible.

- Analyser la méthode numérique et extraire un ensemble de caractéristiques, telle que le nombre des accès mémoires.

- Etape 3 : Etant données la base des caractéristiques, le meilleur format de compression est prédit en se basant sur l'ensemble des règles (base des règles de décision).
- Etape 4 : si l'utilisateur le souhaite, la meilleure implémentation associée aux données en entrées est fournie.
- Etape 5 : une mise à jour de la base des règles de décision est possible si l'utilisateur le souhaite. Dans ce cas, le nouveau cas traité est ajouté à la base des tests. La base des règles de décision est alors régénérée en effectuant une nouvelle analyse de cette dernière.

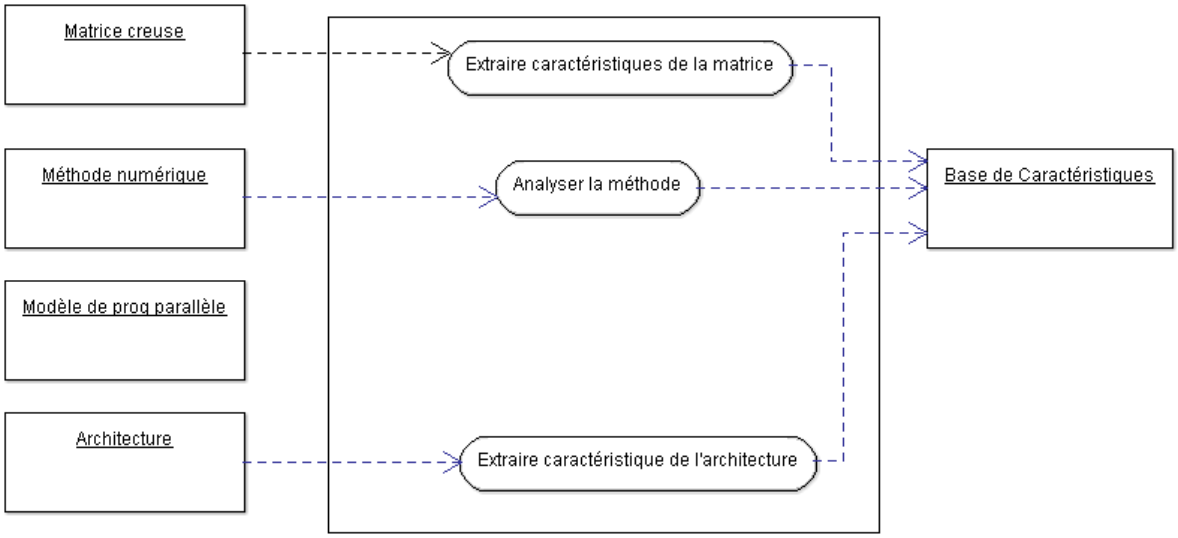

*Figure **Error! No text of specified style in document.**-41: modèle " MAJ base de caractéristiques "*

*Module « utilisateur expert » (Figure Error! **No text of specified style in document.**-42) :*

C'est le module utilisé par l'expert pour la mise à jour de notre système (base de formats, base d'algorithmes, etc.). Nous présentons dans ce qui suit les fonctionnalités possibles.

- L'ajout d'un nouveau format de compression qui sera supporté par notre système. A chaque ajout d'un nouveau format, il faut mettre à jour la base des algorithmes

(versions associées au nouveau format). Dans ce cas de nouveaux tests doivent être effectués et par la suite une mise à jour de la base des règles de décision.

- L'ajout d'une nouvelle version d'un algorithme associé à un format existant. Dans ce cas de nouveaux tests doivent être effectués et par la suite une mise à jour de la base des règles de décision.
- L'ajout d'un nouveau modèle de programmation. Dans ce cas l'expert doit fournir un ensemble de caractéristiques (métriques d'évaluation) associées à ce modèle. Dans ce cas de nouveaux tests doivent être effectués et par la suite une mise à jour de la base des règles de décision.

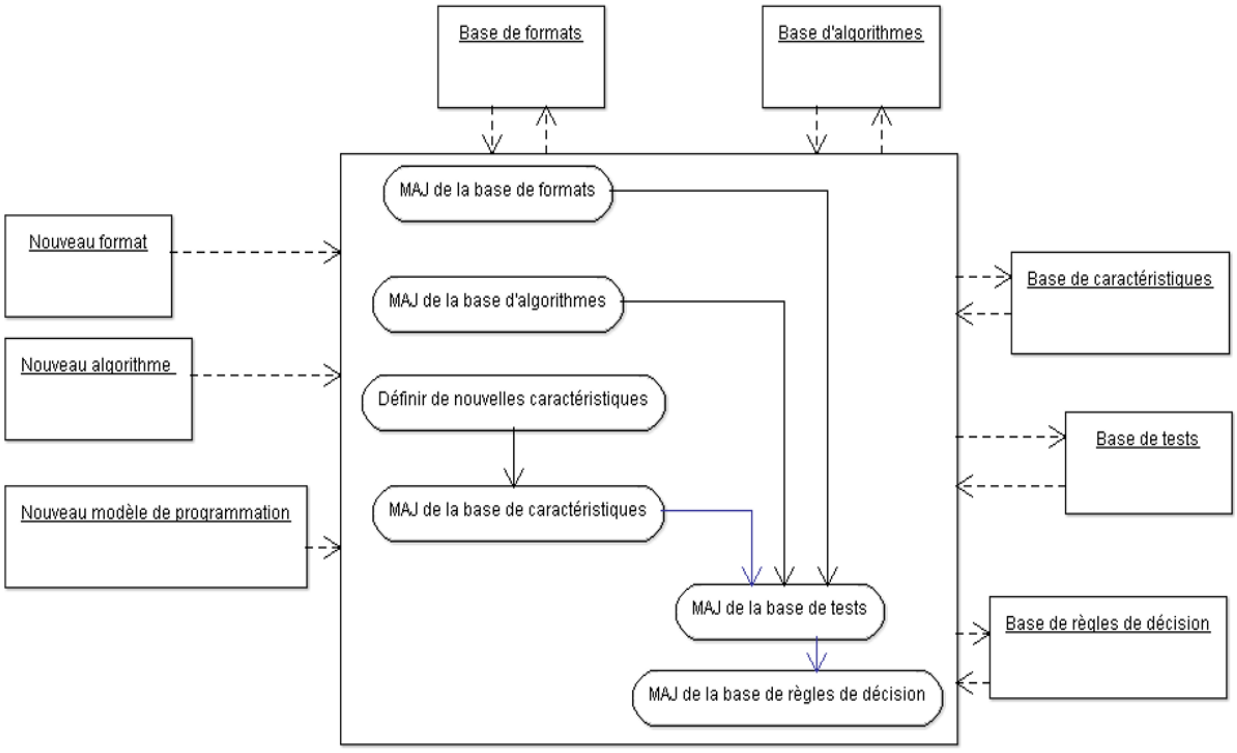

*Figure **Error! No text of specified style in document.**-42: Module " utilisateur expert"*

Conclusion

Nous avons montré dans ce chapitre la difficulté du choix du format de compression creux vu la diversité des paramètres qui interviennent dans ce processus. Nous avons alors passé en revue les travaux proposés dans la littérature pour la sélection automatique du meilleur format de compression. Nous avons mis en évidence que les méthodes actuelles reposent sur une analyse des données de la matrice ou du noyau de calcul uniquement. Nous avons ainsi

adopté une approche orientée composante pour la proposition d'un système de détection automatique du meilleur format de compression. Afin de pouvoir prendre en compte tous les paramètres impactant le choix du format de compression, nous avons introduit l'expertise utilisateur dans notre système.

Nous proposons dans le chapitre suivant d'évaluer notre proposition. Pour cela, nous présentons notre approche dans le cas où l'utilisateur fournit des informations sur des métriques issues du modèle data parallel et de l'analyse de l'algorithme. Une fois que ces métriques sont définies, il est possible de coupler cela avec des techniques d'aide à la décision pour détecter le meilleur format de compression.

ETUDE DES METRIQUES POUR LE CHOIX DU MEILLEUR FORMAT DE COMPRESSION

Introduction

Nous nous intéressons dans ce chapitre à la présentation d'un cas d'étude pour la détection automatique du meilleur format de compression. La section 0 est dévolue à la présentation et la définition des différents types d'entrée à prendre en compte dans cette étude. Nous détaillons en particulier le modèle de programmation data parallel ainsi que les méthodes numériques, à savoir le schéma de Horner et le PMVC.

Une étude de cas est présentée dans la section 0 de ce chapitre. Nous présentons en premier lieu une description détaillée de notre analyse du modèle data parallel. Deux approches différentes sont présentées. La première consiste à associer un élément non nul à chaque processeur virtuel. La deuxième approche consiste à attribuer à chaque processeur virtuel un vecteur i.e. une ligne ou une colonne (selon la direction de stockage des éléments non nuls) de la matrice. Dans une deuxième partie, un ensemble de métriques est défini suite à une analyse des algorithmes. Cette analyse est liée essentiellement aux accès mémoire.

Présentation de notre cas d'étude

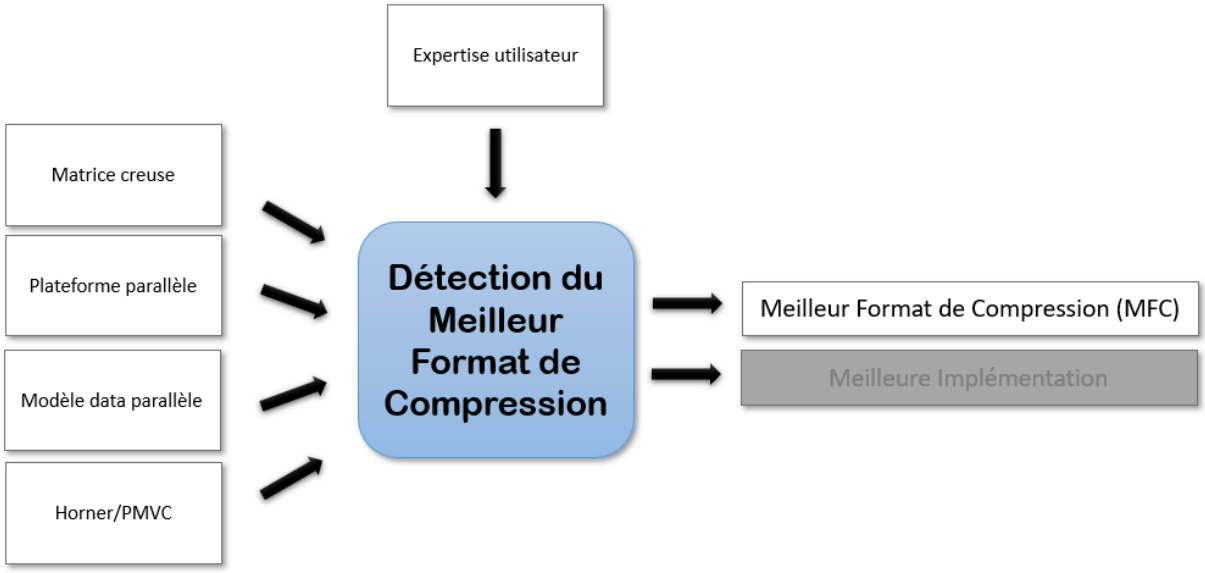

*Figure **Error! No text of specified style in document.**-43 Présentation simplifiée de notre cas d'étude*

Nous nous intéressons dans ce chapitre à la présentation d'une étude de cas de notre système proposé pour la détection du meilleur format de compression (MFC). Etant donnés une matrice creuse et une architecture parallèle, nous cherchons à prédire le MFC pour le calcul du schéma de Horner, et par la suite du PMVC. Notons que nous utilisons le modèle *data parallel* pour la parallélisation des deux méthodes numériques (*Figure **Error! No text of specified style in***

document.-43). Dans ce cas, l'expertise utilisateur est illustrée dans le fait de fournir des informations liées à l'analyse du modèle de programmation et à l'analyse des algorithmes. Cette étape d'analyse permet d'extraire les métriques permettant le choix du MFC. Notre étude sera limitée à quatre formats de compression, à savoir CSR, CSC, ELL et COO. En effet, ce sont les formats les plus utilisés dans la littérature pour la compression de matrices creuses. A ce niveau, nous ne nous intéressons qu'à la sélection du MFC. L'étude des implémentations sera effectuée dans des travaux futurs.

Présentation du modèle de programmation data parallel

L'idée est de concevoir une géométrie virtuelle G, à une ou deux dimensions, dont chaque élément est un processeur virtuel Pv. Les données à traiter sont par la suite affectées à cette géométrie. La taille et/ou le type de ces données est à déterminer par l'utilisateur (concepteur de la géométrie). Le nombre de processeurs virtuels de la géométrie (taille de la géométrie) est supposé égal au nombre des données. Néanmoins, la taille des données est souvent beaucoup plus grande que le nombre de processeurs physiques P. Il faut donc simuler de nombreux processeurs virtuels sur chaque processeur physique. Dans ce cas, le parallélisme entre les opérations effectuées sur le même processeur physique est détruit [PeE06].

Exemples d'algorithmes data parallel

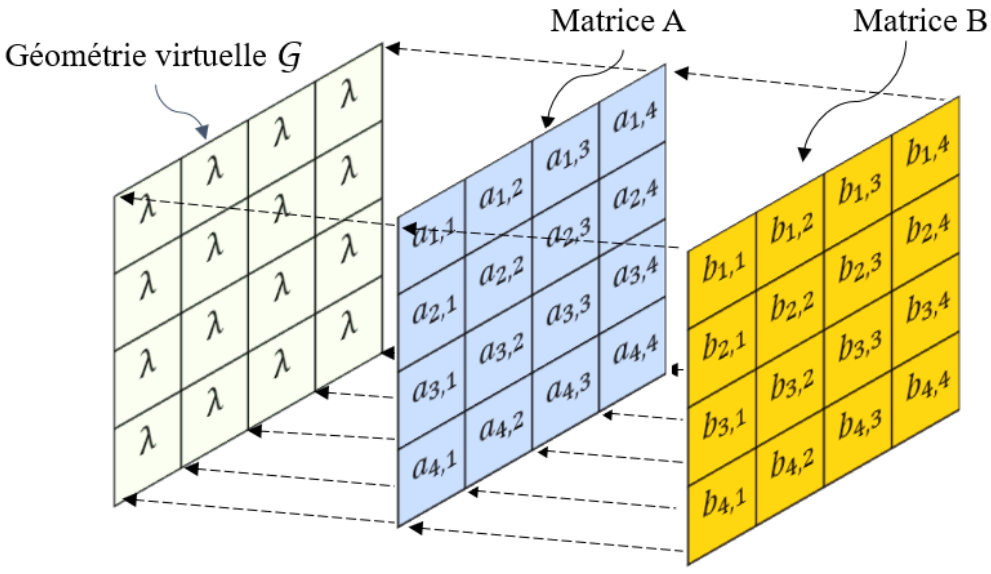

*Figure **Error! No text of specified style in document.**-44 Exemple d'une géométrie virtuelle pour le calcul de APλA*

i) *$APλA$: Matrice + (scalaire × Matrice)*

Soient A et B deux matrices carrées ($A, B \in \mathbb{R}^{n\times n}$). Nous présentons dans cette partie un exemple simple d'utilisation de la géométrie virtuelle. Cet exemple correspond au calcul de la somme de deux matrices. Soit G une géométrie virtuelle à deux dimensions de taille $(n \times n)$. Chaque élément de la géométrie est représenté par ses indices i et j.

Les matrices A et B sont affectées à la géométrie tels que chaque élément $a_{i,j}$ et chaque élément $b_{i,j}$ est affecté à l'élément (i,j) de la géométrie. Chaque processeur virtuel (i,j) de la géométrie calcule donc une addition de deux scalaires $(a_{i,j} + b_{i,j})$. L'addition des deux matrices A et B est faite donc en une seule opération d'addition data parallel [PeE06]. Pour calculer $A + \lambda \times B$, avec λ un scalaire, on affecte λ à chaque élément de la géométrie et on calcule alors une opération triadique data parallel (+, ×). En d'autres termes, chaque processeur virtuel (i,j) calcule une seule opération $a_{i,j} + \lambda \times b_{i,j}$ (*Figure* ***Error! No text of specified style in document.****-44*).

Ax : Matrice× Vecteur

Soient A une matrice carrée, $A \in \mathbb{R}^{n\times n}$, et x un vecteur de taille n. Nous nous intéressons au calcul du produit matrice vecteur $y = A \times x$, avec y un vecteur résultat de taille n. Soit G une géométrie virtuelle à deux dimensions de taille $(n \times n)$. La matrice A est affectée à la géométrie telle que chaque élément de la matrice $a_{i,j}$ est affecté au processeur (i,j) de G. Le vecteur x doit être projeté sur chaque ligne de la géométrie. Dans une première étape, chaque processeur virtuel (i,j) calcule un résultat partiel $Y_{i,j} = a_{i,j} \times x_j$, avec Y une matrice carrée $\in \mathbb{R}^{n\times n}$ (*Figure* ***Error! No text of specified style in document.****-45*). A ce stade, dans chaque $i^{ème}$ ligne de la géométrie, une addition est effectuée, en $log_2(n)$ opérations data parallel, pour avoir la $i^{ème}$ composante de $A \times x$ (*Figure* ***Error! No text of specified style in document.****-46*) [PeE06].

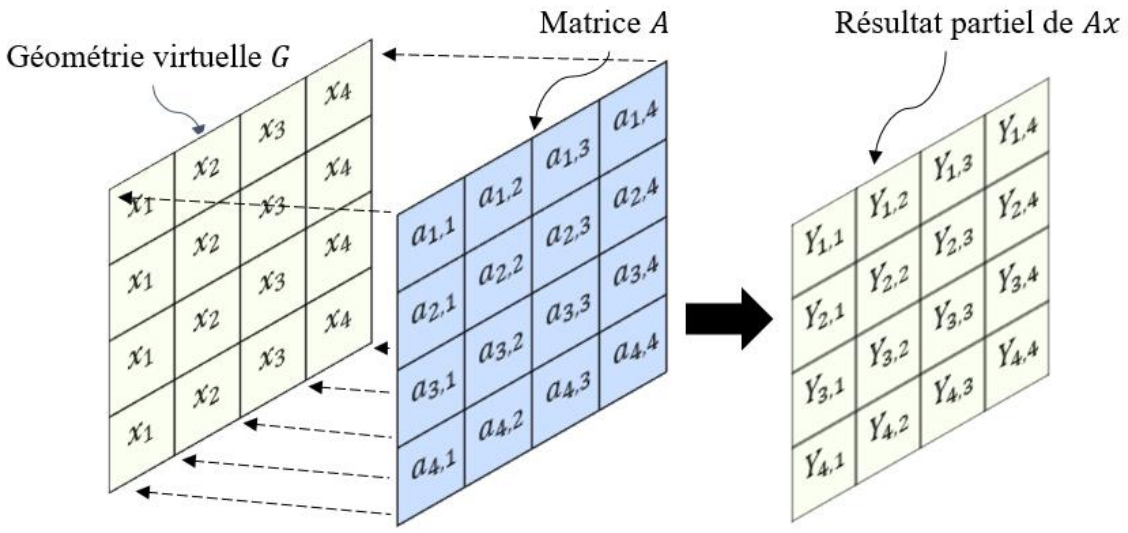

Figure ***Error! No text of specified style in document.****-45 Première étape dans le calcul de Ax*

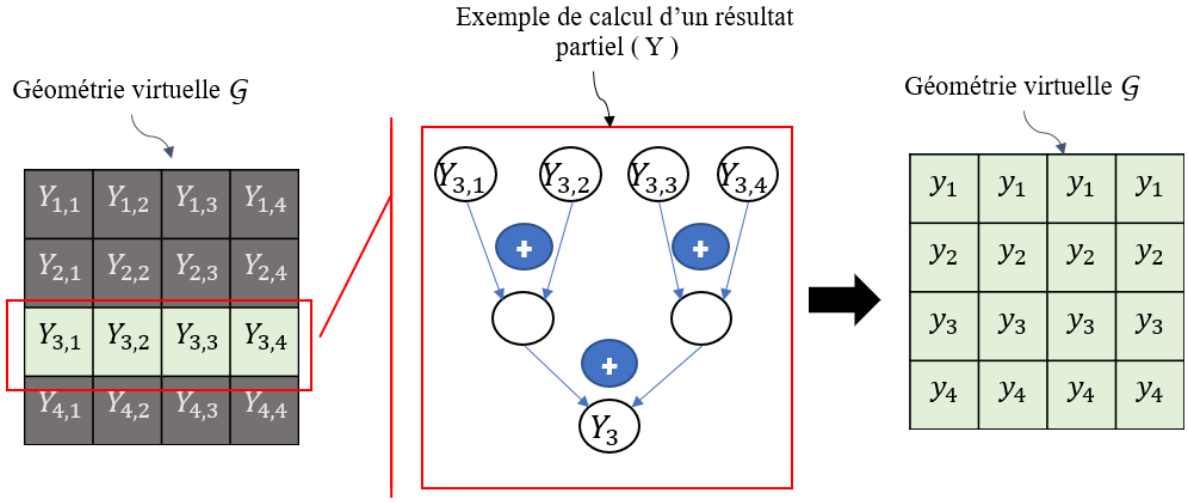

Figure ***Error! No text of specified style in document.****-46 Deuxième étape dans le calcul de AX*

Evaluation d'algorithmes data parallel

Pour l'évaluation des algorithmes data parallel, nous nous basons dans notre étude sur un certain nombre de paramètres présentés dans [PeE06]. Ces paramètres sont présentés sous quelques hypothèses :

- *Hypothèse 1* : Les données sont projetées sur une seule géométrie virtuelle,
- *Hypothèse 2* : Soit G une géométrie virtuelle de taille n. La complexité des opérations préfixe, tels que la somme des éléments d'une même ligne de variables data parallel

affectées à G, avec un résultat par ligne est $log_2(n)$. La complexité de cette opération est considérée la même que celle dans le cas où on a le résultat dans chaque processeur virtuel.

D'où alors la définition des paramètres suivants pour un algorithme data parallel et une géométrie virtuelle :

Pv: nombre de processeurs virtuels de la géométrie virtuelles,

cx : le nombre d'opérations data parallel (autres que les communications),

α : le nombre de processeurs virtuels concernés par une opération data parallel,

De même, au niveau de la machine physique, on a :

P : nombre de processeurs physiques de la machine parallèle ($Pv \gg$ P),

vpr : le ratio des processeurs virtuels avec

$$vpr = \frac{Pv}{\mathrm{P}}$$

Le rapport de calcul d'une opération data parallel *ops* est défini par :

$$\varphi_{ops} = \frac{\alpha_{ops}}{Pv} \leq 1$$

D'où la moyenne des ratios des calculs :

$$\bar{\varphi} = \frac{1}{cx} \sum_{ops=1}^{ops=cx} \varphi_{ops} \leq 1$$

$\bar{\varphi}$ dépend uniquement de l'algorithme data parallel.

Pour un algorithme data parallel donné, le nombre maximal de données impliquées dans une opération data parallel est appelé degré de parallélisme, noté D avec

$$D = \max_{ops=1,cx} \alpha_{ops}$$

Nous présentons dans le *Tableau 5* les résultats numériques qui correspond à l'application du modèle data parallel au algorithmes $AP\lambda A$ et Ax dans le cas où A est une matrice dense.

Tableau 5 métriques data parallel de $AP\lambda A$ et Ax

	$\boldsymbol{Pv}$	$\boldsymbol{cx}$	$\boldsymbol{\bar{\varphi}}$	$\boldsymbol{D}$
APλA	n^2	1 (+, ×)	1	n^2
Ax	n^2	1	1	n^2

Algorithme du schéma de Horner

Soit A une matrice creuse, $A \in \mathbb{R}^{n \times n}$, ayant nnz éléments non nuls. Soient x et y deux vecteurs de tailles n. Le calcul du schéma de Horner consiste à calculer un vecteur résultat y tel que :

$$y = A \times (A \times x + x)$$

Le calcul du schéma de Horner est basé principalement sur le calcul du produit matrice-vecteur creux (PMVC). Ce noyau est invoqué deux fois. ***Algorithme*** *Error! No text of specified style in document.**-3*** présente l'algorithme de calcul du schéma de Horner. La fonction $PMVC(A,x)$ correspond au calcul d'un Produit Matrice Vecteur Creux impliquant la matrice A et le vecteur x. Notons que l'algorithme de calcul du PMVC dépend du format de compression utilisé. Ainsi, nous présentons dans la section suivante les différentes versions de cette algorithme associés aux différents formats de compression à étudier.

Algorithme *Error! No text of specified style in document.**-3** Calcul du schéma de Horner*
w=x ; // w un vecteur de taille n *w=w+PMVC(A,x) ;* *y=PMVC(A,w) ;*

Algorithmes du PMVC pour les formats de compression étudiés

La structure de l'algorithme du PMVC dépend du format de compression utilisé. Ainsi, nous présentons dans cette section quatre versions différentes du PMVC optimisées associées aux formats de compression à étudier CSR, CSC, ELL et COO. Ces algorithmes sont représentés respectivement par ***Algorithme*** *Error! No text of specified style in document.**-4***, ***Algorithme*** *Error! No text of specified style in document.**-5***, ***Algorithme*** *Error! No text of specified style in document.**-6*** et ***Algorithme*** *Error! No text of specified style in document.**-7***. Pour l'optimisation de ces versions du PMVC, nous avons appliqué deux techniques d'optimisation du code :

- *Remplacement scalaire* : remplacer un accès indirect (accès à un élément de tableau) par un accès à une variable scalaire afin d'accélérer les accès.
- Elimination des invariants des boucles : l'objectif de cette optimisation est d'éliminer les instructions inutiles en éliminant les invariants de boucle.

PMVC associé au format de compression CSR

Soit A une matrice carrée, $A \in \mathbb{R}^{n\times n}$, stockée selon le format de compression CSR (voir Chapitre1, la section 2.1.3 -b). Les éléments non nuls sont stockés dans val. Les indices de colonne sont stockés dans ind_col . Un tableau d'entiers $ligne\_ptr$ contient les pointeurs sur le début de chaque ligne dans les tableaux val et ind_col. ***Algorithme*** *Error! No text of specified style in document.**-4*** représente l'algorithme du PMVC associé au format CSR. La complexité de calcul est de $O(nnz)$.

Algorithme *Error! No text of specified style in document.**-4** PMVC associé au format CSR*

```
POUR i = 1, n
    s = 0;
    e = ligne_ptr [i + 1] − 1;
    POUR j = ligne_ptr[i], e
        indX = ind_col[j];
        s = s + val[j] × x[indX];
    FIN POUR
    y [i] = s;
FIN POUR
```

PMVC associé au format de compression CSC

Soit A une matrice carrée, $A \in \mathbb{R}^{n \times n}$, stockée selon le format de compression CSC (voir Chapitre1, la section 2.1.3 -c). Les éléments non nuls sont stockés dans val. Les indices de lignes sont stockés dans ind_ligne . Un tableau d'entiers col_ptr contient les pointeurs sur le début de chaque colonne dans les tableaux val et ind_ligne. ***Algorithme*** *Error! No text of specified style in document.**-5*** représente l'algorithme du PMVC associé au format CSC. La complexité de calcul est de $O(nnz)$.

PMVC associé au format de compression COO

Soit A une matrice carrée, $A \in \mathbb{R}^{n \times n}$, stockée selon le format de compression COO (voir Chapitre1, la section 2.1.3 -a). Les éléments non nuls sont stockés dans val. Les indices de lignes et de colonnes sont stockés respectivement dans les tableaux ind_ligne et ind_col. ***Algorithme*** *Error! No text of specified style in document.**-6*** représente l'algorithme du PMVC associé au format COO. La complexité de calcul est de $O(nnz)$.

Algorithme *Error! No text of specified style in document.**-5** PMVC associé au format CSC*

```
POUR i = 1, n
    w = x[i];
    e = col_ptr [i + 1] − 1;
    POUR j = col_ptr[i], e
        indR = ind_ligne[j] ;
        y [indR] = y [indR] + val[j] × w ;
    FIN POUR
    y [i] = s ;
FIN POUR
```

Algorithme *Error! No text of specified style in document.**-6** PMVC associé au format COO*

```
POUR i = 1, nnz
    indR = ind_ligne[i]
    y[indR] = y[indR] + val[i] × x [ind_col[i]] ;
FIN POUR
```

PMVC associé au format de compression ELL

Soit A une matrice carrée, $A \in \mathbb{R}^{n \times n}$, stockée selon le format de compression ELL (voir Chapitre1, la section 2.1.3 -b). Les éléments non nuls sont compressés par ligne et stockés dans une matrice val de dimension $n \times nz_{maxR}$. Une matrice ind, de même dimension que val, contient les indices des colonnes de chaque élément dans val. ***Algorithme*** *Error! No text of specified style in document.**-7*** représente l'algorithme du PMVC associé au format ELL. La complexité de calcul est de $O(nnz)$.

Algorithme *Error! No text of specified style in document.**-7** PMVC associé au format ELL*

```
POUR i = 1, n
    s = 0;
    j = 0;
    TANT QUE (j < nz_maxR ET 0 < val[i][j])
        s = s + val[i][j] × X[ind[i][j]];
        j++;
    FIN TANT QUE
    y [i] = s ;
FIN POUR
```

Analyse théorique

Nous présentons dans cette section notre analyse théorique associée au cas d'étude proposé. Nous commençons par une analyse du modèle *data parallel* dans le cas de calcul du schéma de Horner afin d'extraire un ensemble de métriques. Deux approches sont présentées : données scalaires dans le cas où nous traitons les éléments non nuls de la matrice, et une deuxième approche où nous traitons des vecteurs (lignes et colonnes de la matrice) au lieu des scalaires. Dans la deuxième partie de cette section, nous présentons une analyse des algorithmes du PMVC afin d'extraire un ensemble complémentaire de métriques. Après la description de

l'analyse du modèle d'expression de parallélisme, Nous présentons notre premier processus de décision déduit de l'analyse du modèle *data parallel*.

Analyse du modèle data parallel pour le schéma Horner

Soit A une matrice creuse, $A \in \mathbb{R}^{n \times n}$, ayant nnz éléments non nuls. Soient x et y deux vecteurs de tailles n. Nous présentons dans cette partie notre méthode utilisée pour la sélection du Meilleur Format de Compression (MFC), entre CSR, CSC, COO et ELL, et ce dans le cas de calcul du schéma de Horner (schéma 2) $y = A \times (A \times x + x)$. Pour faire, nous nous basons sur l'extraction des métriques relatives au modèle de programmation *data parallel* dans le cas du schéma de Horner. Nous commençons par un premier niveau de représentation des données : les données scalaires où chaque élément de la matrice est géré individuellement. Nous observons que, selon ce modèle de représentation, les métriques extraites ne sont pas suffisantes pour prendre une décision. Ainsi, nous introduisons un second niveau de présentation de données où nous traitons des vecteurs (lignes /colonnes) au lieu de scalaires[MHD16] [MHD17].

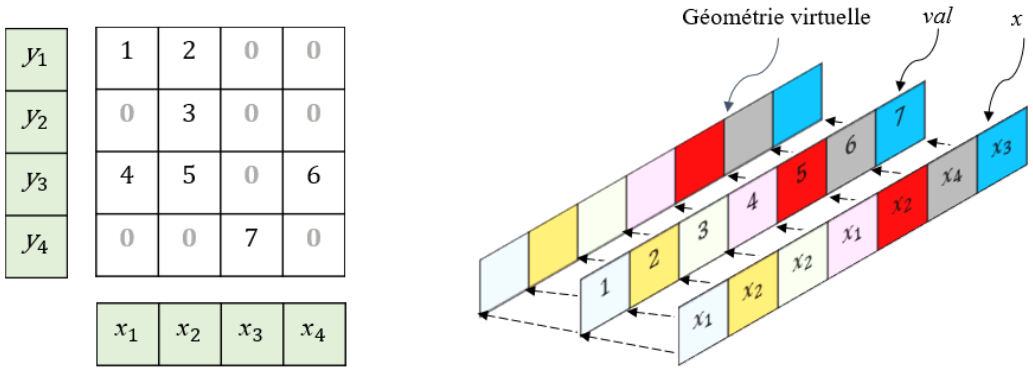

Figure ***Error! No text of specified style in document.****-47 : Affectation des données pour le format CSR*

Données scalaires

Dans ce premier niveau de représentation de données, nous considérons que chaque élément de la matrice creuse (donnée scalaire) est affecté à un processeur de la géométrie virtuelle G_{scal} à une ou deux dimensions (selon le format de compression utilisé). En d'autres termes, chaque processeur virtuel traite une seule et unique donnée scalaire. Nous détaillons dans ce qui suit l'organisation de cette géométrie pour chaque format.

j) CSR

Soit G_{scal} une géométrie virtuelle à une dimension ($1D$) de taille nnz. Pour implémenter le schéma de Horner en utilisant le format CSR, chaque élément val_i ainsi que l'élément de x correspondant sont affectés au même élément i de G_{scal} (*Figure* ***Error! No text of specified style in document.****-47*).

Nous commençons par effectuer une opération de multiplication (×) impliquant val et x en utilisant une seule opération *data parallel*. A savoir, chaque processeur virtuel Pv_i de la géométrie G_{scal} calcule une opération scalaire :

$$z_i = val_i \times x_i$$

Avec z un vecteur de résultat partiel de taille nnz.

Soit w un vecteur de taille n, pour calculer $w = A \times x$, les éléments de z appartenant à une même ligne k de la matrice A sont additionnés (on note cette opération red) dans w_k en $log_2(\mathrm{nz}_{\mathrm{maxR}})$ (*Figure* ***Error! No text of specified style in document.****-48* (a)). Ceci est dû au fait que la complexité de calcul dépend de la ligne la plus chargée (c'est la ligne ayant $\mathrm{nz}_{\mathrm{maxR}}$ éléments non nuls). De plus, la complexité d'une addition des éléments d'une même ligne des variables *data parallel*, est considérée la même que celle dans le cas où on a le résultat dans chaque processeur virtuel de la ligne (*Hypothèse 2, la section 0*).

Pour pouvoir calculer $(A \times x + x)$, les éléments de w doivent être réaffecté aux processeurs de la géométrie tels que chaque élément i de G_{scal} contient la ind_col_i composante de w. Cette opération pourra être optimisée par rapport à la plateforme cible et le langage utilisé [PeE06]. Chaque processeur virtuel Pv_i de la géométrie G_{scal} est amené à calculer une opération d'addition scalaire (+) :

$$w_i = w_i + x_i$$

Pour pouvoir calculer $y = A \times (A \times x + x) = A \times w$, Chaque processeur virtuel Pv_i de la géométrie G_{scal} est amené donc à calculer une deuxième opération de multiplication (×) :

$$u_i = val_i \times w_i$$

Avec u un vecteur de résultat partiel de taille nnz. Pour calculer $y = A \times w$, les éléments de u appartenant à une même ligne k de la matrice A sont additionnés dans y_k en $log_2(\mathrm{nz}_{\mathrm{maxR}})$ (*Figure* ***Error! No text of specified style in document.****-48* (b)).

