Parallélisation de l’assemblage et de la résolution matricielle directe de H2D2 : Développement de critères de choix de configuration

Rapport de recherche R-1052 Mars 2009

PARALLÉLISATION DE L'ASSEMBLAGE ET DE LA RÉSOLUTION MATRICIELLE DIRECTE DE H2D2 : DÉVELOPPEMENT DE CRITÈRES DE CHOIX DE CONFIGURATION

par

Yves SECRETAN
Pascal MATTE

Institut national de la recherche scientifique, INRS-ETE
Québec (Québec), Canada
G1K 9A9

Rapport de recherche N° R-1052

Mars 2009

ISBN: 978-2-89146-595-3

Pour fins de citations :
Secretan, Y. et Matte, P. (2009). Parallélisation de l’assemblage et de la résolution matricielle directe de H2D2 : Développement de critères de choix de configuration. Rapport de recherche R-1052, INRS-ETE, 37 pp.

Table des matières

Liste des figures

1. Introduction

L'assemblage des matrices et des résidus ainsi que la résolution matricielle sont les parties de H2D2 qui consomment la plus grande partie du temps de calcul. Grâce à l'utilisation de librairies externes, H2D2 peut diminuer ses temps de calcul en tournant en mémoire partagée et/ou en mémoire distribuée sur plusieurs processus et/ou tâches (cf. Glossaire).

La résolution en mémoire partagée demande une répartition de la tâche de travail sur différentes tâches qui partagent la mémoire et donc les données. Comme il y a accès concurrent en mémoire, il faut gérer les écritures en mémoire pour éviter que deux tâches modifient simultanément la même place mémoire. Dans H2D2, cette répartition du travail entre les tâches est faite à l'aide des directives OpenMP, un standard pour ce type de besoins. L'utilisateur contrôle ce parallélisme par la variable d'environnement OMP_NUM_THREADS.

La répartition, quant à elle, vise à diviser le maillage en parties les plus égales possible en termes de nombre de nœuds, tout en minimisant la taille des frontières entre les parties, afin de minimiser les transferts d'information. Ce travail est fait à l'aide de la librairie MPI. La partie du maillage global associée à un processus doit être renumérotée pour que localement les numéros de nœuds soient contigus et sans trous. On profite de cette renumérotation pour optimiser celle-ci en vue de minimiser la largeur de bande ou de minimiser la proximité en mémoire des données et ainsi réduire les cache-miss. Tant la distribution du maillage global que la renumérotation font appel à des librairies externes (ParMetis ou Scotch). Au passage, on renumérote également les éléments afin de profiter de la renumérotation des nœuds. En distribuant le maillage, on se retrouve sur chaque processus avec des maillages plus petits. Il est donc pensable et possible que les renumérotations locales occasionnent des gains d'efficacité, les données étant plus proche en mémoire et occasionnant moins de cache-miss.

Dans H2D2, il a été choisi que chaque nœud soit complètement assemblé par un seul processus qui en est le « propriétaire ». Il s'ensuit que la frontière entre deux sous-domaines du maillage traverse les éléments de cette frontière en séparant les nœuds appartenant à un même élément dans des sous-domaines distincts. Afin d'assembler tous les nœuds d'un sous-domaine,

on a besoin d'information des nœuds voisins. Les nœuds aux frontières entre sous-domaines se trouvent donc partagés entre différents processus, bien qu'un seul en soit le propriétaire.

Différents tests ont été effectués afin d'évaluer l'efficacité de la parallélisation de l'assemblage dans H2D2 et les gains en temps de calcul associés au partage des tâches en mémoire et à la distribution des calculs sur plusieurs processus. Les tests ont tous été effectués pour des résolutions matricielles par méthodes directes, la méthode privilégiée actuellement pour H2D2. Ils visent le développement de critères de choix de configuration. Tous les temps présentés correspondent aux temps réels (wall time), à l'opposé des temps CPU qui réfèrent aux temps consommés par le processeur. C'est en effet le temps d'attente que l'on cherche à diminuer.

Les tests ont été effectués sur un cluster composé de deux ordinateurs, chacun muni d'un processeur double-cœur Pentium D de 2,4 GHz et de 1 Go de mémoire vive. Un lien Gigabit lie les machines entre elles.

Ce rapport présente une première section portant sur les tests de parallélisme effectués en mémoire partagée (OMP), suivie d'une section traitant des tests de parallélisme effectués en mémoire distribuée (MPI) et d'une section présentant des tests comparatifs pour différentes configurations OMP et MPI avec résolution complète. Une conclusion viendra clore le rapport en exposant certaines recommandations sur l'exploitation du parallélisme dans H2D2. Un glossaire et une bibliographie sont également présentés.

2. Tests de parallélisme en mémoire partagée (OMP)

La parallélisation des boucles d'assemblage du résidu et de la matrice permet leur exécution en mémoire partagée. Le travail d'assemblage se trouve ainsi réparti en un nombre de tâches spécifiées à l'aide des directives OpenMP. Cette section présente les résultats des tests de parallélisme effectués en mémoire partagée, sur maillages réguliers avec résolution minimale.

2.1. But

Ces tests ont pout but de :

- Contrôler que les résultats obtenus sont les mêmes pour tous les compilateurs;
- Comparer l'efficacité des compilateurs;
- Comparer des configurations de parallélisme;
- Déterminer l'efficacité de la renumérotation.

2.2. Protocole d'analyse

Les tests ont été effectués avec la formulation de l'élément de Saint-Venant 2D (SV2D_Conservatif_CDYS_NN) de H2D2. Une résolution par GMRES avec une matrice de préconditionnement Identité et un schéma de résolution composé de 3 préconditionnements, 7 redémarrages et 10 itérations a été utilisée afin de minimiser la partie liée à la résolution, comme nous cherchons à déterminer ici la qualité de la parallélisation des boucles d'assemblage.

La résolution par des méthodes internes (GMRES) est parallélisée en s'appuyant sur les routines BLAS optimisée, alors que la résolution par des méthodes externes s'appuie sur des méthodes de résolution matricielle optimisées (SuperLU, Pardiso).

Les trois compilateurs suivants ont été testés :

- gcc version 4.2
- intel 10.1

- intel 11.0

Les trois configurations de parallélisme suivantes ont été testées, sans renumérotation, et donc avec leur numérotation naturelle, qui cependant est optimale :

- noatomic - nocolor
- color
- atomic

Avec le compilateur intel 11.0 et une configuration de parallélisme *color*, des calculs avec et sans renumérotation des nœuds et des éléments ont été réalisés.

