Optimisation de l'utilisation du cache dans EUROPLEXUS

Marwa Sridi, Vincent Faucher, Bruno Raffin, Thierry Gautier

ComPAS'2014 : Parallélisme / Architecture / Système
Neuchâtel, Suisse, du 23 au 25 avril 2014

Optimisation de l'utilisation du cache dans EUROPLEXUS

Marwa Sridi, Vincent Faucher, Bruno Raffin, Thierry Gautier

CEA Saclay, DEN/DANS/DM2S/SEMT
Laboratoire DYN - Bâtiment 607
91191 Gif Sur Yvette - France
Marwa.Sridi@cea.fr

Résumé
Dans cet article, nous proposons une nouvelle organisation de la structure de données d'EUROPLEXUS, un code de simulation en dynamique rapide des fluides et des structures développé par le CEA. Cette nouvelle organisation est construite de telle sorte que les données consultées par le processeur travaillant sur une partie du calcul pendant un pas de temps T_i soient le plus contigües possible afin qu'elles tiennent dans le cache de ce dernier. Cette nouvelle répartition nous permettra de minimiser le nombre de défauts de cache comparé à celui obtenu avec l'organisation actuelle de la structure de données. Les études de performance ont validé le gain réalisé avec la nouvelle organisation des données dans le cas des problèmes de grande taille.

Mots-clés : Mémoire cache, défaut de cache, EUROPLEXUS, localité spatiale, structure de données.

---

1. Introduction

Les architectures actuelles se caractérisent par la complexité de leurs espaces de stockage intégrant différents types de mémoires. Ces mémoires se distinguent par leurs capacités et leurs vitesses d'accès. Généralement, ces deux critères sont inversement proportionnels. Les mémoires les plus rapides ont une capacité plus réduite.
Plusieurs développements d'optimisation portent sur l'utilisation du cache afin d'éviter une fréquence élevée de consultation des mémoires distantes et pour que toutes les données demandées soient le plus rapidement accessibles par le processeur. L'influence indéniable que peut avoir l'organisation de la structure de données sur les performances des codes de calcul de grande taille manipulant un jeu de données assez volumineux, nous a mené à veiller à la bonne structuration des données accédées dans le code de calcul industriel EUROPLEXUS.
Ce code met en œuvre des modèles industriels fortement hétérogènes (multiples modélisations, contraintes cinématiques pour les couplages...), rendant sa structure de données complexe et son exécution délicate à optimiser en termes d'utilisation de la mémoire.
Nous commençons dans la section 2, État de l'art, par donner un bref aperçu sur les travaux existants dans le domaine d'optimisation des codes de calcul industriels. Ensuite, nous présentons le code utilisé, et nous détaillons notamment sa structure de données. Dans la troisième section, nous exposons notre technique de réorganisation dans le cadre de l'optimisation de la structure de données d'EUROPLEXUS. Cette approche nous a permis de réaliser un gain de 75% en terme de taux de défaut de cache commis dans les routines de calcul par rapport

à la méthode d'organisation actuelle. Le travail que nous présentons, dans ce papier, est une préparation à la parallélisation du code avec la librairie KAAPI. Les threads de KAAPI vont pouvoir travailler sur des groupes de données contigües à l'intérieur de la boucle de calcul là où la parallélisation aura lieu.

2. État de l'art

La résolution numérique d'un problème dans un domaine continu nécessite le passage par une étape de discrétisation. A l'issue de cette étape, nous définissons des éléments géométriques qui décrivent de la manière la plus fidèle possible le milieu continu d'origine. Cette discrétisation est, communément, la tâche confiée à un logiciel spécifique appelé le mailleur. Il utilise des algorithmes spécifiques qui respectent des méthodes permettant de bien optimiser les mailles. Ces optimisations sont faites selon des considérations bien définies (telles que la frontière du domaine, la finesse exigée [1], ...).
Cependant, malgré les optimisations apportées par le mailleur sur sa première version du maillage, il est très compliqué de pousser les optimisations encore plus loin pour faire en sorte que les numéros des mailles générés répondent aussi bien à la contrainte de proximité en mémoire [2] qu'aux autres contraintes dictées par les algorithmes de maillage.
Conscients de la complexité de cette tâche d'optimisation, les développeurs des codes de mécanique ont opté pour la dissociation de cette étape de l'ensemble des tâches confiées au mailleur. Ils se sont penchés sur le posttraitement des fichiers de maillage en exploitant des techniques de renumérotation avant de les passer en entrée du code du calcul [3] . Pendant cette phase, plusieurs techniques de renumérotation sont exploitées.
Dans la littérature, parmi les techniques les plus répandues on cite celles basées sur une indexation spatiale respectant le fait que, dans un intervalle donné d'indices, la majorité des éléments appartiennent à la liste des voisins directs.
Cette technique utilise les courbes remplissantes (space filling curves) de Peano pour tracer un chemin continu passant par tous les éléments du maillage de manière à garantir le principe d'indexation spatiale cités ci-dessus [1].
Un des mailleurs utilisés avec le code de calcul EPX est CAST3M [7]. C'est un logiciel développé par le CEA. Il emploie une méthode de renumérotation pour les nœuds de type Cuthill McKee inverse [4, 3].
Dans cet article, nous n'allons pas nous attarder sur l'étude de ces techniques de renumérotation. En revanche, nous allons mettre l'accent sur l'optimisation qu'on pourrait apporter à l'intérieur du code du calcul. Ce choix est justifié par la robustesse et l'efficacité des techniques d'optimisation intervenant à l'intérieur du code que ce soit sur sa structure de données ou sur les boucles de calcul. Dans ce travail, nous admettons que nous disposons déjà en entrée d'un maillage optimisé sur lequel des travaux de renumérotation ont été effectués et nous allons nous intéresser aux optimisations que nous pouvons implémenter au niveau de l'utilisation de la mémoire cache.
Dans un problème d'optimisation du cache nous cherchons principalement à réaliser deux objectifs : le premier objectif c'est de trouver la bonne méthode qui nous garantit que la donnée accédée récemment, sera très prochainement réutilisée dans la boucle du calcul. C'est le pro-