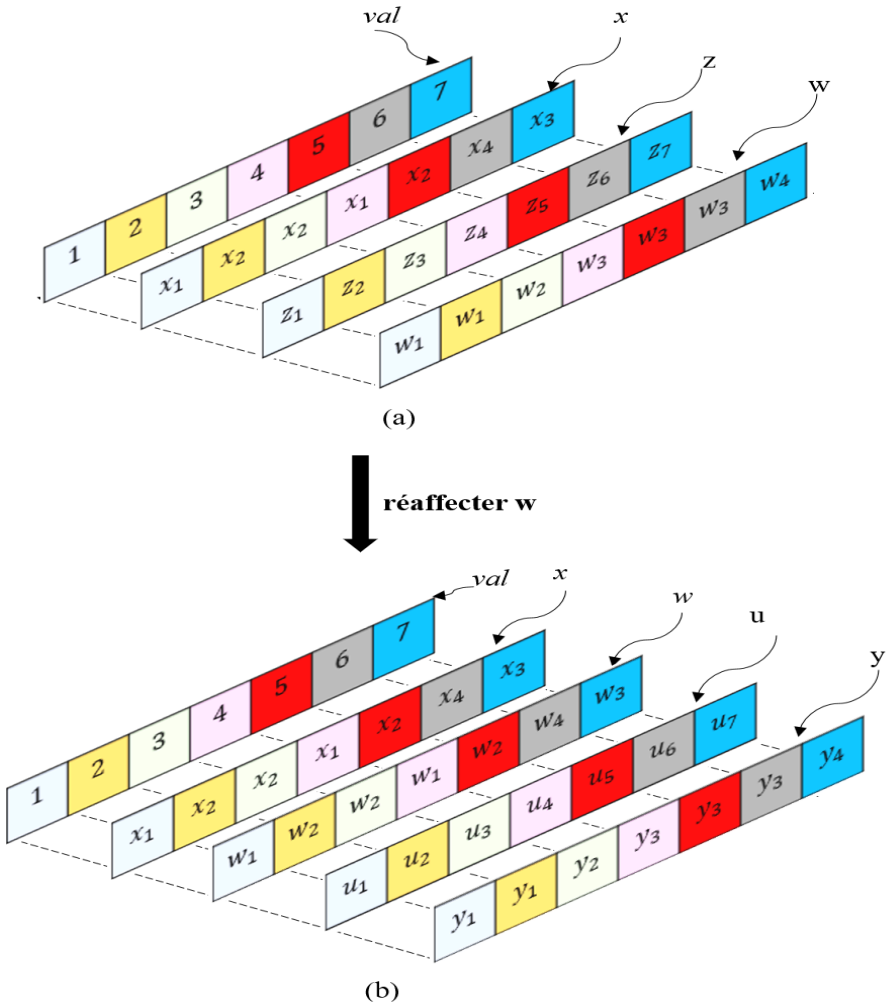

*Figure **Error! No text of specified style in document.**-48: Application du schéma de Horner dans le cas CSR*

CSC

Soit G_{scal}, une géométrie virtuelle à une dimension ($1D$) de taille nnz. Pour implémenter le schéma de Horner en utilisant le format CSC, chaque élément val_i ainsi que l'élément de x correspondant sont affectés au même élément i de G_{scal} (*Figure **Error! No text of specified style in document.**-49*).

De même que pour le format CSR, nous commençons par effectuer une opération de multiplication (×) impliquant val et x en utilisant une seule opération *data parallel*. A savoir, chaque processeur virtuel Pv_i de la géométrie G_{scal} calcule une opération scalaire :

$$z_i = val_i \times x_i$$

Avec z un vecteur de résultat partiel de taille nnz (*Figure **Error! No text of specified style in document.**-50* (a)).

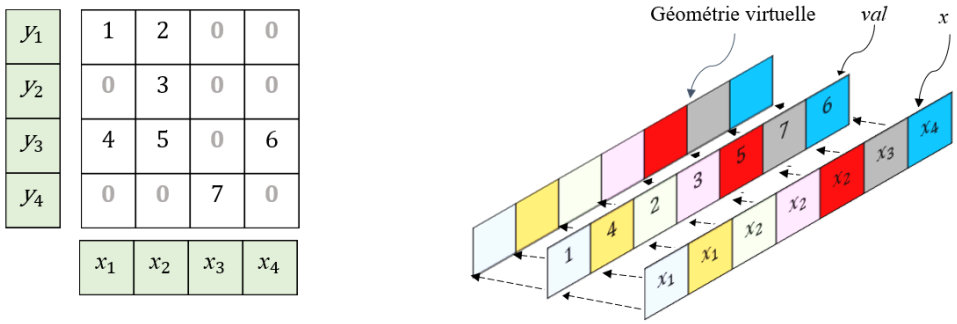

Figure ***Error! No text of specified style in document.****-49 Affectation des données pour le format CSC*

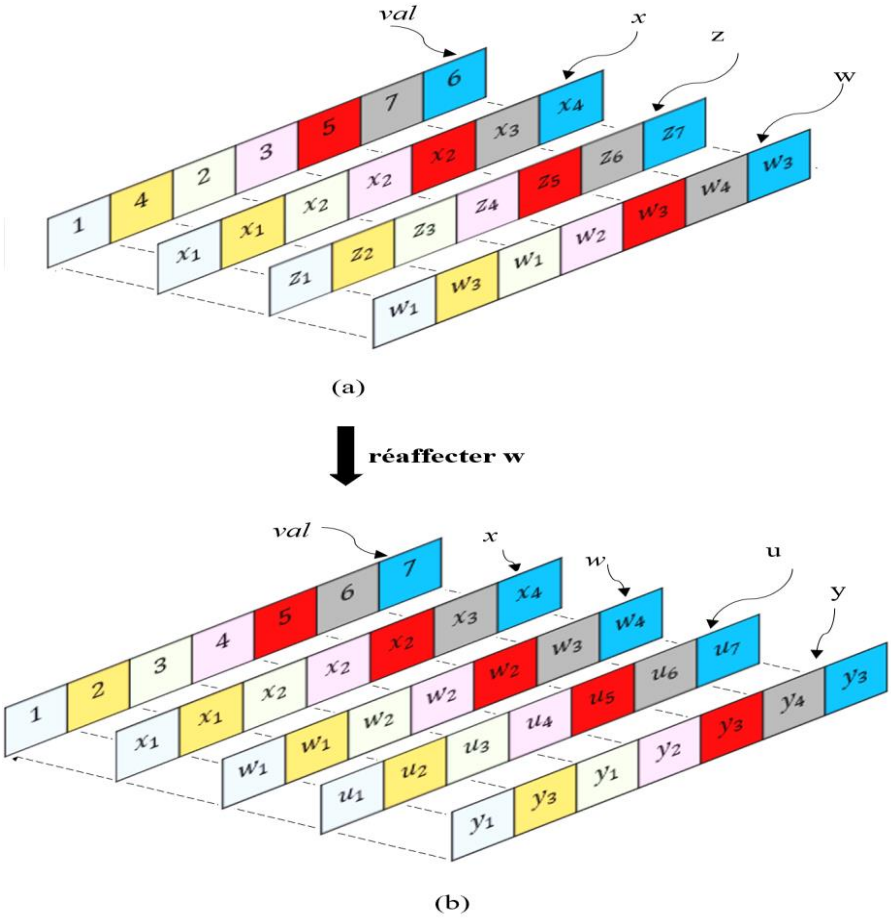

Figure ***Error! No text of specified style in document.****-50: Application du schéma de Horner dans le cas CSC*

Pour calculer $w = A \times x$, les éléments de z appartenant à une même ligne k de la matrice A sont additionnés dans w_k en $log_2(\mathrm{nz}_{\mathrm{maxR}})$ (*Figure* ***Error! No text of specified style in document.****-50* (a)). Ceci est dû au fait que la complexité de calcul dépend de la ligne la plus chargée (avec $\mathrm{nz}_{\mathrm{maxR}}$ éléments non nuls). De plus, la

complexité d'une addition des éléments d'une même ligne des variables *data parallel* est considérée la même que celle dans le cas où on a le résultat dans chaque processeur virtuel de la ligne (*Hypothèse 2, la section 0*).

Pour pouvoir calculer $(A \times x + x)$, les éléments de w doivent être réaffectés aux processeurs de la géométrie. Chaque processeur virtuel Pv_i de la géométrie G_{scal} est amené à calculer une opération d'addition scalaire (+) :

$$w_i = w_i + x_i$$

Pour pouvoir calculer $y = A \times (A \times x + x) = A \times w$, chaque processeur virtuel Pv_i de la géométrie G_{scal} est mené donc à calculer une deuxième opération de multiplication (×) :

$$u_i = val_i \times w_i$$

Avec u un vecteur de résultat partiel de taille nnz. Pour calculer $y = A \times w$, les éléments de u appartenant à une même ligne k de la matrice A sont additionnés dans y_k en $log_2(\text{nnz}_{\text{maxR}})$ *Figure* ***Error! No text of specified style in document.****-50*(b)).

COO

Dans ce travail, nous supposons que dans le cas du format de compression COO, les éléments non nuls de la matrice creuse sont stockés dans val ligne par ligne (respectivement colonne par colonne) et dans ce cas on note COOR (respectivement COOC). Dans le cas de COOR (respectivement COOC) notre implémentation de Horner est identique à celle du format CSR présentée dans la section 0-j) de ce chapitre (respectivement CSC présenté dans la section 0-0 de ce chapitre).

ELL

Dans le cas du format de compression ELL, nous utilisons une géométrie virtuelle G_{scal} à deux dimensions ($2D$) de taille $\text{n} \times \text{nz}_{\text{maxR}}$. Sur cette géométrie sont projetés val et X (*Figure* ***Error! No text of specified style in document.****-51*). X une matrice contenant les éléments de x tels que $X_{i,j} = x[ind_{i,j}]$.

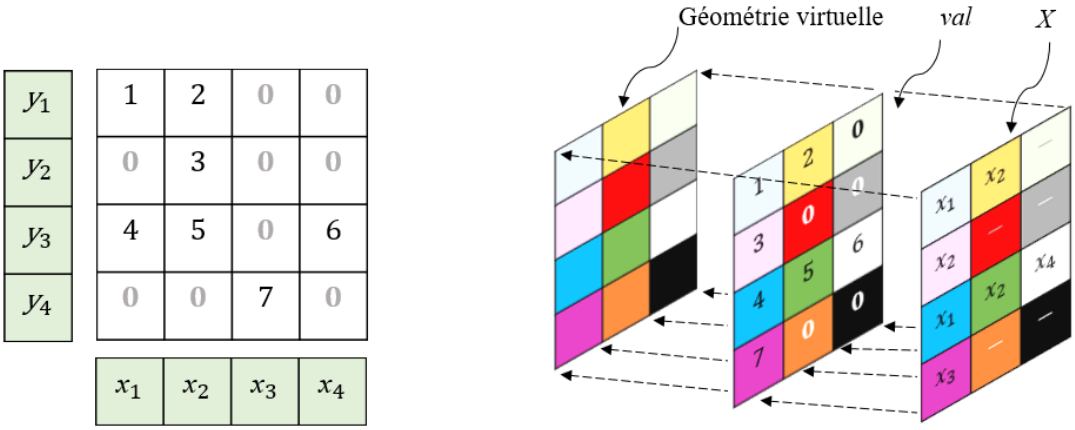

Figure ***Error! No text of specified style in document.****-51 Affectation des données pour le format ELL*

De la même manière que CSR et CSC, pour calculer Horner, la première étape consiste à effectuer une opération de multiplication (×) impliquant val et X en utilisant une seule opération *data parallel*. A savoir, chaque processeur virtuel $Pv_{i,j}$ de la géométrie G_{scal} calcule une opération scalaire :

$$Z_{i,j} = val_{i,j} \times X_{i,j}$$

Avec Z une matrice de résultat partiel de taille n $\times$ $\mathrm{nz_{maxR}}$. Pour calculer $A \times x$, les éléments de chaque ligne de Z sont additionnés en $log_2(\mathrm{nz_{maxR}})$. On obtient comme résultat la matrice W de taille $n \times nz_{maxR}$ avec :

$$W_{i,j} = \sum_{j=1}^{nz_{maxR}} Z_{i,j}$$

Puisque les éléments d'une même ligne i de W sont égaux, on note alors les éléments de cette ligne W_i (Figure Error! ***No text of specified style in document.****-52*).

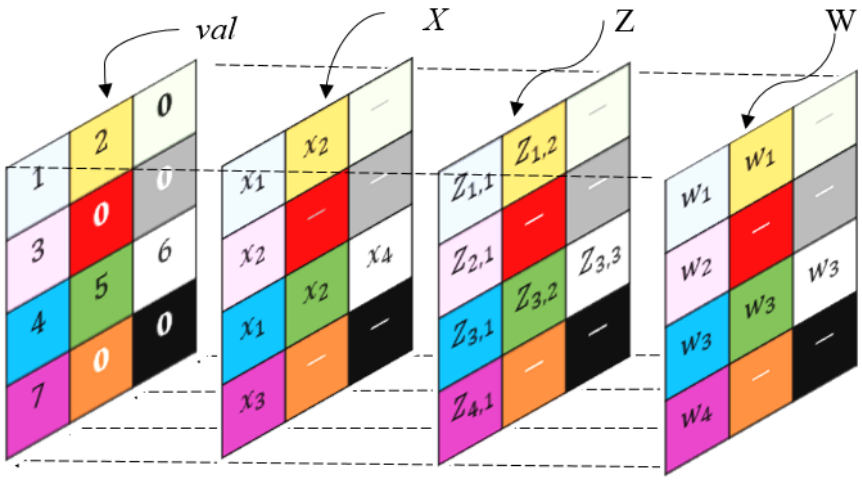

Figure ***Error! No text of specified style in document.****-52 calcul de* $A \times x$ *dans le cas du format ELL*

A cette étape, les éléments de W doivent être réaffectés aux processeurs virtuels tels que à chaque $Pv_{i,j}$ est affecté W_j. Nous effectuons alors une deuxième opération de multiplication (×) impliquant cette fois val et W en utilisant une seule opération *data parallel*.

$$U_{i,j} = val_{i,j} \times W_{i,j}$$

Avec U une matrice de résultat partiel de taille n $\times$ nz_{maxR} (*Figure* ***Error! No text of specified style in document.****-53*). Au niveau de chaque ligne de la géométrie, une opération de réduction (red) des éléments de U est réalisée avec une complexité O $(log_2(nz_{maxR}))$. On obtient donc comme résultat la matrice Y de taille n $\times$ nz_{maxR} avec :

$$Y_{i,j} = \sum_{j=1}^{nz_{maxR}} U_{i,j} = y_i$$

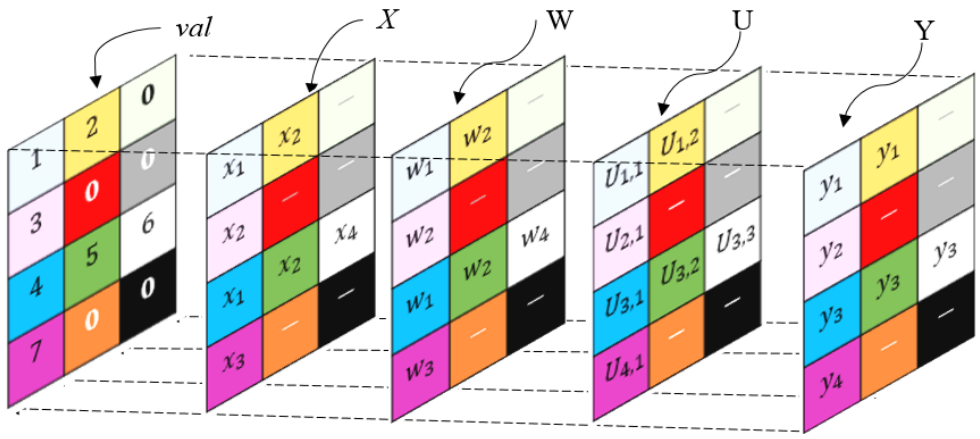

*Figure **Error! No text of specified style in document.**-53 Application du schéma de Horner dans le cas ELL*

Récapitulatif

Le *Tableau 6* représente les résultats numériques des métriques *data parallel* de données scalaires pour les formats CSC, CSR, COO et ELL. Pour Horner (schéma 2), trois types d'opérations *data parallel* sont impliquées : deux multiplications (×), une addition (+) et $2 \times log_2(\text{nz}_{\text{maxR}})$ réductions (red). Le nombre de ces opérations parallèles de données scalaires cx_{scal} est le même pour les quatre formats testés :

$$cx_{scal} = 3 + 2 \times log_2(\text{nz}_{\text{maxR}})$$

Etant donné que les données ELL sont affectées aux processeurs virtuels d'une géométrie virtuelle $2D$ de taille $\text{n} \times \text{nz}_{\text{maxR}}$ alors que celles de CSR, CSC et COO sont affectés à des processeurs d'une géométrie virtuelle $1D$ de taille nnz, nous présentons dans ce qui suit les résultats numériques de l'application du modèle *data parallel* aux algorithmes de Horner associés aux formats CSR, CSC et COO, et en deuxième partie celles du format ELL.

Soient :

- $\boldsymbol{\alpha_{\times}}$: le nombre de processeurs virtuels concernés par une opération de multiplication *data parallel*,
- $\boldsymbol{\alpha_{+}}$: le nombre de processeurs virtuels concernés par une opération d'addition *data parallel*,
- $\boldsymbol{\alpha_{red}}$: le nombre de processeurs virtuels concernés par une opération de réduction *data parallel.*

Dans le cas des formats CSR, CSC et COO, nous avons :

$$\boldsymbol{\alpha}_{\times} = \boldsymbol{\alpha}_{+} = nnz \quad par\ la\ suite \quad \varphi_{\times} = \varphi_{+} = \frac{nnz}{Pv} = \frac{nnz}{nnz} = 1$$

$$\boldsymbol{\alpha}_{\boldsymbol{red}} = \text{nnz} \ \ par\ la\ suite \ \ \varphi_{red} = \frac{\text{nnz}}{Pv} = 1$$

On a alors

$$\bar{\varphi} = \frac{1}{cx_{scal}} \sum_{ops} \varphi_{ops} \qquad avec\ ops \in \{+,\times, red\}$$
$$= \frac{1}{cx_{scal}} \left(2 \times \varphi_{\times} + \varphi_{+} + 2 \times log_2(\text{nz}_{\text{maxR}}) \times \varphi_{red} \right)$$
$$= \frac{3 + 2 \times log_2(\text{nz}_{\text{maxR}})}{3 + 2 \times log_2(\text{nz}_{\text{maxR}})} = 1$$

Le nombre maximal de données impliquées dans une opération *data parallel* (degré de parallélisme) est alors :

$$D = \max\{\boldsymbol{\alpha}_{\times}, \boldsymbol{\alpha}_{+}, \boldsymbol{\alpha}_{\boldsymbol{red}}\} = nnz$$

Dans le cas du format ELL, nous avons :

$$\boldsymbol{\alpha}_{\times} = \boldsymbol{\alpha}_{+} = \text{n} \times \text{nz}_{\text{maxR}} \quad par\ la\ suite \quad \varphi_{\times} = \varphi_{+} = \frac{\text{n} \times \text{nz}_{\text{maxR}}}{Pv} = \frac{\text{n} \times \text{nz}_{\text{maxR}}}{\text{n} \times \text{nz}_{\text{maxR}}} = 1$$

$$\boldsymbol{\alpha}_{\boldsymbol{red}} = n \times \text{nz}_{\text{maxR}} \ \ par\ la\ suite \ \ \varphi_{red} = \frac{n \times \text{nz}_{\text{maxR}}}{\text{n} \times \text{nz}_{\text{maxR}}}$$
$$= 1$$

On a alors

$$\bar{\varphi} = \frac{1}{cx_{scal}} \sum_{ops} \varphi_{ops} \qquad avec\ ops \in \{+,\times, red\}$$
$$= \frac{1}{cx_{scal}} \left(2 \times \varphi_{\times} + \varphi_{+} + 2 \times log_2(\text{nz}_{\text{maxR}}) \times \varphi_{red} \right)$$
$$= \frac{3 + 2 \times log_2(\text{nz}_{\text{maxR}})}{3 + 2 \times log_2(\text{nz}_{\text{maxR}})} = 1$$

Le nombre maximal de données impliquées dans une opération *data parallel* (degré de parallélisme) est alors :

$$D = \max\{\boldsymbol{\alpha}_{\times}, \boldsymbol{\alpha}_{+}, \boldsymbol{\alpha}_{\boldsymbol{red}}\} = \mathrm{n} \times \mathrm{nz}_{\mathrm{maxR}}$$

Tableau 6 résultats numériques de l'application du modèle data parallel à Horner : données scalaires

	CSR/CSC/COO	**ELL**
$\boldsymbol{P_v}$	nnz	$n \times nz_{max\mathrm{R}}$
$\boldsymbol{cx_{scal}}$	$3 + 2 \times log_2(\mathrm{nz}_{\mathrm{maxR}})$	$3 + 2 \times log_2(\mathrm{nz}_{\mathrm{maxR}})$
$\boldsymbol{\alpha}_{\times}$	nnz	$n \times nz_{max\mathrm{R}}$
$\boldsymbol{\alpha}_{+}$	nnz	$n \times nz_{max\mathrm{R}}$
$\boldsymbol{\alpha_{red}}$	nnz	$n \times nz_{max\mathrm{R}}$
$\boldsymbol{\overline{\varphi}}$	1	1
$\boldsymbol{D}$	nnz	$\mathrm{n} \times \mathrm{nz}_{\mathrm{maxR}}$

Nous remarquons, d'après le *Tableau 6*, que cette première étude basée sur les données scalaires ne nous permet pas de faire un choix entre les formats étudiés. Néanmoins, nous pouvons classer les formats selon deux classes : la première contenant CSR, CSC et COO alors que la deuxième correspond au format ELL. Nous définissons à ce niveau un nouveau paramètre qu'on note ELL_{ratio} avec :

$$ELL_{ratio} = \frac{nnz}{n \times nz_{max\mathrm{R}}} \times 100$$

Le paramètre ELL_{ratio} représente la densité de la matrice ELL. Plus ce ratio est proche de 100%, plus les performances de calcul du schéma de Horner en utilisant le format ELL sont proches de celles en utilisant CSR, CSC et COO

Données vectorielles

Nous remarquons que le calcul du schéma de Horner implique deux opérations vectorielles principales : le produit scalaire (***Algorithme*** *Error! No text of specified style in document.****-8***) et l'opération $saxpy$ (***Algorithme*** *Error! No text of specified style in document.****-9***). Ainsi, nous proposons d'étudier dans cette partie un deuxième niveau du modèle *data parallel* où nous traitant des vecteurs au lieu de scalaires. Pour cela, nous organisons une géométrie virtuelle G_{vect} à une dimension $(1D)$ de taille n. Chaque vecteur (ligne / colonne de la matrice) est

affecté à un processeur virtuel. Afin d'avoir la même quantité de données communiquées dans le cas des quatre formats testés, les vecteurs y et x sont affectés à chaque processeur virtuel.

Algorithme *Error! No text of specified style in document.**-8** Dot product :* $\alpha = dot\,(x\,;\,y) = x^T\,y$

$\alpha = 0$
POUR $i = 1, n$
$\alpha = \alpha + X[i] \times Y[i];$
FIN POUR

Algorithme *Error! No text of specified style in document.**-9** opération* $saxpy$*:* $y = saxpy(\alpha; x; y) = \alpha x + y$

POUR $i = 1, n$
$Y[i] = \alpha X[i] + Y[i];$
FIN POUR

k) <u>CSR</u>

Pour implémenter le schéma de Horner en utilisant le format CSR, les éléments de val appartenant à la $i^{ème}$ ligne de A (noté $A_{i,:}$) sont affectés à l'élément i de G_{vect}. Le vecteur x est affecté à tous les processeurs virtuels de G_{vect} (*Figure **Error! No text of specified style in document.**-54*).

Nous commençons par calculer $w = A \times x + x$, avec w un vecteur résultat de taille n affecté à chaque processeur virtuel de la géométrie G_{vect}, c-à-d chaque processeur virtuel Pv_i possède sa propre copie de w. Dans une première étape, chaque processeur virtuel Pv_i initialise la $i^{ème}$ composante de w par la $i^{ème}$ composante de x. Les autres éléments de w seront initialisés à zéro (*Figure **Error! No text of specified style in document.**-54*). Au niveau de chaque processeur Pv_i on a alors :

$$w_i = x_i$$

$$w_k = 0 \quad \forall\, k \in [1..n] \text{ et } k \neq i$$

A cette étape, nous calculons des produits scalaires (***Algorithme*** *Error! No text of specified style in document.**-8***) entre chaque ligne i de la matrice ($A_{i,:}$) et le vecteur x, et ce en utilisant une seule opération vectorielle *data parallel* ($\times$) avec la complexité $O(\mathrm{nz_{maxR}})$.

$$w_i = w_i + dot\,(A_{i,:}\,;\,x) = x_i + dot\,(A_{i,:}\,;\,x)$$

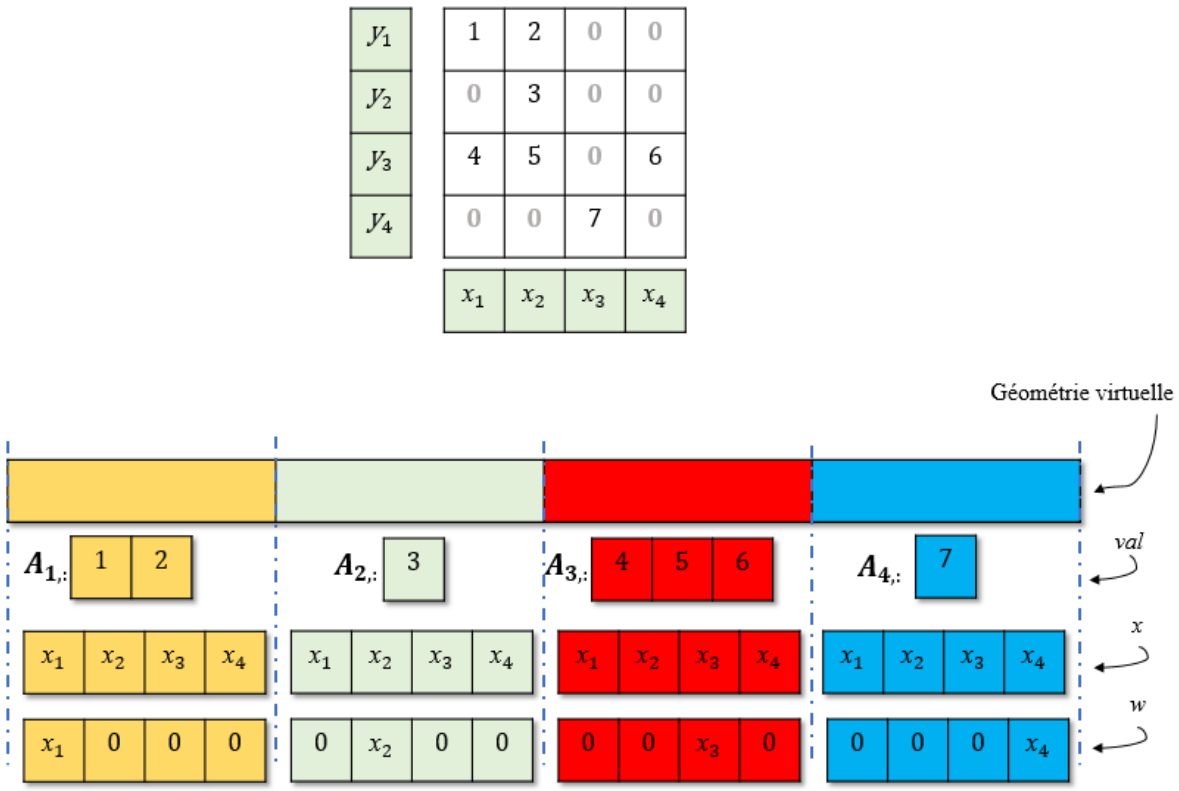

*Figure **Error! No text of specified style in document.**-54 Affectation des données vectorielles pour le format CSR*

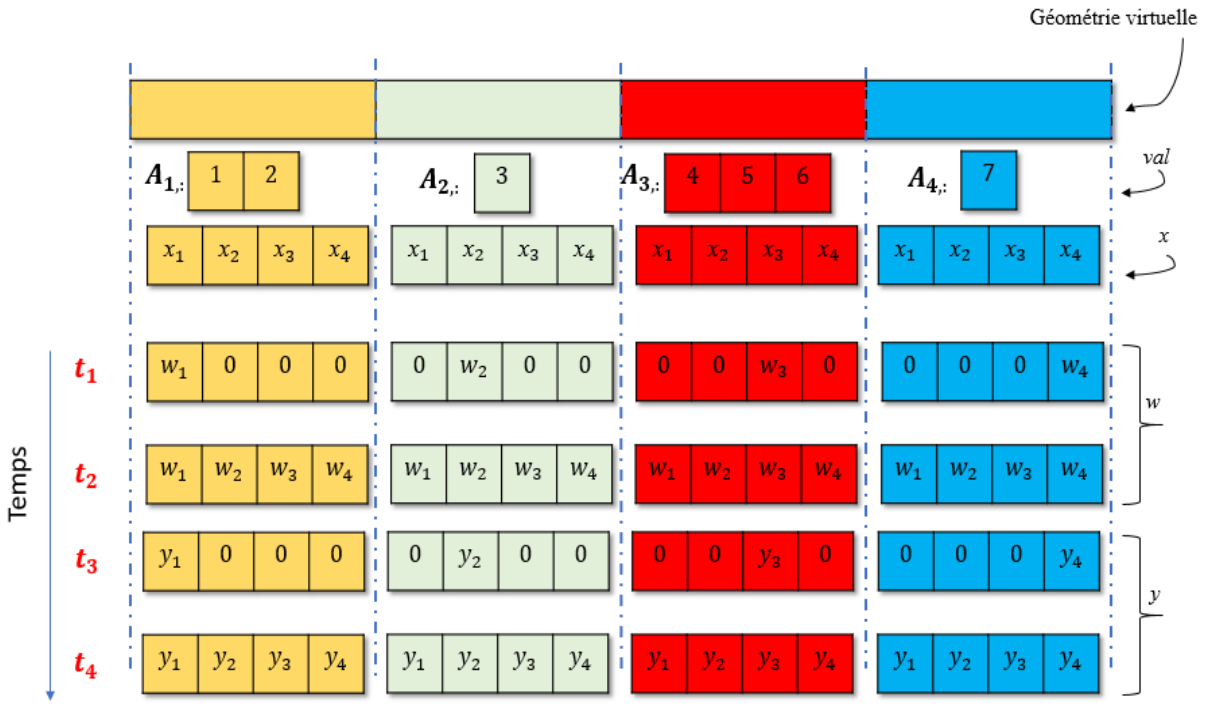

Figure ***Error! No text of specified style in document.****-55 Etapes de calcul de Y*

A ce niveau, dans chaque Pv_i , tous les éléments de w sont égaux à zéro sauf le $i^{ème}$ élément qui stocke la $i^{ème}$ composante de $A \times x + x$ (*Figure* ***Error! No text of specified style in document.****-55* à $\boldsymbol{t_1}$). Pour construire w final, nous devons additionner les résultats partiels stockés au niveau de chaque processeur virtuel, et ce en utilisant $log_2(n)$ réduction *data parallel* de vecteurs $(red +)$. (*Figure* ***Error! No text of specified style in document.****-55* à $\boldsymbol{t_2}$).

Pour pouvoir calculer $y = A \times (A \times x + x) = A \times w$, chaque processeur virtuel Pv_i de la géométrie G_{vect} est amené donc à calculer une deuxième opération vectorielle *data parallel* $(\times)$ avec la complexité $O(\mathrm{nz}_{\mathrm{maxR}})$ (*Figure* ***Error! No text of specified style in document.****-55* à $\boldsymbol{t_3}$).

$$y_i = dot\ (A_{i,:}\ ; w)$$

Pour construire y final, nous devons additionner les résultats partiels stockés au niveau de chaque processeur virtuel, et ce en utilisant $log_2(n)$ réduction *data parallel* de vecteurs $(red +)$ (*Figure* ***Error! No text of specified style in document.****-55* à $\boldsymbol{t_4}$).

CSC

Pour implémenter le schéma de Horner en utilisant le format CSC, les éléments de val appartenant à la $j^{ème}$ colonne de A (noté $A_{:,j}$) sont affectés au même processeur virtuel j de G_{vect}. Le vecteur x est affecté à tous les processeurs virtuels de G_{vect} (*Figure* ***Error! No text of specified style in document.****-56*).

Comme pour le cas du format CSR, nous commençons par calculer $w = A \times x + x$, avec w un vecteur résultat de taille n affecté à chaque processeur virtuel de la géométrie G_{vect}. Chaque processeur virtuel Pv_j possède sa propre copie de w. Dans une première étape, chaque processeur virtuel Pv_j calcule un résultat partiel $u = A_{:,j} \times x_j + x_j$. u un vecteur de résultat partiel de taille n. Chaque Pv_j initialise la $j^{ème}$ composante de sa copie de u par la $j^{ème}$ composante de x. Les autres éléments de u seront initialisés à zéro (*Figure* ***Error! No text of specified style in document.****-56*). Au niveau de chaque processeur Pv_j on a alors :

$$u_j = x_j$$

$$u_k = 0 \quad \forall \, \mathrm{k} \in [1..\mathrm{n}] \text{ et } \mathrm{k} \neq \mathrm{j}$$

Chaque processeur virtuel Pv_i met à jour sa copie de u, et ce en calculant une opération $saxpy$ (***Algorithme*** *Error! No text of specified style in document.****-9***) impliquant la $j^{ème}$ colonne de la matrice ($A_{:,j}$) et x_j, et ce en utilisant une seule opération vectorielle *data parallel* ($\times$) avec la complexité $O(\mathrm{nz_{maxC}})$ (Figure Error! ***No text of specified style in document.****-57* à $\mathbf{t_1}$).

$$u = saxpy(x_j; A_{:,j};\ u)$$

Pour construire w, nous devons additionner les résultats partiels u stockés au niveau de chaque processeur virtuel, et ce en utilisant $log_2(n)$ réduction *data parallel* de vecteurs ($red+$). Le résultat de cette opération de réduction est stocké dans w (Figure Error! ***No text of specified style in document.****-57* à $\mathbf{t_2}$). A ce stade, les processeurs Pv_j possèdent la même copie de w.

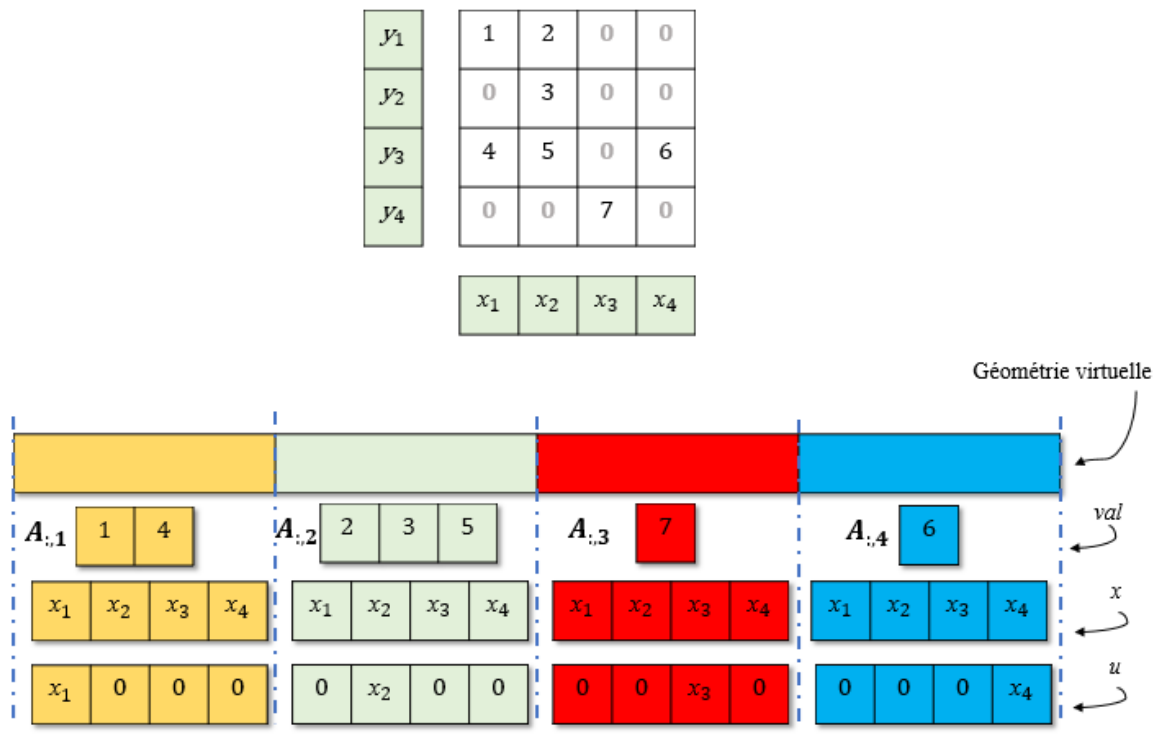

*Figure **Error! No text of specified style in document.**-56 Affectation des données vectorielles pour le format CSC*

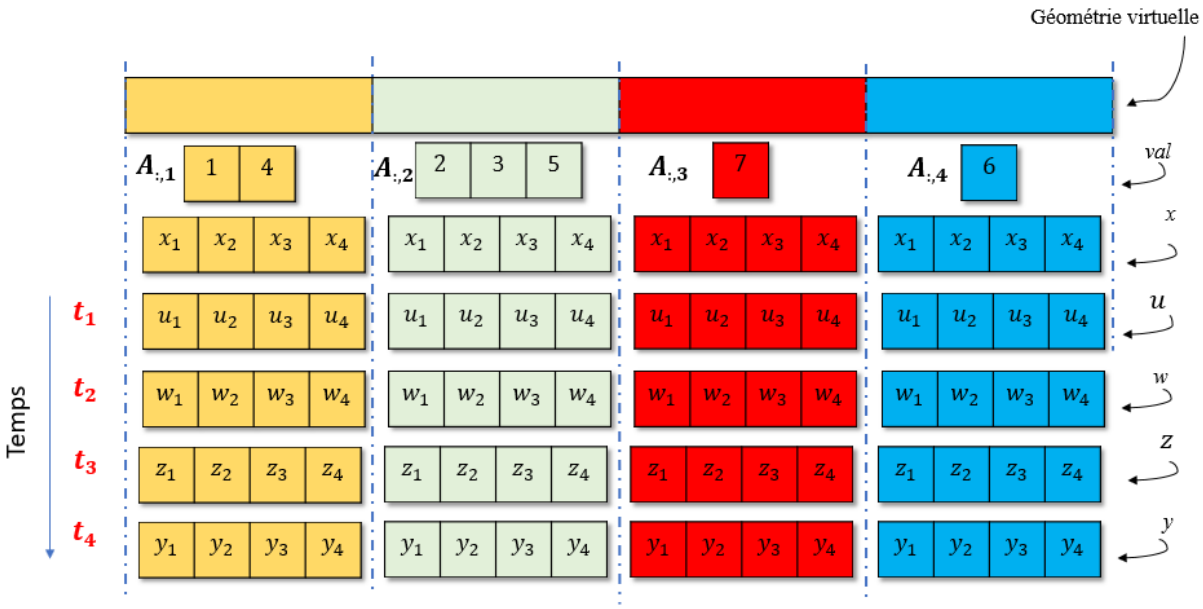

*Figure **Error! No text of specified style in document.**-57 Etapes de calcul de Y*

Pour pouvoir calculer $y = A \times (A \times x + x) = A \times w$, chaque processeur virtuel Pv_j de la géométrie G_{vect} est mené donc à calculer une deuxième opération $saxpy$ (***Algorithme** Error! No text of specified style in document.**-9***) impliquant la $j^{ème}$

colonne de la matrice ($A_{:,j}$) et w_j. La complexité de cette opération *data parallel* est de $O(\mathrm{nz_{maxC}})$ (Figure Error! ***No text of specified style in document.****-57* à t_3).

$$z = saxpy(w_j; A_{:,j};\ z\ )$$

Avec z un vecteur de résultat partiel de taille n et chaque processeur virtuel Pv_j possède sa propre copie de z initialisé à zéros. Pour construire y final, nous devons additionner les résultats partiels stockés dans z au niveau de chaque processeur virtuel, et ce en utilisant $log_2(n)$ réduction *data parallel* de vecteurs ($red +$) (Figure Error! ***No text of specified style in document.****-57* à t_4).