Les maillages suivants, de taille croissante, ont été utilisés pour chacune des configurations de parallélisme et pour les tests de renumérotation :

- Quadrillage de 32x32 : soit 2 048 éléments, 4 225 nœuds
- Quadrillage de 64x64 : soit 8 192 éléments, 16 641 nœuds
- Quadrillage de 128x128 : soit 32 768 éléments, 66 049 nœuds
- Quadrillage de 256x256 : soit 131 072 éléments, 263 169 nœuds
- Quadrillage de 256x512 : soit 262 144 éléments, 525 825 nœuds

2.3. Directives OpenMP

Dans le but de paralléliser les boucles d'assemblage (de la matrice ou du résidu), on utilise un certain nombre de directives OpenMP (Open Multi-Processing). OpenMP est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Cette API est supportée sur de nombreuses plateformes, incluant Unix et Windows, pour les langages de programmation C/C++ et Fortran. Il se présente sous la forme d'un ensemble de directives, d'une bibliothèque logicielle et de variables d'environnement (Wikipedia). OpenMP est portable et dimensionnable. Il permet de développer rapidement des applications parallèles à petite granularité en restant proche du code séquentiel. On peut facilement contrôler le nombre de tâches utilisés par openMP en assignant le nombre voulu à la variable d'environnement OMP_NUM_THREADS.

2.4. Les configurations de parallélisme

L'assemblage est une boucle sur les éléments qui somme la contribution des éléments dans un vecteur ou dans une matrice globale. Si la partie qui concerne le calcul élémentaire est facilement parallélisable, la sommation et l'écriture dans la table globale doivent être faites avec soin afin d'éviter les conflits où deux tâches veulent en même temps modifier la même donnée. Il existe deux façons d'éviter ces conflits:

1. ***color***: en renumérotant les éléments en « couleurs », c'est-à-dire en traitant en parallèle des groupes d'éléments qui sont garantis disjoints;
2. ***atomic***: en restreignant l'accès à l'écriture à une seule tâche à la fois, ce qu'on appelle une opération « atomic ».

2.4.1. Coloriage du graphe des éléments

La première solution consiste à colorier le graphe des éléments. Les sommets du graphe représentent les éléments et deux éléments sont reliés s'ils partagent au moins un nœud. En coloriant ce graphe, on s'assure que les éléments d'une même couleur sont tous indépendants. Par la définition du coloriage, deux éléments partageant un même nœud, donc liés par une arrête, ne peuvent être de la même couleur. Ainsi, on peut traiter tous les éléments d'une couleur de manière indépendante. Il suffit donc de faire une boucle sur les couleurs et de paralléliser la boucle sur les éléments d'une couleur (la boucle interne).

Cette solution a le désavantage d'être plus compliquée à implémenter et de modifier la renumérotation courante. En effet, si on a numéroté les nœuds et les éléments de façon à minimiser les cache-miss, on ne respecte plus cet ordre en traitant les éléments par couleur. Par ailleurs, de par la nature du coloriage, on est garanti d'accéder à chaque nœud au maximum une fois par couleur traité, réduisant ainsi le nombre de cache-hit.

2.4.2. Ajout de la directive *omp atomic*

La seconde solution est d'ajouter la directive *omp atomic* lors de l'actualisation de la matrice ou du vecteur global. L'**atomicité** est une propriété utilisée en programmation concurrente pour désigner une opération ou un ensemble d'opérations d'un programme qui s'exécute entièrement sans que le processus ou le thread qui la gère cède la place à un autre processus pendant tout le déroulement. Une opération qui vérifie cette propriété est qualifiée d'**atomique** (Wikipedia). La directive *omp atomic* assure qu'un seul thread à la fois peut faire une assignation à la matrice ou au vecteur global et garantie donc l'intégrité du résultat. En principe, ces opérations peuvent profiter d'instructions spéciales du processeur. Comme on le verra dans les résultats, le coût est trop élevé pour que la solution puisse être retenue.

2.5. Résultats

2.5.1. Comparaison des compilateurs

Les Figure 1, Figure 2 et Figure 3 présentent, pour différentes tailles de maillage, les temps de calcul obtenus en fonction du nombre de tâches OMP, pour les configurations *noatomic-nocolor*, *color* et *atomic*, respectivement. À noter que l'échelle de temps de la Figure 3 (*atomic*) diffère des deux autres configurations par un facteur 2. Tous les résultats obtenus ont été contrôlés de façon à s'assurer qu'on obtienne la même solution indépendamment du compilateur utilisé; ils se sont avérés identiques entre eux.