1. La finesse du maillage se traduit par la variation de la taille des formes selon la complexité du calcul dans quelques zones par rapport à d'autres
2. Les données des mailles dont l'écart entre les numéros est faible, doivent être spatialement proches sur la grille du maillage.
3. Dans certains cas le mailleur est intégré dans le code du calcul.

blème de la localité temporelle. Le deuxième but consiste à assurer que les données contigües en mémoire soient consécutivement utilisées au cours du calcul. Nous parlons de la localité spatiale.
De nombreux travaux de recherche ont porté sur la première voie d'optimisation qui s'intéresse à la localité temporelle. Dans ce cadre, M.E. Wolf et M.S. Lam [10] ont travaillé sur les algorithmes de ré-ordonnancement des boucles imbriquées pour optimiser l'utilisation des données du cache. Cette approche nécessite de faire des transformations sur le code en travaillant sur les bornes des boucles ainsi que sur leurs corps. Cette méthode n'est pas toujours applicable surtout dans le cas des codes de calcul ayant des dépendances complexes entre les différentes boucles. M.Kandemir et A.Choudhary [9] ont privilégié la réorganisation de données en mémoire par rapport à la transformation des nids de boucles. Ils se sont servis des techniques d'algèbre linéaire pour effectuer des transformations sur les matrices des données. Dans ce même cadre, P.Clauss [2] a mis en œuvre une méthode plus générale pour la réorganisation des données dans la mémoire, il a proposé une méthode basée sur des transformations polynomiales appliquées sur les accès aux données. Cette démarche nécessite un paramétrage des bornes des boucles de calcul et des tailles des tableaux.
L'avantage des approches basées sur la réorganisation des tableaux de données est qu'elles ne demandent pas des interventions de transformation sur tout le code. Les modifications effectuées sont restreintes aux tableaux de données accédés et n'affectent pas les routines internes.

3. Réorganisation des données à la volée

3.1. Présentation du code EUROPLEXUS

Le code EUROPLEXUS [8] (abrégé EPX dans la suite du texte) est un logiciel de simulation en dynamique rapide des structures et des fluides en interaction, copropriété du CEA et de la Commission Européenne (*Joint Research Center*). EPX est utilisé en particulier pour l'analyse des conséquences des accidents dans le nucléaire civil et pour l'analyse de la protection des citoyens face à diverses malveillances.

3.2. L'organisation hiérarchique du code EUROPLEXUS

Le code EPX possède une structure de données complexe de par son large domaine d'application et la complexité des phénomènes qu'il permet de simuler. Dans la suite de cette section, nous expliquons l'organisation générale de cette structure et nous optons pour une classification en familles des différentes variables manipulées pour justifier l'approche de réorganisation que nous allons proposer par la suite.
Comme la plupart des codes de dynamique rapide, EPX est basé sur un ensemble de routines qui obéissent à une certaine hiérarchie. Les routines de haut niveau prennent en charge la lecture des données, les initialisations globales et le lancement du calcul. Les calculs élémentaires commencent au niveau de la routine principale CELEM. Dans cette routine, se fait le calcul élémentaire par appel à la routine LOOPELM à l'intérieur d'une boucle sur tous les éléments du maillage. LOOPELM effectue les tâches élémentaires en appelant les routines secondaires spécialisées. Par exemple, dans le cas des éléments cubiques, la routine CUBE est appelée par LOOPELM (Figure 1).

3.3. La structure de données du code EUROPLEXUS

A l'issu d'une opération de discrétisation et de maillage, les éléments obtenus possèdent leurs propres caractéristiques. En effet, si nous focalisons notre étude sur une maille, nous trouvons que pour mener un calcul à terme, nous devons fournir au code de calcul plusieurs informa-