COO

Dans le cas de COOR (COOC) notre implémentation du schéma de Horner est identique à celle du format CSR présentée dans la section 00-k) de ce chapitre (respectivement CSC présenté dans la section 0-0 de ce chapitre).

ELL

Dans le cas de traitement de données vectorielle, notre implémentation du schéma de Horner, dans le cas où la matrice est compressée selon format ELL, est identique à celle associée au format CSR présenté dans la section 0-k) de ce chapitre.

Récapitulatif

Le

Tableau 7 représente les résultats numériques des métriques *data parallel* de données vectorielles pour les formats CSC, CSR, COO et ELL. Dans le cas de calcul de la méthode de Horner (schéma 2), deux types d'opérations vectorielles *data parallel* sont impliquées : deux multiplications (dot ou $saxpy$) notés (×) et $2 \times log_2(\mathrm{n})$ réductions avec addition de vecteurs ($red +$). Le nombre de ces opérations parallèles de données vectorielles cx_{vect} est le même pour les quatre formats testés :

$$cx_{vect} = 2 + 2 \times log_2(n)$$

Soient :

- $\boldsymbol{\beta}_{\times}$: le nombre de processeurs virtuels concernés par une opération de multiplication *data parallel* (dot ou $saxpy$),

- $\boldsymbol{\beta_{red+}}$: le nombre de processeurs virtuels concernés par une opération de réduction avec addition de vecteur

Dans le cas des quatre formats, nous avons :

$$\boldsymbol{\beta}_{\times} = n \quad par\ la\ suite \quad \varphi_{\times} = \frac{\boldsymbol{\beta}_{\times}}{Pv} = \frac{n}{n} = 1$$

$$\boldsymbol{\beta_{red+}} = \mathrm{n} \quad par\ la\ suite \quad \varphi_{red+} = \frac{\boldsymbol{\beta_{red+}}}{Pv} = \frac{\mathrm{n} \times\ log_2(\mathrm{n})}{n} = 1$$

On a alors

$$\bar{\varphi} = \frac{1}{cx_{vect}} \sum_{ops} \varphi_{ops} \qquad avec\ ops \in \{\times, red+\}$$

$$= \frac{1}{cx_{vect}} \left(2 \times \varphi_{\times} + 2 \times\ log_2(\mathrm{n}) \times\ \varphi_{red+} \right)$$

$$= \frac{2 + 2 \times\ log_2(\mathrm{n})}{2\ +\ 2\ \times log_2(n)} = 1$$

Le nombre maximal de données impliquées dans une opération *data parallel* (degré de parallélisme) est alors :

$$D = \max\{\boldsymbol{\beta}_{\times}, \boldsymbol{\beta_{red+}}\} = n$$

Tableau 7 résultats numériques de l'application du modèle data parallel à Horner : données vectorielles

	CSR/COOR/ELL	**CSC/COOC**
$\boldsymbol{P_v}$	n	n
$\boldsymbol{cx_{vect}}$	$2 + 2 \times log_2(n)$	$2 + 2 \times log_2(\mathrm{n})$
$\boldsymbol{\beta}_{\times}$	n	n
$\boldsymbol{\beta_{red+}}$	n	n
$\boldsymbol{co\hat{u}t}_{\times}$	$\mathbf{nz_{maxR}}$	$\mathbf{nz_{maxC}}$
$\bar{\boldsymbol{\varphi}}$	1	1
$\boldsymbol{D}$	n	n

Nous remarquons que les résultats numériques de l'applications du modèle *data parallel* dans le cas de traitement des données vectorielles sont les mêmes pour tous les formats étudiés. Néanmoins, le coût de chaque opération de multiplication de vecteurs $\boldsymbol{co\hat{u}t}_{\times}$ n'est pas le même. Cela dépend de nz_{maxR} (pour les formats stockage ligne) ou de nz_{maxC} (pour les formats stockage colonne). Ainsi, nous pouvons différencier deux classes de formats grâce à ce paramètre : la première classe contenant les formats de compression dont la direction de stockage des éléments non nuls est lignes (CSR/COOR/ELL) et une deuxième classe dont la direction de stockage des éléments non nuls est colonnes (CSC/COOC).

Cette première étude indique ce que permet de différencier l'analyse du modèle *data parallel*. La prise en compte du modèle *data parallel* dans le processus de décision entre CSR, CSC, ELL et COO amènent à considérer les métriques : ELL_{raio} et $\boldsymbol{co\hat{u}t}_{\times}$. Dans ce qui suit nous analysons l'algorithme afin d'extraire un deuxième ensemble de métrique.

Analyse des algorithmes de PMVC associés aux différents formats de compression étudiés

Nous présentons dans cette partie un récapitulatif comparatif des algorithmes des PMVC associées aux différents formats de compression CSR, CSC, ELL et COO (*Tableau 8*). Une analyse des algorithmes nous permet de remarquer que la complexité de calcul est la même dans le cas des quatre versions du PMVC ($O(nnz)$). Néanmoins, la différence majeure est au niveau des accès mémoire : nombre de lectures, nombre d'écritures et nombre des accès indirectes (calcul d'adresse).

Tableau 8 Analyse des algorithmes du PMVC dans le cas séquentiel

Format	# Lectures	# Ecritures	#accès indirects	#opérations
CSR	$3 \times n + 6 \times \text{nnz}$	$4 \times n + 3 \times \text{nnz}$	$2 \times n + 4 \times nnz$	$2 \times nnz$
CSC	$3 \times n + 6 \times nnz$	$3 \times n + 3 \times nnz$	$2 \times n + 5 \times nnz$	$2 \times nnz$
ELL	$2 \times n + 6 \times nnz$	$4 \times n + 3 \times nnz$	$1 \times n + 4 \times nnz$	$2 \times nnz$
COO	$6 \times nnz$	$4 \times nnz$	$6 \times nnz$	$2 \times nnz$

Analyse du modèle d'expression du parallélisme

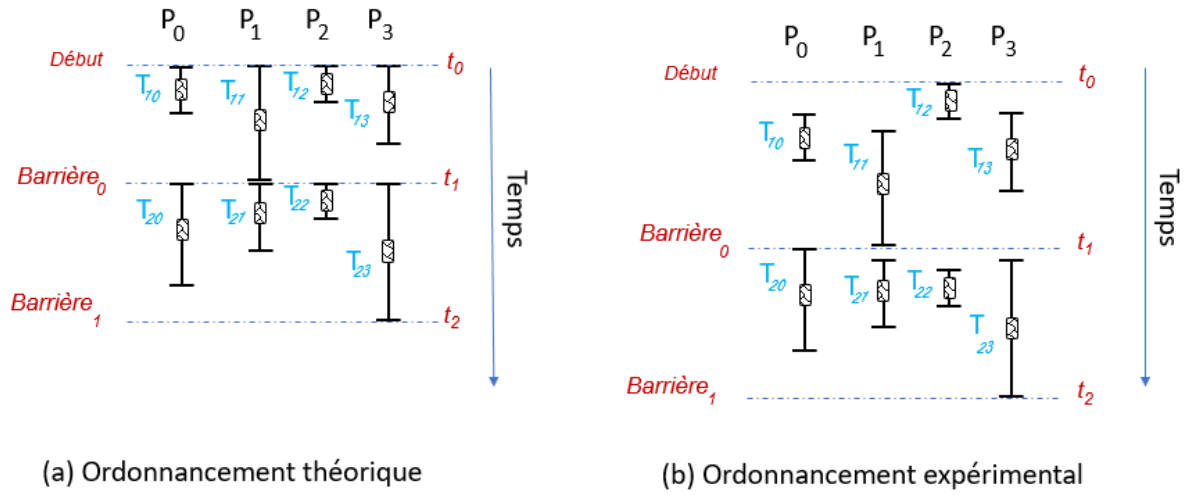

Figure ***Error! No text of specified style in document.****-58 Ordonnancement des processus parallèles*

Théoriquement, les processeurs physiques commencent une tâche à la même date t_0. Ce qui est également le cas au niveau d'une région de synchronisation (Barrière). Chaque processeur peut exécuter des tâches de différentes tailles/temps (*Figure* ***Error! No text of specified style in document.****-58* (a)). Le temps total d'exécution est donc limité par le temps d'exécution du processeur dont la tâche est plus longue. Néanmoins, n'est pas le cas en réalité (*Figure* ***Error! No text of specified style in document.****-58* (b)). Cela dépend de l'environnement d'exécution qui peut provoquer un décalage de démarrage des processus. Prenons le cas de calcul d'un produit matrice vecteur creux sur une plateforme parallèle à PP processeurs physiques. Chaque processeur calcule un résultat partiel *produit fragment vecteur*, avec chaque fragment étant un ensemble de lignes de la matrice. Théoriquement, tous les processeurs commencent le calcul de leurs résultats à la même date alors qu'en pratique ce n'est pas vrai. Ainsi, pour évaluer les performances d'une méthode numérique, nous considérons le temps maximal d'exécution des processeurs au lieu du temps d'exécution total du programme.

Définition d'un processus de décision

Soit A une matrice creuse, $A \in \mathbb{R}^{n \times n}$, nous cherchons à trouver le format de compression optimal entre CSR, CSC, ELL et COO et ce dans le cas de calcul du schéma de Horner. Nous avons montré dans la section 0 que l'application du modèle *data parallel* peut mener à un premier pas dans le processus de décision. Dans le cas où nous traitons des vecteurs (lignes ou colonnes de la matrice), nous classons les formats en entrée en deux classe $C1$ et $C2$.

- La classe $C1$ contient les formats où les éléments non nuls de la matrice sont stockés dans un ordre de lignes (CSR, ELL, COOR), dans ce cas la complexité du temps de calcul dépend de $\mathrm{nz}_{\mathrm{maxR}}$.
- La deuxième classe $C2$ contient l'ensemble des formats où les éléments sont stockés dans un ordre de colonnes (CSC et COOC), dans ce cas la complexité du temps de calcul dépend de $\mathrm{nz}_{\mathrm{maxC}}$.

Nous pouvons à ce stade conclure que, pour une matrice creuse donnée, si $\mathrm{nz}_{\mathrm{maxR}} < \mathrm{nz}_{\mathrm{maxC}}$, alors le format de compression optimal est l'un des formats appartenant à $C1$, sinon il correspond à l'un des formats appartenant à $C2$.

Ainsi, nous choisissons de limiter notre étude à deux formats représentatifs CSR $\in C1$ et CSC $\in C2$. Les résultats obtenus pour CSR (respectivement CSC) seront valables pour COOR et ELL (respectivement COOC).

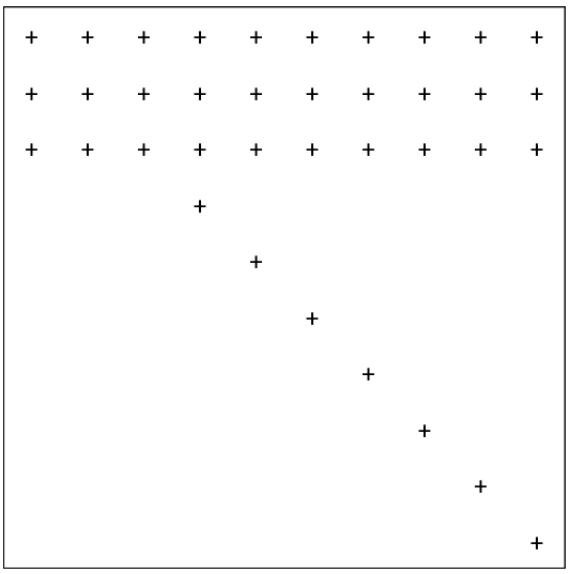

*Figure **Error! No text of specified style in document.**-59 Exemple de matrice pathologique $MatRow$ avec $n = 10$ et $larg = 3$*

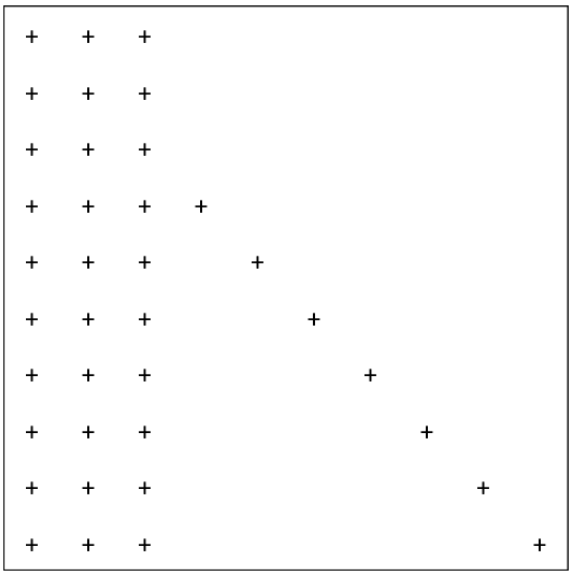

*Figure **Error! No text of specified style in document.**-60 Exemple de matrice pathologique $MatCol$ avec $n = 10$ et $larg = 3$*

Pour valider cette première conclusion, et être conforme à notre étude, nous proposons un cas pathologique de matrice creuse qu'on note $MatRow$ et $MatCol$ tels que :

- $MatRow$: une matrice creuse carrée avec $larg$ premières lignes (de la ligne numéro 1 à la ligne numéro $larg$) ne contenant pas des zéros. Les éléments de la diagonale sont non nuls (Figure Error! ***No text of specified style in document.**-59*),

- $MatCol$: une matrice creuse carrée avec $larg$ premières colonnes (de la colonne numéro 1 à la colonne numéro $larg$) ne contenant pas des zéros. Les éléments de la diagonale sont non nuls (Figure Error! ***No text of specified style in document.****-60*).

Ces matrices pathologiques ont la particularité de satisfaire le critère de $\mathrm{nz}_{\mathrm{maxR}}$ et $\mathrm{nz}_{\mathrm{maxC}}$. Ainsi, lorsque nous traitons des matrices creuses avec des structures proches de $MatRow$ et $MatCol$ le format choisi dépend des valeurs de $\mathrm{nz}_{\mathrm{maxR}} et$ $\mathrm{nz}_{\mathrm{maxC}}$.

Algorithme *Error! No text of specified style in document.****-10*** *Processus de décision basé sur les métriques data parallel*

$Extraire\_caractéristiques\_matrice(A)$;
SI $EstProche(Structure(A), Structure(matrice\_Pathologique))$ **ALORS**
SI ($\mathrm{nz}_{\mathrm{maxR}} < \mathrm{nz}_{\mathrm{maxC}}$) **ALORS**
$format = CSR$;
SINON
$format = CSC$;
FIN SI
SINON
$MAJ\ base\_de\_test()$;
$MAJ\ base\_de\_règles\_de\_décision()$;
$format = Selectionner\_format(A)$;
FIN SI

Ainsi, nous définissons un premier processus de décision (***Algorithme*** *Error! No text of specified style in document.****-10***) pour le choix du format de compression (entre CSR et CSC). Etant donnée une matrice creuse $A \in \mathbb{R}^{n \times n}$, On cherche à déterminer son meilleur format de compression. La première étape consiste à extraire un certain nombre de caractéristiques de la matrice en entrée. Dans une deuxième étape, et en se basant sur les caractéristiques extraites précédemment, nous analysons la structure de la matrice. Ainsi, si cette structure est similaire à l'un des cas pathologiques alors le choix sera automatiquement fait entre CSR et CSC, et ce en se basant les métriques *data parallel*. Dans le cas contraire, une mise à jour automatique du système sera effectuée pour ajouter le cas étudié à notre base. La mise à jour consiste à calculer le schéma de Horner pour tous les formats de compression puis choisir celui qui minimise le temps d'exécution.

Conclusion

Nous avons présenté dans ce chapitre une étude de cas pour valider notre système proposé pour la détection automatique du meilleur format de compression. Cette étude concerne le modèle de programmation *data parallel* et les noyaux de calcul schémas de Horner et le PMVC. Nous avons commencé ce chapitre par une présentation générale du modèle de programmation *data parallel* ainsi que l'ensemble des métriques d'évaluation qu'on peut extraire pour évaluer les performances de calcul. Nous avons par la suite présenté les algorithmes à étudier, à savoir le schéma de Horner et le PMVC.

Nous avons présenté par la suite notre étude sur la définition des métriques affectant le choix du format de compression. Notre étude est limitée à quatre formats de compression, à savoir, les formats CSR, CSC, COO et ELL. Nous avons commencé cette étude par une analyse de modèle de programmation *data parallel* [PeE06], et ce dans le cas de calcul de Horner. L'idée principale consiste à concevoir une géométrie virtuelle constituée d'un ensemble de processeurs virtuels. Les éléments non nuls de la matrice sont affectés par la suite à cette géométrie. Deux démarches différentes sont présentées. Selon la première démarche, à chaque processeur virtuel, nous affectons un élément non nul de la matrice. Selon la deuxième démarche, chaque processeur virtuel reçoit un vecteur i.e. une ligne ou une colonne (selon la direction de stockage des éléments non nuls) de la matrice. Dans un deuxième lieu, nous avons présenté une étude concernant l'analyse des algorithmes où nous avons pu extraire un ensemble de métriques représentant le nombre des accès mémoire.

ETUDE EXPERIMENTALE POUR L’EVALUATION DES METRIQUES

Introduction

Nous nous intéressons dans ce chapitre à la validation de notre cas d'étude proposé pour la détection automatique du meilleur format de compression. La section 0 est dévolue à l'illustration de notre étude sur des cas de tests pathologiques. L'étude expérimentale montre que l'analyse des modèles *data parallel* seule ne permet pas de prendre une décision. Ainsi, nous proposons dans la section 0 un modèle de coût et nous discutons de la possibilité de la présentation exacte de ce dernier.

Etude expérimentale : calcul du schéma de Horner

Nous présentons dans cette partie notre étude expérimentale réalisée afin de valider notre processus de décision (voir Chapitre 3, **Algorithme Error!** *No text of specified style in document.**-10***). Cette étude est effectuée sur un ensemble de matrices pathologiques dont on varie la taille n et la largeur $larg$ (voir Chapitre 3, Figure Error! ***No text of specified style in document.**-59* et Figure Error! ***No text of specified style in document.**-60*).

Plateforme de test et méthodologie

Nous avons effectué nos tests sur des machines appartenant au site Grid'5000 qui est une plate-forme expérimentale pour la recherche sur les systèmes distribués [MNM11]. Nos tests présentés dans cette partie ont été effectués sur le site de Nancy à travers deux nœuds de son cluster 'Grimoire' (voir Tableau 9). Ce cluster utilise le processeur Intel Xeon E5-2630 v3 d'Intel, basé sur l'architecture $Haswell$, et intégrant huit cœurs par CPU. On a alors $PP = 32$ processeurs physiques.

Tableau 9 Plateforme de test

CPU :	**Intel Xeon E5-2630 v3 (2 CPU/nœud, 8 cœurs/CPU)**
# nœuds	2
Fréquence	2.40 GHz
Mémoire :	128 GB

Les matrices utilisées dans cette première étude sont de deux types :

- Des matrices pathologiques $MatRow$ et $MatCol$ qu'on a générées aléatoirement. Ces matrices correspondent aux cas extrêmes détectés lors de l'étude du modèle *data parallel*

pour un format donné. Les matrices testées sont de tailles $n = 100\,000$ et de largeur $larg$ variant entre 10 et 7000.

$larg \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000\}$

- Des matrices réelles sélectionnées à partir de la collection des matrices réelles de l'université de Floride (matrices avec structures régulières et irrégulières) [DaY11].

Nous avons choisi d'affecter à chaque processeur physique $\frac{n}{PP}$ processeurs virtuels. Ainsi, nous avons un nombre équilibré de lignes (respectivement de colonnes) entre les processeurs dans le cas de CSR (respectivement CSC).

Nos algorithmes sont implémentés en langage C. Nous avons utilisé l'environnement MPI (bibliothèque MPICH2) pour la mise en œuvre du modèle de programmation *data parallel*.

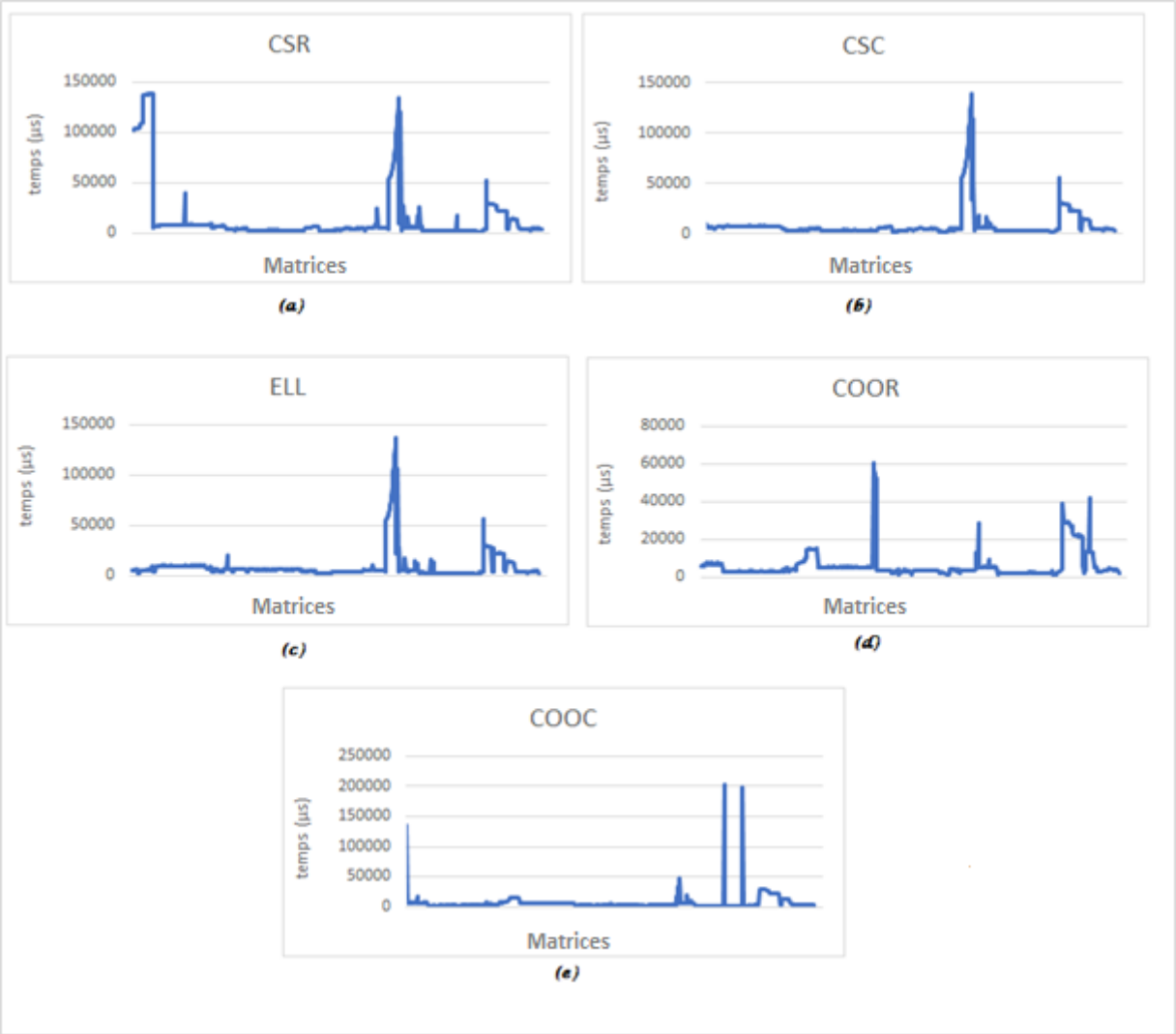

Figure ***Error! No text of specified style in document.****-61 Décalage des dates de démarrage des processus MPI dans le cas de calcul du PMVC*

Synchronisation des processus MPI

Nous présentons dans cette partie une étude expérimentale pour valider ce qui a été présenté dans la section 3.3 du chapitre 3, et ce dans le cas de l'utilisation de MPI. Figure Error! ***No text of specified style in document.****-61* illustre l'écart entre la date de démarrage du premier et du dernier processeur pour différentes implémentations du PMVC (associées aux différents formats de compression CSR, COO, ELL, COOR, COOC). Cet écart peut produire une fausse prédiction du MFC qui dépend des temps de calcul. Ainsi, pour le reste des expérimentations, le temps d'exécution présenté correspond à celui du processeur ayant mis plus de temps.

Résultats numériques et interprétation

Figure Error! ***No text of specified style in document.****-62* et Figure Error! ***No text of specified style in document.****-63* illustrent les résultats expérimentaux obtenus pour $MatRow$ et $MatCol$ lorsque les matrices sont compressées selon les formats CSR et CSC.

Dans le cas de $MatRow$, nous avons $\mathrm{nz}_{\mathrm{maxC}} = larg + 1 < \mathrm{nz}_{\mathrm{maxR}} = 100\,000$, avec $larg \in [10\ 7000]$. Selon l'algorithme de décision (voir Chapitre 3, **Algorithme Error!** *No text of specified style in document.****-10***), le MFC est le format CSC. Ce résultat est confirmé par le résultat expérimental (Figure Error! ***No text of specified style in document.****-62*).

Dans le cas de $MatCol$, nous avons $\mathrm{nz}_{\mathrm{maxR}} = larg + 1 < \mathrm{nz}_{\mathrm{maxC}} = 100\,000$, avec $larg \in [10\ 7000]$. Selon l'algorithme de décision (voir Chapitre 3, **Algorithme Error!** *No text of specified style in document.****-10***), le MFC est le format CSR. Ce résultat est confirmé par le résultat expérimental (Figure Error! ***No text of specified style in document.****-63*).

Nous avons ainsi prouvé la cohérence de notre étude avec les résultats expérimentaux, et ce dans le cas des matrices pathologiques. Nous essayons dans un deuxième cas de vérifier la validité de ce résultat dans le cas des matrices réelles de différentes structures. Nos tests ont été effectués sur 26 matrices représentatives de différentes tailles. Le Tableau 10 représente les résultats obtenus pour la prédiction du MFC. Le $\mathbf{MFC_{théorique}}$ correspond au format de compression sélectionné à partir de notre étude (voir Chapitre 3, **Algorithme Error!** *No text of specified style in document.****-10***) alors que $\mathbf{MFC_{pratique}}$ représente celui obtenu de notre étude expérimentale. Nous remarquons que 23% des cas correspondent à une mauvaise prédiction. Ce résultat peut être dû à plusieurs facteurs :

- Défauts de caches : l'un des problèmes les plus rencontrés lors des calculs est le problème de défauts de caches. Ceci peut être dû à plusieurs causes tels que la taille des données par rapport à la taille mémoire ou encore l'algorithme utilisé qui fait appel à

des données non contiguës dans la mémoire. Ces problèmes provoquent des temps d'exécution supplémentaires.

Ceci est illustré dans Figure Error! ***No text of specified style in document.****-64* où sont représentés les défauts de cache pour le calcul parallèle de la méthode de Horner dans le cas de la matrice ***webbase-1M***. La simulation du cache a été effectuée à l'aide de l'outil de profilage et de débogage mémoire ***Valgrind*** [VGD]. Nous remarquons que le nombre de défauts de cache L1 provoqué par le format CSC est largement supérieur à celui obtenu en utilisant CSR. Ceci peut expliquer le résultat obtenu dans le Tableau 10 où il a été démontré que, dans le cas de la matrice *webbase-1M*, le format CSR est meilleur que CSC.

- Mouvement de données et communications : nous avons dès le début choisi d'utiliser le schéma de Horner comme noyau pour notre étude de cas car il représente bien les communications et les mouvements des données. Néanmoins, ces dernières peuvent provoquer des problèmes lors de l'évaluation des performances puisqu'elles ne sont pas optimisées dans nos tests.
- Ordonnancement des processeurs virtuels : dans cette partie, nous avons choisi d'affecter à chaque processeur physique $\frac{n}{PP}$ processeurs virtuels. Nous remarquons bien que cette méthode naïve d'ordonnancement peut provoquer un déséquilibre de charge entre les processeurs et par la suite affecter l'exactitude des résultats. Dans le cas des matrices pathologiques, cet ordonnancement peut être la cause de la cohérence entre notre étude théorique et les résultats expérimentaux. Prenons par exemple le cas de la matrice $MatRow$:
    - En utilisant CSR, il y aura toujours au moins un processeur avec une charge maximale de 100000×31250 éléments non nuls,
    - En utilisant CSC, la charge des processeurs est équilibrée. Cette charge est de l'ordre de $larg \times 31250$ éléments non nuls, avec $larg \leq 7000$.

  Ainsi, il est évident que dans ce cas l'utilisation du format de compression CSC donne de meilleures performances.

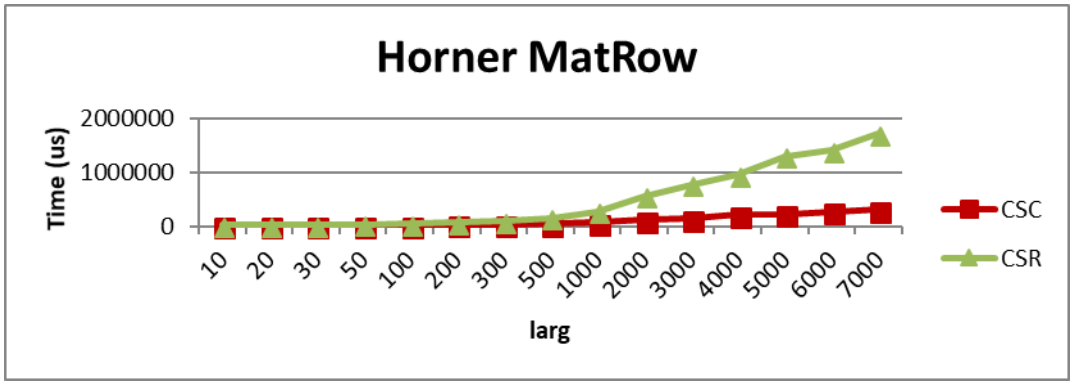

Figure ***Error! No text of specified style in document.****-62 Comparaison des formats CSR et CSC dans le cas de calcul du schéma de Horner sur une plateforme parallèle où les matrices en entrée sont de types MatRow*

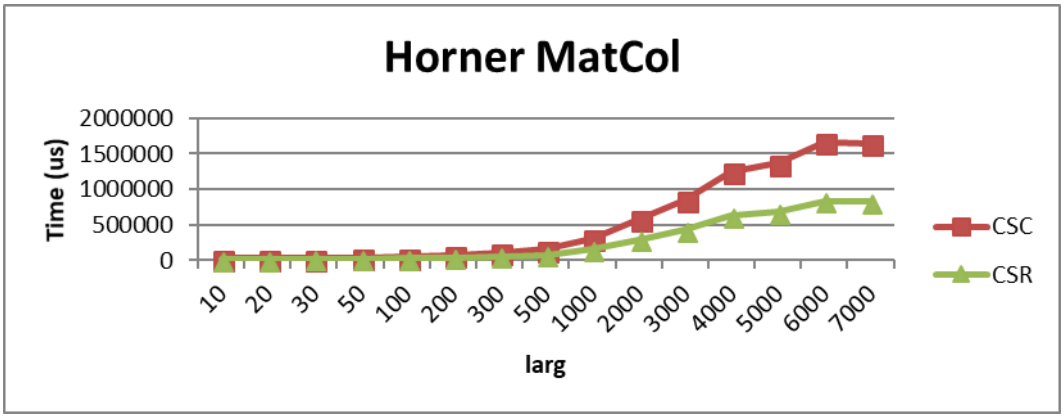

Figure ***Error! No text of specified style in document.****-63 Comparaison des formats CSR et CSC dans le cas de calcul du schéma de Horner sur une plateforme parallèle où les matrices en entrée sont de types MatCol*

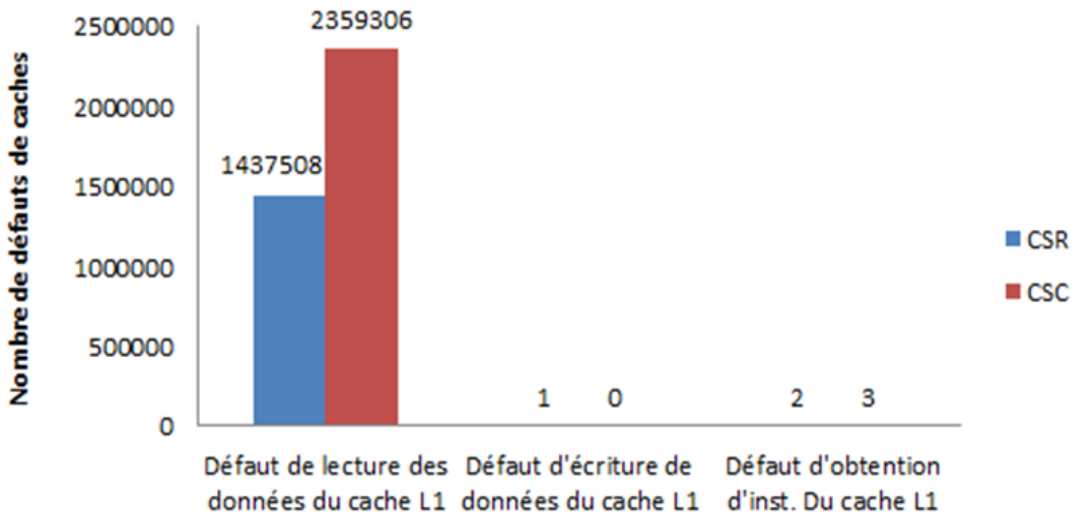

Figure ***Error! No text of specified style in document.****-64Nombre de défauts de cache L lors de calcul du schéma de Horner dans le cas de la matrice webbase-1M*

Pour remédier à ces problèmes, nous proposons dans la section suivante de faire une étude expérimentale en utilisant le noyau de calcul PMVC au lieu du schéma de Horner afin d'éviter le problème de mouvement des données. Nous rappelons que notre étude effectuée pour le cas de Horner reste valable pour le cas du PMVC. La métrique qui nous intéresse est la métrique $\boldsymbol{co\hat{u}t}_{\times}$ représentant le coût d'une opération de multiplication *data parallel*.
En outre, différentes approches d'ordonnancement de processeurs virtuels seront utilisées.