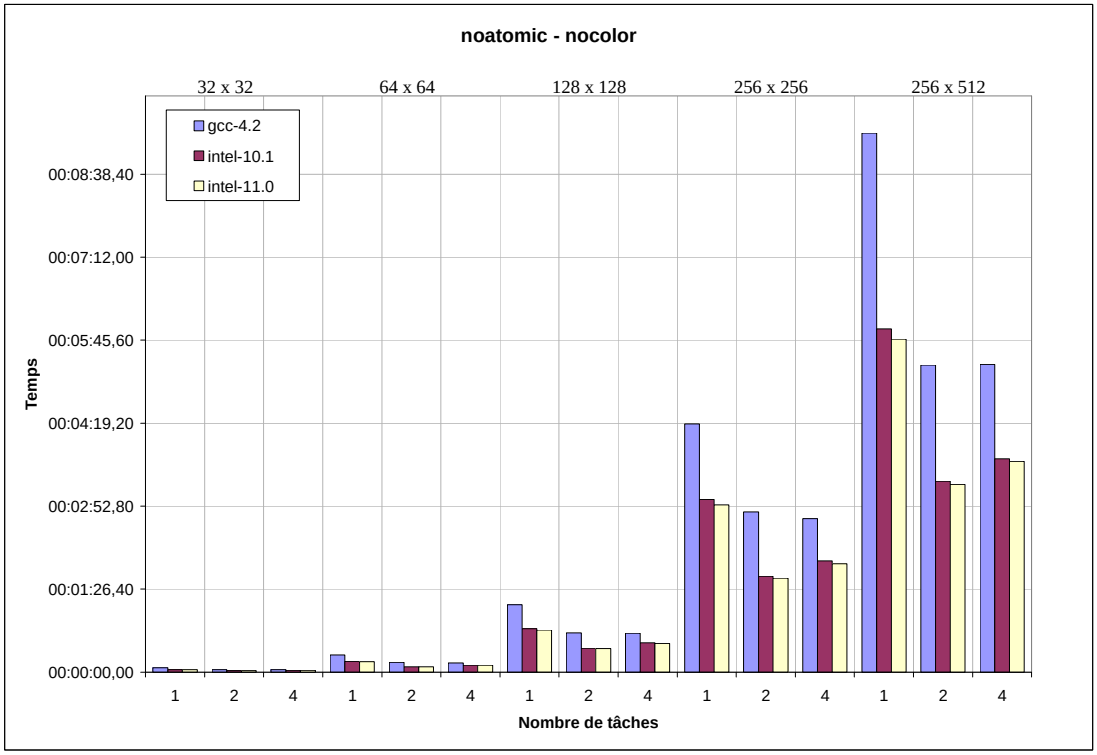

Figure 1 – Temps de calcul en fonction du nombre de tâches pour différentes tailles de maillage, configuration *noatomic - nocolor*

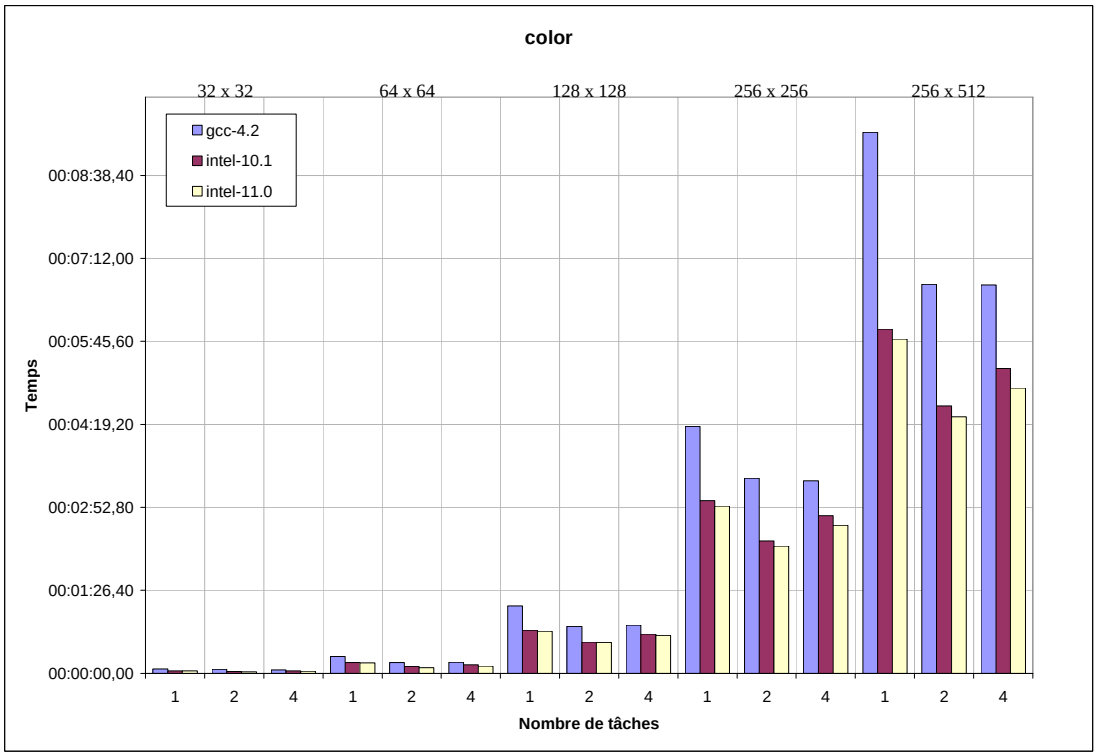

Figure 2 – Temps de calcul en fonction du nombre de tâches pour différentes tailles de maillage, configuration *color*

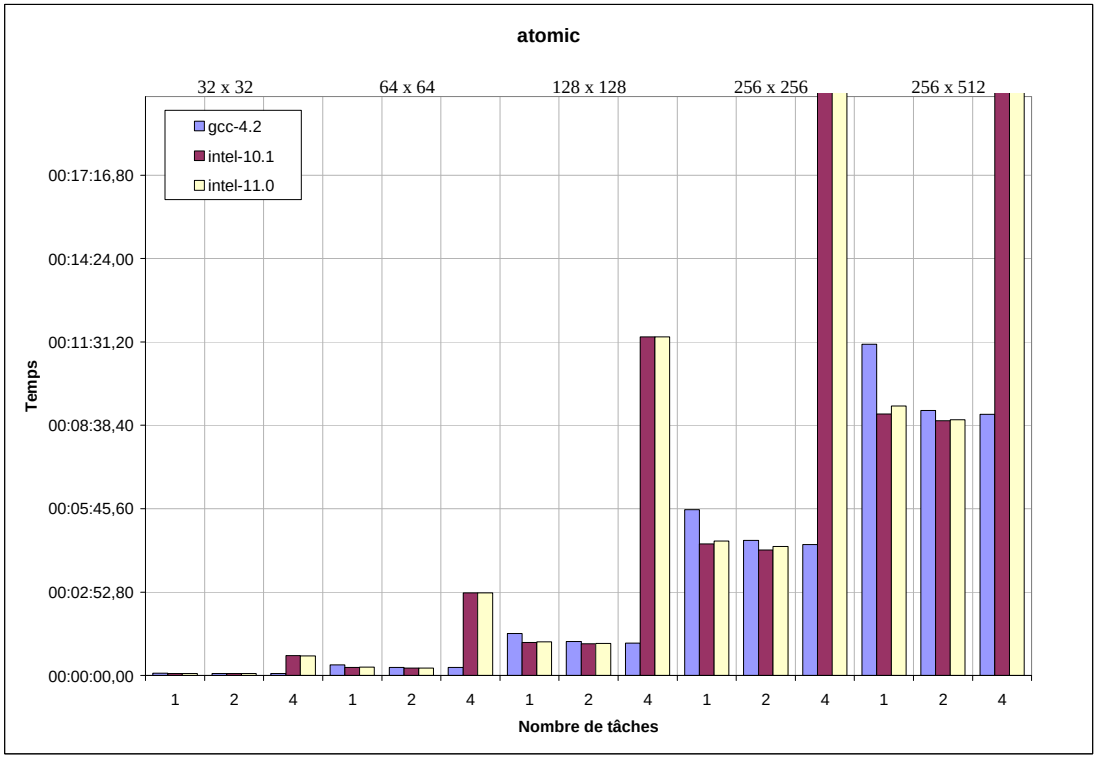

Figure 3 – Temps de calcul en fonction du nombre de tâches pour différentes tailles de maillage, configuration *atomic*