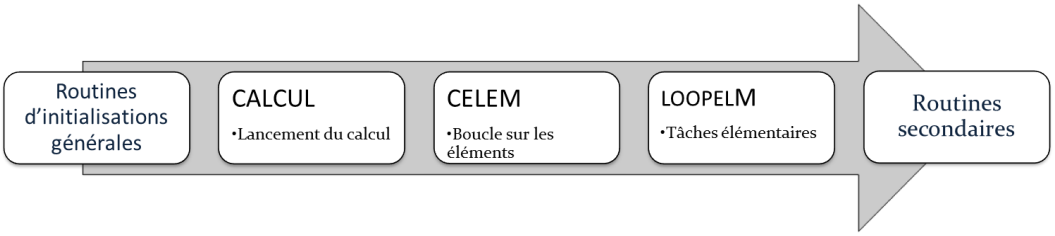


FIGURE 1 – Organisation des routines du code EPX

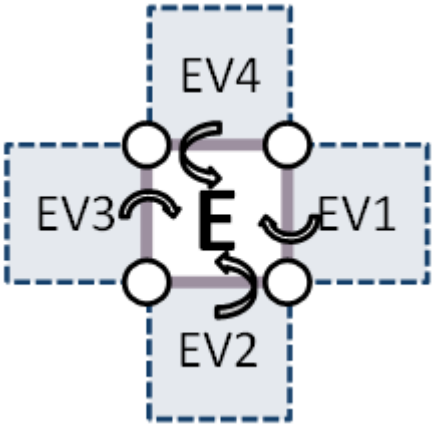


FIGURE 2 – Représentation d'une maille et de ses voisins en 2D

tions (Figure 2) :
- des données élémentaires : ce sont les informations qui définissent directement l'élément en question ;
- des données nodales : ce sont les informations relatives aux nœuds de chaque maille ;
- des données de voisinage : ce sont les variables qui décrivent tout échange entre l'élément et ses cellules voisines, que ce soit sous forme de transfert d'énergie ou de masse (cas d'interaction fluide/structure).

Les trois familles que nous venons de construire, forment le paquet complet des données nécessaires pour le déroulement d'un calcul dans EPX. Ces notions restent valables aussi bien dans le cas d'interaction structure-structure que dans le cas d'interaction fluide-structure. Les différentes variables de ces familles sont stockées dans des tableaux dynamiques séparés et elles sont, par conséquent, logées dans des blocs non contigus en mémoire.
Dans EPX, pendant un pas de temps T_i, et pour un élément du maillage IEL, le processeur fait appel à chacune de ces variables selon les besoins du calcul. Comme ces données sont stockées sur différents tableaux, chaque fois que le processeur demande une donnée, il charge dans son cache tout le bloc mémoire correspondant. La donnée demandée appartient à une ligne du cache contenant d'autres données inutiles pour le calcul de l'élément courant.
La manière avec laquelle les données élémentaires sont stockées dans les tableaux (Figure 3), n'assure pas une utilisation efficace du contenu du cache. Dans la plupart des cas, nous remplissons le cache par une petite portion de données utiles pour le calcul courant (c'est la partie qui correspond aux cases réellement demandées par le processeur de chaque tableau de don-