Tableau 10 Calcul de Horner dans le cas de matrices réelles

Matrice	n	nnz	nz_{maxR}	nz_{maxC}	$MFC_{théorique}$	$MFC_{pratique}$
ASIC_680ks	682712	2329176	210	210	-	CSR
Bcircuit	68902	375558	34	34	-	CSR
bcsstk35	30237	740200	**66**	166	CSR	CSR
c-66b	49989	274498	5237	**2914**	CSC	CSR
circuit_4	80209	307604	8900	**6750**	CSC	CSR
g7jac160	47430	656616	**120**	153	CSR	CSR
minsurfo	40806	122214	4	4	-	CSR
rajat18	94294	485143	**28711**	28747	CSR	CSR
RFdevice	74104	365580	**9**	271	CSR	CSR
webbase-1M	1000005	3105536	28685	**4700**	CSC	CSR

ASIC_100k	99340	954163	99029	99029	-	CSR
c-big	345241	1343126	**6875**	19578	CSR	CSR
lung2	109460	492564	8	8	-	CSR
para-9	155924	5416358	6931	6931	-	CSR
shyy161	76480	329762	6	6	-	CSR
ASIC_320ks	321671	1827807	412	412	-	CSR
Hamrle3	1447360	5514242	9	**6**	CSC	CSR
rajat21	411676	1893370	**100470**	118689	CSR	CSR
boneS01	127224	3421188	42	42	-	CSR
mario002	389874	1167685	7	**5**	CSC	CSR
rajat23	110355	556938	**3269**	3401	CSR	CSR
shallow_water1	81920	204800	4	4	-	CSR
boyd2	466316	890091	93262	**10992**	CSC	CSR
language	399130	1216334	**107**	11555	CSR	CSR
rajat29	643994	4866270	**454099**	454521	CSR	CSR
shipsec5	179860	5146478	**78**	96	CSR	CSR
					38.5% CSR 23% CSC	**100% CSR**

Etude expérimentale : calcul du produit matrice vecteur creux

Notre plateforme de test est constituée de PP processeurs physiques. À chaque processeur physique est affecté un ou plusieurs processeurs virtuels. Nous présentons donc deux cas d'étude comme suit :

- Dans un premier cas, nous essayons d'évaluer notre solution sur une plateforme physique équivalente à la géométrie virtuelle et ce en utilisant un nombre de processeurs physiques $PP = Pv$.
- Dans le deuxième cas, nous analysons les performances des noyaux de calcul dans le cas où le nombre de processeurs physiques est inférieur à celui du processeurs virtuels (Ce cas correspond au type d'architecture utilisé réellement). Ainsi, chaque processeur physique traite un ensemble de lignes (respectivement colonnes) de la matrice dans le cas du format CSR (respectivement CSC). Pour l'ordonnancement des tâches, nous avons utilisé deux approches : l'algorithme NEZGT et l'ordonnanceur MPI.

Plate-forme de Test

Tableau 11 Plateforme de test

Cluster	**paravance**
CPU :	**Intel Xeon E5-2630 v3 (2 CPU/nœud, 8 cœurs/CPU)**
Architecture	Haswell
# total de nœuds	72
# total de CPUs	144 (2 CPU/nœud)
# total de cœurs	1152 (8coeurs / CPU)
Fréquence	2.40 GHz
Mémoire :	128 GB

Nous avons effectué nos tests sur des machines appartenant au site Grid'5000 qui est une plate-forme expérimentale pour la recherche sur les systèmes distribués [MNM11]. Nos tests ont été effectués sur le site de Rennes à travers son cluster 'Paravance' (voir Tableau 11). Ce cluster utilise le processeur Intel Xeon E5-2630 v3 d'Intel, basé sur l'architecture $Haswel$, et intégrant huit cœurs par CPU.

Nos algorithmes sont implémentés en langage C. Nous avons utilisé l'environnement MPI (bibliothèque MPICH2) pour la mise en œuvre du modèle de programmation *data parallel*.

Premier cas : $PP = Pv$

Dans un premier cas, nous essayons d'évaluer notre solution sur une plateforme physique équivalente à la géométrie virtuelle et ce en utilisant un nombre de processeurs physiques $PP = Pv$. Suite aux contraintes de la plateforme expérimentale, l'étude expérimentale sera effectuée sur une plateforme composée de $PP = 1152$ processeurs physiques. C'est le nombre maximal de processeurs physiques qu'on a pu avoir sur la plateforme Grid5000 (Tableau 11). Le temps d'exécution total de calcul du PMVC est enregistré pour chaque matrice.

Les matrices de tests sont des matrices pathologiques $MatRow$ et $MatCol$ qu'on a générées aléatoirement. Les matrices sont de tailles $n = PP = Pv = 1152$ et de largeurs $larg$ variant entre 10 et 900.

Algorithme *Error! No text of specified style*	***Algorithme*** *Error! No text of specified style*

*in document.**-11** Calcul de l'opération « dot product » au niveau du processeur le plus chargé (* $\boldsymbol{PP_{load\_mx}}$*) dans le cas où* $PP = Pv$	*in document.**-12** Calcul de l'opération « saxpy » au niveau du processeur le plus chargé (* $\boldsymbol{PP_{load\_mx}}$*) dans le cas où* $PP = Pv$
POUR $j = 1, \text{nz}_{\text{maxR}}$ $indX = ind\_col[j];$ $s = s + val[j] \times x[indX];$ ***FIN POUR*** $\boldsymbol{y}[PP_{load\_mx}] = \boldsymbol{s}$;	$\boldsymbol{w} = \boldsymbol{x}[PP_{load\_mx}]$; ***POUR*** $j = 1, \text{nz}_{\text{maxC}}$ $indR = ind\_ligne[j]$; $y\,[indR] = y\,[indR] + val[j] \times w$; ***FIN POUR***

Notons que lorsque $PP = Pv$, le temps total d'exécution est théoriquement égal au temps d'exécution du processeur le plus chargé. Dans le cas du format CSR (respectivement CSC), le processeur le plus chargé est celui qui traite nz_{maxR} (respectivement nz_{maxC}) éléments non nuls. Selon notre étude (voir Chapitre 3, **Algorithme Error!** *No text of specified style in document.**-10***) qui correspond à l'application du modèle *data parallel*, on a :

- Pour les matrices ayant une structure proche de celle de la matrice $MatRow$, le MFC correspond au format CSC.
- Pour les matrices ayant une structure proche de celle de la matrice $MatCol$, le MFC correspond au format CSR.

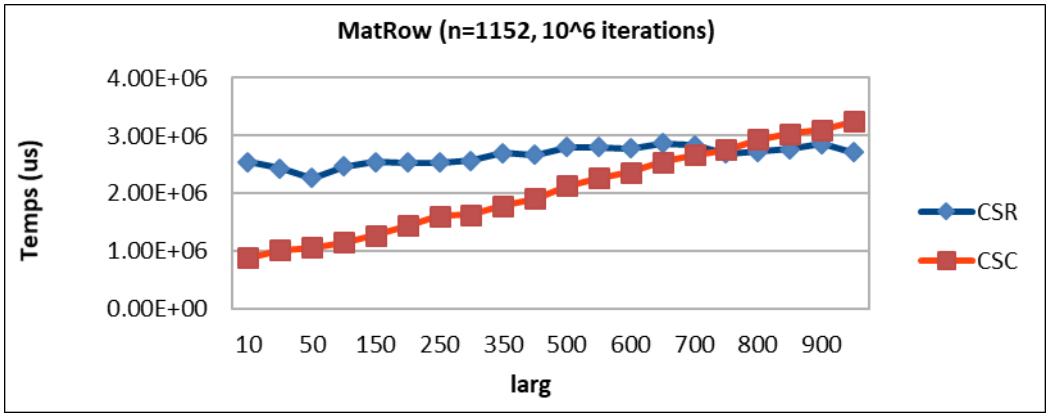

Figure ***Error! No text of specified style in document.****-65 Temps d'exécution du PMVC dans le cas de MatRow (*$PP = Pv = 1152$*)*

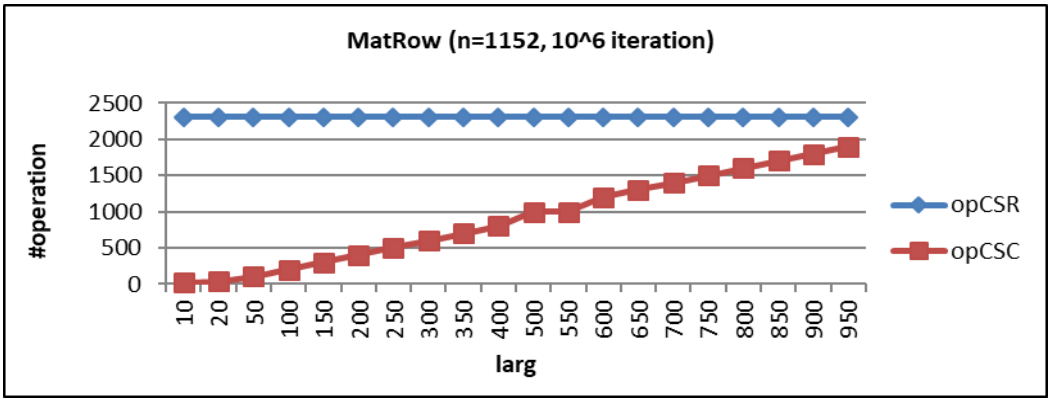

Figure ***Error! No text of specified style in document.****-66 nombre des opérations du PMVC dans le cas de MatRow (*$PP = Pv = 1152$*)*

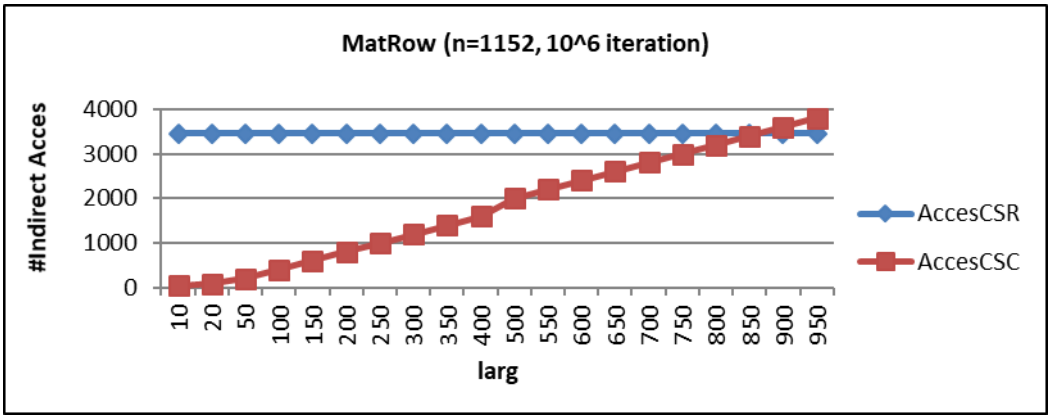

Figure ***Error! No text of specified style in document.****-67 nombre des accès mémoire (accès indirects) lors du calcul du PMVC dans le cas de MatRow (PP=Pv=1152)*

Figure Error! ***No text of specified style in document.****-65* (respectivement Figure Error! ***No text of specified style in document.****-68*) illustre le temps d'exécution du PMVC dans le cas des matrices $MatRow$ (respectivement $MatCol$) lorsque la matrice A est compressée selon le format CSR puis CSC.

- Dans le cas de $MatCol$, quel que soit la valeur de $larg$, les résultats expérimentaux sont conformes à notre étude.
- Cependant, pour le cas des matrices $MatRow$, les résultats expérimentaux et théoriques ne sont cohérents que pour certaines valeurs de $larg$ ($larg < 750$).

  Lorsque la matrice $MatRow$ est compressée selon le format CSC, nous remarquons que le nombre d'opérations au niveau du processeur le plus chargé augmente avec l'augmentation de la valeur de $larg$. Cependant, il reste toujours inférieur à celui du cas où la matrice est compressée selon le format CSR (Figure Error! ***No text of specified***

***style in document.**-66*).

Néanmoins, nous remarquons que, à partir d'une certaine valeur de $larg$ ($larg$ =750), le PMVC correspondant au format CSR donne de meilleures performances que celles du PMVC correspondant au format CSC (Figure Error! ***No text of specified style in document.**-65*). Ainsi, nous pouvons conclure que les métriques extraites à partir de l'analyse du modèle *data parallel* (la charge maximale des processeurs physiques) ne sont pas suffisantes pour pouvoir sélectionner le MFC.

- Une analyse des algorithmes de calcul du PMVC associés aux formats CSR et CSC (**Algorithme Error!** *No text of specified style in document.**-11*** et **Algorithme Error!** *No text of specified style in document.**-12***) montre qu'un autre paramètre peut affecter le temps de calcul. Ce paramètre correspond au nombre des accès mémoires qui dépend du format utilisé, plus particulièrement il dépend de nz_{maxR} et nz_{maxC}.

Tableau 12 Nombre des accès mémoire lors de calcul du PMVC dans le cas où $PP = Pv$

	#Accès indirects	**#Lectures**	**#Ecritures**	**#Opérations**
CSR	$3 \times nz_{maxR}$	$4 \times nz_{maxR}$	$2 \times nz_{maxR}$	$2 \times nz_{maxR}$
CSC	$4 \times nz_{maxC}$	$4 \times nz_{maxC}$	$2 \times nz_{maxC}$	$2 \times nz_{maxC}$

Le Tableau 12 représente une étude concernant le nombre des accès mémoires et des opérations dans le cas de calcul du PMVC sur une plateforme parallèle avec $PP = Pv$ processeurs physiques. Ces valeurs sont associées au processeur le plus chargé puisque le temps d'exécution totale dépend de ce dernier (**Algorithme Error!** *No text of specified style in document.**-11*** et **Algorithme Error!** *No text of specified style in document.**-12***). Le processeur le plus chargé dans le cas de CSR (respectivement CSC) traite une ligne (respectivement une colonne) de la matrice avec nz_{maxR} (respectivement nz_{maxC}) éléments non nuls. Un accès indirect correspond à l'accès à un élément de tableau (par exemple $val[j]$).

Nous remarquons à partir de Figure Error! ***No text of specified style in document.**-69* que le nombre de lectures et d'écritures des données de $MatRow$ dans le cas du format CSC est toujours inférieur à celui dans le cas du format CSR quel que soit la valeur de $larg$. Malgré que le nombre des accès mémoire en lecture ou écriture affecte le temps total d'exécution, il ne peut dans ce cas étudié expliquer les résultats obtenus dans

Figure Error! ***No text of specified style in document.****-65*. Néanmoins, le nombre des accès indirects aux éléments de tableau peut être l'origine de ce résultat (*Figure* ***Error! No text of specified style in document.****-67*). En effet, chaque accès indirect nécessite un temps supplémentaire pour le calcul d'adresse.

Nous remarquons à partir de la Figure Error! ***No text of specified style in document.****-67* que le nombre total des accès indirects lors de calcul du PMVC associé au format CSC est supérieur à celui du PMVC associé au format CSR, et ce à partir d'une valeur de $larg$ =850. Notons que cette borne n'est pas la même borne détectée dans la Figure Error! ***No text of specified style in document.****-65* ($larg$ =750). Ainsi, nous pouvons conclure que le temps de calcul du PMVC n'est pas affecté par un seul facteur. En effet, le nombre croissant d'opérations, de lectures/écritures des données et le nombre des accès indirects affectent en même temps les performances de ce noyau.

Puisque chaque format a son propre coût, nous définissons alors $CSR_{coût}$ et $CSC_{coût}$ dans le cas où $PP = Pv$ tels que :

$$CSR_{coût} = 3 \times \mathrm{nz}_{\mathrm{maxR}} \times AI_{coût} + 4 \times \mathrm{nz}_{\mathrm{maxR}} \times L_{coût} + 2 \times \mathrm{nz}_{\mathrm{maxR}} \times E_{coût} + 2 \times \mathrm{nz}_{\mathrm{maxR}} \times OP_{coût}$$

$$CSC_{coût} = 4 \times \mathrm{nz}_{\mathrm{maxC}} \times AI_{coût} + 4 \times \mathrm{nz}_{\mathrm{maxC}} \times L_{coût} + 2 \times \mathrm{nz}_{\mathrm{maxC}} \times E_{coût} + 2 \times \mathrm{nz}_{\mathrm{maxC}} \times OP_{coût}$$

Avec:

- $AI_{coût}$ est le coût de calcul de l'adresse d'un élément de tableau (Accès Indirect),
- $L_{coût}$ est le coût d'un accès mémoire en Lecture,
- $E_{coût}$ est le coût d'un accès mémoire en Ecriture,
- $OP_{coût}$ est le coût d'une opération arithmétique (multiplication ou addition).

Les différents poids associés aux composantes des deux équations ($AI_{coût}$, $L_{coût}$, $E_{coût}$ et $OP_{coût}$) sont déterminés à partir du Tableau 12.

Nous définissons alors un nouveau processus de décision présenté dans **Algorithme Error!** *No text of specified style in document.****-13***. On note que $AI_{coût}$, $L_{coût}$, $E_{coût}$ et $OP_{coût}$ dépendent de l'architecture utilisée.

Algorithme *Error! No text of specified style in document.****-13*** *Processus de décision basé*

sur les métriques des coûts $(PP = Pv)$

$Extraire\_caractéristiques\_matrice(A);$

SI $EstProche(Structure(A), Structure(matrice\_Pathologique))$ **ALORS**

SI $(CSR_{coût} < CSC_{coût})$ **ALORS**

$format = CSR;$

SINON

$format = CSC;$

FIN SI

SINON

$MAJ\ base\_de\_test();$

$MAJ\ base\_de\_règles\_de\_décision();$

$format = Selectionner\_format(A);$

FIN SI

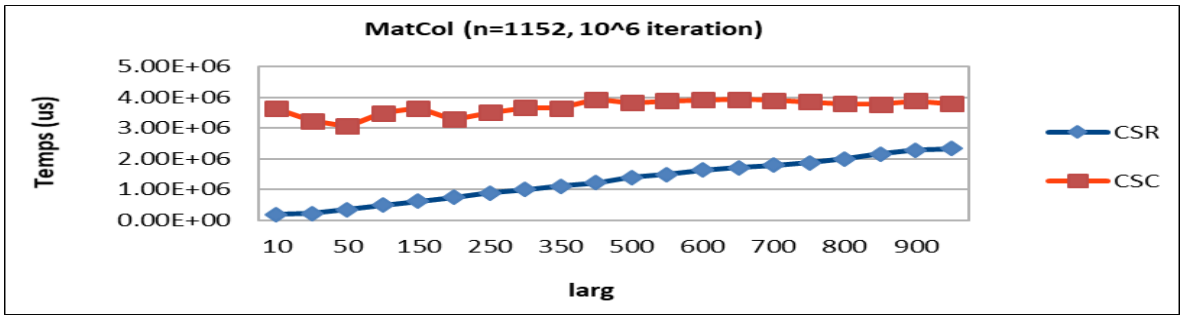

Figure ***Error! No text of specified style in document.****-68 Temps d'exécution du PMVC dans le cas de MatCol* $(PP = Pv = 1152)$

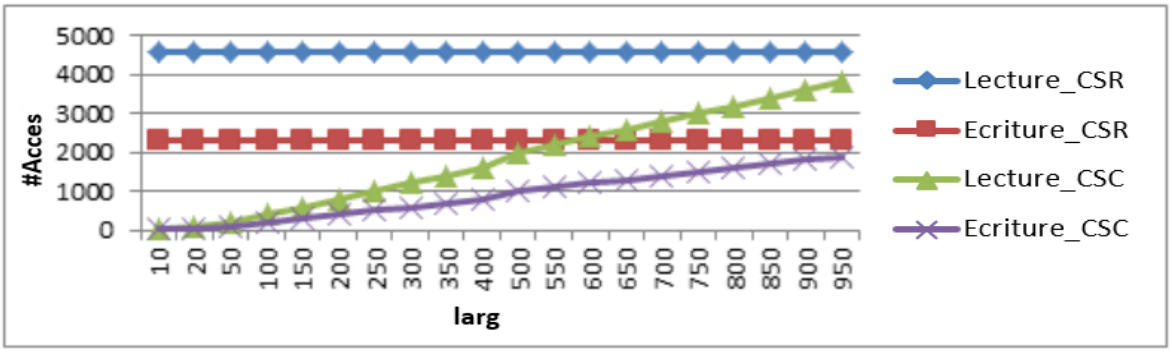

Figure ***Error! No text of specified style in document.****-69 nombre des accès mémoire en lecture/écriture lors du calcul du PMVC par le processeur le plus chargé dans le cas de MatRow (PP=Pv=1152)*

Deuxième cas : $PP < Pv$

Nous avons pu démontrer dans la section précédente que les performances du PMVC ne dépendent pas uniquement de la charge des processeurs. En effet, le nombre des accès aux mémoires affecte aussi les performances de ce noyau. Cette conclusion a été tirée grâce à l'étude d'un cas pathologique de matrice $MatRow$ qu'on a défini suite à notre étude du modèle *data parallel*. Pour valider cette conclusion, nous analysons dans cette partie les performances de calcul du PMVC dans le cas des mêmes matrices pathologiques $MatRow$, sauf que cette fois ci nous utilisons une architecture parallèle à $PP < Pv$ processeurs physiques. Ainsi, nous affectons à chaque processeur physique un ensemble de processeurs virtuels.

- Dans le cas du format CSR : un ensemble de lignes est affecté au même processeur physique,
- Dans le cas du format CSC : un ensemble de colonnes est affecté au même processeur physique.

Pour l'ordonnancement des processeurs virtuels nous avons utilisé deux approches différentes :

- Un ordonnancement MPI : c'est à l'environnement MPI de faire l'affectation des processeurs virtuels aux processeurs physiques,
- Un ordonnancement en utilisant l'algorithme NEZGT (voir la section 0): c'est un algorithme présenté pour la fragmentation des matrices creuses qu'on a utilisé pour ordonnancer les processeurs virtuels [Ham10].

Ordonnancement MPI

l) Stratégie d'ordonnancement MPI

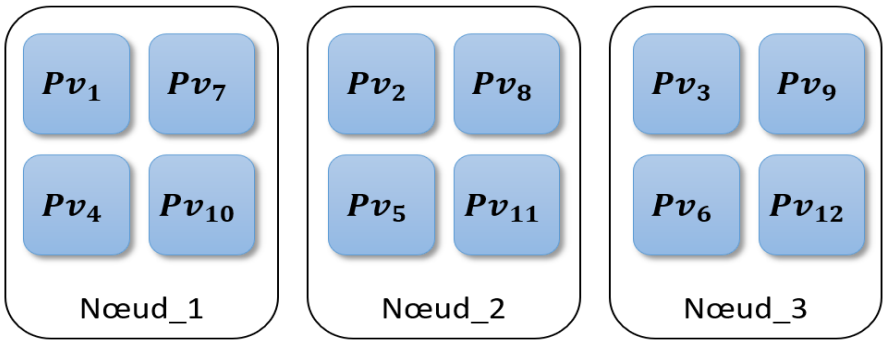

*Figure **Error! No text of specified style in document.**-70 Ordonnancement des processus MPI*

La stratégie d'ordonnancement des processus MPI, utilisée dans notre plateforme de test, consiste à affecter ces derniers à des processeurs physiques à tour de rôle. Dans notre cas, chaque processeur virtuel correspond à un processus MPI. Les premiers PP processus MPI seront affectés aux PP processeurs physiques. Cette étape est répétée jusqu'à épuisement des processus. Dans notre cas, lorsque la matrice est stockée selon le format CSR (respectivement CSC), chaque ligne (respectivement colonne) de la matrice est gérée par un processus MPI.

Données de tests

Nous évaluons les performances du PMVC lorsque la matrice $MatRow$ est compressée selon le format CSR puis CSC. Etant donnée qu'on est limité par le nombre de processus MPI qui peuvent être gérés par un cœur physique, nous travaillons avec des matrices $MatRow$ de tailles $n = 10\,000$ avec des valeurs de largeur $larg$ variant entre 10 et 8000.

Nos expérimentations sont réalisées sur une plateforme parallèle contenant 72 nœuds, chaque nœud avec deux processeurs Intel Xeon E5-2630 v3 (avec 8 cœurs chacun). Le nombre total des processeurs physiques est égal à 1152. Le temps d'exécution total de calcul du PMVC est enregistré pour chaque matrice.

Ordonnancement en utilisant l'algorithme NEZGT

m) Algorithme NEZGT

L'approche NEZGT est une approche proposée dans [EHM06b] [Ham10] pour la distribution du PMVC sur une grille de calcul. C'est une version **G**énéralisée avec **T**ri décroissant de l'approche **NEZ** (**N**ombre **E**quilibré de non-**Z**éros) qui vise à décomposer une matrice donnée en un ensemble de fragments de lignes contiguës avec nombre équilibré des éléments non nuls. L'approche NEZGT se décompose en trois phases qui se détaillent comme suit :

- La phase 0 est une phase de tri, elle consiste à trier les lignes de la matrice creuse par ordre décroissant de nombre d'éléments non nuls.
- La phase 1 est basée sur l'heuristique LS (*List Scheduling*): Elle consiste d'abord, à affecter les *f* premières lignes de la matrice en entrée aux *f* fragments (*f* est le nombre

de fragments souhaité): chaque ligne i (i=1…*f*) est affectée au fragment i. Par la suite, le reste des lignes sont affectées aux fragments en se basant sur leurs charges (nombre d'éléments non nuls). En d'autres termes, la ligne courante non affectée est toujours attribuée au fragment le moins chargé jusqu'à épuisement des lignes.

- La phase 2, 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 [Ham10].

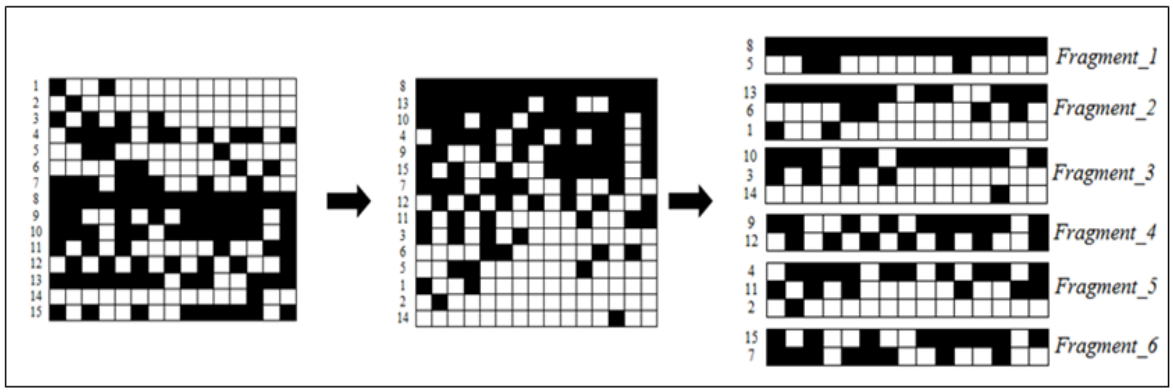

Figure ***Error! No text of specified style in document.****-71: Exemple d'application de la méthode NEZGT*

Données de tests

Nous évaluons les performances du PMVC lorsque la matrice $MatRow$ est compressée selon le format CSR puis CSC. Les matrices de tests sont des matrices de tailles $n = 50\,000$ avec des valeurs de largeur $larg$ variant entre 10 et 20 000.

Nos expérimentations sont réalisées sur une plateforme parallèle contenant 10 nœuds, chaque nœud avec deux processeurs Intel Xeon E5-2630 v3 (avec 8 cœurs chacun). Le nombre total des processeurs physiques est égal à 160. Le temps d'exécution total de calcul du PMVC est enregistré pour chaque matrice.

Dans le cas du format de compression CSC, l'algorithme NEZGT traite des colonnes au lieu des lignes.

Résultats Numériques et Interprétations

Les résultats présentés dans cette partie correspondent au processeur avec le temps d'exécution maximal qu'on note $\boldsymbol{PP_{tmax}}$.

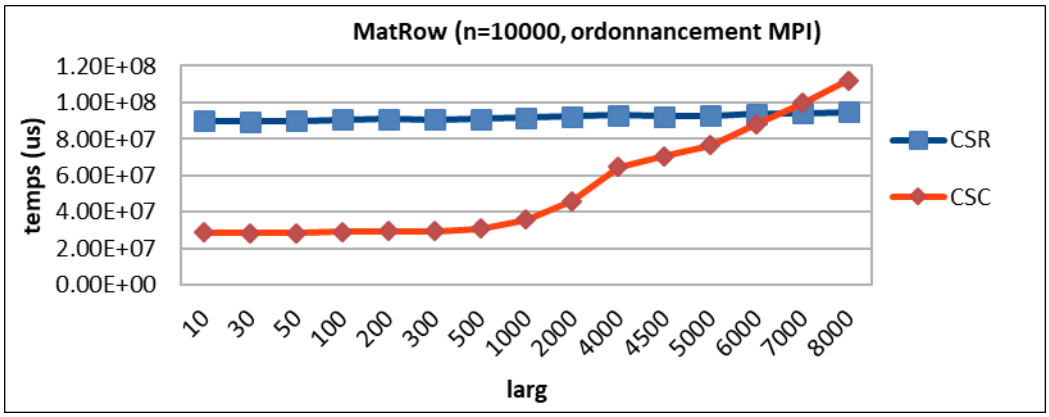

Figure ***Error! No text of specified style in document.****-72 Temps d'exécution du PMVC dans le cas de MatRow (PP<Pv, ordonnancement MPI)*

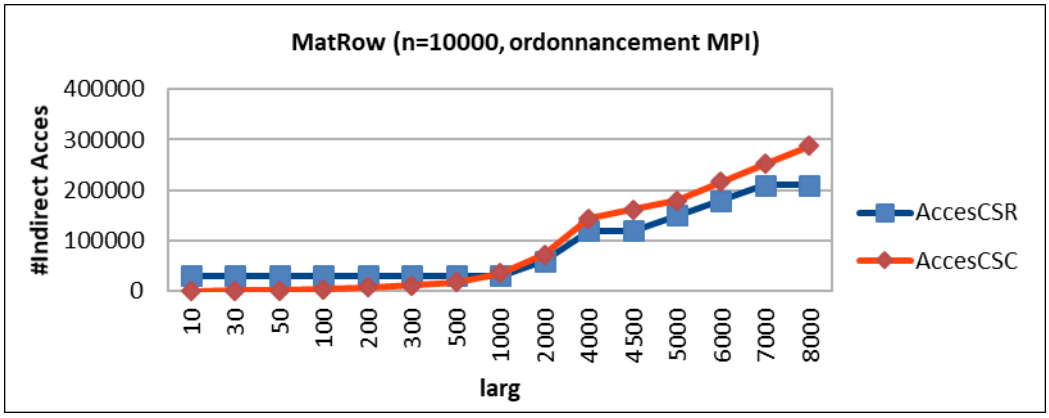

Figure ***Error! No text of specified style in document.****-73 nombre des accès mémoire (accès indirects) lors du calcul du PMVC dans le cas de MatRow (PP<Pv, ordonnancement MPI)*

L'étude expérimentale, utilisant les ordonnancements MPI ou NEZGT, conduit aux conclusions suivantes :

- Dans 50% des cas, le processeur avec le temps d'exécution maximal ne correspond pas au processeur le plus chargé. Ces résultats prouvent que le nombre des opérations (déduit de l'analyse du modèle *data parallel*) n'est pas une métrique suffisante pour prédire les performances du PMVC et sélectionner le MFC.
- Nous remarquons à partir de Figure Error! ***No text of specified style in document.****-73* et Figure Error! ***No text of specified style in document.****-76* que, à partir d'une certaine valeur de $larg$, le nombre des accès indirects lorsque la matrice est stockée selon le

format CSC devient supérieur à celui du cas où la matrice est stockée selon le format CSR.

- Dans le cas de l'ordonnancement MPI, nous remarquons que lorsque $larg < 1000$, le PMVC associé au format de compression CSC a de meilleures performances par rapport à celui associé au format CSR (Figure Error! ***No text of specified style in document.***-*72*). Ceci est dû au fait que le nombre des opérations et des accès mémoires (accès indirects) dans le cas de CSC sont faibles par rapport à celui du format CSR (voir Figure Error! ***No text of specified style in document.***-*73* et Figure Error! ***No text of specified style in document.***-*74*). A partir d'une valeur $larg = 1000$, la variation au niveau du temps d'exécution est expliquée par le nombre croissant des accès indirects (qui devient dans le cas de CSC supérieur à celui de CSR).
- Dans le cas de l'ordonnancement NEZGT, nous remarquons d'après Figure Error! ***No text of specified style in document.***-*75* que les performances des PMVC associés aux formats de compression CSR et CSC sont proches quel que soit la valeur de $larg$. Pour $larg \leq 500$, cela est dû au nombre restreint d'opérations et des accès indirects pour les deux formats. A partir d'une valeur $larg = 1000$, on remarque que le nombre des accès indirects dans le cas de CSC devient légèrement supérieur à celui de CSR. Néanmoins, on n'obtient pas de différence dans les performances en raison du nombre égal d'opérations (Figure Error! ***No text of specified style in document.***-*77*). En effet, on peut conclure que le nombre des accès ne peut pas aider dans le processus de sélection du MFC.

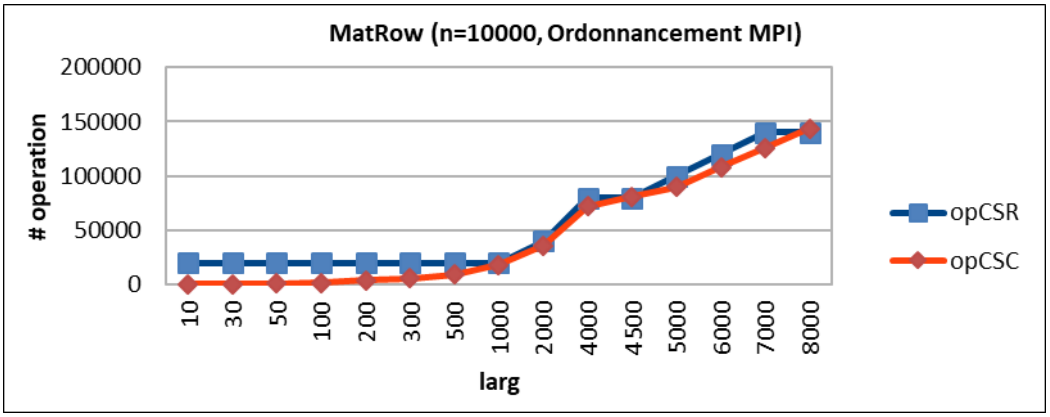

Figure ***Error! No text of specified style in document.****-74 nombre des opérations du PMVC dans le cas de MatRow (PP<Pv, ordonnancement MPI)*

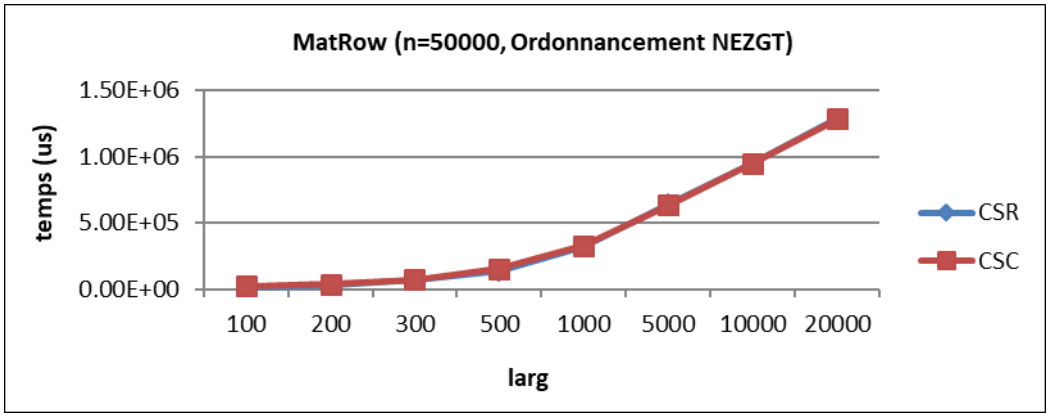

*Figure **Error! No text of specified style in document.**-75 Temps d'exécution du PMVC dans le cas de MatRow (PP<Pv, ordonnancement NEZGT)*

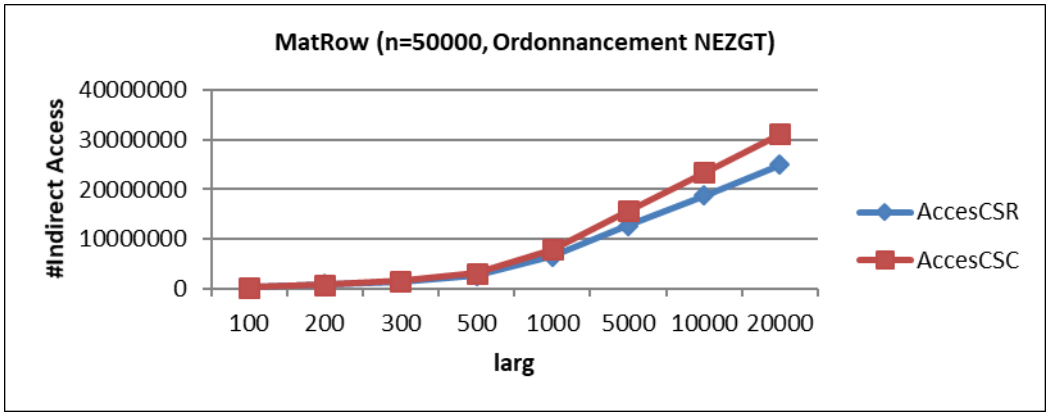

*Figure **Error! No text of specified style in document.**-76 nombre des accès mémoire (accès indirects) lors du calcul du PMVC dans le cas de MatRow (PP<Pv, ordonnancement NEZGT)*

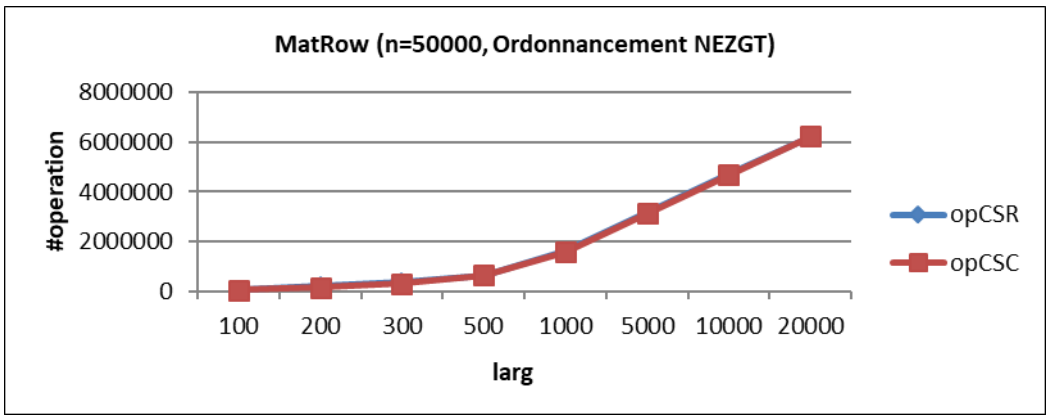

*Figure **Error! No text of specified style in document.**-77 nombre des opérations du PMVC dans le cas de MatRow (PP<Pv, ordonnancement NEZGT)*

Modèle de coût pour la sélection du Meilleur Format de Compression

Nous présentons dans cette partie un processus de décision permettant la sélection du meilleur format de compression (**Algorithme Error!** *No text of specified style in document.**-14***). Ce processus est basé sur les métriques des coûts extraites de l'analyse de l'algorithme. Etant donnée une matrice creuse A, la première étape consiste à extraire un ensemble de caractéristiques de cette matrice ($Extraire\_caractéristiques\_matrice(A)$). Par la suite, on chercher le coût de calcul du PMVC pour chaque format de compression utilisé ($calculer\_coût(A, PMVC, format)$). Le MFC est celui dont la valeur de coût est minimale ($Selectionner\_min(CCSR, CCSC, CELL, CCOOR, CCOOC)$;).