Les compilateurs intel (10.1 et 11.0) donnent systématiquement des résultats meilleurs que le compilateur gcc-4.2 avec un léger avantage avec la version 11.0. La configuration *noatomic - nocolor*, qui n'est pas valide pour le calcul parce qu'elle donne des résultats erronés, permet de déterminer le gain maximum atteignable à l'aide de la parallélisation OMP et de contrôler ainsi si le travail est fait adéquatement. Ainsi, on remarque que la configuration *color* donne des résultats comparables à la configuration *noatomic - nocolor*. Alors qu'avec une seule tâche OMP les temps sont identiques à ceux de la configuration *noatomic - nocolor*, les plus grandes différences apparaissent pour le maillage 256x512 avec 2 et 4 tâches. Avec la directive *omp atomic*, tous les compilateurs ajoutent un overhead important, qui correspond à plus du double des temps des autres configurations. La configuration *atomic* est donc à rejeter.

2.5.2. Comparaison des configurations

Les résultats qui suivent ont été obtenus avec le compilateur intel 11.0, jugé optimal (voir section 2.5.1). Ils présentent les speed-up pour les deux configurations *color* (Figure 5) et *atomic*

(Figure 6), avec comme référence la configuration *noatomic - nocolor* (Figure 4), qui permet de déterminer le gain maximum atteignable à l'aide de la parallélisation OMP. Les speed-up sont calculés par rapport aux temps de calcul à une tâche. Les résultats pour 4 tâches sont présentés, bien que chacun des ordinateurs composant le cluster ne possède qu'un seul processeur double-cœur. Tel que mentionné précédemment, tous les résultats obtenus ont été contrôlés de façon à s'assurer qu'on obtienne la même solution indépendamment de la configuration testée, pour un maillage donné.

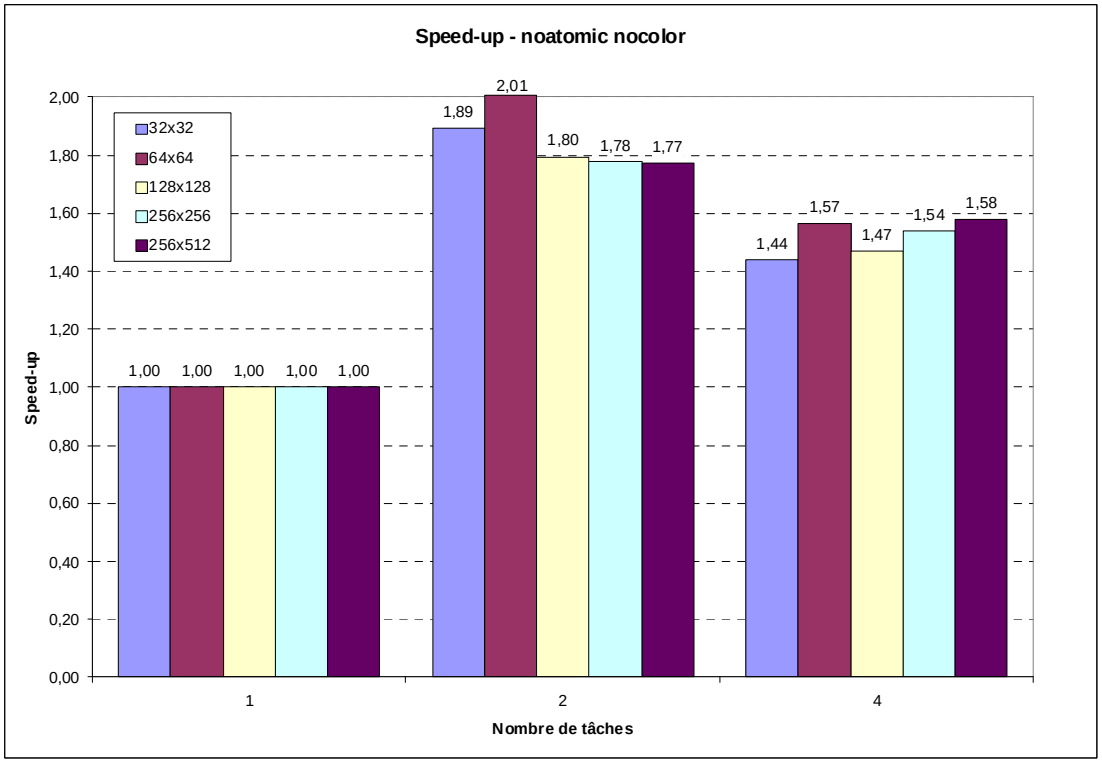

Figure 4 – Speed-up en fonction du nombre de tâches pour différents maillages avec la configuration *noatomic - nocolor*