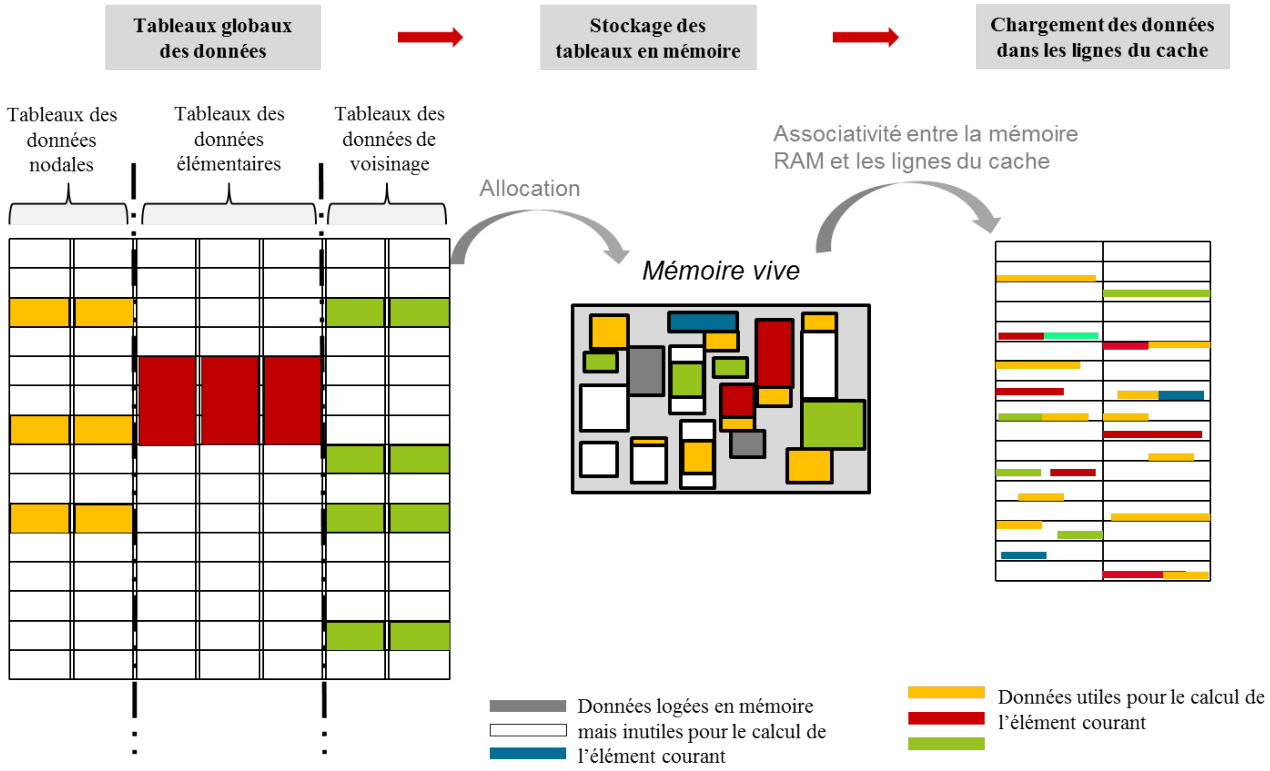


FIGURE 3 – Schématisation de la disposition des données dans le cache

nées). La partie restante sera remplie inutilement par des données qui ont été copiées car elles appartiennent à la même ligne de cache que la donnée utile.
L'organisation des données dans les tableaux telle qu'elle est faite dans EPX est coûteuse en terme de ressources mémoire. Ce type de structure est utilisé d'une façon répétitive tout au long du calcul. A cause de cette structuration, une grande partie du temps d'exécution est dépensée dans des défauts de cache. Ces défauts pourraient être significativement réduits si nous créons une nouvelle structure de données qui obéit à la règle de localité spatiale des données. La nouvelle structure doit garantir la présence d'un maximum de données utiles dans le cache. De plus, la complexité de ce code nous incite à intervenir en amont des routines de calcul. Par conséquent, l'optimisation que nous allons apporter doit se limiter à des modifications dans les routines principales et ne doit affecter que les arguments des routines secondaires concernées lors des appels.

3.4. Présentation de l'approche proposée

La technique que nous avons adoptée pour remplir notre objectif d'optimisation de la localité spatiale des données dans le cache est basée sur le passage d'une structure de données globale à des données de structure locales. Cela consiste à travailler sur des blocs de données contigus en mémoire qui décrivent la structure en question (la maille). Nous parlons ainsi des données caractérisant la maille et provenant des trois familles de données que nous avons définies dans la sous-section 3.2.
Notre but est d'assurer que, pendant chaque pas de temps T_i, toutes les données qui occupent le cache soient utiles. Une fois la boucle de calcul entamée (dans la routine CELEM), nous de-

Version de référence

```
SUBROUTINE CELEM (...)

...

DO IELOOP = 1 , NELEM    ! Boucle sur tous les
éléments du maillage où NELEM est le nombre
total d'éléments

        ...
        CALL LOOPELM(...)

ENDDO
...

END SUBROUTINE CELEM
```

Version optimisée

```
SUBROUTINE CELEM (...)
...
  DO IELOOP_GR = 1, NGR    ! Boucle sur les groupes
d'éléments

      CALL GR_WPGEN(...)  ! Routine de copie des
données des tableaux globaux vers les
                              !tableaux locaux
      DO IELOOP = ISTART , IEND    ! Boucle sur les
éléments du groupe où ISTART est l'indice du
premier élément du groupe et IEND est l'indice du
dernier élément du groupe
          ...
          CALL LOOPELM(...)
      ENDDO
      CALL GR_WPCOP(...)  ! Routine de copie des
données des tableaux locaux vers les !
                              !tableaux globaux

ENDDO
...

END SUBROUTINE CELEM
```

FIGURE 4 – Structure de la routine CELEM pour la méthode d'organisation par groupe

vrions disposer de tout le paquet d'informations qui caractérisent les éléments à traiter, afin de permettre au processeur d'achever son calcul sans avoir besoin de charger des données supplémentaires à partir des tableaux globaux.
Cette approche est basée sur la création d'un espace de travail local. Nous copions, pendant la phase d'initialisation, toutes les données issues des trois familles qui caractérisent l'élément en tenant compte de son voisinage. L'opération de copie est effectuée par la routine GR_*W*PGEN à l'intérieur d'une boucle sur un nombre d'éléments que nous définissons au préalable. Ce nombre sera considéré comme un paramètre réglable sur lequel nous pouvons agir, pour gérer la taille des groupes d'éléments à considérer tout en gardant l'aspect contigu des données manipulées. Une fois la boucle du calcul élémentaire (routine LOOPELM) achevée, une deuxième copie doit être effectuée pour mettre à jour les valeurs des tableaux globaux. Cette opération de copie des tableaux globaux vers les tableaux locaux est assurée par la routine GR_*W*PCOP (Figure 4).
Les numéros des éléments de chaque groupe correspondent à des indices consécutifs des mailles à traiter. Ces éléments seront traités successivement au moment de l'exécution. Cette méthode impose l'attribution d'indices locaux pour les éléments du groupe courant.
Procéder par un basculement entre les indices locaux des éléments et leurs indices globaux est un bon moyen pour éviter de faire trop de modifications sur les routines appelées par la routine de calcul. En ce qui concerne les routines internes, les grandes modifications à apporter se limitent au niveau des arguments lors des appels, tout en gardant un fonctionnement identique.
Pour un groupe Gr_i, on boucle d'abord sur ses éléments pour identifier leurs voisins. Nous