Pour cela on associe à chaque format de compression un coût correspondant au calcul du PMVC dans le cas où $PP < Pv$. Ce coût dépend du nombre des accès mémoires et du nombre des opérations qui peuvent être extraites des algorithmes **Algorithme Error!** *No text of specified style in document.**-11*** et **Algorithme Error!** *No text of specified style in document.**-12***.

Algorithme *Error! No text of specified style in document.**-14** Processus de décision basé sur les métriques des coûts* ($PP < Pv$)

$Extraire\_caractéristiques\_matrice(A)$;
$CCSR = calculer\_coût(A, PMVC, CSR)$;
$CCSC = calculer\_coût(A, PMVC, CSC)$;
$CELL = calculer\_coût(A, PMVC, ELL)$;
$CCOOR = calculer\_coût(A, PMVC, COOR)$;
$CCOOC = calculer\_coût(A, PMVC, COOC)$;
$format = Selectionner\_min(CCSR, CCSC, CELL, CCOOR, CCOOC)$;
FIN SI

Tableau 14 représente la valeur de chacune de ces métriques dans le cas de calcul du PMVC associés aux différents formats de compression CSR, CSC, ELL et COO.

Nous avons présenté dans cette partie un processus de sélection de MFC basé sur un critère de coût. Néanmoins, ces coûts dépendent de l'architecture et des algorithmes d'ordonnancement et donc ne peuvent pas être prédit à l'avance. Ainsi, nous pensons à avoir recours au système d'apprentissage automatique pour bénéficier de toutes les métriques extraites dans ce chapitre et pouvoir sélectionner le MFC.

Tableau 13 Notations utilisées dans les équations

Attribut	Signification
$\boldsymbol{PP}$	Nombre de processeurs physiques
$\boldsymbol{PP_{load\_mx}}$	Processeur avec charge ($load$) maximale
$\boldsymbol{PP_{tmax}}$	Processeur avec temps d'exécution maximale ($tmax$)
$\boldsymbol{nzPP_{tmax}}$	Nombre des éléments non nuls dans $\boldsymbol{PP_{tmax}}$
$\boldsymbol{nrPP_{tmax}}$	Nombre de lignes dans $\boldsymbol{PP_{tmax}}$
$\boldsymbol{ncPP_{tmax}}$	Nombre de colonnes dans $\boldsymbol{PP_{tmax}}$
$\boldsymbol{AI_{coût}}$	Coût d'accès à un élément de tableau (Accès indirect)
$\boldsymbol{L_{coût}}$	Coût d'un accès mémoire en lecture
$\boldsymbol{E_{coût}}$	Coût d'un accès mémoire en écriture
$\boldsymbol{Op_{coût}}$	Coût d'une Opération arithmétique (addition ou multiplication)

Tableau 14 Nombre des accès aux données

Format	# Lectures	# Ecriture	#accès indirects	#opérations
$\boldsymbol{CSR}$	$3 \times \mathrm{nrPP}_{tmax} + 6 \times \mathrm{nzPP}_{tmax}$	$4 \times \mathrm{nrPP}_{tmax} + 3 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{nrPP}_{tmax} + 4 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{nzPP}_{tmax}$
$\boldsymbol{CSC}$	$3 \times \mathrm{ncPP}_{tmax} + 6 \times \mathrm{nzPP}_{tmax}$	$3 \times \mathrm{ncPP}_{tmax} + 3 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{ncPP}_{tmax} + 5 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{nzPP}_{tmax}$
$\boldsymbol{ELL}$	$2 \times \mathrm{nrPP}_{tmax} + 6 \times \mathrm{nzPP}_{tmax}$	$4 \times \mathrm{nrPP}_{tmax} + 3 \times \mathrm{nzPP}_{tmax}$	$1 \times \mathrm{nrPP}_{tmax} + 4 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{nzPP}_{tmax}$
$\boldsymbol{COO}$	$6 \times \mathrm{nzPP}_{tmax}$	$4 \times \mathrm{nzPP}_{tmax}$	$6 \times \mathrm{nzPP}_{tmax}$	$2 \times \mathrm{nzPP}_{tmax}$

$$CSR_{coût} = (3 \times \mathrm{nrPP}_{tmax} + 6 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{L_{coût}} + (4 \times \mathrm{nrPP}_{tmax} + 3 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{E_{coût}} + (2 \times \mathrm{nrPP}_{tmax} + 4 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{AI_{coût}} + 2 \times \mathrm{nzPP}_{tmax} \times \boldsymbol{Op_{coût}} \quad (1)$$

$$CSC_{coût} = (3 \times \mathrm{ncPP}_{tmax} + 6 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{L_{coût}} + (3 \times \mathrm{ncPP}_{tmax} + 3 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{E_{coût}} + (2 \times \mathrm{ncPP}_{tmax} + 5 \times \mathrm{nzPP}_{tmax}) \times \boldsymbol{AI_{coût}} + 2 \times \mathrm{nzPP}_{tmax} \times \boldsymbol{Op_{coût}} \quad (2)$$

$$ELL_{coût} = (2 \times \text{nrPP}_{tmax} + 6 \times \text{nzPP}_{tmax}) \times \boldsymbol{L_{coût}} + (4 \times \text{nrPP}_{tmax} + 3 \times \text{nzPP}_{tmax}) \times \boldsymbol{E_{coût}} + (1 \times \text{nrPP}_{tmax} + 4 \times \text{nzPP}_{tmax}) \times \boldsymbol{AI_{coût}} + 2 \times \text{nzPP}_{tmax} \times \boldsymbol{Op_{coût}} \quad (3)$$

$$COO_{coût} = (6 \times \text{nzPP}_{tmax}) \times \boldsymbol{L_{coût}} + (4 \times \text{nzPP}_{tmax}) \times \boldsymbol{E_{coût}} + (6 \times \text{nzPP}_{tmax}) \times \boldsymbol{AI_{coût}} + 2 \times \text{nzPP}_{tmax} \times \boldsymbol{Op_{coût}} \quad (4)$$

Conclusion

Nous avons présenté dans ce chapitre une étude expérimentale afin de valider notre étude de cas présentée dans le chapitre précédent. Une première étude expérimentale illustrée sur des cas de tests pathologiques a montré que les métriques extraites seulement du modèle *data parallel* ne suffisent pas pour faire un choix. Nous avons ainsi introduit les métriques de l'analyse de l'algorithme dans notre étude expérimentale. A l'issue des résultats expérimentaux, nous avons pu définir un nouveau processus de décision basé sur un modèle de coût. Néanmoins, vu que ce modèle dépend de l'architecture et de l'environnement de l'exécution, il est difficile de le prédire. Nous pensons ainsi à la combinaison de l'expertise utilisateur (analyse des modèles de programmation et de l'algorithme) et du *machine learning* pour la mise en œuvre d'un système automatique de sélection du meilleur format de compression.

MACHINE LEARNING POUR LA SELECTION DU MEILLEUR FORMAT DE COMPRESSION

1. Introduction

Nous nous intéressons dans cette thèse au problème de la sélection automatique du meilleur format de compression d'une matrice creuse. Nous avons présenté dans le chapitre précédent une étude permettant d'avoir une idée sur les principaux paramètres impactant ce processus de choix. Nous avons montré que plusieurs facteurs peuvent intervenir dans le choix du meilleur format de compression (MFC) tels que la structure de la matrice, l'architecture, etc. D'autres facteurs liés à l'environnement de développement peuvent aussi avoir un impact sur le choix à faire. Ainsi, nous avons montré que le processus du choix nécessite un nombre trop important d'exécutions avec de nombreux paramètres pour être réalisé complétement par l'être humain. Nous nous sommes alors naturellement orientés vers les techniques de d'apprentissage automatique. Ainsi nous avons investigué des solutions basées sur des systèmes d'apprentissage automatique pour pouvoir sélectionner le MFC.

Nous commençons ce chapitre par une présentation générale du *machine learning* (système d'apprentissage). Nous passons en revue les principaux modèles d'apprentissage automatique et nous détaillons en particulier les classificateurs. En effet, le problème de sélection du meilleur format de compression peut être présenté sous la forme d'un problème de classification où on cherche à déterminer la classe (format de compression) à laquelle appartient une entrée (matrice creuse + architecture + modèle de programmation + noyau de calcul).

Nous détaillons dans la section 3 l'ensemble des étapes qu'on a suivi pour la mise en œuvre de notre système d'apprentissage. La dernière partie de ce chapitre est consacrée à la présentation des résultats numériques obtenus et de leurs interprétations.

2. Présentation du Machine Learning

La *Machine Learning* (ML) ou encore l'apprentissage automatique fait référence à la détection automatique de modèles dans les données. L'apprentissage automatique est également utilisé dans de nombreuses applications scientifiques telles que la bio-informatique, la médecine, l'astronomie, etc. [ShB14]. Il consiste à concevoir des algorithmes permettant à un ordinateur d'apprendre.

2.1. Modèles d'apprentissage automatique

2.1.1 Supervisé

L'apprentissage supervisé est le modèle d'apprentissage le plus simple à comprendre. L'algorithme d'apprentissage supervisé génère une fonction qui affecte les entrées aux sorties désirées [Ayo10]. En d'autres termes, l'objectif de l'apprentissage supervisé est de construire un modèle concis de la distribution des étiquettes de classe en termes de caractéristiques prédictives. Le classificateur résultant est ensuite utilisé pour attribuer des étiquettes de classe aux instances de test dans lesquelles les valeurs des entités de prédiction sont connues, mais la valeur de l'étiquette de classe est inconnue [Kot07]. Par exemple, si nous voulons apprendre à l'ordinateur comment faire la distinction entre les images de chats et de chiens, nous allons exécuter l'algorithme sur beaucoup de photos de chats et de chiens. Afin de superviser l'algorithme pour apprendre la bonne façon de classer les images, nous allons étiqueter les images comme des chats et des chiens. Une fois que notre algorithme apprend à classer les images, nous pouvons l'utiliser sur de nouvelles données et prédire des étiquettes (chat ou chien dans notre cas) sur des images non vues. Nous pouvons classer les problèmes de l'apprentissage supervisé en deux grandes classes : classification et régression.

- Classification : Les problèmes de classification catégorisent toutes les variables qui forment la sortie. Des exemples de ces catégories formées par classification incluraient des données démographiques telles que l'état matrimonial, le sexe ou l'âge. Le modèle le plus couramment utilisé pour ce type est la machine à vecteurs de support.
- Régression : Les problèmes qui peuvent être classés comme des problèmes de régression comprennent les types dans lesquels les variables de sortie sont définies comme un nombre réel. Le format de ce problème suit souvent un format linéaire.

2.1.2 Non supervisé

Au cours du processus d'apprentissage non supervisé, le système ne dispose pas d'ensembles de données concrets et les résultats de la plupart des problèmes sont en grande partie inconnus. L'objectif est que l'ordinateur apprenne à faire quelque chose sans que nous lui disons comment [Ayo10]. L'apprentissage non supervisé signifie que l'algorithme utilisé n'a pas d'étiquettes attachées pour superviser l'apprentissage. Nous fournissons simplement un algorithme avec une grande quantité de données et caractéristiques de chaque observation. Le système d'apprentissage non supervisé reconnaîtra alors tous les objets similaires et les regroupera. Les étiquettes qu'il donnera à ces objets seront conçues par la machine elle-même.

Revenons à l'exemple des chats et des chiens, dans ce cas on n'a pas d'étiquettes pour les images de chats et de chiens. L'algorithme lui-même ne peut pas décider ce qu'est un visage, mais il peut diviser les données en groupes. Un apprentissage non supervisé peut être utilisé dans ce cas pour séparer les images en deux groupes en fonction de certaines caractéristiques inhérentes aux images, telles que la couleur, la taille, la forme, etc.

2.2. Classification supervisée

Dans le cadre de ce travail, nous nous intéressons aux problèmes de classification supervisée. Nous présentons dans cette partie les principaux algorithmes utilisés dans ce cadre ainsi que l'approche utilisée pour résoudre un problème de classification.

2.2.1 Algorithmes

a) *KNN (K-Nearest Neighbors)*

Noté aussi KPPV (*K* Plus Proches Voisins). Il représente généralement le classificateur le plus simple. Dans notre base d'apprentissage les données sont stockées sous la forme (vecteur, étiquette) où vecteur représente les attributs de la donnée et étiquette représente la classe associée. Pour chaque nouvelle donnée, présentée sous forme de vecteur, on cherche la donnée la plus proche. Supposons qu'on a un ensemble de données sous les formes *cercle* et *rectangle*. Le but est de définir la classe à laquelle appartient un nouvel objet. *Figure* ***Error! No text of specified style in document.****-78* illustre un tel exemple de classification où le nouvel objet, dont la classe est à prédire, est représenté par une étoile. Dans cet exemple, si on a $k = 3$, alors la classe associée au nouveau point est *cercle*, par contre si $k = 5$ la classe associée est *rectangle*.

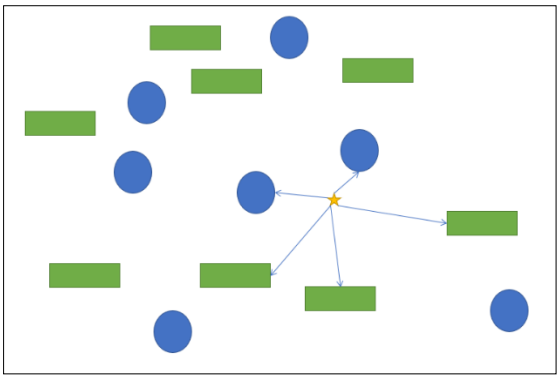

Figure ***Error! No text of specified style in document.****-78 Exemple de classification KNN*

b) Arbres de décisions (Decision Trees)

L'arbre de décision construit des modèles de classification ou de régression sous la forme d'une structure arborescente. Les arbres de décision classent les instances en les triant en fonction des valeurs de caractéristiques. Chaque nœud dans un arbre de décision représente une entité dans une instance à classer, et chaque branche représente une valeur que le nœud peut assumer. Les instances sont classées à partir du nœud racine et triées en fonction de leurs valeurs [Kot07]. Les arbres de décision peuvent gérer à la fois des données catégoriques et numériques.

Un exemple simple est présenté dans [TSK18], il permet de faire la classification d'une nouvelle espèce découverte. Le but est de voir si elle est mammifère ou non. L'approche consiste à poser un ensemble de questions sur les caractéristiques de l'espèce. A chaque fois, la réponse permet de descendre un niveau dans l'arbre jusqu'à arriver à un nœud feuille représentant la classe correspondante (*Figure **Error! No text of specified style in document.**-79*).

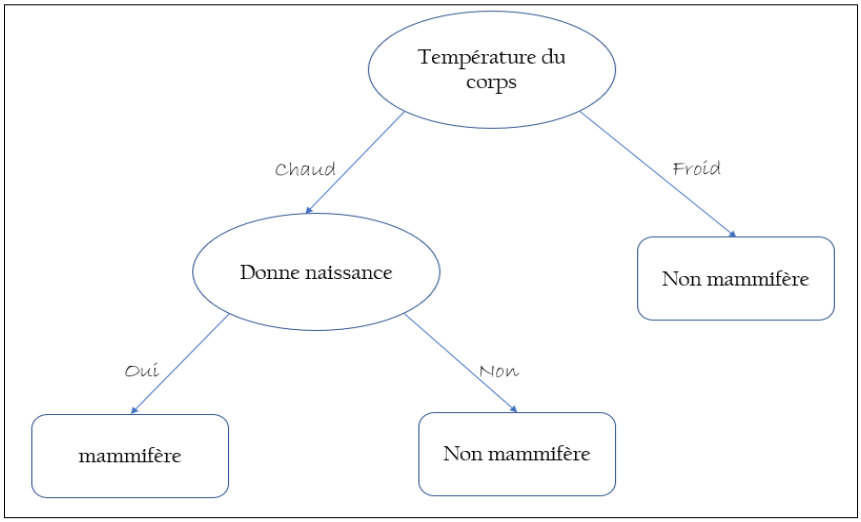

*Figure **Error! No text of specified style in document.**-79 Arbre de décision pour la classification des mammifères*

c) Réseaux de neurones (Neural Networks)

Un réseau de neurones artificiel est inspiré par le système nerveux biologique tels que le cerveau. Dans le cas de classification supervisée, la sortie d'un réseau de neurone artificiel est comparée à la sortie désirée. Au début, des poids aléatoires sont définis puis ajusté à chaque itération du processus d'apprentissage afin d'avoir la précision souhaitée (correspondance entre la sortie obtenue et celle désirée) [KsR12]. Un neurone est donc caractérisé par les signaux d'entrée (x_1, x_2, …, x_p) et une fonction d'activation *f :*

$$f(w_0 + \sum_{i=1}^{p} w_i x_i)$$

Parmi les réseaux de neurones les plus décrit dans la littérature, on reconnaît le Perceptron MuliCouches (PMC), en anglais Multi-Layer Perceptron (MLP). Le PMC est représenté par un ensemble de couches successives (*Figure* ***Error! No text of specified style in document.****-80*). Chaque couche contient un ensemble de neurones non connectés entre eux. Les couches entre la couche d'entrée et celle de sortie sont appelées des couches cachées. Chaque neurone de la couche cachée transforme les valeurs de la couche précédente en utilisant la fonction *f*. La couche de sortie reçoit les valeurs de la dernière couche cachée et les transforme en valeurs de sortie.

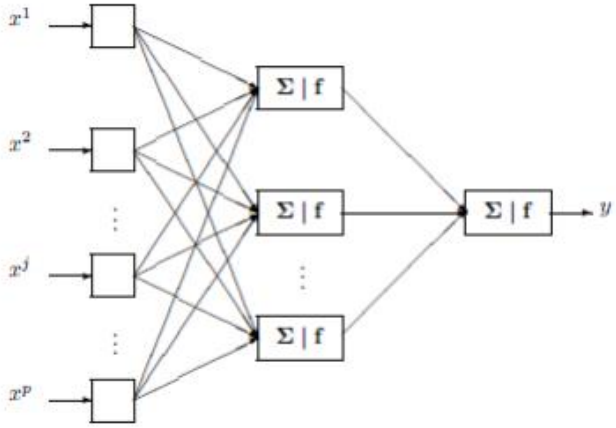

Figure ***Error! No text of specified style in document.****-80 Exemple de Perceptron MultiCouches élémentaire avec une couche cachée et une couche de sortie [Res]*

d) <u>Machine à support de vecteur</u>

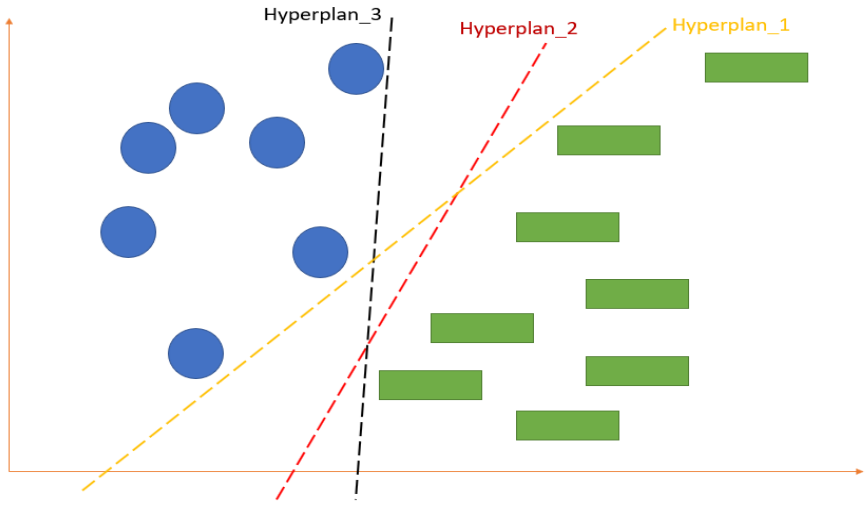

Figure ***Error! No text of specified style in document.****-81 Exemple d'hyperplan utilisés dans SVM*

Une machine à vecteurs de support, *Support Vector Machines (SVM) en anglais*, est une machine d'apprentissage supervisé qui peut être utilisée pour les problèmes de classification ou de régression.

Généralement, SVM est utilisée pour les problèmes de classification. Dans ce cas, on construit un hyperplan n-dimensionnel qui sépare de manière optimale les données en sous-ensembles. Dans un espace à deux dimensions, cet hyperplan est une ligne qui divise un plan en deux parties où chaque classe se trouve dans une partie. Dans l'exemple de la distinction entre des cercles et des rectangles, plusieurs lignes (hyperplan) peuvent être définie pour la séparation des deux classes (*Figure* ***Error! No text of specified style in document.****-81*). Le but de SVM est de trouver l'hyperplan qui divise le plan en deux sous-ensembles d'une manière optimale.

2.2.2 Approche générale pour la résolution d'un problème de classification.

L'approche générale pour la résolution d'un problème de classification consiste à mettre en œuvre un modèle de classification pour un ensemble de données en entrée (*Input data set*). Chaque technique de classification utilise un algorithme d'apprentissage afin d'identifier le meilleur modèle permettant de trouver une relation entre l'ensemble des attributs et l'ensemble des classes (les étiquettes) des données en entrée. Le modèle généré par l'algorithme d'apprentissage doit prédire correctement les classes des nouvelles données [TSK18].

e) Définition des données d'entrainement et des données de test

La première étape pour la mise en œuvre d'un modèle de classification est la définition des données d'entrainement et celles de test. Les données d'entrainement (*Training dataset*) représentent l'ensemble des entrées dont les classes auxquelles ils appartiennent sont connues. Les données de test ou d'évaluation (*Test dataset*) sont celles dont les classes sont inconnues (étiquettes à prédire).

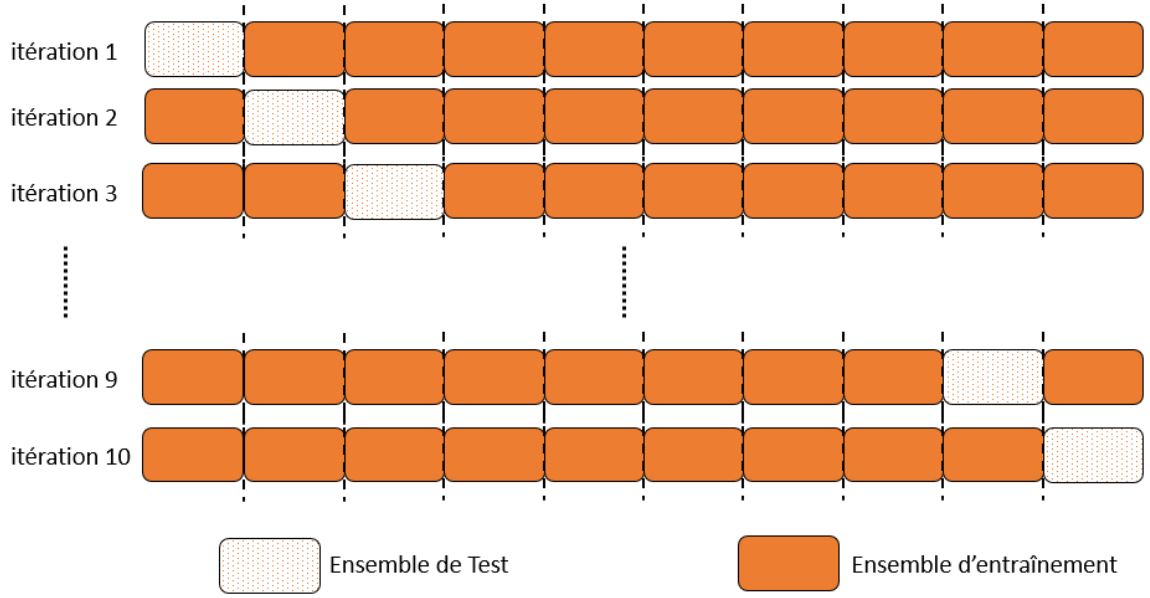

*Figure **Error! No text of specified style in document.**-82 Méthode de la validation croisée à k plis, un exemple pour k=10*

Généralement, un simple partitionnement aléatoire est utilisé pour choisir les données d'entrainement et celles de test ($70\% - 30\%$ ou $80\% - 20\%$ dans la plupart des cas).

Une autre stratégie souvent utilisée pour la division des données est la méthode de validation croisée à $k - plis$ ($k - fold\ cross\ validation$), où k est le nombre d'échantillons [TSK18]. Cette méthode consiste à diviser l'ensemble des données en k sous-ensembles, généralement on a $k = 5$ ou $k = 10$. Pour chaque itération, un sous-ensemble est sélectionné pour représenter le jeu de données d'entrainement pendant que le modèle est formé en utilisant les autres sous-ensembles restants (*Figure **Error! No text of specified style in document.**-82*). Ce processus est répété jusqu'à ce que la précision soit déterminée pour chaque instance de l'ensemble de données. Une moyenne des précisions est fournie à la fin.

f) Evaluation des performances

L'évaluation des performances d'un modèle de classification est basée sur le nombre des prédictions correctes et incorrectes qu'il fournit. Ces informations sont représentées sous la forme d'une matrice appelée *matrice de confusion* (*confusion matrix*). Dans ce qui suit cette matrice est notée *C*. Chaque élément (i, j) de *C* représente le nombre des enregistrements appartenant à une classe i et qui ont *j* comme classe prédite. La somme des éléments diagonaux de *C* représente le nombre des prédictions correctes.

$$nombre\ de\ prédictions\ correctes = \sum_{i=j} C_{ij}$$

$$nombre\ de\ prédictions\ incorrectes = \sum_{i \neq j} C_{ij}$$

A partir de la *matrice de confusion*, on peut directement déduire la *précision* de notre modèle de classification. La *précision* ($Accuracy$) représente la métrique de performance la plus utilisée pour l'évaluation des performances d'un modèle de classification. Cette métrique est définie par :

$$Accuracy = \frac{nombre\ de\ prédictions\ correctes}{nombre\ total\ des\ prédictions}$$

Généralement, on cherche à avoir des modèles de classification avec une *précision* maximale.

3. Machine Learning pour la sélection du meilleur format de compression

Le problème de la sélection du meilleur format de compression de matrice creuse peut être formulé comme un problème de classification où chaque format représente une classe. Pour cela, nous constituons d'abord notre ensemble de données contenant des matrices creuses représentatives représentées sous la forme d'un ensemble d'attributs. Dans un deuxième temps, nous utilisons un algorithme de classification pour construire un modèle d'apprentissage qui sera validé sur un ensemble de tests. Ce modèle sera utilisé pour prédire le meilleur format de compression de matrice creuse.

3.1. Présentation du processus d'apprentissage pour la sélection du meilleur format de compression

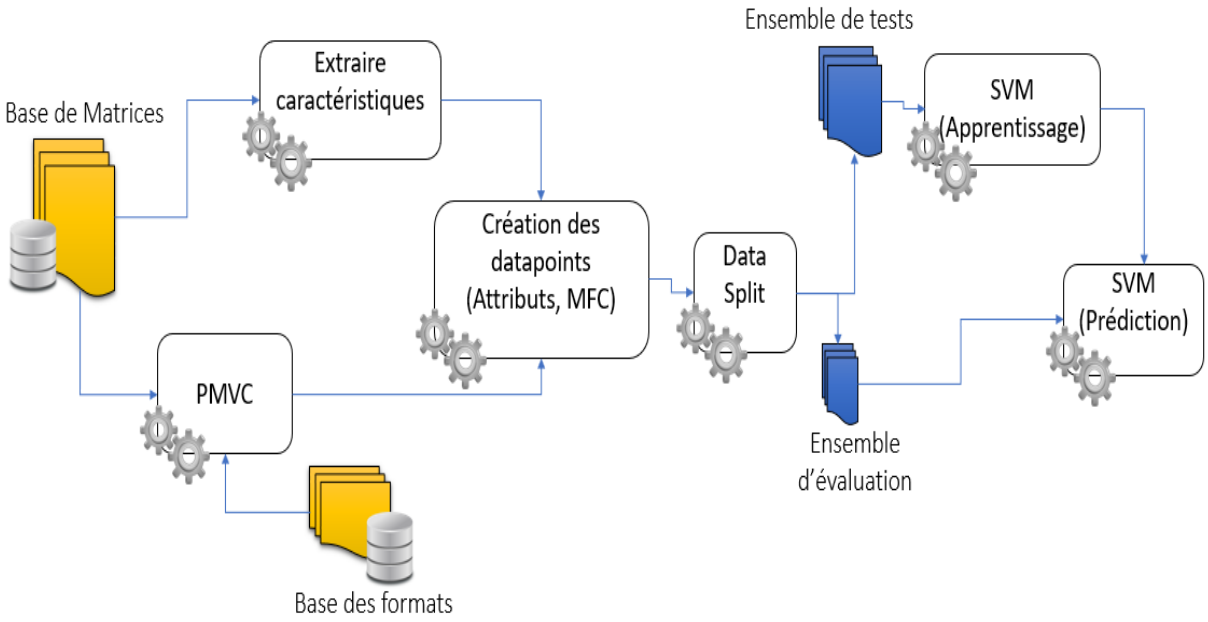

Figure ***Error! No text of specified style in document.****-83 Modèle de décision*

Soit A une matrice creuse $\in \mathbb{R}^{n \times n}$. Nous cherchons à déterminer le meilleur format de compression pour cette matrice pour un noyau de calcul, une architecture et un modèle de programmation données. Nous avons démontré dans le chapitre précédent que les métriques extraites à partir de l'application du modèle *data parallel* et de l'analyse de l'algorithme peuvent mener à un processus de décision. Nous proposons dans cette partie l'utilisation du *machine learning* pour la mise en œuvre d'un système pour le choix automatique du MFC.

Notre système sera alimenté par les métriques prédéfinies dans le précédent chapitre. La mise en œuvre d'un système utilisant le *machine learning* est composée de plusieurs étapes. La phase la plus importante d'un tel système est la phase d'apprentissage ou d'entrainement. Plus le système est mieux entraîné plus il peut prédire correctement. La phase d'apprentissage consiste à appliquer un algorithme d'apprentissage (classification dans notre cas) sur un ensemble de données. Nous présentons dans ce travail un processus d'apprentissage pour la sélection du MFC en utilisant l'algorithme de classification SVM (Figure Error! ***No text of specified style in document.****-83*).

Nous détaillons dans ce qui suit les étapes à suivre pour la définition de notre modèle d'apprentissage :

- Etape 1 : Etant donnée une base de matrices creuses représentatives, la première étape consiste à analyser l'ensemble de ces matrices et définir leurs caractéristiques. Pour cela, nous avons utilisé des fonctions de la bibliothèque Sparskit [Saa94] ainsi qu'un ensemble d'algorithmes personnalisés.
- Etape 2 : Etant donnés la base des matrices creuses de test, le noyau de calcul (PMVC) et la base des formats de compression à laquelle le MFC appartient, pour chaque format de la base, nous calculons le PMVC pour toutes les matrices de test. Le MFC est sélectionné pour chaque cas de test.
- Etape 3 : Etant donnés les caractéristiques des matrices et le MFC correspondant, on construit nos points de données qui seront présentés d'un ensemble de couples {caractéristiques matrice, MFC}.
- Etape 4 : Pour commencer la phase d'apprentissage, l'ensemble des points de données doit être divisé en deux groupes : un ensemble d'entrainement et un ensemble de test. Pour cela plusieurs méthodes existent (par exemple méthode naïve 80%-20%, méthode de cross validation [TSK18]).
- Etape 5 : C'est la phase d'apprentissage, elle consiste à appliquer un algorithme de classification, SVM dans notre cas, sur l'ensemble des données de tests. Ceci permet d'établir une relation entre les attributs et la classe (MFC) ce qui rend le système capable de prédire dans l'étape suivante.
- Etape 6 : Etape de prédiction : Dans cette étape, on suppose qu'on n'a pas d'informations sur la bonne classe (MFC) associée aux données de la base d'évaluation. Le système utilise donc ce qu'il a appris à l'étape d'apprentissage pour pouvoir prédire le MFC. Nous serons ainsi capables de comparer la valeur prédite par rapport à la bonne valeur et par la suite évaluer les performances de notre système.

3.2. Définition des paramètres de notre système d'apprentissage

Nous présentons dans cette partie l'ensemble des paramètres qu'on a défini pour la mise en œuvre de notre système. Nous commençons par la présentation de l'ensemble des matrices qui vont constituer notre base de données. Nous allons par la suite extraire un certain nombre de caractéristiques représentant les attributs. L'étape suivante consiste à répartir les données en un ensemble d'entrainement et un autre de test. Pour chaque entrée, le MFC doit être déterminé suite aux expérimentations.

3.2.1 Définition de l'ensemble des données

Nous présentons dans cette partie notre base de matrices (*Figure* ***Error! No text of specified style in document.****-84* et *Figure* ***Error! No text of specified style in document.****-85*).