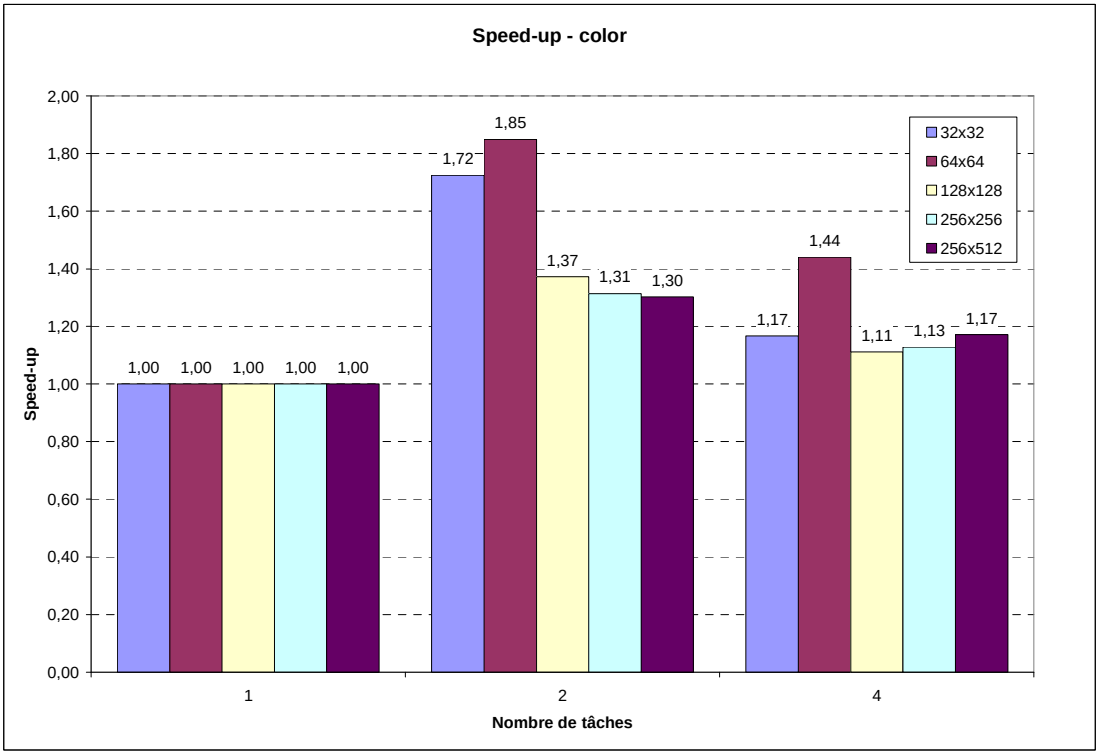

Figure 5 – Speed-up en fonction du nombre de tâches pour différents maillages avec la configuration *color*

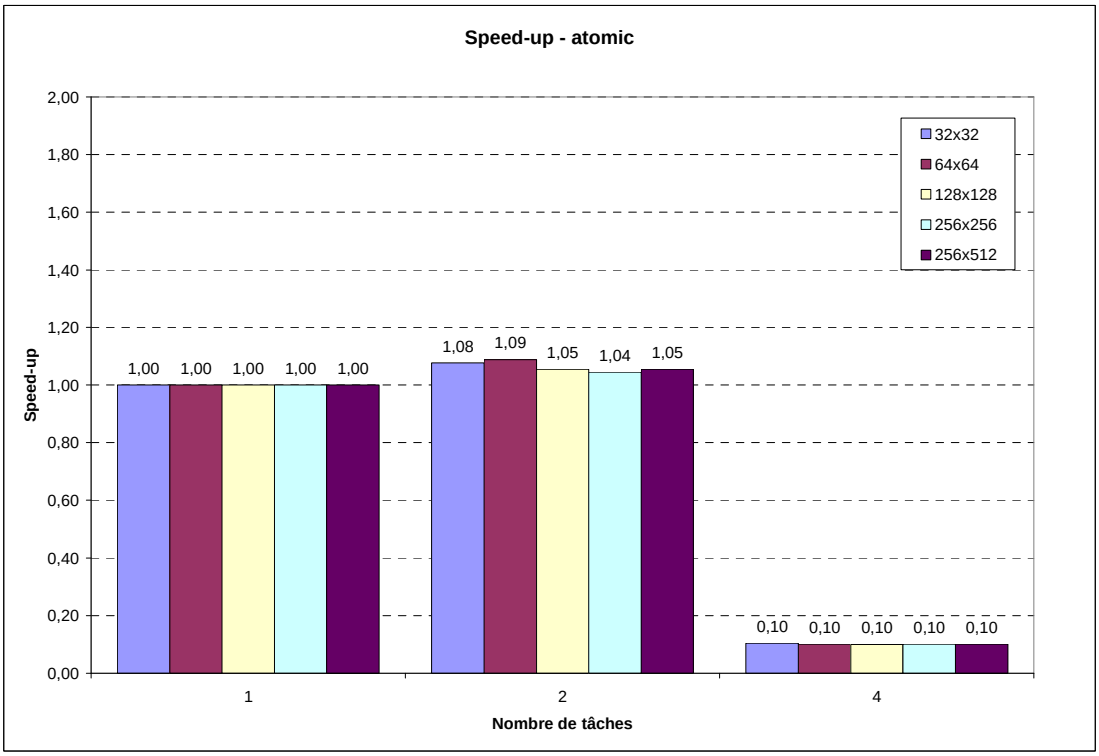

Figure 6 – Speed-up en fonction du nombre de tâches pour différents maillages avec la configuration *atomic*

Il est intéressant de noter qu'un speed-up maximum de 1,85 et un speed-up moyen de 1,5 sont atteints avec deux tâches pour la configuration *color*. L'optimum de 2 tâches est atteint avec un maillage de 64x64, ce qui est un maillage relativement petit. Les speed-up tendent à diminuer à mesure que le maillage augmente en taille, une fois l'optimum dépassé. De plus, même avec 4 tâches, on a encore un gain avec un speed-up moyen de 1,2 pour la configuration *color*. Les speed-up pour la configuration *atomic*, quant à eux, n'atteignent jamais la barre des 1,1 avec deux tâches.

La Figure 7 présente, pour le compilateur intel 11.0 et un maillage de 256x256, une comparaison des temps de calcul obtenus en fonction du nombre de tâches pour les trois configurations de parallélisme. Les résultats pour un seul maillage sont présentés, le choix du maillage étant arbitraire.