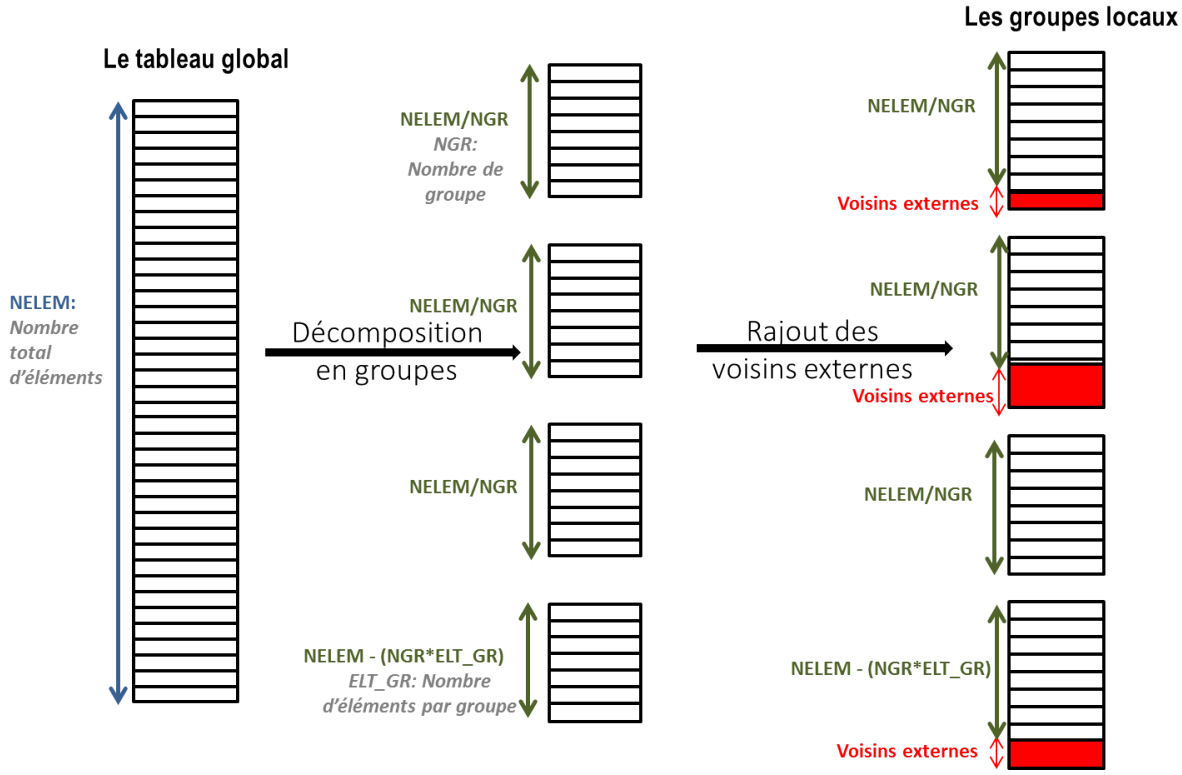


FIGURE 5 – Schéma du principe de la méthode d'organisation par groupe

considérons que l'élément V est un voisin de l'élément IEL s'ils ont au moins une face commune sur la grille du maillage. De ce fait, nous déterminons la liste de tous les voisins de IEL en vérifiant toutes ses faces communes avec les cellules voisines. Ensuite, nous testons si l'indice global du voisin est présent dans la liste des éléments effectivement utilisés dans le groupe que nous avons construit au départ :

– si le voisin fait partie de la liste, alors il sera doublement exploité une fois chargé dans le cache ;
– sinon, nous ne le traitons que comme simple voisin et dans ce cas, nous ne lui chargeons que les données relatives à son rôle de voisin et nous le rajoutons au groupe en question (Figure 5).

4. Expérimentations

Toutes les expérimentations ont été effectuées sur un processeur Intel ayant deux niveaux de cache inclusifs (L1 et L2) et un niveau non inclusif (L3). La taille du cache L1 est 32 KB, celle du cache L2 est égale à 256 KB et celle du L3 est égale à 16 MB.
Les cas tests que nous avons utilisés simulent des interactions de type fluide/structure. Ce type de simulations nous donne la possibilité de vérifier la validité de notre approche dans le cas d'un éventuel échange entre l'élément courant et les cellules voisines (exploitation des données de voisinage).
La taille des maillages utilisés dans les différents cas tests varie de l'ordre d'une centaine jusqu'à 10^6 éléments. L'utilisation des tests dont la taille dépasse la capacité de stockage du cache

nous permet de tirer profil de l'approche de décomposition par groupes.
L'outil de mesure de performance que nous avons utilisé est LIKWID [5]. La métrique sur laquelle nous nous sommes basés pour évaluer les performances de l'approche d'organisation des données par groupes est le taux de défauts de cache (Miss rate). Cet indicateur de performance représente le rapport entre le nombre de défauts de cache et le nombre total d'accès :

$$taux\ de\ défauts\ de\ cache = \frac{nombre\ total\ de\ défauts}{nombre\ total\ d'accès}$$