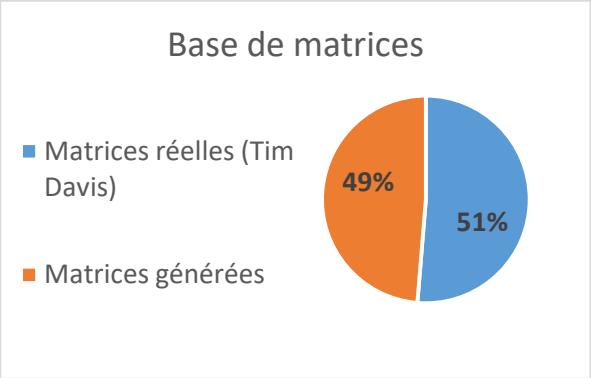

Figure ***Error! No text of specified style in document.****-84 Types des matrices utilisées dans les tests*

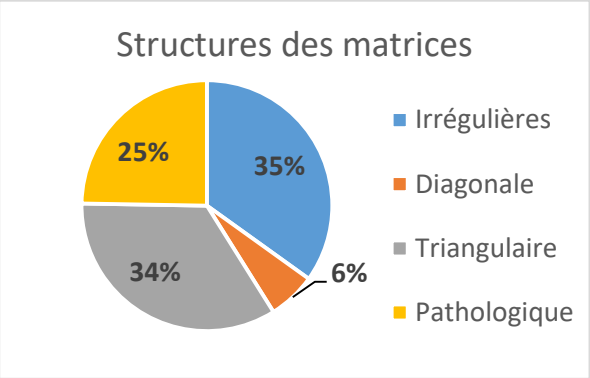

Figure ***Error! No text of specified style in document.****-85 Structures des matrices utilisées dans les tests*

Tableau 15 statistique sur les matrices

n	nnz	$densité$
$[362 : 2.99E+06]$	$[1000 : 2.1E+09]$	$[0.0001 : 80]\%$

Pour avoir une base assez variée et avoir le plus possible de cas de tests, nous avons utilisé trois types de matrices :

- Matrices réelles : Un ensemble de matrices sélectionnées à partir de la collection des matrices réelles de l'université de Floride (matrices avec structures régulières et irrégulières) [DaY11]. Ces matrices proviennent de différents domaines d'application tels que la dynamique des fluides, l'électromagnétique où la simulation de circuit.
- Des matrices creuses avec structures régulières qu'on a générées : matrices diagonales et triangulaires,
- Des matrices pathologiques qui correspondent aux cas extrêmes détectés lors de l'étude du modèle *data parallel* pour un format donné (voir chapitre 3).

Le Tableau 15 représente des statistiques sur les matrices utilisées dans notre étude.

3.2.2 Définition de l'ensemble des attributs

En utilisant une approche d'apprentissage automatique, deux étapes sont requises : entraînement et évaluation. Pour cela, un ensemble d'attributs (caractéristiques) doit être définit. Ces attributs permettent de définir la correspondance d'une matrice creuse à un format de compression.

Tableau 16 Ensemble des attributs utilisés dans notre classificateur

Attribut	Signification	CM	CA	MDP
n	Nombre de lignes	✓	✓	✓
nnz	Nombre des éléments non nuls	✓	✓	✓
d	Densité de la matrice	✓		
$diag$	Indique si la matrice est diagonale	✓		
$triang$	Indique si la matrice est triangulaire	✓		
$\mathbf{nz_{maxR}}$	Nombre maximal d'éléments non nuls par ligne			✓
$\mathbf{nz_{minR}}$	Nombre minimal d'éléments non nuls par ligne			✓
$\mathbf{nz_{maxC}}$	Nombre maximal d'éléments non nuls par colonne			✓
$\mathbf{nz_{minC}}$	Nombre minimal d'éléments non nuls par colonne			✓
$Op_{coût}$	Coût d'une Opération *data parallel*			✓
PP	Nombre de processeurs physiques			✓
Pv	Nombre de processeurs virtuels			✓

Dans la définition de ses attributs nous nous sommes basés sur les caractéristiques des matrices (CM), les caractéristiques de l'algorithme (CA) et métriques *data parallel* (MDP). Le choix de ses attributs est basé sur notre étude présentée dans le chapitre précédent.

- CM : représente un ensemble de caractéristiques de la matrice creuse telles que la tailles (n), le nombre des éléments non nuls (nnz), etc. Pour extraire ces caractéristiques nous avons utilisé la bibliothèque Sparskit [Saa94] ainsi qu'un ensemble de nos algorithmes.
- CA : représente un ensemble d'attributs qu'on a pu extraire en analysant les algorithmes de calcul du PMVC associés aux différents formats de compression. En effet, nous avons prouvé dans notre étude expérimentale (chapitre 3) qu'un certain nombre de caractéristiques de l'algorithme peuvent affecter les performances du noyau de calcul. Nous citons principalement le nombre des accès InDirects (ID), le nombre de lecture mémoire (R, pour Read) et le nombre d'écriture mémoire (W, pour Write). Pour cela, ceux-ci sont inclus dans notre étude.
- MDP : représente l'ensemble des métriques qu'on a pu définir en appliquant le modèle *data parallel* aux noyaux de calcul PMVC associés aux différents formats de compression (CSR, CSC, ELL, COOR et COOC).

Afin d'avoir les meilleures performances de notre système d'apprentissage, on a intérêt à définir un ensemble d'attributs aussi simple que possibles. Le Tableau 16 représente l'ensemble des attributs qu'on a maintenu pour la définition de notre système.

3.2.3 Data points

Un point de donnée ($data\ point$ ou $datapoint$) représente une entrée de notre système d'apprentissage. Chaque point de données peut être défini comme un couple {attribut, MFC}. Pour déterminer la classe (le meilleur format) de chaque point de donnée, nous exécutons le noyau PMVC sous chacun des quatre formats de compression pour toutes les matrices creuses en entrée. La valeur de GFLOP/s est enregistrée pour chaque format. Ainsi, chaque matrice est affectée à sa classe selon la plus haute valeur de GFLOP/s. Dans ce travail, nous avons utilisé un nombre total de 600 points de données.

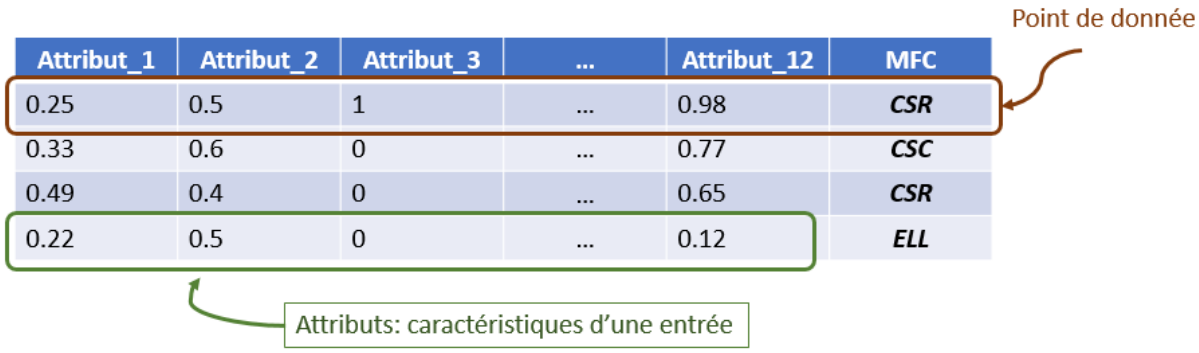

*Figure **Error! No text of specified style in document.**-86 Structure d'une base d'apprentissage (points de données)*

3.2.4 Répartition des données

Pour pouvoir évaluer les performances de notre système, l'ensemble des points de données doit être divisé en deux sous-ensembles : un sous ensemble pour l'étape d'entrainement et un autre pour l'évaluation. Pour cela, nous choisissons d'utiliser la méthode de validation croisée ($k-fold$) puisqu'elle est la méthode la plus utilisée à nos jours. Nous choisissons de travailler avec un nombre d'échantillons $k = 10$. Ainsi, tous l'ensemble des points de données sera divisé en 10 sous-ensembles et nous aurons 10 itérations d'apprentissage. A chaque itération, un sous-ensemble est sélectionné pour représenter l'ensemble d'évaluation et le reste (9 sous-ensembles) est utilisé pour l'entraînement. De cette manière, notre système sera entrainé sur tout l'ensemble des points de données. La précision ($accuracy$) est déterminée pour chaque

itération. A la fin des itérations, la précision du système est la moyenne des précisions obtenues à chaque itération.

3.2.5 Algorithme de classification

Le problème de sélection du meilleur format de compression est un problème de classification où, étant donnée un ensemble en entrée (matrice, plateforme, etc.), nous cherchons à déterminer le format associé (classe). Pour le traitement de notre problème, nous avons choisi d'utiliser l'algorithme de classification SVM (Support Vector Machine). En particulier, nous avons utilisé l'implémentation LibSVMs [HCL16].

g) Data scaling

Une étape importante avant de commencer le processus d'apprentissage est l'étape de la mise à l'échelle ($data\ scaling$). Le principal avantage de la mise à l'échelle est d'éviter que les attributs ayant des valeurs numériques grandes dominent ceux dont les valeurs sont plus petites [HCL16]. Comme recommandé dans [HCL16], nous choisissons d'adapter linéairement chaque donnée à la plage [0,1].

h) La fonction noyau : le noyau RBF

Dans le cas des problèmes de classification, deux modèles des SVM peuvent être rencontrés :

- Cas linéairement séparable : c'est le cas le plus facile à traiter car les données sont linéairement séparables,
- Cas non linéairement séparable : c'est le cas correspondant à la majorité des problèmes réels. Dans ce cas les données dans l'espace d'origine sont non linéairement séparables. Pour cela une transformation de l'espace d'origine vers un autre où les données sont linéairement séparables est nécessaire. Plusieurs fonctions noyaux (notées k) ont été proposés pour faire cette transformation, les plus connues sont de type linéaire, polynomiale, fonction gaussienne radiale et sigmoïde.
  - Linéaire

$$k(x, x') = x.x'$$

  - Polynomial

$$k(x, x') = (coef + x.x')^{degré}$$

  - Fonction gaussienne radiale (*RBF, pour Radial Basis Function*)

$$k(x, x') = exp^{\frac{-||x-x'||^2}{\gamma}} \quad ; \text{avec } \gamma > 0$$

- Sigmoïde

$$k(x, x') = \tanh(\gamma \times x.x' + coef)$$

$degré$, $coef$ et γ sont des paramètres des noyaux.

x, x' deux points d'observation.

La fonction RBF est la fonction noyau la plus utilisée dans la machine SVM [HCL16]. Ceci est dû à ses réponses localisées et finies sur toute l'étendue de l'axe réel x. Ainsi, nous choisissons d'utiliser RBF pour notre problème. Dans cette fonction noyau, deux paramètres sont à déterminer : C et γ, avec $C > 0$ représente le paramètre de pénalité du terme d'erreur.

i) Les paramètre C et gamma

Comme recommandé dans [HCL16], nous utilisons une recherche « $gridsearch$ » sur C et gamma (γ) en utilisant une validation croisée $10 - fold$. Nous essayons différentes paires de valeurs (C, γ) et nous choisissons celle avec les meilleures performances ($C = 100$ et $\gamma = 0.8$).

3.3. Etude expérimentale

3.3.1 Méthodologie et plateforme de test

Pour préparer l'ensemble des données d'apprentissage, nous exécutons le PMVC pour chaque matrice en entrée compressée sous chaque format de notre base (CSR, CSC, ELL, COOC, et COOR).

Tableau 17 Plateforme de test

CPU:	Intel Xeon E5-2630 v4 (2 CPUs/nœud, 10 cœurs/CPU)
Fréquence	2.20 GHZ
Mémoire:	256 GB

Nous avons effectué nos tests sur des machines appartenant au site Grid'5000 qui est une plate-forme expérimentale pour la recherche sur les systèmes distribués. Nos tests ont été effectués sur le site de Lille à travers son cluster 'chetemi'. Cette grappe utilise le processeur Xeon E5-2630 v4 d'Intel (*Tableau 17*). Pour chaque entrée (voir la section 3.2.1), nous exécutons le PMVC 10 fois sur 100, 140, 160 puis 200 processeurs physiques. Pour chaque entrée, la moyenne des temps des exécutions est maintenue.

Nous avons utilisé l'algorithme NEZGT [MeH12][MeH13-a][MeH13-b][MHD17][MHD18-a][MHD18-b] pour l'ordonnancement des données : pour chaque processeur virtuel on associe un poids w_i représentant le nombre des éléments non nuls qu'il traite. L'algorithme NEZGT nous permet d'affecter un ensemble de processeurs virtuels à un processeur physique tels que la somme des poids de chaque processeur physique sont plus au moins égaux.

Nous avons utilisé le langage R pour la réalisation de notre système de sélection du MFC, et ce à travers le logiciel de développement R Studio [Tea15]. Pour l'application de l'algorithme SVM, nous avons utilisé la bibliothèque LibSVMs via son implémentation dans le package e1071 du logiciel R Studio [Tea15] [Mey17]. Le package e1071 propose la fonction $tune.svm$ pour la recherche automatique des meilleurs paramètres que nous avons utilisés pour trouver les valeurs optimales de C et γ.

3.3.2 Résultats numériques et interprétations

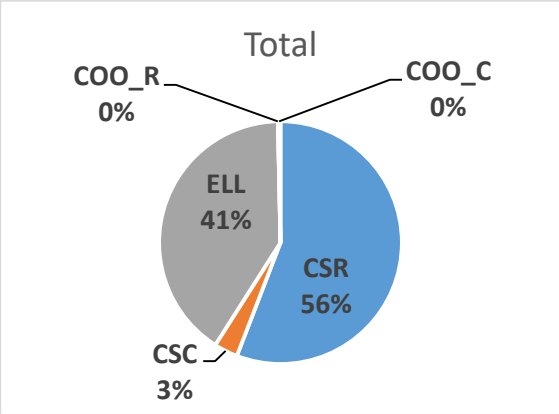

Figure ***Error! No text of specified style in document.****-87 Répartitions des MFC des matrices testées*

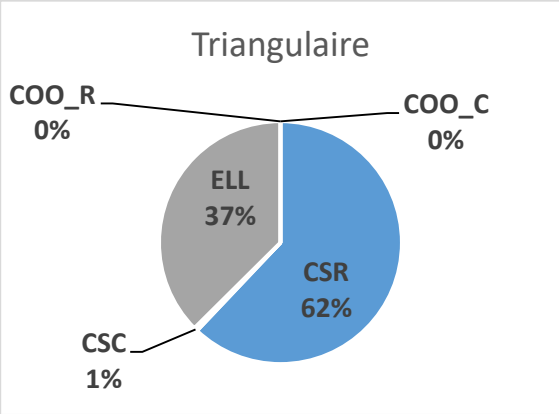

Figure ***Error! No text of specified style in document.****-88 Répartition des MFC dans le cas des matrices triangulaires*

Figure ***Error! No text of specified style in document.****-87*, *Figure* ***Error! No text of specified style in document.****-88*, Figure Error! ***No text of specified style in document.****-89*, Figure Error! ***No text of specified style in document.****-90*, Figure Error! ***No text of specified style in document.****-91*, et Figure Error! ***No text of specified style in document.****-92* représentent quelques statistiques sur la sélection de MFC en fonction des structures des matrices et des domaines d'application. Pour chaque format est illustré le nombre (pourcentage) des points de données pour lesquels il est sélectionné comme le MFC. Nous remarquons qu'une simple

caractéristique, telle que la structure de la matrice par exemple, n'est pas d'une grande aide pour la sélection du meilleur format de compression.

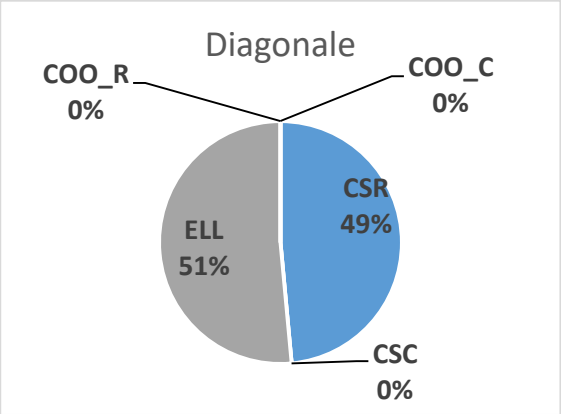

Figure ***Error! No text of specified style in document.****-89 Répartition des MFC dans le cas des matrices diagonales*

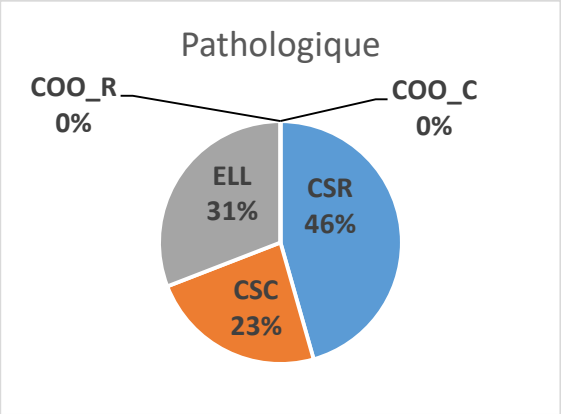

Figure ***Error! No text of specified style in document.****-90 Répartition des MFC dans le cas des matrices pathologiques*

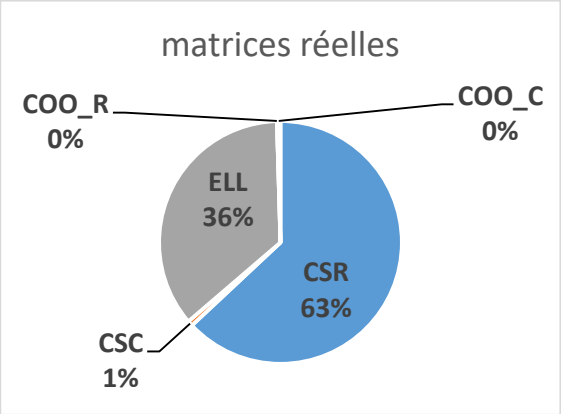

Figure ***Error! No text of specified style in document.****-91Répartition des MFC dans le cas des matrices provenant de problèmes réels*

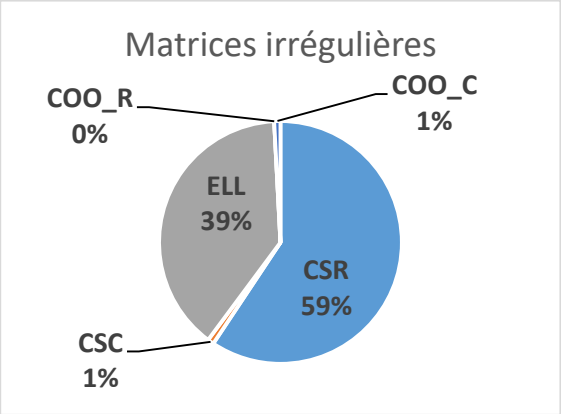

Figure ***Error! No text of specified style in document.****-92 Répartition des MFC dans le cas des matrices avec structures irrégulières*

En outre, Le *Tableau 18* montre la Perte Moyenne des Performances (PMP) dans le cas où on décide d'utiliser un format fixe (CSR, CSC, COO ou ELL) pour tous l'ensemble des données de test, au lieu d'utiliser le MFC correspondant. Nous notons que dans ce cas nous pouvons perdre jusqu'à 84% de performances si le mauvais format est utilisé (au lieu du MFC).

Ainsi, nous prouvons conclure, à partir de ces premiers résultats, que la sélection du MFC est une étape très importante dans le calcul creux permettant d'obtenir les meilleures performances

des noyaux de calcul. De plus, ce processus doit être automatisé vu la complexité des critères de sélection.

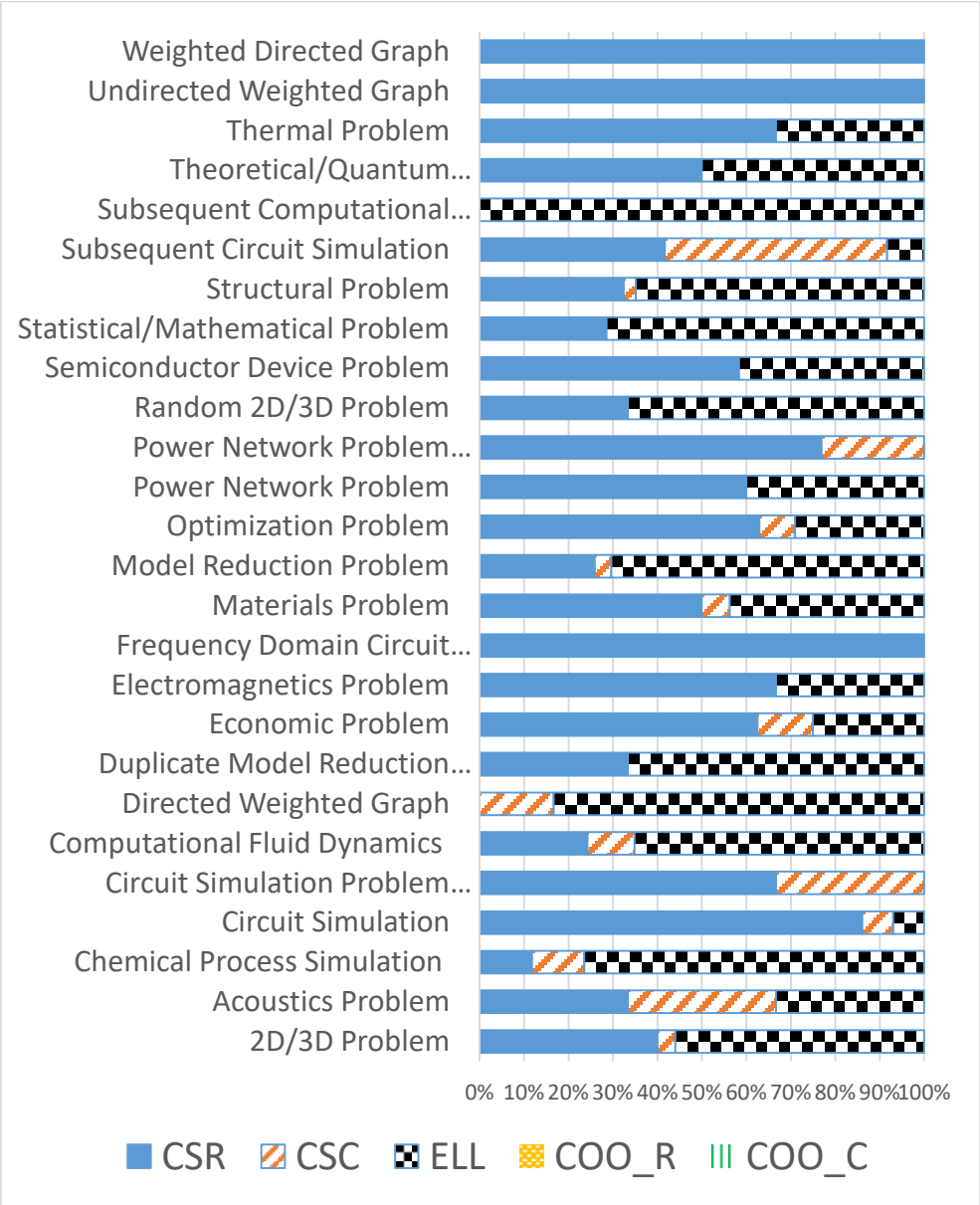

Figure ***Error! No text of specified style in document.****-93 MFC des matrices réelles en fonction des domaine d'application*

Tableau 18 Perte moyenne des performances dans le cas d'utilisation d'un format fixe

Format	CSR	CSC	ELL	COOR	COOC
PMP	27.1%	58.19%	38%	84.96%	82.82%

En analysant les résultats expérimentaux, nous remarquons que les performances de calcul sont faibles lorsque la matrice en entrée est compressée selon le format COO, avec ses deux versions

COOR et COOC. Le format COO ne donne les meilleures performances que dans 3 cas de l'ensemble des tests. En d'autres termes, le format COO n'est sélectionné comme MFC que dans 3 cas testés (sur les 600 cas de tests). C'est pour cette raison, nous avons décidé d'éliminer le format COO du processus de la prédiction, et ce afin d'augmenter les performances de notre classificateur. Les points de données pour lesquels le format COO est sélectionné comme MFC sont affectés à la prochaine classe (CSR, CSC ou ELL) avec les meilleures performances. Ainsi, nous obtenons un classificateur à trois classes CSR, CSC et ELL.

Pour analyser les performances de notre classificateur, trois métriques d'évaluation sont utilisées : la précision, LUBS et GOWS. Le *Tableau 19* représente les notations utilisées dans cette section.

Tableau 19 Notations utilisées dans les équations

Notation	**Signification**
$\boldsymbol{tp}$	Nombre des points de données utilisé dans les tests
$\boldsymbol{PPF_i}$	Performances du format prédit dans le point de données de test i
$\boldsymbol{POF_i}$	Performances du format optimal dans le point de données de test i
$\boldsymbol{PWF_i}$	Performances du pire format dans le point de données de test i
$\boldsymbol{LUBS}$	Perte par rapport à la sélection du format optimal (Loss Under Best Selection)
$\boldsymbol{GOWS}$	Gain par rapport à la sélection du pire format (Gain Over Worst Selection)

j) *La précision (Accuracy)*

Pour évaluer la performance d'un classificateur, nous mesurons toujours sa précision qui représente le nombre de prédictions correctes de toutes les prédictions faites. Nous divisons simplement le nombre de points de données de test correctement classifiés par le nombre total de points de données de test. Plus cette valeur est proche de 100%, plus notre classificateur est performant.

$$Accuracy = \frac{nombre\ de\ prédictions\ correctes}{nombre\ total\ des\ prédictions}$$

La précision obtenue pour notre classificateur est de l'ordre de 95.65% (Tableau 20). Nous rappelons que nous avons commencé par une étude théorique afin de pouvoir choisir la liste des attributs à utiliser dans notre classificateur. Cette étude consiste à utiliser une

géométrie virtuelle sur laquelle les données sont projetées. Par la suite un ensemble de métriques sont extraites afin d'évaluer les performances des formats de compression. Ces métriques représentent une grande partie des attributs utilisés dans notre classificateur. Le reste des attributs est sélectionné grâce à une analyse des algorithmes. Ainsi, cette étape de sélection de métriques (attributs) a permis d'avoir de meilleures performances de notre classificateur.

Pour prouver encore l'efficacité et l'importance de notre stratégie de sélection des attributs, nous évaluons les performances (la précision) de notre classificateur en utilisant d'autres ensemble d'attributs : Pour prouver l'importance de notre stratégie (utiliser notre étude pour sélectionner des fonctionnalités), nous mesurons la précision en utilisant d'autres fonctionnalités de collecte :

- Dans le cas où on élimine l'un des attributs sélectionnés à partir de notre étude théorique, la précision de notre système décroit jusqu'à 79.26%.
- Dans le cas où on ajoute un ou plusieurs autres attributs, telles que autres caractéristiques de la matrice, la précision du classificateur décroit aussi. Dans ce cas la précision maximale qu'on a pu obtenir est de l'ordre de 90%.

k) LUBS

Comme dans [SEF15] et [BJW16], notre objectif n'est pas seulement d'avoir un classificateur de haute précision ($accuracy$ proche de 100%), mais aussi de minimiser le taux de perte par rapport à la meilleure sélection. Ainsi nous mesurons le $LUBS$ ($Loss\ Under\ Best\ Selection$) de notre classificateur. Cette métrique permet de mesurer de combien notre classificateur est loin d'un classificateur parfait (dont la précision est de 100%). Cette perte de performances provient des mauvaises classifications (classifications erronées). Plus la valeur de LUBS est proche de 0%, meilleur est le classificateur.

$$LUBS = \frac{1}{tp} \sum_{i=1}^{i=tp} \frac{POF_i - PPF_i}{POF_i}$$

Dans le cas de notre classificateur, la perte moyenne de performances résultant d'une mauvaise classification est de l'ordre 1.96% (Tableau 20). Ainsi, nous concluons que même si un point de données est mal classé, il est affecté à une autre classe avec des performances très proches de celles obtenues dans le cas d'utilisation du MFC.

l) GOWS

De même que pour la métrique LUBS, les auteurs dans [SEF15] et [BJW16] proposent une troisième métrique pour l'évaluation des performances d'un classificateur qui est la métrique GOWS. La métrique $GOWS$, pour $Gain\ Over\ Worst\ Selection$, permet de mesurer le gain en performance lorsque notre classificateur est utilisé pour sélectionner le format de compression au lieu d'utiliser celui avec les performances les plus faibles.

$$GOWS = \frac{1}{tp} \sum_{i=1}^{i=tp} \frac{PPF_i - PWF_i}{PWF_i}$$

Nous remarquons qu'en utilisant notre classificateur pour le choix du format de compression, le taux moyen de gain en performance des points de données correctement classés est de l'ordre de 14% (Tableau 20). Ainsi, en utilisant notre classificateur pour le choix du MFC, notre noyau de calcul s'exécute 14% plus vite que lorsqu'on utilise le format de compression le moins adéquat.

Tableau 20 Résultats numériques de l'évaluation des performances de notre système d'apprentissage

Accuracy	***LUBS***	***GOWS***
95.65%	1.96%	14%

4. Conclusion

Le but final de notre travail est la mise en œuvre d'un système permettant la sélection automatique du meilleur format de compression pour une matrice creuse, une architecture, un noyau de calcul et un modèle de programmation donnés. Dans ce chapitre, nous avons utilisé des outils du *machine learning* afin de prédire le MFC (entre CSR, CSC, ELL et COO) d'une matrice creuse dans le cas de calcul du PMVC sur une architecture parallèle multiprocesseurs. Pour cela, nous avons utilisé des métriques extraites de l'application du modèle *data parallel* et de l'analyse des algorithmes pour définir les attributs utilisés dans le système d'apprentissage. Nous utilisons l'algorithme SVM pour l'étape d'apprentissage et d'évaluation du système.

Une suite d'expérimentions a été réalisée sur une plateforme parallèle sur des matrices de structures différentes afin de déterminer le MFC pour chaque format. Nous avons alors construit notre base de points de données pour l'utiliser dans l'apprentissage de notre système. Les

résultats expérimentaux obtenus ont montré que le format COO n'est sélectionné comme MFC que dans trois cas, c'est pourquoi nous l'avons éliminé de notre processus de décision.
L'évaluation des performances de notre classificateur montre que notre système peut atteindre une valeur de précision de l'ordre de 95.65%. Nous confirmons ainsi que la connaissance du modèle de programmation et de l'algorithme permet de bien définir les métriques impactant le choix du meilleur format de compression. Notre contribution dans ce travail peut se résumer dans les deux points suivants :

- Utilisation du modèle *data parallel* et de l'analyse de l'algorithme pour définir l'ensemble des attributs à utiliser dans le processus d'apprentissage.
- Prédiction du meilleur format de compression dans le cas d'une plateforme multiprocesseurs tandis que les travaux existants sont réalisés sur un GPU avec des cœurs partageant la même mémoire [SMP15] [BJW16] [BJW16-a] [LBE16].

CONCLUSION GENERALE ET PERSPECTIVES

Un système de sélection automatique du meilleur format de compression (MFC) a été proposé dans cette thèse. Etant donnés une matrice creuse, une méthode numérique, une architecture et un modèle de programmation parallèle, ce système doit prédire automatiquement le format de compression le plus adéquat.

Dans ce cadre nous avons commencé par un état de l'art sur les calculs creux. Nous avons détaillé les formats de compression les plus connus dans la littérature. Ces formats sont regroupés selon deux ensembles, à savoir des formats de base et des formats dérivés qui sont l'optimisation de ces derniers ou encore la composition d'un ou plusieurs formats. Nous nous sommes penchés ensuite sur la présentation de l'algorithme du produit matrice-vecteur creux (PMVC) qui constitue le noyau pour plusieurs problèmes d'algèbre linéaire creux. Ceci a été suivi par la présentation du schéma de Horner utilisant PMVC comme noyau de base.
Le traitement de matrices creuses de grandes tailles nécessite aussi l'utilisation des architectures haute performance. Ceci nous a amené à donner un aperçu sur les différentes architectures parallèles et distribuées. Nous avons détaillé en particulier les architectures des grappes qui forment aujourd'hui une bonne alternative pour des applications scientifiques gourmandes en performances de calcul.

Notre objectif étant d'étudier la sélection automatique du meilleur format de compression creux, nous avons alors présenté un état de l'art sur les travaux portant sur le même sujet. Nous avons constaté que les travaux existants cherchent à optimiser un critère de recherche alors qu'un autre n'est pas pris en considération. Les systèmes proposés dans la littérature prennent en compte l'architecture utilisée et les caractéristiques de la matrice en entrée. Nous proposons d'ajouter l'expertise utilisateur pour pouvoir ajouter d'autres facteurs pouvant affecter les performances du noyau de calcul tel que le modèle de programmation. Ainsi, nous proposons un système intelligent permettant le choix automatique du MFC, et ce étant donnée comme entrée une matrice creuse, une méthode numérique, une architecture et un

modèle de programmation parallèle. Une conception détaillée de notre système ainsi que ses principaux composants est présenté.

Une étude de cas est présentée pour la validation de notre système. Cette étude est réalisée dans un cadre limité correspondant au modèle de programmation *data parallel* et aux deux méthodes numériques : calcul du schéma de Horner et calcul du PMVC. Nous avons commencé par une étude permettant de définir les principales métriques affectant le choix du MFC. Pour cela, une analyse du modèle date parallèle a été effectuée. L'idée est de définir une géométrie virtuelle composée d'un ensemble de processeurs virtuels. Les éléments non nuls de la matrice sont affectés à cette géométrie est un ensemble de métriques sont extraites. Cette analyse dépend de la méthode numérique et des formats de compression utilisés. Notre étude de cas concerne le schéma de Horner et les formats de compression CSR, CSC, ELL et COO. Deux démarches différentes sont présentées. Selon la première démarche, à chaque processeur virtuel est affecté un élément non nul de la matrice. L'analyse de cette première approche ne permet pas de donner une décision pour faire le choix entre les formats étudiés. La deuxième approche consiste à attribuer à chaque processeur virtuel une ligne ou une colonne de la matrice (selon le format de compression) au lieu d'un seul élément. Ainsi, nous traitons des vecteurs au lieu des scalaires. Dans un deuxième lieu, nous avons effectué une analyse de l'algorithme. Cette analyse nous a permis d'extraire un ensemble de métriques liées aux accès à la mémoire. L'analyse du modèle *data parallel* nous a mené à la définition d'un premier processus de décision où nous avons classé les formats en deux groupes : un premier groupe où les éléments de la matrice sont stockés dans un ordre de lignes (CSR, ELL, COOR) et un deuxième où elles sont stockées dans un ordre de colonnes (CSC, COOC). Pour cela une première validation expérimentale a été limité aux deux formats représentatifs CSC et CSR.

Pour l'évaluation de notre, nous avons effectué un ensemble de tests sur une grappe de machines appartenant à la plateforme GRID5000. Nous avons présenté en premier lieu le cas de calcul du schéma de Horner. L'ensemble des tests a été effectué sur des matrices creuses pathologiques. Ces matrices correspondent aux cas extrêmes détectés lors de l'étude du modèle *data parallel* pour un format donné. Les premiers résultats approuvent notre étude. Néanmoins, les tests effectués sur des matrices provenant de problèmes réels contredit ces dernières. Plusieurs facteurs peuvent être l'origine de ce résultat tels que les mouvements des données lors de calcul de Horner et la méthode utilisée pour l'ordonnancement des processeurs virtuels.

Pour remédier un ce problème, une deuxième étude de cas est présentée. Dans ce cas nous avons choisi de tester l'algorithme de calcul du Produit Matrice Vecteur Creux afin d'éviter les problèmes de mouvement des données. Trois démarches d'ordonnancement des processeurs virtuels ont été utilisées. Dans un premier lieu, nous avons essayé d'avoir une plateforme physique équivalent à notre géométrie virtuelle, ainsi chaque processeur physique traite une seule chaque ligne/colonne de la matrice. Dans un deuxième lieu, nous avons opté pour l'utilisation de l'ordonnancement de MPI puis en utilisant un algorithme de fragmentation NEZGT. Ce dernier vise à décomposer une matrice donnée en un ensemble de fragments de lignes ou de colonnes contiguës avec nombre équilibré des éléments non nuls.
L'étude des résultats expérimentaux a montré que l'analyse du modèle *data parallel* seule ne permet pas de sélectionner le MFC. Nous avons démontré que les accès en mémoire et plus particulièrement les accès indirects peuvent affecter les performances de calcul. A la fin de cette étude de cas, un nouveau processus de décision basé sur un critère de coût est proposé.