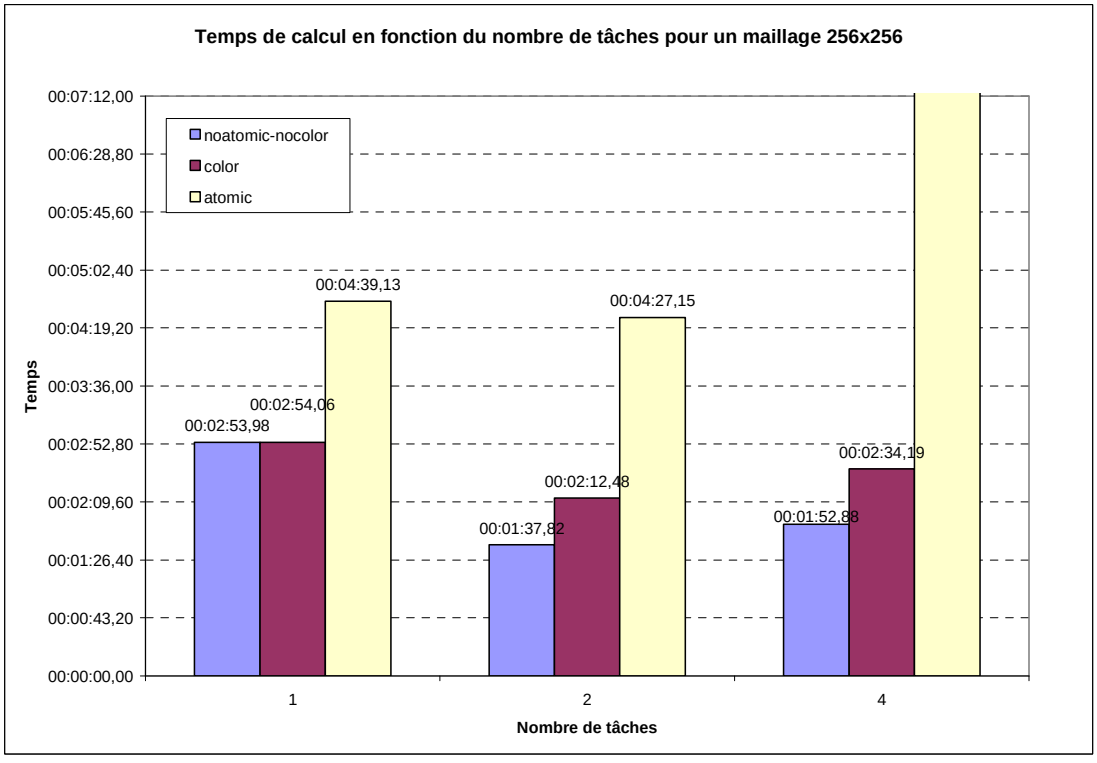

Figure 7 – Comparaison des temps de calcul en fonction du nombre de tâches pour les trois configurations de parallélisme avec un maillage de 256x256 et le compilateur Intel 11.0

Globalement l'option *atomic* coûte 50% à 60% plus cher que la référence (*noatomic - nocolor*) avec une seule tâche, ce qui n'est pas acceptable. Bien que détruisant la numérotation et donc l'alignement mémoire, l'option *color* montre une légère accélération, avec un speed-up

moyen de l'ordre de 1,5 pour 2 tâches. C'est l'option qui est implantée dans H2D2 et activée par la variable d'environnement OMP_NUM_THREADS.

2.5.3. Impact de la numérotation

Les Figure 8 et Figure 9 présentent les speed-up obtenus avec la configuration de parallélisme *color* sans renumérotation et avec renumérotation, respectivement, pour différentes tailles de maillage.

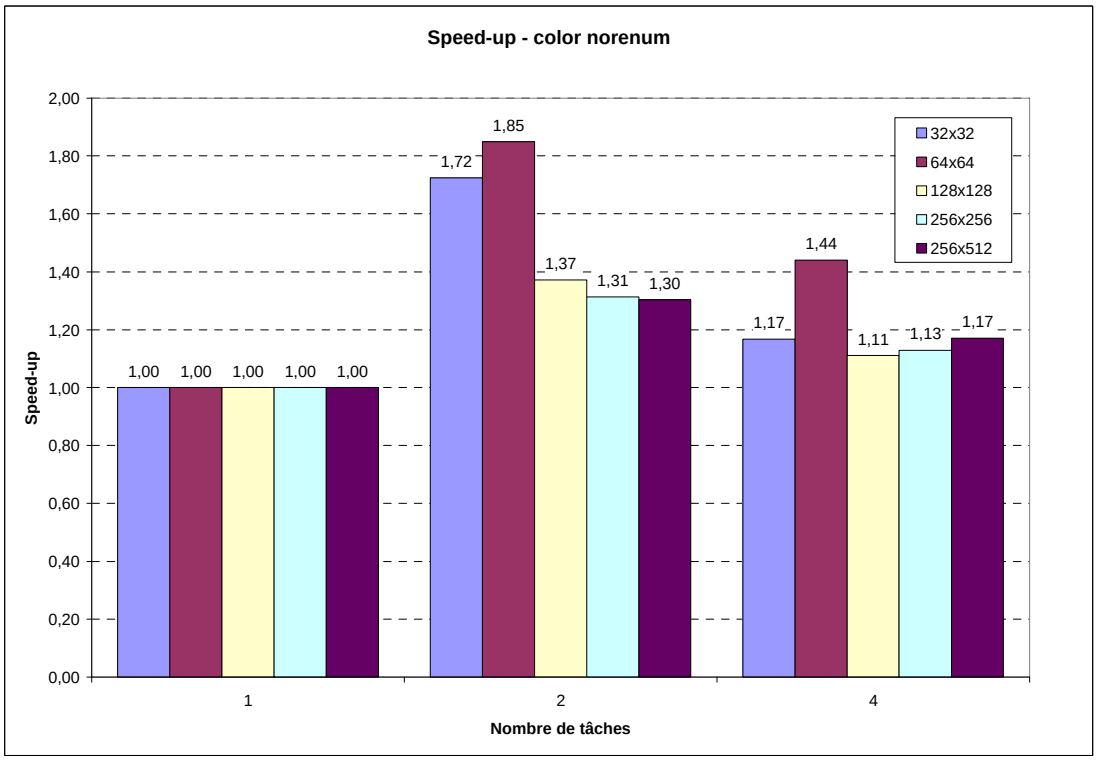

Figure 8 – Speed-up en fonction du nombre de tâches pour la configuration color sans renumérotation des nœuds et des éléments