Les résultats expérimentaux ont montré que l'utilisation de cette méthode est bénéfique dans les problèmes manipulant un jeu de données de taille suffisamment importante dépassant la capacité du cache. Dans le cas contraire, les coûts dus à la construction des groupes [4] et à la mise à jour des valeurs des tableaux globaux une fois le calcul terminé [5]sont pénalisant en termes de défauts de cache.
Pour les problèmes de grande taille, la version avec réorganisation des données sous formes de groupes s'est avérée plus efficace pour la gestion du cache que la version de référence du code EPX.
Dans la figure 6, nous comparons le taux de défauts de cache dans la routine principale (CELEM), dans la boucle du calcul élémentaire (LOOPELM) et dans une routine interne (CUBE)des deux versions. Nous constatons que le taux de défauts de cache de la version optimisée est inférieur à celui de la version de référence pour les routines internes (LOOPELM et CUBE). Cependant, pour CELEM, il est supérieur à celui de la version de référence du code.
Ce taux élevé résulte des opérations de copie préalable des données des tableaux globaux vers les tableaux locaux avant la boucle des calculs élémentaires et des opérations inverses après la boucle.
Nous avons analysé les taux de défauts de cache dans les routines de copie des données pour certaines variables de transfert de flux entre l'élément courant et ces voisins. Dans la figure 7, nous comparons le taux de défauts de cache commis dans GR_WPGEN et GR_WPCOP pour la variable DMASS la variable de transfert de masse entre les cellules dans EPX.
Cette analyse nous a montré que les copies des données des voisins externes au groupe présentent un taux de défauts de cache plus élevé que celui des copies des données pour les éléments du groupe.
De plus, le nombre de voisins externes au groupe s'est avéré élevé par rapport à la taille du groupe surtout pour les groupes de petites tailles (Figure 8). Ces valeurs demandent des optimisations qui feront l'objet de travaux évoqués dans les perspectives.
Le choix de la taille des groupes dépend du problème à traiter. Il n'y a pas une taille optimale qui peut être généralisée pour tous les cas tests pour une taille donnée du cache. Ce manque de généralisation pourrait être expliqué par la dissemblance des caractéristiques des éléments à traiter du fait de leurs propriétés mécaniques et physiques (par exemple, on ne traite pas de la même manière un problème d'interaction fluide-structure et un problème d'interaction structure-structure). A ce titre, nous avons adopté une démarche empirique pour la détermination de la taille du groupe optimale pour le jeu de données à considérer.
Nous avons lancé une série de tests sur des groupes de tailles variées. L'ensemble de ces tests a été effectué sur le même jeu de données afin de visualiser l'influence de ce paramètre sur les performances du cache. L'analyse des résultats obtenus avec les différents cas tests a montré que le rapport de défaut du cache L2 varie en fonction de la taille du groupe. La figure 9 représente la courbe de variation du rapport de défaut de cache (L2 MISS RATIO) en fonction de la

4. copie des données des tableaux globaux vers les tableaux locaux dans la routine GR_WPGEN
5. copie des données des tableaux locaux vers les tableaux globaux dans la routine GR_WPCOP

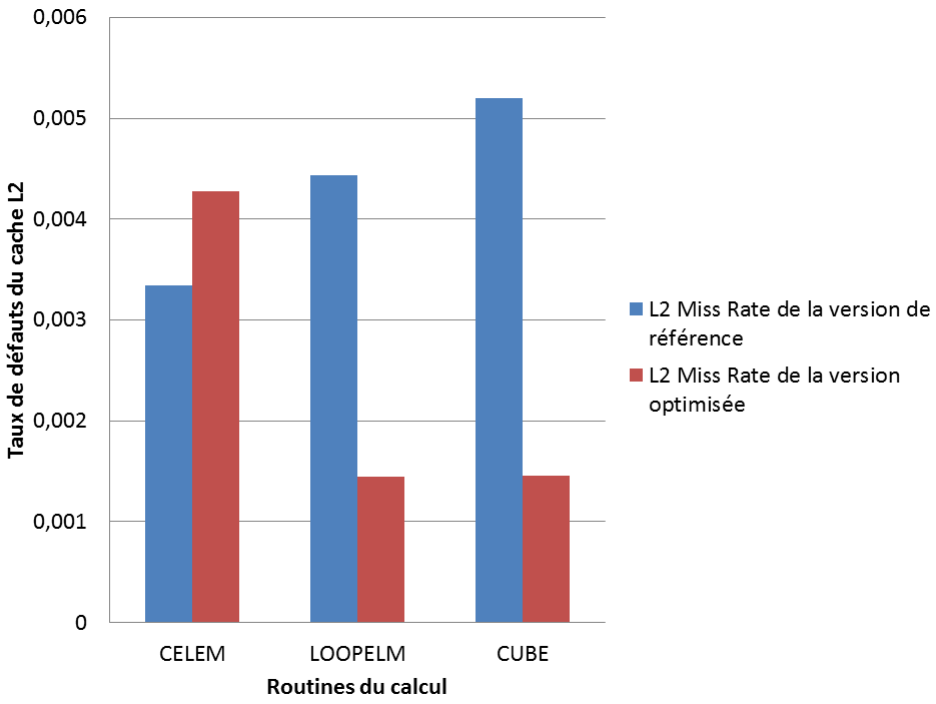


FIGURE 6 – Taux de défauts de cache des routines de la version optimisée comparé à la version de référence