En effet, la métrique de coût présentée dépend de l'architecture utilisée ainsi que de l'environnement de l'exécution. Ces coûts ne peuvent pas alors être déterminés à l'avance. Toutefois, ils nous permettent de définir les différents paramètres affectants le choix du MFC.
Vu le nombre de paramètres à déterminer, le processus de choix devient compliqué est difficile à mettre en œuvre par l'être humain. C'est pourquoi nous nous sommes orientés vers l'utilisation des algorithmes du Machine Learning. Notre but est de coupler l'expertise utilisateurs (métriques extraites) et les algorithmes d'apprentissage automatique. Nous avons commencé par la présentation des algorithmes du *machine learning* et leurs types. Nous remarquons que le problème de choix du meilleur format de compression correspond à un problème de classification où la classe à déterminée est le format de compression. Pour cela nous avons préparé une base d'apprentissage formée d'un ensemble de points des données. Ces ensembles correspondent au calcul du PMVC associé au différent format de compression étudiés, à savoir CSR, CSC, ELL, COOR et COOC. Ces tests ont été effectué sur des matrices réels, des matrices pathologiques et des matrices de structures particulières (triangulaire ou diagonale). L'analyse des modèles *data parallel* et l'analyse des algorithmes nous ont servi pour la définition des attributs à utiliser. Une phase d'apprentissage a été effectuée en utilisant l'algorithme SVM (Machines à Support de Vecteurs). Nous avons opté pour l'utilisation de la méthode de validation croisée $k - fold$ afin de diviser notre ensemble de données en deux sous-ensembles : l'un pour l'entrainement et l'autre pour l'évaluation. Nous avons réalisé une

recherche « $gridsearch$ » sur C et gamma (γ) du noyau RBF de SVM, et ce afin d'avoir les meilleures performances de notre classificateur.
Trois métriques ont été proposées pour évaluer les performances de notre classificateur : la précision (représentant le pourcentage des prédictions correctes), LUBS (représentant la perte par rapport à la sélection du format optimal) et GOWS (représentant le gain par rapport à la sélection du pire format). L'évaluation des performances de notre classificateur montre que notre système peut atteindre une valeur de précision de l'ordre de 95.65% [MHD18-b].

Les résultats que nous avons obtenus à l'issue de ces travaux de thèse sur le choix automatique du format de compression sont encourageants. Toutefois certains points sont encore à améliorer et de nombreux axes de recherche apparaissent. Ainsi, nous citons comme perspectives :

- Validation de notre système dans le cas de l'utilisation d'autre architectures parallèles tels que les architectures GPUs,
- Etude de l'impact d'utilisation de différents environnements d'exécution tels que l'utilisation de l'approche hybride MPI/Open MP dans le cas des cluster multi-cœurs,
- Incorporer l'étude des communications et des mouvements des données dans le processus de sélection du format de compression. Dans ce cas, l'utilisation d'autre outils d'ordonnancement peut servir. Nous citons par exemple l'utilisation des méthodes de l'hypergraphe qui permettent de minimiser les communications [AKA12],
- Etude et évaluation des performances de méthodes numériques, autres que le schéma de Horner et le PMVC
- Intégrer le choix de la meilleure implémentation dans le processus de décision. Ceci nécessite aussi l'étude de différentes techniques d'optimisation tels que les techniques d'optimisation du compilateur,
- Etude plus approfondie dans le cas des matrices avec structures régulières. Dans ce cas, d'autres formats de compression peuvent être intégrés dans notre système.
- L'étude et l'évaluation des performances d'autres algorithmes d'apprentissage et les comparer à SVM, tels que les méthodes du *deep learning*.

BIBLIOGRAPHIE

[AKA12] K. Akbudak, E. Kayaaslan & C. Aykanat, “Technical Report on Hypergraph-Partitionning-Based Models and Methods for Exploiting Cache Locality in Sparse-Matrix Vector Multiplication”, Rapport technique, Bilkent University, Faculty of Engineering, Ankara - Turquie, Février 2012.

[ArP10] W. Armstronga & Alistair P. Rendella, “Runtime Sparse Matrix Format Selection”, Procedia Computer Science, 1(1), pages 135–144, 2010

[Ayo10] T.O. Ayodele, “Types of Machine Learning Algorithms”, 2010.

[BeG09] N. Bell & M. Garland, “Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors”, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland-Oregon, Novembre 2009.

[Bis04] R. Bisseling, “Parallel Scientific Computation, a Structured Approach Using Bsp and Mpi”, Utrecht University, Oxford University Press, Rapport technique, Mai 2004.

[BJW16] A. Benatia, W. Ji, Y. Wang, & F. Shi, “Sparse Matrix Format Selection with Multiclass SVM for SPMV on GPU”, 45th International Conference on Parallel Processing (ICPP), Philadelphia- États-Unis, Août 2016.

[BJW16-a] A. Benatia, W. Ji, Y. Wang, & F. Shi, “Machine Learning Approach for the Predicting Performance of SpMV on GPU”, IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan-Chine, Décembre 2016.

[CoT93] M. Cosnard & D. Trystram, “Algorithmes et Architectures Parallèles”, Paris, 1993.

[Cou14] R. Couturier, “Designing scientific applications on GPUs”, 2014.

[CSV10] J.W. Choi, A. Singh & R.W. Vuduc, “Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs”, Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 115-126, Bangalore-Inde , Janvier 2010.

[CYL10] W. Cao, L. Yao, Z. Li, Y. Wang & Z Wang, “Implementing Sparse Matrix-Vector Multiplication using CUDA Based on a Hybrid Sparse Matrix Format”, International Conference on Computer Application and System Modeling (ICCASM 2010), Taïwan - Chine, Octobre 2010.

[DaY11] T. Davis & H. Yifan, “The University of Florida Sparse Matrix Collection”, ACM Transactions on Mathematical Software, 38 (1), pages 1–25, 2011.

[DeS08] S. Delage Santacreu, “Introduction au Calcul Parallèle avec la Bibliothèque MPI (Message Passing Interface) ”, Cours IDRIS (Institut du Développement et des Ressources en Informatique Scientifique), France, Novembre 2008.

[EHM06-a] N. Emad, O. Hamdi-Larbi & Z. Mahjoub, “Autour des Calculs Creux et des Calculs Creux Parallèles”, Rapport de recherche, laboratoire PRISM, France, 2006.

[EHM06-b] N. Emad, O. Hamdi-Larbi & Z. Mahjoub, “Vers la Détection du Meilleur Format de Compression pour Matrice Creuse dans un Environnement Parallèle”, Rapport de recherche, laboratoire PRISM, France, Octobre 2006.

[EkM03] A. Ekambaram & E. Montagne, “An Alternative Compressed Storage Format for Sparse Matrices”, Proceedings of International Symposium on Computer and Information Sciences, pages 196-203. Antalya - Turquie, Novembre 2003.

[FKA09] A. Farzaneh, H. Kheiri & M. Abbaspour-shahmersi, “An Efficient Storage Format for Large Sparse Matrices”, Communications Series A1 Mathematics & Statistics, 58(2), page 1, Ankara - Turquie, 2009.

[GuL16] P. Guo & C.W. Lee, “A Performance Prediction and Analysis Integrated Framework for SPMV on GPUs”, Procedia Computer Science, 80, pages 178 – 189, 2016.

[GWC14] P. Guo, L. Wang & P. Chen, “A Performance Modeling and Optimization Analysis Tool for Sparse Matrix-Vector Multiplication on GPUs”, IEEE Transactions on Parallel and Distributed Systems, 25(5), pages 1112 – 1123, Mai 2014.

[Ham10] O. Hamdi-Larbi, “Etude de la Distribution, sur Système à Grande Echelle, de Calcul Numérique Traitant des Matrices Creuses Compressées”, Thèse de doctorat en Informatique, FST (Tunisie)-UVSQ (France), Mars 2010.

[HCE09] K. Hamidouche, F. Cappello & D. Etiemble, “Comparaison de MPI, OpenMP et MPI+OpenMP sur un Nœud Multiprocesseur Multi-Cœurs AMD à Mémoire Partagée”, RenPar’19, Rencontre Francophone de Parallelisme, Toulouse - France, Septembre 2009.

[HCL16] C. W. Hsu, C.C. Chang & C. J. Lin, “A Practical Guide to Support Vector Classification”, 2016.

[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, California - États-Unis, 2007.

[ImY01] E. Im & K.Yelick, “Optimizing Sparse Matrix Computations for Register Reuse in SPARSITY”, In Proceedings of the International Conference on Computational Science - LNCS, 2073, pages 127-136, 2001.

[IYV04] E. Im, K.Yelick & R. Vuduc, “SPARSITY: Optimization Framework for Sparse Matrix Kernels”, International Journal of High Performance Computing Applications, 18(1), pages135 – 158, Février 2004.

[Jai09] A. Jain, “pOSKI: An Extensible Autotuning Framework to Perform Optimized SPMVs on Multicore Architectures”, Rapport de recherche, University of California, États-Unis, 2008.

[KGK08] K. Kourtis, G.Goumas, and N. Koziris, “Improving The Performance of Multithreaded Sparse Matrix-Vector Multiplication Using Index and Value Compression”, 37th International Conference on Parallel Processing, september 2008.

[KhU09] M.T. Khan & A. Usman, “Technique Detection Software for Sparse Matrices”, Annals. Computer Science Series, 7 (2), pages 57-66, 2009.

[KHW12] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. Basermann, A. R. Bishop, “Sparse Matrix-Vector Multiplication on GPGPU Clusters: A New Storage Format and a Scalable Implementation”, IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pages 1696-1702, Shanghai-Chine, Mai 2012.

[KoJ02] J. Koster, “Parallel Templates for Numerical Linear Algebra, A High-Performance Computation Library”, Thèse de doctorat, Utrecht University, Pays-Bas, Juillet 2002.

[Kot07] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques”, Informatica, 31(3), pages 3-24, 2007.

[KsR12] A.P. Kshirsagar & M.N. Rathod, “Artificial Neural Network”, IJCA Proceedings on National Conference on Recent Trends in Computing, Mai 2012.

[LaW10] P. Lavallee & P. Wautelet, “Programmation Hybride MPI-OpenMP”, Cours IDRIS (Institut du Développement et des Ressources en Informatique Scientifique), France, Avril 2012.

[LBE16] C. Lehnert, R. Berrendorf, J. P. Ecker, & F. Mannuss, “Performance Prediction and Ranking of SPMV Kernels on GPU Architectures”, Proceedings of the 22nd International Conference on Euro-Par, pages 90–102, New York- États-Unis, Août 2016.

[LeE08] S. Lee & R. Eigenmann, “Adaptive Runtime Tuning of Parallel Sparse Matrix-Vector Multiplication on Distributed Memory Systems”, Proceedings of the 22nd annual International Conference on Supercomputing, pages 195-204, Ile de kos, Grèce, Juin 2008.

[LZT12] J. Li, X. Zhang, G. Tan, M. Chen, “SMAT: An Input Adaptive Sparse Matrix-Vector Multiplication Auto-Tuner”, Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, pages 117-126, Washington- États-Unis, Juin 2013.

[MeH12] I. Mehrez & O. Hamdi-Larbi, “Sparse Matrix Vector Product distribution on a cluster of multicore nodes”, 3rd ALAMA meeting (Linear Algebra, Matrix Analysis and Applications), Madrid-Espagne, Juin 2012.

[MeH13-a] I. Mehrez & O. Hamdi-Larbi, “Compromis Equilibrage des Charges-Optimisation des Communications pour la Distribution de Calcul Creux sur une Grappe Multi-cœurs”, ComPAS’2013/RenPar’21, Grenoble- France, Janvier 2013.

[MeH13-b] I. Mehrez & O. Hamdi-Larbi “SMVP Distribution Using Hypergraph Model and S-GBNZ Algorithm”, 8th international conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Compiegne-France, Octobre 2013.

[Mey17] David Meyer, “Support Vector Machines: The Interface to libsvm in package e1071”, FH Technikum Wien, Austria, février 2017.

[MHD16] I. Mehrez, O. Hamdi-Larbi-Larbi, T. Dufaud & N. Emad, “Towards an Auto-Tuning System Design for Optimal Sparse Compression Format Selection with User Expertise”, 13th ACS/IEEE International Conference on Computer Systems and Applications AICCSA, Agadir-Maroc, Novembre-Décembre 2016.

[MHD17] I. Mehrez, O. Hamdi-Larbi, T. Dufaud & N. Emad, “Understanding the Performances of Sparse Compression Formats Using Data Parallel Programming Model”, International Conference on High Performance Computing & Simulation (HPCS), Gênes-Italie, 2017.

[MHD18-a] I. Mehrez, O. Hamdi-Larbi, T. Dufaud & N. Emad, “Optimal Sparse Compression Format Selection on Multicore Cluster”, Second Tunisian Workshop on High Performance Computing (HPC), Tunis-Tunisie, Avril 2018.

[MHD18-b] I. Mehrez, O. Hamdi-Larbi, T. Dufaud & N. Emad, “Machine Learning for Optimal Compression Format Prediction on Multiprocessor Platform”, International Conference on High Performance Computing & Simulation (HPCS), Orléans-France, Juillet 2018.

[MHD18-c] I. Mehrez, O. Hamdi-Larbi, T. Dufaud & N. Emad, “Understanding the Performances of SMVP on Multiprocessor Platform”, The 24th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas- États-Unis, Juillet-Août 2018.

[MLA10] A. Monakov, A. Lokhmotov & A. Avetisyan, “Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures”, Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers, pages 111–125, Pisa-Italie, Janvier 2010.

[MNM11] P. Morillon, L. Nussbaum & D. Margery, “Gestion d'une infrastructure expérimentale de grande échelle avec Puppet et Git”, 9èmes Journées Réseaux - JRES 2011, Toulouse - France, 2011.

[MoE04] E. Montagne & A. Ekambaram, “An Optimal Storage Format for Sparse Matrices”, Information Processing Letters, 90 (2), pages 87-92, 2004.

[NRR14] B. Neelima, G. R. M. Reddy, & P. S. Raghavendra, “Predicting an Optimal Sparse Matrix Format for SPMV Computation On GPU”, IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), AZ- États-Unis, Mai 2014.

[PeE06] S. Petiton & N. Emad, “A Data Parallel Scientific Computing Introduction”, Computer Science, 1132, pages 45–60, 1996.

[RHJ09] R. Rabenseifner, G. Hager, G. Jost, “Hybrid MPI/OpenMP Parallel Programming on Cluster of Multi-Core SMP Nodes”, Proceedings of the 17th Euromicro International Conference on Parallel Distributed and Network-based Processing, pages 427-436, Weimar-Allemagne, Février 2009.

[Saa03] Y. Saad, “Iterative Methods for Sparse Linear Systems, Second Edition”, 2003.

[Saa94] Y. Saad “SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations - Version 2”. Rapport technique, 1994.

[SEF15] O. Spillinger, D. Eliahu, A. Fox, and J. Demmel, “Matrix Multiplication Algorithm Selection with Support Vector Machines”, Rapport technique, University of California at Berkeley, 2015.

[SFH11] G. Schubert, H. Fehske, G. Hager & G. Wellein, “Hybrid-Parallel Sparse Matrix-Vector Multiplication with Explicit Communication Overlap on Current Multicore-Based Systems”, Parallel Processing Letters, 21, pages 339-358, 2011.

[SGF09] G. Schubert, G. Hager & H. Fehske, “Performance Limitations for Sparse Matrix-Vector Multiplications on Current Multicore Environments”, High Performance Computing in Science and Engineering, Munich - Allemagne, Décembre 2009.

[ShB14] S. Shalev-Shwartz & S. Ben-David, “Understanding Machine Learning: From Theory to Algorithms”, 2014.

[ShU07] R. Shahnaz & A. Usman, “An Efficient Sparse Matrix-Vector Multiplication on Distributed”, International Journal of Computer Science and Network Security, 7(1), pages 77-82, Janvier 2007.

[ShU11] R. Shahnaz & A. Usman, “Blocked-Based Sparse Matrix-Vector Multiplication on Distributed Memory Parallel Computers”, International Arab Journal of Information Technology, 8(2), pages 130-136, Avril 2011.

[SMP15] N. Sedaghati, T. Mu, L.N. Pouchet, S. Parthasarathy & P. Sadayappan, “Automatic Selection of Sparse Matrix Representation on GPUs”, Proceedings of the 29th ACM on International Conference on Supercomputing, pages 99-108, CA- États-Unis, Juin 2015.

[SrS11] A. Srinivasa & M. Sosonkina, “Non-Uniform Memory Affinity Strategy in Multi-Threaded Sparse Matrix Computations”, Proceedings of the 20th Symposium on High Performance Computing, Orlando - États-Unis, Mars 2012.

[Tea15] RStudio Team, “RStudio: Integrated Development for R”. RStudio, Inc., Boston, MA, 2015.
http://www.rstudio.com/.

[TSK18] P.N. Tan, M. Steinbach, A. Karpatne & V. Kumar, “Introduction to Data Mining, 2nd Edition”, Janvier 2018.

[VDY05] R. Vuduc, J.W. Demmel & K. Yelick, “OSKI: A Library of Automatically Tuned Sparse Matrix Kernels”, Journal of Physics: Conference Series, 16(1), pages 521, 2005.

[VFG11] F. Vázquez, J. J. Fernández & E. M. Garzon, “A New Approach for Sparse Matrix Vector Product on NVIDIA GPUs”, Concurrency and Computation: Practice and Experience, 23(8), pages 815-826, Juin 2011.

[VGM09] F. Vázquez, E. M. Garzon, J. A. Martinez & J.J Fernández, “The Sparse Matrix Vector Product on GPUs”, Proceedings of the 2009 International Conference on Computational and Mathematical Methods in Science and Engineering, 2, pages 1081-1092, Juin 2009.

[WGL17] R. Wang, T. Gu & M. Li, “Performance Prediction Based on Statistics of Sparse Matrix-Vector Multiplication on GPUs”, Journal of Computer and Communications, 5 (6), pages 65-83, Avril 2017.

[WOV07] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick & J. Demmel, “Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms”, Proceedings of the 2007 ACM/IEEE conference on SuperComputing, Nevada - États-Unis, Novembre 2007.

[YzB09] A.J. Yzelmen & R. Bisseling, “Cache-Oblivious Sparse Matrix-Vector Multiplication by Using Sparse Matrix Partitioning Methods”, SIAM Journal on Scientific Computing, 31, pages 3128-3154, États-Unis, Juin 2009.

[YzB11] A.J. Yzelmen & R. Bisseling, “Two-Dimentional Cache-Oblivious Sparse Matrix-Vector Multiplication”, Parallel Computing, 37, pages 806-819, Décembre 2011.

NETOGRAPHIE

[ALR] Algèbre linéaire, matrice bandes
http://gilles.dubois10.free.fr/algebre_lineaire/bandes.html

[AMP] D. Dureau "Architecture des machines parallèles : Quelques définitions et notions de base", 2007
http://www.e-campus.uvsq.fr/claroline/backends/download.php?url=LzIwMDgvRm9ybWF0aW9uX2luaXRpYWxlL01QSS9jb3VyczAxMDJfYXJjaGlfcGFyYWwucGRm&cidReset=true&cidReq=M1INFOUE13

[DTS] http://www.dataswift.fr/reseaux/myrinet/

[HPC] http://www.hpccommunity.org/content/ethernet-cluster-170/

[INF] Institut Fourier
http://www-fourier.ujf-grenoble.fr/~eiserm/enseignement.html#google

[MIG] MIGALE, Le cluster de calcul
http://migale.jouy.inra.fr/?q=cluster

[MIR] MIRAGE, Généralités sur le parallélisme
https://ciment.ujf-grenoble.fr/mirage/parallelisme/generalites-sur-le-parallelisme/generalites-sur-le-parallelisme

[MNH] Grappe de calcul parallèle (cluster)
http://www.mnhn.fr/oseb/Grappe-de-calcul-parallele-cluster

[MUM] http://graal.ens-lyon.fr/MUMPS/

[NRI] Julien Laval, "L'avenir en accélération : le multi-cœur devient la norme et l'informatique est poussée à l'extrême", 2011
http://newsroom.intel.com/docs/DOC-2372

[PCB] http://www.pcbureau.com/comment-fonctionnent-les-processeurs-multi-cores-504.html,

[Res] https://docplayer.fr/15387120-Reseaux-de-neurones-1-introduction-resume-1-1-historique-1-2-reseaux-de-neurones-1-reseaux-de-neurones.html

[TST] http://www.transtec.fr/fr/high-performance-computing/qlogic-infiniband

ANNEXE A : STATISTIQUES SUR LES POINTS DE DONNEES UTILISES DANS NOS TESTS

Matrices	P	n	nnz	diag	triang	nnzlower	nnzupper	nnzdiag	nzmaxC	nzminC	nzmaxR	nzminR
ASIC_680ks	200	682712	2329176	0	0	823232	823232	682712	210	1	210	1
bcsstk21	200	3600	15100	0	1	11500	0	3600	5	1	5	1
c-42	200	10471	60378	0	1	49907	0	10471	772	1	1282	1
plat362	200	362	3074	0	1	2712	0	362	15	1	15	1
steam2	200	600	13760	0	0	6580	6580	600	28	16	28	16
g7jac180	200	53370	747276	0	0	347631	346275	53370	153	2	120	2
watt__1	200	1856	11360	0	0	4752	4752	1856	7	1	7	2
bayer05	200	3268	27836	0	0	16346	11488	2	73	1	68	1
bcsstm09	200	1083	1083	1	0	0	0	1083	1	1	1	1
cavity26	200	4562	138187	0	0	66701	68123	3363	62	8	62	1
goodwin	200	7320	324784	0	0	147662	170881	6241	112	1	112	11
lung2	200	109460	492564	0	0	164187	218917	109460	8	2	8	2
rajat28	200	87190	607235	0	0	261686	260646	84903	29105	1	29105	1
bayer10	200	13436	94926	0	0	55385	39538	3	27	1	32	1
bloweybq	200	10001	39996	0	1	29995	0	10001	4	1	10001	1
c-big	200	345241	1343126	0	1	997885	0	345241	19578	1	6875	1
graham1	200	9035	335504	0	0	123354	205185	6965	406	1	398	5
memplus	200	17758	126150	0	0	54196	54196	17758	574	2	574	2
sherman2	200	1080	23094	0	0	9960	12054	1080	42	1	34	1
adder_dcop_43	200	1813	11246	0	0	3764	5681	1801	1310	1	1332	1
c-19	200	2327	12072	0	1	9745	0	2327	362	1	249	1
circuit_2	200	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
GT01R	200	7980	430909	0	0	212234	210695	7980	90	1	75	21
orsreg_1	200	2205	14133	0	0	5964	5964	2205	7	4	7	4

sherman4	200	1104	3786	0	0	1341	1341	1104	7	1	7	1
af23560	200	23560	484256	0	0	230348	230348	23560	21	11	21	10
c-31	200	5339	41955	0	1	36616	0	5339	458	1	332	1
ex19	200	12005	259879	0	0	123937	123937	12005	50	1	50	1
plat1919	200	1919	17159	0	1	15240	0	1919	15	1	15	1
spmsrtls	200	29995	129971	0	1	99976	0	29995	5	1	5	1
af_0_k101	200	503625	9027150	0	1	8523525	0	503625	25	1	25	1
af_3_k101	200	503625	9027150	0	1	8523525	0	503625	25	1	25	1
circuit_2	200	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
crashbasis	200	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
dc1	200	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc2	200	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc3	200	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
2D_27628_bjtcai	200	27628	442898	0	0	207723	207547	27628	90	9	90	3
ABACUS_shell_md_E	200	23412	19216	0	0	8806	8806	1604	12	0	12	0
ABACUS_shell_md_K	200	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_md_M	200	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_md	200	23412	218484	0	0	97536	97536	23412	15	6	15	6
ABACUS_shell_ud_K	200	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_ud_M	200	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_ud	200	23412	218484	0	0	97536	97536	23412	15	6	15	6
bayer02	200	13935	63679	0	0	31092	32583	4	22	1	28	1
bcsstk35	200	30237	740200	0	1	709963	0	30237	166	1	66	1
bcsstm37	200	25503	14765	0	1	760	0	14005	5	0	3	0
bloweybl	200	30003	70001	0	1	49999	0	20002	10001	1	5	0
bodyy6	200	19366	77057	0	1	57691	0	19366	9	1	8	1
brainpc2	200	27607	96601	0	1	82794	0	13807	7	1	13799	0
bratu3d	200	27792	88627	0	1	85169	0	3458	7	0	7	0
c-43	200	11125	67400	0	1	56275	0	11125	220	1	2620	1
c-44	200	10728	47864	0	1	37136	0	10728	780	1	122	1

c-47	200	15343	113372	0	1	98029	0	15343	630	1	2133	1
c-49	200	21132	89087	0	1	67955	0	21132	1532	1	197	1
c-51	200	23196	113123	0	1	89927	0	23196	1570	1	584	1
c-52	200	23948	113332	0	1	89384	0	23948	1714	1	224	1
c-53	200	30235	201224	0	1	170989	0	30235	362	1	4468	1
c-54	200	31793	211743	0	1	179950	0	31793	1702	1	1808	1
cage10	200	11397	150645	0	0	69624	69624	11397	25	5	25	5
case9_A_01	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_02	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_03	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_04	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_05	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_06	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_07	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_08	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_09	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_10	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_11	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_12	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9	200	14454	75801	0	1	72171	0	3630	12	0	7208	0
cond-mat-2003	200	31163	120029	0	1	120029	0	0	65	0	189	0
crystm03	200	24696	304233	0	1	279537	0	24696	14	1	14	1
cvxqp3	200	17500	69981	0	1	52481	0	17500	8	1	16	1
dtoc	200	24993	34986	0	1	34986	0	0	4	0	3	0
Dubcova1	200	16129	134569	0	1	118440	0	16129	12	1	25	1
epb2	200	25228	175027	0	0	65922	83877	25228	87	3	87	3
ex35	200	19716	228208	0	0	104246	104246	19716	18	1	18	1
fd15	200	11532	44206	0	0	17096	27110	0	6	1	8	1
g7jac040	200	11790	114671	0	0	59868	43013	11790	120	2	99	2
g7jac100	200	29610	384636	0	0	182911	172115	29610	153	2	120	2

Ill_Stokes	200	20896	191368	0	0	88465	87231	15672	12	3	12	3
inlet_E	200	11730	95316	0	0	58643	28223	8450	14	0	14	1
inlet	200	11730	328323	0	0	209830	106763	11730	41	1	39	9
mark3jac060sc	200	27449	170695	0	0	78478	64768	27449	44	2	47	2
memplus	200	17758	126150	0	0	54196	54196	17758	574	2	574	2
msc23052	200	23052	588933	0	1	565881	0	23052	178	1	54	1
ncvxqp9	200	16554	31547	0	1	22493	0	9054	3	1	9	0
ns3Da	200	20414	1679599	0	0	830185	830178	19236	306	9	306	9
OPF_6000	200	29902	166160	0	1	136258	0	29902	74	1	20	1
qpband	200	20000	30000	0	1	15000	0	15000	3	1	2	0
raefsky3	200	21200	1488768	0	0	733784	733784	21200	80	32	80	32
Si10H16	200	17077	446500	0	1	429423	0	17077	297	1	297	1
Si5H12	200	19896	379247	0	1	359351	0	19896	281	1	294	1
sinc15	200	11532	568526	0	0	247066	321460	0	93	1	94	1
smt	200	25710	1889447	0	1	1863737	0	25710	228	1	375	1
spmsrtls	200	29995	129971	0	1	99976	0	29995	5	1	5	1
std1_Jac2_db	200	21982	498771	0	0	233444	243345	21982	211	1	212	1
stokes64	200	12546	74242	0	1	65792	0	8450	8	1	9	0
t2dah_a	200	11445	93781	0	1	82336	0	11445	12	1	20	1
thread	200	29736	2249892	0	1	2220156	0	29736	255	1	291	1
tuma1	200	22967	50560	0	1	37200	0	13360	5	1	5	0
vibrobox	200	12328	177578	0	1	165250	0	12328	78	1	115	1
wang4	200	26068	177196	0	0	75564	75564	26068	7	4	7	4
wathen100	200	30401	251001	0	1	220600	0	30401	11	1	11	1
waveguide3D	200	21036	303468	0	0	141216	141216	21036	28	5	28	5
Zd_Jac2_db	200	22835	676439	0	0	284333	369271	22835	224	1	225	1
inline_1	200	503712	18660027	0	1	18156315	0	503712	843	1	333	1
mac_econ_fwd500	200	206500	1273389	0	0	741556	509333	22500	44	1	47	1
majorbasis	200	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
matrix_9	200	103430	2121550	0	0	1012440	1005680	103430	4057	12	677	13

matrix-new_3	200	125329	2678750	0	0	1293165	1260256	125329	9358	12	1139	12
patents_main	200	240547	560943	0	0	560784	159	0	206	0	173	0
PR02R	200	161070	8185136	0	0	4013166	4010900	161070	92	1	88	5
pre2	200	659033	5959282	0	0	3151218	2574288	233776	628	1	745	1
Raj1	200	263743	1302464	0	0	519769	519036	263659	40468	1	40468	1
rajat21	200	411676	1893370	0	0	865943	618293	409134	118689	1	100470	1
rajat23	200	110355	556938	0	0	225573	224533	106832	3401	1	3269	1
rajat24	200	358172	1948235	0	0	796695	795655	355885	105296	1	105296	1
rajat26	200	51032	249302	0	0	100618	100116	48568	3401	1	3269	1
rajat30	200	643994	6175377	0	0	2774857	2766155	634365	454746	1	454746	1
RM07R	200	381689	37464962	0	0	18566920	18516353	381689	295	1	245	1
sherman3	200	5005	20033	0	0	7514	7514	5005	7	1	7	1
stomach	200	213360	3021648	0	0	1404144	1404144	213360	19	7	22	6
torso1	200	116158	8516500	0	0	4396322	4004020	116158	3263	9	1224	8
torso2	200	115967	1033473	0	0	458751	458755	115967	10	5	10	6
torso3	200	259156	4429042	0	0	2084557	2085329	259156	22	7	21	6
trans4	200	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
trans5	200	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
twotone	200	120750	1224224	0	0	628411	544438	51375	185	1	188	1
hvdc2	200	189860	1347273	0	0	577592	579937	189744	60	1	60	1
webbase-1M	200	1000005	3105536	0	0	1348437	757094	1000005	4700	1	28685	1
memchip	200	2987012	26621990	0	0	11817558	11817567	2986865	2312481	1	2312476	1
ASIC_680ks	100	682712	2329176	0	0	823232	823232	682712	210	1	210	1
bcsstk21	100	3600	15100	0	1	11500	0	3600	5	1	5	1
c-42	100	10471	60378	0	1	49907	0	10471	772	1	1282	1
plat362	100	362	3074	0	1	2712	0	362	15	1	15	1
steam2	100	600	13760	0	0	6580	6580	600	28	16	28	16
g7jac180	100	53370	747276	0	0	347631	346275	53370	153	2	120	2
watt__1	100	1856	11360	0	0	4752	4752	1856	7	1	7	2
bayer05	100	3268	27836	0	0	16346	11488	2	73	1	68	1

bcsstm09	100	1083	1083	1	0	0	0	1083	1	1	1	1
cavity26	100	4562	138187	0	0	66701	68123	3363	62	8	62	1
goodwin	100	7320	324784	0	0	147662	170881	6241	112	1	112	11
lung2	100	109460	492564	0	0	164187	218917	109460	8	2	8	2
rajat28	100	87190	607235	0	0	261686	260646	84903	29105	1	29105	1
bayer10	100	13436	94926	0	0	55385	39538	3	27	1	32	1
bloweybq	100	10001	39996	0	1	29995	0	10001	4	1	10001	1
c-big	100	345241	1343126	0	1	997885	0	345241	19578	1	6875	1
graham1	100	9035	335504	0	0	123354	205185	6965	406	1	398	5
memplus	100	17758	126150	0	0	54196	54196	17758	574	2	574	2
sherman2	100	1080	23094	0	0	9960	12054	1080	42	1	34	1
adder_dcop_43	100	1813	11246	0	0	3764	5681	1801	1310	1	1332	1
c-19	100	2327	12072	0	1	9745	0	2327	362	1	249	1
circuit_2	100	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
GT01R	100	7980	430909	0	0	212234	210695	7980	90	1	75	21
orsreg_1	100	2205	14133	0	0	5964	5964	2205	7	4	7	4
sherman4	100	1104	3786	0	0	1341	1341	1104	7	1	7	1
af23560	100	23560	484256	0	0	230348	230348	23560	21	11	21	10
c-31	100	5339	41955	0	1	36616	0	5339	458	1	332	1
ex19	100	12005	259879	0	0	123937	123937	12005	50	1	50	1
plat1919	100	1919	17159	0	1	15240	0	1919	15	1	15	1
spmsrtls	100	29995	129971	0	1	99976	0	29995	5	1	5	1
af_0_k101	100	503625	9027150	0	1	8523525	0	503625	25	1	25	1
af_3_k101	100	503625	9027150	0	1	8523525	0	503625	25	1	25	1
circuit_2	100	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
crashbasis	100	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
dc1	100	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc2	100	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc3	100	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
2D_27628_bjtcai	100	27628	442898	0	0	207723	207547	27628	90	9	90	3