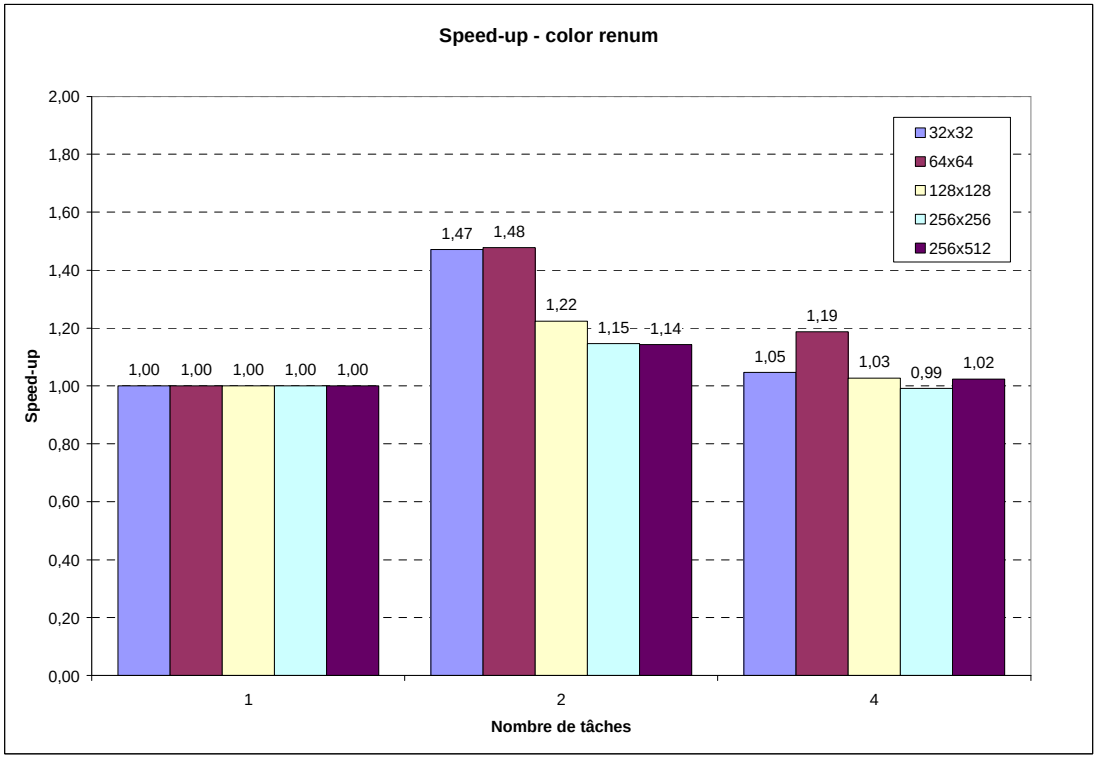

Figure 9 – Speed-up en fonction du nombre de tâches pour la configuration color avec renumérotation des nœuds et des éléments

Il ressort des Figure 8 et Figure 9 que l'ensemble « renumérotation des nœuds, renumérotation des éléments, coloriage des éléments » a un impact majeur sur les temps de calcul. La bonne performance de la version *norenum* montre qu'il doit être possible d'optimiser globalement la renumérotation, de la voir comme une seule étape qui optimise l'accès aux données plutôt que trois étapes distinctes. Dans sa configuration actuelle, H2D2 ne peut pas optimiser l'assemblage une fois les éléments colorié. Seuls des tests peuvent permettre, au cas par cas, de trouver la combinaison optimale.

2.6. Conclusion

Les tests de parallélisme en mémoire partagée ont fait ressortir les points suivants :

- Le compilateur intel 11.0 et la configuration de parallélisme *color* sont à privilégier;
- Un speed-up moyen de 1,5 est atteint avec 2 tâches pour la configuration *color* sur différents maillages réguliers;

- L’option *atomic* coûte 50% à 60% plus cher que la référence (*noatomic* - *nocolor*) avec une seule tâche, ce qui n’est pas acceptable;
- Les résultats sont les mêmes pour chaque compilateur et configuration de parallélisme testés;
- La renumérotation dans le cas de maillages très réguliers n’apporte rien, au contraire elle dégrade le comportement; ce test montre la sensibilité de l’assemblage à la perturbation de la numérotation induite par le coloriage.

3. Tests de parallélisme en mémoire distribuée (MPI)

La distribution des calculs sur plusieurs processus permet de subdiviser la charge de travail en de plus petits problèmes de tailles semblables. Une résolution en parallèle du problème est ainsi réalisée et le partage de l'information doit être assuré à la frontière des sous-domaines de calcul. Cette section présente les résultats des tests de parallélisme effectués en mémoire distribuée, à l'aide de la librairie MPI, appliquant les configurations optimales à une résolution typique sur un domaine réel de calcul, soit celui du fleuve Saint-Laurent.

3.1. But

Ces tests ont pour but de :

- Déterminer les speed-up pour différentes configurations MPI;
- Déterminer les temps utilisés par MPI (temps CPU versus Wall time) et consacrés aux échanges réseau (cluster versus local);
- Établir des seuils d'efficacité entre les gains en temps de résolution versus les pertes en temps d'échange d'information entre processus.

3.2. Protocole d'analyse

Les tests ont été effectués avec la formulation de l'élément de Saint-Venant 2D (SV2D_Conservatif_CDYS_NN) de H2D2 appliquée au fleuve Saint-Laurent dans le secteur Grondines – Île-aux-Grues. Le maillage d'analyse est présenté à la Figure 10. Il est composé de 95547 nœuds et de 46174 éléments. Le scénario d'analyse correspond à un événement de faible débit et de grande marée. La résolution est non-stationnaire et s'étend sur 2 heures. Un schéma temporel implicite d'Euler avec des pas de temps de 5 minutes est utilisé. Une résolution par Newton et le solveur distribué MUMPS est réalisée, avec une part de résolution réelle afin de déterminer les vrais speed-up associés autant à la résolution qu'à l'assemblage. Le compilateur intel 11.0 et la configuration *color* ont été utilisés.

Afin d'obtenir des temps de calcul comparables, il faut s'assurer que la charge de calcul soit constante (même nombre d'itérations) d'une configuration à l'autre. Pour ce faire, la norme sur les incréments de solution a été fixée à une valeur très basse (10^{-20}) et le nombre d'itérations limité à 8 pour chaque pas de temps.

Dans tous les cas, le nombre de tâches OMP a été fixé à 1. Les configurations suivantes ont été testées avec le solveur MUMPS:

- MPI=1, OMP=1 (ou 1x1)
- MPI=2, OMP=1 (ou 2x1)
- MPI=4, OMP=1 (ou 4x1)
- MPI=6, OMP=1 (ou 6x1)
- MPI=8, OMP=1 (ou 8x1)
- MPI=10, OMP=1 (ou 10x1)
- MPI=12, OMP=1 (ou 12x1)

Des tests supplémentaires ont été réalisés sur un seul ordinateur (processeur double-cœur) avec les mêmes configurations (1x1 à 12x1), afin de quantifier l'impact sur les temps de calcul de l'échange des données entre deux ordinateurs d'un même cluster.