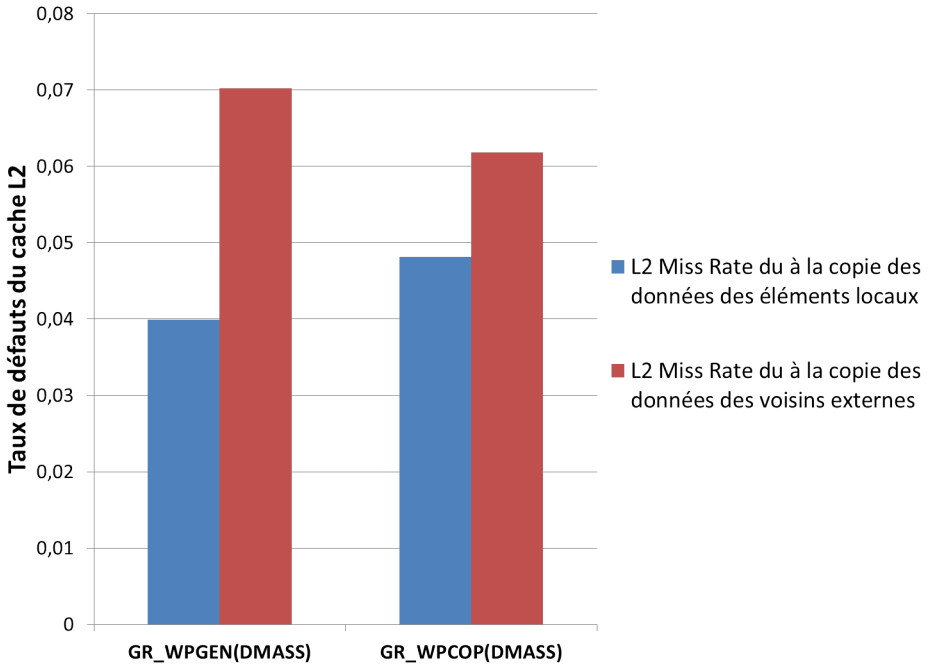
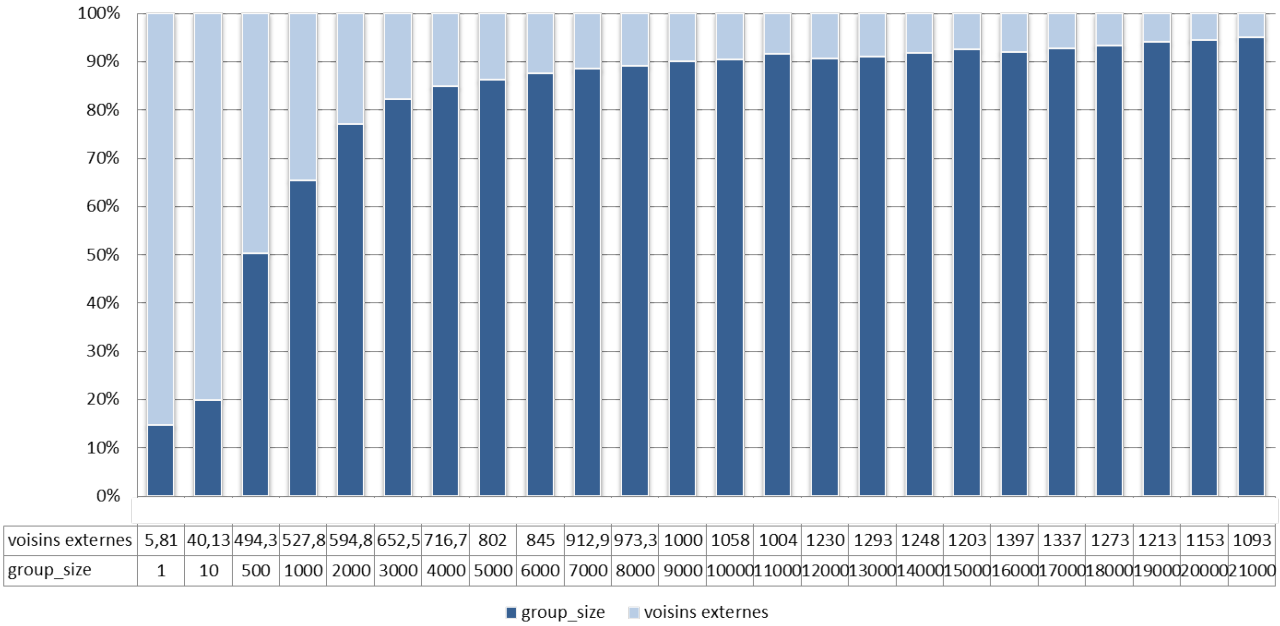


FIGURE 7 – Taux de défauts de cache dans les routines de copie



voisins externes	5,81	40,13	494,3	527,8	594,8	652,5	716,7	802	845	912,9	973,3	1000	1058	1004	1230	1293	1248	1203	1397	1337	1273	1213	1153	1093
group_size	1	10	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	15000	16000	17000	18000	19000	20000	21000

FIGURE 8 – Nombre de voisins externes par rapport à la taille du groupe

taille du groupe. L'axe des abscisse est gradué en échelle logarithmique.
Le cas test que nous avons utilisé pour obtenir cette courbe contient 46000 éléments. Le rapport de défaut de cache obtenu avec la version de référence est supérieur à celui obtenu avec la nouvelle version du code. Le premier point de la courbe (Figure 9) représente le rapport de défaut de cache global obtenu avec la version de référence. Il est presque 4 fois plus grand que celui obtenu avec la nouvelle version du code.
Nous avons constaté que lorsque nous augmentons la taille du groupe de 1 élément jusqu'à environ 1000 éléments par groupe (Zone A de la courbe), le rapport de défaut de cache diminue. Ces valeurs montrent l'amélioration de la localité spatiale et temporelle grâce à l'augmentation du nombre de données utiles contenues dans les blocs contigus. Au-delà de 1000 éléments par groupe (Zone B sur la courbe), nous constatons la présence d'un palier dans la courbe. En fait, une fois le cache rempli, l'utilisation de groupes de taille plus grande que la capacité du cache n'améliore plus les performances puisque nous avons atteint la limite de stockage du cache.

5. Conclusion et perspectives

Nous avons étudié l'organisation de la structure de données du code de simulation en dynamique rapide EUROPLEXUS. Afin d'optimiser l'utilisation du cache pour ce code, nous avons proposé une nouvelle approche d'organisation de sa structure de données. Notre méthode est basée sur la réorganisation des données des tableaux élémentaires sous forme de groupes. Le but de cette approche est d'augmenter la localité des données dans le cache. Nous avons vérifié expérimentalement l'apport de cette approche, dans le cas des problèmes de grande taille, pour diminuer le taux de défauts de cache dans la boucle des calculs élémentaires.