ABACUS_shell_md_E	100	23412	19216	0	0	8806	8806	1604	12	0	12	0
ABACUS_shell_md_K	100	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_md_M	100	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_md	100	23412	218484	0	0	97536	97536	23412	15	6	15	6
ABACUS_shell_ud_K	100	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_ud_M	100	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_ud	100	23412	218484	0	0	97536	97536	23412	15	6	15	6
bayer02	100	13935	63679	0	0	31092	32583	4	22	1	28	1
bcsstk35	100	30237	740200	0	1	709963	0	30237	166	1	66	1
bcsstm37	100	25503	14765	0	1	760	0	14005	5	0	3	0
bloweybl	100	30003	70001	0	1	49999	0	20002	10001	1	5	0
bodyy6	100	19366	77057	0	1	57691	0	19366	9	1	8	1
brainpc2	100	27607	96601	0	1	82794	0	13807	7	1	13799	0
bratu3d	100	27792	88627	0	1	85169	0	3458	7	0	7	0
c-43	100	11125	67400	0	1	56275	0	11125	220	1	2620	1
c-44	100	10728	47864	0	1	37136	0	10728	780	1	122	1
c-47	100	15343	113372	0	1	98029	0	15343	630	1	2133	1
c-49	100	21132	89087	0	1	67955	0	21132	1532	1	197	1
c-51	100	23196	113123	0	1	89927	0	23196	1570	1	584	1
c-52	100	23948	113332	0	1	89384	0	23948	1714	1	224	1
c-53	100	30235	201224	0	1	170989	0	30235	362	1	4468	1
c-54	100	31793	211743	0	1	179950	0	31793	1702	1	1808	1
cage10	100	11397	150645	0	0	69624	69624	11397	25	5	25	5
case9_A_01	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_02	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_03	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_04	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_05	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_06	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_07	100	14454	75801	0	1	72171	0	3630	12	0	7208	0

case9_A_08	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_09	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_10	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_11	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_12	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9	100	14454	75801	0	1	72171	0	3630	12	0	7208	0
cond-mat-2003	100	31163	120029	0	1	120029	0	0	65	0	189	0
crystm03	100	24696	304233	0	1	279537	0	24696	14	1	14	1
cvxqp3	100	17500	69981	0	1	52481	0	17500	8	1	16	1
dtoc	100	24993	34986	0	1	34986	0	0	4	0	3	0
Dubcova1	100	16129	134569	0	1	118440	0	16129	12	1	25	1
epb2	100	25228	175027	0	0	65922	83877	25228	87	3	87	3
ex35	100	19716	228208	0	0	104246	104246	19716	18	1	18	1
fd15	100	11532	44206	0	0	17096	27110	0	6	1	8	1
g7jac040	100	11790	114671	0	0	59868	43013	11790	120	2	99	2
g7jac100	100	29610	384636	0	0	182911	172115	29610	153	2	120	2
Ill_Stokes	100	20896	191368	0	0	88465	87231	15672	12	3	12	3
inlet_E	100	11730	95316	0	0	58643	28223	8450	14	0	14	1
inlet	100	11730	328323	0	0	209830	106763	11730	41	1	39	9
mark3jac060sc	100	27449	170695	0	0	78478	64768	27449	44	2	47	2
memplus	100	17758	126150	0	0	54196	54196	17758	574	2	574	2
msc23052	100	23052	588933	0	1	565881	0	23052	178	1	54	1
ncvxqp9	100	16554	31547	0	1	22493	0	9054	3	1	9	0
ns3Da	100	20414	1679599	0	0	830185	830178	19236	306	9	306	9
OPF_6000	100	29902	166160	0	1	136258	0	29902	74	1	20	1
qpband	100	20000	30000	0	1	15000	0	15000	3	1	2	0
raefsky3	100	21200	1488768	0	0	733784	733784	21200	80	32	80	32
Si10H16	100	17077	446500	0	1	429423	0	17077	297	1	297	1
Si5H12	100	19896	379247	0	1	359351	0	19896	281	1	294	1
sinc15	100	11532	568526	0	0	247066	321460	0	93	1	94	1

smt	100	25710	1889447	0	1	1863737	0	25710	228	1	375	1
spmsrtls	100	29995	129971	0	1	99976	0	29995	5	1	5	1
std1_Jac2_db	100	21982	498771	0	0	233444	243345	21982	211	1	212	1
stokes64	100	12546	74242	0	1	65792	0	8450	8	1	9	0
t2dah_a	100	11445	93781	0	1	82336	0	11445	12	1	20	1
thread	100	29736	2249892	0	1	2220156	0	29736	255	1	291	1
tuma1	100	22967	50560	0	1	37200	0	13360	5	1	5	0
vibrobox	100	12328	177578	0	1	165250	0	12328	78	1	115	1
wang4	100	26068	177196	0	0	75564	75564	26068	7	4	7	4
wathen100	100	30401	251001	0	1	220600	0	30401	11	1	11	1
waveguide3D	100	21036	303468	0	0	141216	141216	21036	28	5	28	5
Zd_Jac2_db	100	22835	676439	0	0	284333	369271	22835	224	1	225	1
inline_1	100	503712	18660027	0	1	18156315	0	503712	843	1	333	1
mac_econ_fwd500	100	206500	1273389	0	0	741556	509333	22500	44	1	47	1
majorbasis	100	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
matrix_9	100	103430	2121550	0	0	1012440	1005680	103430	4057	12	677	13
matrix-new_3	100	125329	2678750	0	0	1293165	1260256	125329	9358	12	1139	12
memchip	100	2987012	26621990	0	0	11817558	11817567	2986865	2312481	1	2312476	1
patents_main	100	240547	560943	0	0	560784	159	0	206	0	173	0
PR02R	100	161070	8185136	0	0	4013166	4010900	161070	92	1	88	5
pre2	100	659033	5959282	0	0	3151218	2574288	233776	628	1	745	1
Raj1	100	263743	1302464	0	0	519769	519036	263659	40468	1	40468	1
rajat21	100	411676	1893370	0	0	865943	618293	409134	118689	1	100470	1
rajat23	100	110355	556938	0	0	225573	224533	106832	3401	1	3269	1
rajat24	100	358172	1948235	0	0	796695	795655	355885	105296	1	105296	1
rajat26	100	51032	249302	0	0	100618	100116	48568	3401	1	3269	1
rajat30	100	643994	6175377	0	0	2774857	2766155	634365	454746	1	454746	1
RM07R	100	381689	37464962	0	0	18566920	18516353	381689	295	1	245	1
sherman3	100	5005	20033	0	0	7514	7514	5005	7	1	7	1
stomach	100	213360	3021648	0	0	1404144	1404144	213360	19	7	22	6

torso1	100	116158	8516500	0	0	4396322	4004020	116158	3263	9	1224	8
torso2	100	115967	1033473	0	0	458751	458755	115967	10	5	10	6
torso3	100	259156	4429042	0	0	2084557	2085329	259156	22	7	21	6
trans4	100	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
trans5	100	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
twotone	100	120750	1224224	0	0	628411	544438	51375	185	1	188	1
hvdc2	100	189860	1347273	0	0	577592	579937	189744	60	1	60	1
webbase-1M	100	1000005	3105536	0	0	1348437	757094	1000005	4700	1	28685	1
MatRow	100	10000	1009900	0	0	994950	4950	10000	101	100	10000	10000
MatRow	100	10000	2009800	0	0	1979900	19900	10000	201	200	10000	10000
MatRow	100	10000	3009700	0	0	2954850	44850	10000	301	300	10000	10000
MatRow	100	10000	5009500	0	0	4874750	124750	10000	501	500	10000	10000
MatRow	100	10000	10009000	0	0	9499500	499500	10000	1001	1000	10000	10000
MatRow	100	10000	20008000	0	0	17999000	1999000	10000	2001	2000	10000	10000
MatRow	100	10000	30007000	0	0	25498500	4498500	10000	3001	3000	10000	10000
MatRow	100	10000	40006000	0	0	31998000	7998000	10000	4001	4000	10000	10000
MatRow	100	10000	50005000	0	0	37497500	12497500	10000	5001	5000	10000	10000
MatRow	100	10000	60004000	0	0	41997000	17997000	10000	6001	6000	10000	10000
MatRow	100	10000	70003000	0	0	45496500	24496500	10000	7001	7000	10000	10000
MatRow	100	10000	80002000	0	0	47996000	31996000	10000	8001	8000	10000	10000
MatCol	100	10000	1009900	0	0	994950	4950	10000	10000	10000	101	100
MatCol	100	10000	2009800	0	0	1979900	19900	10000	10000	10000	201	200
MatCol	100	10000	3009700	0	0	2954850	44850	10000	10000	10000	301	300
MatCol	100	10000	5009500	0	0	4874750	124750	10000	10000	10000	501	500
MatCol	100	10000	10009000	0	0	9499500	499500	10000	10000	10000	1001	1000
MatCol	100	10000	20008000	0	0	17999000	1999000	10000	10000	10000	2001	2000
MatCol	100	10000	30007000	0	0	25498500	4498500	10000	10000	10000	3001	3000
MatCol	100	10000	40006000	0	0	31998000	7998000	10000	10000	10000	4001	4000
MatCol	100	10000	50005000	0	0	37497500	12497500	10000	10000	10000	5001	5000
MatCol	100	10000	60004000	0	0	41997000	17997000	10000	10000	10000	6001	6000

MatCol	100	10000	70003000	0	0	45496500	24496500	10000	10000	10000	7001	7000
MatCol	100	10000	80002000	0	0	47996000	31996000	10000	10000	10000	8001	8000
MatRow	100	50000	5049900	0	0	4994950	4950	50000	101	100	50000	50000
MatRow	100	50000	25049500	0	0	24874750	124750	50000	501	500	50000	50000
MatRow	100	50000	50049000	0	0	49499500	499500	50000	1001	1000	50000	50000
MatRow	100	50000	100048000	0	0	97999000	1999000	50000	2001	2000	50000	50000
MatRow	100	50000	250045000	0	0	237497500	12497500	50000	5001	5000	50000	50000
MatRow	100	50000	500040000	0	0	449995000	49995000	50000	10001	10000	50000	50000
MatRow	100	50000	750035000	0	0	637492500	1.12E+08	50000	15001	15000	50000	50000
MatRow	100	50000	1000030000	0	0	799990000	2E+08	50000	20001	20000	50000	50000
MatCol	100	50000	5049900	0	0	4994950	4950	50000	50000	50000	101	100
MatCol	100	50000	25049500	0	0	24874750	124750	50000	50000	50000	501	500
MatCol	100	50000	50049000	0	0	49499500	499500	50000	50000	50000	1001	1000
MatCol	100	50000	100048000	0	0	97999000	1999000	50000	50000	50000	2001	2000
MatCol	100	50000	250045000	0	0	237497500	12497500	50000	50000	50000	5001	5000
MatCol	100	50000	500040000	0	0	449995000	49995000	50000	50000	50000	10001	10000
MatCol	100	50000	750035000	0	0	637492500	1.12E+08	50000	50000	50000	15001	15000
MatCol	100	50000	1000030000	0	0	799990000	2E+08	50000	50000	50000	20001	20000
MatRow	100	100000	50099500	0	0	49874750	124750	100000	501	500	100000	100000
MatRow	100	100000	100099000	0	0	99499500	499500	100000	1001	1000	100000	100000
MatRow	100	100000	500095000	0	0	487497500	12497500	100000	5001	5000	100000	100000
MatRow	100	100000	700093000	0	0	675496500	24496500	100000	7001	7000	100000	100000
MatRow	100	100000	1000090000	0	0	949995000	49995000	100000	10001	10000	100000	100000
MatRow	100	100000	2000080000	0	0	1799990000	2E+08	100000	20001	20000	100000	100000
MatCol	100	100000	50099500	0	0	49874750	124750	100000	100000	100000	501	500
MatCol	100	100000	100099000	0	0	99499500	499500	100000	100000	100000	1001	1000
MatCol	100	100000	500095000	0	0	487497500	12497500	100000	100000	100000	5001	5000
MatCol	100	100000	700093000	0	0	675496500	24496500	100000	100000	100000	7001	7000
MatCol	100	100000	1000090000	0	0	949995000	49995000	100000	100000	100000	10001	10000
MatCol	100	100000	2000080000	0	0	1799990000	2E+08	100000	100000	100000	20001	20000

MatRow	100	300000	150299500	0	0	149874750	124750	300000	501	500	300000	300000
MatRow	100	300000	300299000	0	0	299499500	499500	300000	1001	1000	300000	300000
MatRow	100	300000	1500295000	0	0	1487497500	12497500	300000	5001	5000	300000	300000
MatRow	100	300000	2100293000	0	0	2075496500	24496500	300000	7001	7000	300000	300000
MatCol	100	300000	150299500	0	0	149874750	124750	300000	300000	300000	501	500
MatCol	100	300000	300299000	0	0	299499500	499500	300000	300000	300000	1001	1000
MatCol	100	300000	1500295000	0	0	1487497500	12497500	300000	300000	300000	5001	5000
MatCol	100	300000	2100293000	0	0	2075496500	24496500	300000	300000	300000	7001	7000
MatRow	100	500000	250499500	0	0	249874750	124750	500000	501	500	500000	500000
MatRow	100	500000	500499000	0	0	499499500	499500	500000	1001	1000	500000	500000
MatCol	100	500000	250499500	0	0	249874750	124750	500000	500000	500000	501	500
MatCol	100	500000	500499000	0	0	499499500	499500	500000	500000	500000	1001	1000
MatRow	100	1000000	500999500	0	0	499874750	124750	1000000	501	500	1000000	1000000
MatRow	100	1000000	1000999000	0	0	999499500	499500	1000000	1001	1000	1000000	1000000
MatCol	100	1000000	500999500	0	0	499874750	124750	1000000	1000000	1000000	501	500
MatCol	100	1000000	1000999000	0	0	999499500	499500	1000000	1000000	1000000	1001	1000
Diag_1000	100	1000	1000	1	0	0	0	1000	1	1	1	1
Diag_2000	100	2000	2000	1	0	0	0	2000	1	1	1	1
Diag_3000	100	3000	3000	1	0	0	0	3000	1	1	1	1
Diag_4000	100	4000	4000	1	0	0	0	4000	1	1	1	1
Diag_5000	100	5000	5000	1	0	0	0	5000	1	1	1	1
Diag_10000	100	10000	10000	1	0	0	0	10000	1	1	1	1
Diag_20000	100	20000	20000	1	0	0	0	20000	1	1	1	1
Diag_30000	100	30000	30000	1	0	0	0	30000	1	1	1	1
Diag_40000	100	40000	40000	1	0	0	0	40000	1	1	1	1
Diag_50000	100	50000	50000	1	0	0	0	50000	1	1	1	1
Diag_100000	100	100000	100000	1	0	0	0	100000	1	1	1	1
Diag_200000	100	200000	200000	1	0	0	0	200000	1	1	1	1
Diag_300000	100	300000	300000	1	0	0	0	300000	1	1	1	1
Diag_400000	100	400000	400000	1	0	0	0	400000	1	1	1	1

Diag_500000	100	500000	500000	1	0	0	0	500000	1	1	1	1
Diag_1000000	100	1000000	1000000	1	0	0	0	1000000	1	1	1	1
T_Sup_1000	100	1000	500500	0	2	0	499500	1000	1000	1	1000	1
T_Inf_1000	100	1000	500500	0	1	499500	0	1000	1000	1	1000	1
T_Sup_2000	100	2000	2001000	0	2	0	1999000	2000	2000	1	2000	1
T_Inf_2000	100	2000	2001000	0	1	1999000	0	2000	2000	1	2000	1
T_Sup_3000	100	3000	4501500	0	2	0	4498500	3000	3000	1	3000	1
T_Inf_3000	100	3000	4501500	0	1	4498500	0	3000	3000	1	3000	1
T_Sup_4000	100	4000	8002000	0	2	0	7998000	4000	4000	1	4000	1
T_Inf_4000	100	4000	8002000	0	1	7998000	0	4000	4000	1	4000	1
T_Sup_5000	100	5000	12502500	0	2	0	12497500	5000	5000	1	5000	1
T_Inf_5000	100	5000	12502500	0	1	12497500	0	5000	5000	1	5000	1
T_Sup_10000	100	10000	50005000	0	2	0	49995000	10000	10000	1	10000	1
T_Inf_10000	100	10000	50005000	0	1	49995000	0	10000	10000	1	10000	1
T_Sup_20000	100	20000	200010000	0	2	0	2E+08	20000	20000	1	20000	1
T_Inf_20000	100	20000	200010000	0	1	199990000	0	20000	20000	1	20000	1
T_Sup_30000	100	30000	450015000	0	2	0	4.5E+08	30000	30000	1	30000	1
T_Inf_30000	100	30000	450015000	0	1	449985000	0	30000	30000	1	30000	1
T_Sup_40000	100	40000	800020000	0	2	0	8E+08	40000	40000	1	40000	1
T_Inf_40000	100	40000	800020000	0	1	799980000	0	40000	40000	1	40000	1
T_Sup_1000	200	1000	500500	0	2	0	499500	1000	1000	1	1000	1
T_Inf_1000	200	1000	500500	0	1	499500	0	1000	1000	1	1000	1
T_Sup_2000	200	2000	2001000	0	2	0	1999000	2000	2000	1	2000	1
T_Inf_2000	200	2000	2001000	0	1	1999000	0	2000	2000	1	2000	1
T_Sup_3000	200	3000	4501500	0	2	0	4498500	3000	3000	1	3000	1
T_Inf_3000	200	3000	4501500	0	1	4498500	0	3000	3000	1	3000	1
T_Sup_4000	200	4000	8002000	0	2	0	7998000	4000	4000	1	4000	1
T_Inf_4000	200	4000	8002000	0	1	7998000	0	4000	4000	1	4000	1
T_Sup_5000	200	5000	12502500	0	2	0	12497500	5000	5000	1	5000	1
T_Inf_5000	200	5000	12502500	0	1	12497500	0	5000	5000	1	5000	1

T_Sup_10000	200	10000	50005000	0	2	0	49995000	10000	10000	1	10000	1
T_Inf_10000	200	10000	50005000	0	1	49995000	0	10000	10000	1	10000	1
T_Sup_20000	200	20000	200010000	0	2	0	2E+08	20000	20000	1	20000	1
T_Inf_20000	200	20000	200010000	0	1	199990000	0	20000	20000	1	20000	1
T_Sup_30000	200	30000	450015000	0	2	0	4.5E+08	30000	30000	1	30000	1
T_Inf_30000	200	30000	450015000	0	1	449985000	0	30000	30000	1	30000	1
T_Sup_40000	200	40000	800020000	0	2	0	8E+08	40000	40000	1	40000	1
T_Inf_40000	200	40000	800020000	0	1	799980000	0	40000	40000	1	40000	1
ASIC_680ks	160	682712	2329176	0	0	823232	823232	682712	210	1	210	1
bcsstk21	160	3600	15100	0	1	11500	0	3600	5	1	5	1
c-42	160	10471	60378	0	1	49907	0	10471	772	1	1282	1
plat362	160	362	3074	0	1	2712	0	362	15	1	15	1
steam2	160	600	13760	0	0	6580	6580	600	28	16	28	16
g7jac180	160	53370	747276	0	0	347631	346275	53370	153	2	120	2
watt__1	160	1856	11360	0	0	4752	4752	1856	7	1	7	2
bayer05	160	3268	27836	0	0	16346	11488	2	73	1	68	1
bcsstm09	160	1083	1083	1	0	0	0	1083	1	1	1	1
cavity26	160	4562	138187	0	0	66701	68123	3363	62	8	62	1
goodwin	160	7320	324784	0	0	147662	170881	6241	112	1	112	11
lung2	160	109460	492564	0	0	164187	218917	109460	8	2	8	2
rajat28	160	87190	607235	0	0	261686	260646	84903	29105	1	29105	1
bayer10	160	13436	94926	0	0	55385	39538	3	27	1	32	1
bloweybq	160	10001	39996	0	1	29995	0	10001	4	1	10001	1
c-big	160	345241	1343126	0	1	997885	0	345241	19578	1	6875	1
graham1	160	9035	335504	0	0	123354	205185	6965	406	1	398	5
memplus	160	17758	126150	0	0	54196	54196	17758	574	2	574	2
sherman2	160	1080	23094	0	0	9960	12054	1080	42	1	34	1
adder_dcop_43	160	1813	11246	0	0	3764	5681	1801	1310	1	1332	1
c-19	160	2327	12072	0	1	9745	0	2327	362	1	249	1
circuit_2	160	4510	21199	0	0	6734	9964	4501	1274	1	1285	1

GT01R	160	7980	430909	0	0	212234	210695	7980	90	1	75	21
orsreg_1	160	2205	14133	0	0	5964	5964	2205	7	4	7	4
sherman4	160	1104	3786	0	0	1341	1341	1104	7	1	7	1
af23560	160	23560	484256	0	0	230348	230348	23560	21	11	21	10
c-31	160	5339	41955	0	1	36616	0	5339	458	1	332	1
ex19	160	12005	259879	0	0	123937	123937	12005	50	1	50	1
plat1919	160	1919	17159	0	1	15240	0	1919	15	1	15	1
spmsrtls	160	29995	129971	0	1	99976	0	29995	5	1	5	1
af_0_k101	160	503625	9027150	0	1	8523525	0	503625	25	1	25	1
af_3_k101	160	503625	9027150	0	1	8523525	0	503625	25	1	25	1
circuit_2	160	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
crashbasis	160	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
dc1	160	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc2	160	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
dc3	160	116835	766396	0	0	324946	324615	116835	114190	1	114174	1
2D_27628_bjtcai	160	27628	442898	0	0	207723	207547	27628	90	9	90	3
ABACUS_shell_md_E	160	23412	19216	0	0	8806	8806	1604	12	0	12	0
ABACUS_shell_md_K	160	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_md_M	160	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_md	160	23412	218484	0	0	97536	97536	23412	15	6	15	6
ABACUS_shell_ud_K	160	23412	216082	0	0	95934	96736	23412	15	4	12	6
ABACUS_shell_ud_M	160	23412	206474	0	0	91932	91130	23412	12	4	11	4
ABACUS_shell_ud	160	23412	218484	0	0	97536	97536	23412	15	6	15	6
bayer02	160	13935	63679	0	0	31092	32583	4	22	1	28	1
bcsstk35	160	30237	740200	0	1	709963	0	30237	166	1	66	1
bcsstm37	160	25503	14765	0	1	760	0	14005	5	0	3	0
bloweybl	160	30003	70001	0	1	49999	0	20002	10001	1	5	0
bodyy6	160	19366	77057	0	1	57691	0	19366	9	1	8	1
brainpc2	160	27607	96601	0	1	82794	0	13807	7	1	13799	0
bratu3d	160	27792	88627	0	1	85169	0	3458	7	0	7	0

c-43	160	11125	67400	0	1	56275	0	11125	220	1	2620	1
c-44	160	10728	47864	0	1	37136	0	10728	780	1	122	1
c-47	160	15343	113372	0	1	98029	0	15343	630	1	2133	1
c-49	160	21132	89087	0	1	67955	0	21132	1532	1	197	1
c-51	160	23196	113123	0	1	89927	0	23196	1570	1	584	1
c-52	160	23948	113332	0	1	89384	0	23948	1714	1	224	1
c-53	160	30235	201224	0	1	170989	0	30235	362	1	4468	1
c-54	160	31793	211743	0	1	179950	0	31793	1702	1	1808	1
cage10	160	11397	150645	0	0	69624	69624	11397	25	5	25	5
case9_A_01	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_02	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_03	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_04	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_05	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_06	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_07	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_08	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_09	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_10	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_11	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9_A_12	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
case9	160	14454	75801	0	1	72171	0	3630	12	0	7208	0
cond-mat-2003	160	31163	120029	0	1	120029	0	0	65	0	189	0
crystm03	160	24696	304233	0	1	279537	0	24696	14	1	14	1
cvxqp3	160	17500	69981	0	1	52481	0	17500	8	1	16	1
dtoc	160	24993	34986	0	1	34986	0	0	4	0	3	0
Dubcova1	160	16129	134569	0	1	118440	0	16129	12	1	25	1
epb2	160	25228	175027	0	0	65922	83877	25228	87	3	87	3
ex35	160	19716	228208	0	0	104246	104246	19716	18	1	18	1
fd15	160	11532	44206	0	0	17096	27110	0	6	1	8	1

g7jac040	160	11790	114671	0	0	59868	43013	11790	120	2	99	2
g7jac100	160	29610	384636	0	0	182911	172115	29610	153	2	120	2
Ill_Stokes	160	20896	191368	0	0	88465	87231	15672	12	3	12	3
inlet_E	160	11730	95316	0	0	58643	28223	8450	14	0	14	1
inlet	160	11730	328323	0	0	209830	106763	11730	41	1	39	9
mark3jac060sc	160	27449	170695	0	0	78478	64768	27449	44	2	47	2
memplus	160	17758	126150	0	0	54196	54196	17758	574	2	574	2
msc23052	160	23052	588933	0	1	565881	0	23052	178	1	54	1
ncvxqp9	160	16554	31547	0	1	22493	0	9054	3	1	9	0
ns3Da	160	20414	1679599	0	0	830185	830178	19236	306	9	306	9
OPF_6000	160	29902	166160	0	1	136258	0	29902	74	1	20	1
qpband	160	20000	30000	0	1	15000	0	15000	3	1	2	0
raefsky3	160	21200	1488768	0	0	733784	733784	21200	80	32	80	32
Si10H16	160	17077	446500	0	1	429423	0	17077	297	1	297	1
Si5H12	160	19896	379247	0	1	359351	0	19896	281	1	294	1
sinc15	160	11532	568526	0	0	247066	321460	0	93	1	94	1
smt	160	25710	1889447	0	1	1863737	0	25710	228	1	375	1
spmsrtls	160	29995	129971	0	1	99976	0	29995	5	1	5	1
std1_Jac2_db	160	21982	498771	0	0	233444	243345	21982	211	1	212	1
stokes64	160	12546	74242	0	1	65792	0	8450	8	1	9	0
t2dah_a	160	11445	93781	0	1	82336	0	11445	12	1	20	1
thread	160	29736	2249892	0	1	2220156	0	29736	255	1	291	1
tuma1	160	22967	50560	0	1	37200	0	13360	5	1	5	0
vibrobox	160	12328	177578	0	1	165250	0	12328	78	1	115	1
wang4	160	26068	177196	0	0	75564	75564	26068	7	4	7	4
wathen100	160	30401	251001	0	1	220600	0	30401	11	1	11	1
waveguide3D	160	21036	303468	0	0	141216	141216	21036	28	5	28	5
Zd_Jac2_db	160	22835	676439	0	0	284333	369271	22835	224	1	225	1
inline_1	160	503712	18660027	0	1	18156315	0	503712	843	1	333	1
mac_econ_fwd500	160	206500	1273389	0	0	741556	509333	22500	44	1	47	1

majorbasis	160	206500	1273389	0	0	741556	509333	22500	44	1	47	1
matrix_9	160	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
matrix-new_3	160	103430	2121550	0	0	1012440	1005680	103430	4057	12	677	13
memchip	160	125329	2678750	0	0	1293165	1260256	125329	9358	12	1139	12
patents_main	160	240547	560943	0	0	560784	159	0	206	0	173	0
PR02R	160	161070	8185136	0	0	4013166	4010900	161070	92	1	88	5
pre2	160	659033	5959282	0	0	3151218	2574288	233776	628	1	745	1
Raj1	160	263743	1302464	0	0	519769	519036	263659	40468	1	40468	1
rajat21	160	411676	1893370	0	0	865943	618293	409134	118689	1	100470	1
rajat23	160	110355	556938	0	0	225573	224533	106832	3401	1	3269	1
rajat24	160	358172	1948235	0	0	796695	795655	355885	105296	1	105296	1
rajat26	160	51032	249302	0	0	100618	100116	48568	3401	1	3269	1
rajat30	160	643994	6175377	0	0	2774857	2766155	634365	454746	1	454746	1
ASIC_680ks	140	682712	2329176	0	0	823232	823232	682712	210	1	210	1
bcsstk21	140	3600	15100	0	1	11500	0	3600	5	1	5	1
c-42	140	10471	60378	0	1	49907	0	10471	772	1	1282	1
plat362	140	362	3074	0	1	2712	0	362	15	1	15	1
steam2	140	600	13760	0	0	6580	6580	600	28	16	28	16
g7jac180	140	53370	747276	0	0	347631	346275	53370	153	2	120	2
watt__1	140	1856	11360	0	0	4752	4752	1856	7	1	7	2
bayer05	140	3268	27836	0	0	16346	11488	2	73	1	68	1
bcsstm09	140	1083	1083	1	0	0	0	1083	1	1	1	1
cavity26	140	4562	138187	0	0	66701	68123	3363	62	8	62	1
goodwin	140	7320	324784	0	0	147662	170881	6241	112	1	112	11
lung2	140	109460	492564	0	0	164187	218917	109460	8	2	8	2
rajat28	140	87190	607235	0	0	261686	260646	84903	29105	1	29105	1
bayer10	140	13436	94926	0	0	55385	39538	3	27	1	32	1
bloweybq	140	10001	39996	0	1	29995	0	10001	4	1	10001	1
c-big	140	345241	1343126	0	1	997885	0	345241	19578	1	6875	1
graham1	140	9035	335504	0	0	123354	205185	6965	406	1	398	5

memplus	140	17758	126150	0	0	54196	54196	17758	574	2	574	2
sherman2	140	1080	23094	0	0	9960	12054	1080	42	1	34	1
adder_dcop_43	140	1813	11246	0	0	3764	5681	1801	1310	1	1332	1
c-19	140	2327	12072	0	1	9745	0	2327	362	1	249	1
circuit_2	140	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
GT01R	140	7980	430909	0	0	212234	210695	7980	90	1	75	21
orsreg_1	140	2205	14133	0	0	5964	5964	2205	7	4	7	4
sherman4	140	1104	3786	0	0	1341	1341	1104	7	1	7	1
af23560	140	23560	484256	0	0	230348	230348	23560	21	11	21	10
c-31	140	5339	41955	0	1	36616	0	5339	458	1	332	1
ex19	140	12005	259879	0	0	123937	123937	12005	50	1	50	1
plat1919	140	1919	17159	0	1	15240	0	1919	15	1	15	1
spmsrtls	140	29995	129971	0	1	99976	0	29995	5	1	5	1
af_0_k101	140	503625	9027150	0	1	8523525	0	503625	25	1	25	1
af_3_k101	140	503625	9027150	0	1	8523525	0	503625	25	1	25	1
circuit_2	140	4510	21199	0	0	6734	9964	4501	1274	1	1285	1
crashbasis	140	160000	1750416	0	0	1072812	517604	160000	11	6	18	4
Diag_1000	200	1000	1000	1	0	0	0	1000	1	1	1	1
Diag_2000	200	2000	2000	1	0	0	0	2000	1	1	1	1
Diag_3000	200	3000	3000	1	0	0	0	3000	1	1	1	1
Diag_4000	200	4000	4000	1	0	0	0	4000	1	1	1	1
Diag_5000	200	5000	5000	1	0	0	0	5000	1	1	1	1
Diag_10000	200	10000	10000	1	0	0	0	10000	1	1	1	1
Diag_20000	200	20000	20000	1	0	0	0	20000	1	1	1	1
Diag_30000	200	30000	30000	1	0	0	0	30000	1	1	1	1
Diag_40000	200	40000	40000	1	0	0	0	40000	1	1	1	1
Diag_50000	200	50000	50000	1	0	0	0	50000	1	1	1	1
Diag_100000	200	100000	100000	1	0	0	0	100000	1	1	1	1
Diag_200000	200	200000	200000	1	0	0	0	200000	1	1	1	1
Diag_300000	200	300000	300000	1	0	0	0	300000	1	1	1	1

Diag_400000	200	400000	400000	1	0	0	0	400000	1	1	1	1
Diag_500000	200	500000	500000	1	0	0	0	500000	1	1	1	1
Diag_1000000	200	1000000	1000000	1	0	0	0	1000000	1	1	1	1
Diag_1000	160	1000	1000	1	0	0	0	1000	1	1	1	1
Diag_2000	160	2000	2000	1	0	0	0	2000	1	1	1	1
Diag_3000	160	3000	3000	1	0	0	0	3000	1	1	1	1
Diag_4000	160	4000	4000	1	0	0	0	4000	1	1	1	1
Diag_5000	160	5000	5000	1	0	0	0	5000	1	1	1	1
Diag_10000	160	10000	10000	1	0	0	0	10000	1	1	1	1
Diag_20000	160	20000	20000	1	0	0	0	20000	1	1	1	1
Diag_30000	160	30000	30000	1	0	0	0	30000	1	1	1	1
Diag_40000	160	40000	40000	1	0	0	0	40000	1	1	1	1
Diag_50000	160	50000	50000	1	0	0	0	50000	1	1	1	1
Diag_100000	160	100000	100000	1	0	0	0	100000	1	1	1	1
Diag_200000	160	200000	200000	1	0	0	0	200000	1	1	1	1
Diag_300000	160	300000	300000	1	0	0	0	300000	1	1	1	1
Diag_400000	160	400000	400000	1	0	0	0	400000	1	1	1	1
Diag_500000	160	500000	500000	1	0	0	0	500000	1	1	1	1
Diag_1000000	160	1000000	1000000	1	0	0	0	1000000	1	1	1	1
Diag_1000	140	1000	1000	1	0	0	0	1000	1	1	1	1
Diag_2000	140	2000	2000	1	0	0	0	2000	1	1	1	1
Diag_3000	140	3000	3000	1	0	0	0	3000	1	1	1	1
Diag_4000	140	4000	4000	1	0	0	0	4000	1	1	1	1
Diag_5000	140	5000	5000	1	0	0	0	5000	1	1	1	1
Diag_10000	140	10000	10000	1	0	0	0	10000	1	1	1	1
Diag_20000	140	20000	20000	1	0	0	0	20000	1	1	1	1
Diag_30000	140	30000	30000	1	0	0	0	30000	1	1	1	1
Diag_40000	140	40000	40000	1	0	0	0	40000	1	1	1	1
Diag_50000	140	50000	50000	1	0	0	0	50000	1	1	1	1
Diag_100000	140	100000	100000	1	0	0	0	100000	1	1	1	1

Diag_200000	140	200000	200000	1	0	0	0	200000	1	1	1	1
Diag_300000	140	300000	300000	1	0	0	0	300000	1	1	1	1
Diag_400000	140	400000	400000	1	0	0	0	400000	1	1	1	1
Diag_500000	140	500000	500000	1	0	0	0	500000	1	1	1	1
Diag_1000000	140	1000000	1000000	1	0	0	0	1000000	1	1	1	1
Diag_200000	200000	0	1	1	0	0						

ANNEXE B: PREPARATION DU CLUSTER DE TEST

Systèmes à utiliser

1. **OAR :** un système de gestion de tâches pour les clusters et d'autres infrastructures informatiques.
2. **Kadeploy 3 :** système de déploiement rapide et évolutif pour grappes et grilles de calcul. Il permet aux utilisateurs de déployer leur propre système d'exploitation sur leurs nœuds réservés.

Etapes générales à suivre

1. Connexion à grid5000 à partir d'un terminal

ichrak@ubuntu: **ssh** user@access.grid5000.fr

2. ***Connexion à un site*** :

access: **ssh** *site*

3. Réservation d'un job:

fgrenoble: **oarsub** -r '*date heure*' –l nodes=*nb-noeuds*, walltime=*durée* -p 'cluster="*nom_cluster*"' -t deploy

4. Déploiement de l'environnement

kadeploy3 -e *nom_env* -f $OAR_FILE_NODES –k

RQ : $OAR_FILE_NODES contient la liste des nœuds réservés.

5. Lancement d'un script

sh script.sh

Génération de la clé ssh

$ssh-keygen -t dsa

Création d'un nouvel utilisateur

ichrak@fgrenoble:~$ id

uid=10591(ichrak) gid=8000(users) groups=8000(users), 9004(user), 9024(ml-users), 10000(orsay)

```
ichrak@fgrenoble:~$ oarsub -I -t deploy -l '{rconsole="YES"}/nodes=1,walltime=3'
    [ADMISSION RULE] Modify resource description with type constraints
    Generate a job key...
    OAR_JOB_ID=931499
    Interactive mode : waiting...
    Starting...
    Connect to OAR job 931499 via the node fgrenoble.grenoble.grid5000.fr
ichrak@fgrenoble:~$ cat $OAR_FILE_NODES
    edel-3.grenoble.grid5000.fr
ichrak@fgrenoble:~$ kadeploy3 -e squeeze-x64-NFS -f $OAR_FILE_NODES -k
ichrak@fgrenoble:~$ ssh root@edel-3.grenoble.grid5000.fr
root@edel-3:~# adduser --uid 10591 --ingroup users ichrak
    Adding user `ichrak' ...
    Adding new user `ichrak' (10591) with group `users' ...oarsub -I -l
cluster=1/nodes=3,walltime=10:0  -t deploy
    Creating home directory `/home/ichrak' ...
    Copying files from `/etc/skel' ...
    Enter new UNIX password:
    Retype new UNIX password:
    passwd: password updated successfully
    Changing the user information for ichrak
    Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
    Is the information correct? [Y/n] Y

root@edel-3:~# su - ichrak
ichrak@edel-3:~$  mkdir ~/.ssh
ichrak@edel-3:~$ exit
    logout
root@edel-3:~# cp /root/.ssh/authorized_keys /home/ichrak/.ssh/
root@edel-3:~# ls /home/ichrak/.ssh
    authorized_keys
root@edel-3:~# chown ichrak:users /home/ichrak/.ssh/authorized_keys
root@edel-3:~# ssh ichrak@edel-3.grenoble.grid5000.fr
root@edel-3:~# exit
    logout
    Connection to edel-3.grenoble.grid5000.fr closed.
root@edel-3:~# rm /etc/udev/rules.d/*-persistent-net.rules
```

Création d'un nouvel environnement

Installation de MICH2

root@edel-3:~# export http_proxy="http://proxy:3128";export https_proxy="http://proxy:3128"
root@edel-3:~# wget --no-check-certificate
http://www.mcs.anl.gov/research/projects/mpich2/
downloads/tarballs/1.4.1p1/mpich2-1.4.1p1.tar.gz
root@edel-3:~# tar xvf mpich2*.tar.gz
root@edel-3:~# cd mpich2-1.4.1p1
root@edel-3:~/mpich2-1.4.1p1# ./configure --prefix=/usr --enable-cxx –
disable-f77 --disable-fc --enable-threads=multiple --enable-debuginfo
root@ edel-3:~/mpich2-1.4.1p1# make
root@ edel-3:~/mpich2-1.4.1p1# make install
Installing MPE2 include files to /opt/include
Installing MPE2 libraries to /opt/lib
Installing MPE2 utility programs to /opt/bin
Installing MPE2 configuration files to /opt/etc
Installing MPE2 system utility programs to /opt/sbin
Installing MPE2 man to /opt/share/man
Installing MPE2 html to /opt/share/doc/
Installed MPE2 in /opt
/opt/sbin/mpeuninstall may be used to remove the installation
root@edel-3:~# export PATH=/home/ichrak/mpi_install/bin:$PATH

ichrak@fgrenoble:~$ ssh root@edel-3 tgz-g5k > ~/Img_Ichrak_NFS_VF.tgz
kaenv3 -p squeeze-x64-NFS -u deploy > Env_Ichrak_NFS_VF.env
ichrak@fgrenoble:~$ vi DescEnvIchrak_NFS_final.dsc
name : Env_Ichrak_NFS_VF
version : 1
description : Debian 6. MPICH2 NUMA NFS
author : support-staff@lists.grid5000.fr
tarball : /home/ichrak/Img_Ichrak_NFS_VF.tgz|tgz
postinstall:/grid5000/postinstalls/debian-x64-nfs-2.4-post.tgz|tgz|traitement.ash/rambin
kernel : /vmlinuz
initrd : /initrd.img
fdisktype : 83
filesystem : ext3
environment_kind : linux
demolishing_env : 0

Exemple réservation

ichrak@fgrenoble:~$ oarsub -r '2016-07-16 10:00:00' –l nodes=64, walltime=2:00:00 -p 'cluster="edel"' -t deploy

1. Réserver 64 nœuds le 16 juillet 2016 du site edel, pour une période 2 heures à partir de 10h

ichrak@fgrenoble:~$ kadeploy3 -e Env_Ichrak_NFS_VF -f $OAR_FILE_NODES –k

2. Déploiement de l’environnement crée sur les nœuds réservés

ichrak@edel-3:~$ Dossier_travail/sh Mon_Script.sh