Dès à présent, les accès principaux à la mémoire centrale ont été ôtés des routines élémentaires appelées de manière répétitive, ce qui répond à l'objectif initial visant à préparer l'accélération

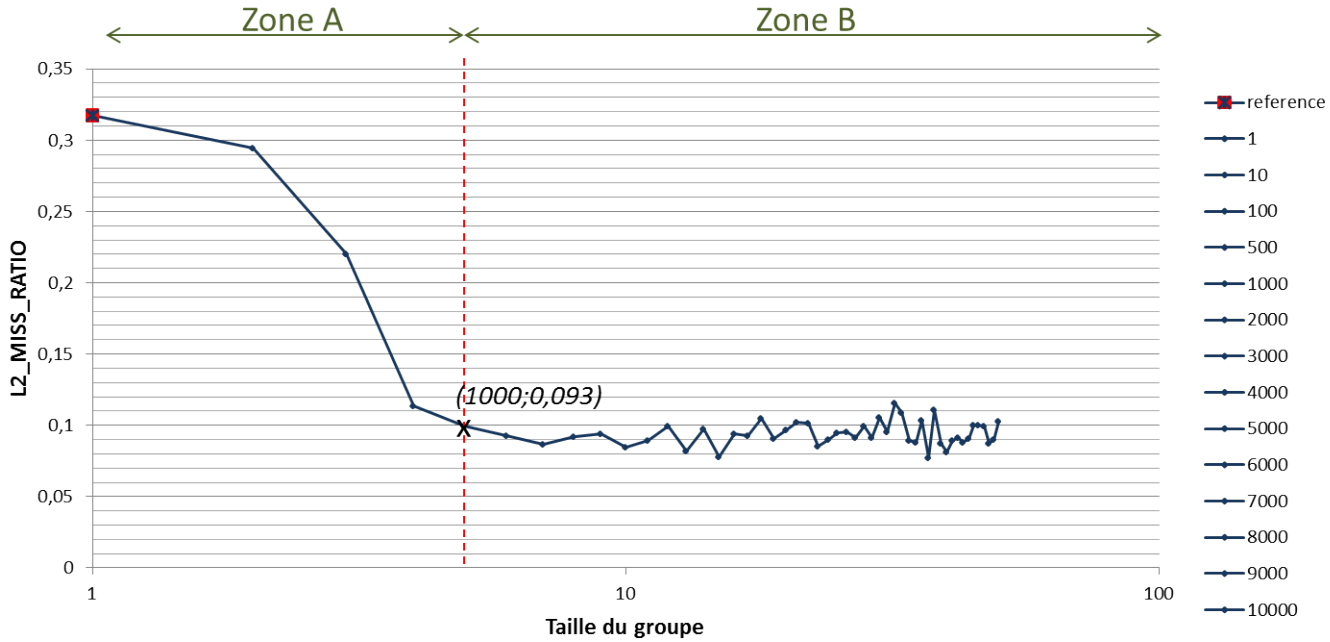


FIGURE 9 – Variation du rapport de défaut de cache en fonction de la taille du groupe

parallèle optimale de la boucle principale via la librairie KAAPI [6] (INRIA), implémentant un ordonnancement dynamique par vol de travail et bénéficiant pleinement de la localité des données sur lesquelles s'exécutent les threads.
Dans le but de baisser les coûts dus à la création des groupes dans la routine principale, nous envisageons de minimiser le nombre de voisins externes dans chaque groupe. Une des améliorations qui pourrait être faite dans ce cadre consiste à renuméroter les éléments. Cette renumérotation devrait garantir que les indices des voisins d'un élément soient proches de l'indice de l'élément courant.

Bibliographie

1. Bader (M.), Franz (R.), Gunther (S.) et Heinecke (A.). – Hardware-oriented implementation of cache oblivious matrix operations based on space-filling curves. *In : Parallel Processing and Applied Mathematics*, éd. par Wyrzykowski (R.), Dongarra (J.), Karczewski (K.) et Wasniewski (J.), pp. 628–638. – Springer Berlin Heidelberg, 2008.
2. Clauss (P.). – *Méthodes polyédriques pour la parallélisation et l'optimisation de programmes*. – Habilitation thesis, Universite Louis Pasteur, janvier 2000.
3. Cuong (P. Q.). – Résolution des systèmes linéaires, Juin 2003.
4. Dureisseix (D.) et Champaney (L.). – Calcul de structures et parallélisme : un bilan et quelques développements récents. *Mécanique & Industries*, vol. 1, n1, 2000, pp. 43 – 60.
5. http ://code.google.com/p/likwid/.
6. http ://kaapi.gforge.inria.fr/dokuwiki/doku.php.
7. http ://www cast3m.cea.fr/.
8. http ://www epx.cea.fr/.

9. Kandemir (M.), Choudhary (A.), Ramanujam (J.) et Banerjee (P.). – Optimizing spatial locality in loop nests using linear algebra. *In : PROC. 7TH INT. WORKSHOP ON COMPILERS FOR PARALLEL COMPUTERS*, p. 167.
10. Wolf (M. E.) et Lam (M. S.). – A data locality optimizing algorithm. Computer Systems Laboratory Stanford University, CA 94305, pp. 30–44.