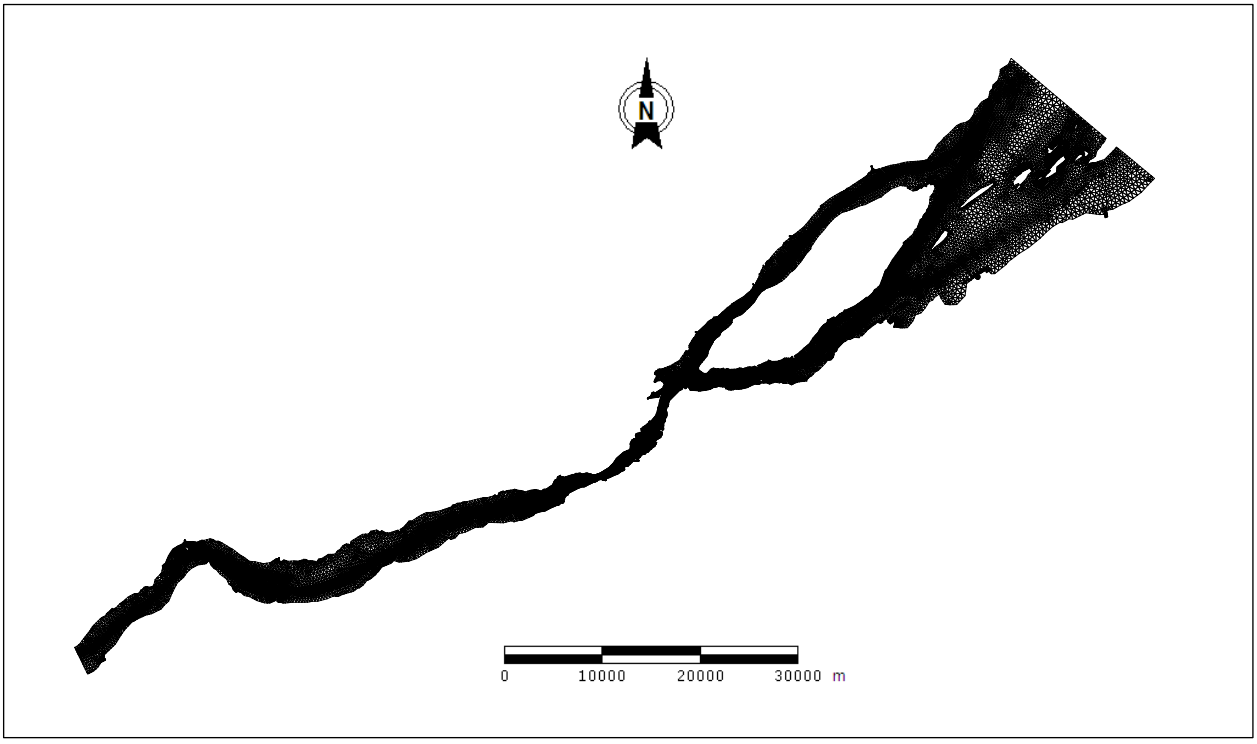

Figure 10 – Maillage d'analyse sur le tronçon Grondines – Île-aux-Grues, composé de 95 547 nœuds et de 46 174 éléments.

3.3. Solveur parallèle MUMPS

MUMPS est un solveur massivement parallèle. Il utilise MPI pour résoudre un système matriciel sur une grappe d'ordinateurs. Il est également possible de l'utiliser séquentiellement. MUMPS est donc très utile puisqu'il permet de résoudre des problèmes de taille considérable autrement impossible à résoudre sur un seul ordinateur (trop de mémoire requise, temps de calcul trop élevés). La documentation de MUMPS est disponible à l'adresse suivante : http://mumps.enseeiht.fr/index.php?page=doc .

MUMPS a été implémenté comme un module de H2D2 et fonctionne, du côté de l'interface, comme les autres solveurs (*mkl_pardiso*, *superlu_sp*, *ldu_memory*, etc.). Le nom de classe dans l'interface H2D2 est *mumps_linear_solver*.

3.4. Partitionnement du maillage

Les sous-domaines de partitionnement du maillage sont divisés de façon à répartir uniformément la charge de calcul et à partager un minimum de nœuds à la frontière des sous-domaines. Chaque sous-domaine possède sensiblement le même nombre de nœuds. Le partitionnement du maillage se fait avec Scotch ou ParMETIS en autant de domaines qu'il y a de processus. Le partitionnement du maillage s'effectue en trois étapes dont deux sont liées :

- le partitionnement et la renumérotation des nœuds;
- la renumérotation des éléments.

3.4.1. Renumérotation des nœuds

Il est possible dans H2D2 de renuméroter les nœuds dans le but de limiter les cache-miss lors de l'assemblage de la matrice globale (ou du résidu) et ainsi d'optimiser les temps de résolution d'un problème. De plus, lors de calculs répartis sur plusieurs processus, une renumérotation locale des nœuds du maillage est nécessaire sur chaque sous-domaine de calcul. Trois modules sont disponibles pour effectuer ce travail : BGL (Boost Graph Library), ParMetis et Scotch. La performance et l'efficacité de chacun de ces modules peut varier selon le contexte (taille de maillage, type de formulation, numérotation existante, etc.).

De façon générale, le module BGL est à éviter dès que l'on a affaire à de grands maillages. En effet, pour des maillages de plus de 250 000 nœuds, le temps de renumérotation dépasse rapidement le temps de résolution. De plus, Scotch donne des résultats d'aussi bonne qualité en une fraction du temps qu'aurait pris BGL. ParMetis est également un choix acceptable, bien que moins efficace que Scotch (et légèrement plus lent). Dans le cas de Scotch et de ParMetis, le temps de renumérotation est négligeable par rapport au temps de résolution; il est donc conseillé de renuméroter. Plus la formulation utilisée est légère en calculs, plus le choix de la numérotation est important. De plus, si on utilise le même maillage pour plusieurs résolutions, il devient de plus en plus avantageux de renuméroter, comme on n'a qu'à le faire une fois et sauvegarder la renumérotation.

3.4.2. Renumérotation des éléments

La renumérotation frontale des éléments permet de tirer avantage de la renumérotation des nœuds et assure une répartition des numéros d'élément à travers le maillage dans le sens de progression de la numérotation des nœuds.

3.5. Résultats

3.5.1. Partitionnement du maillage

La Figure 11 présente une division du maillage en sous-domaines de calcul, effectuée à l'aide du module Scotch.

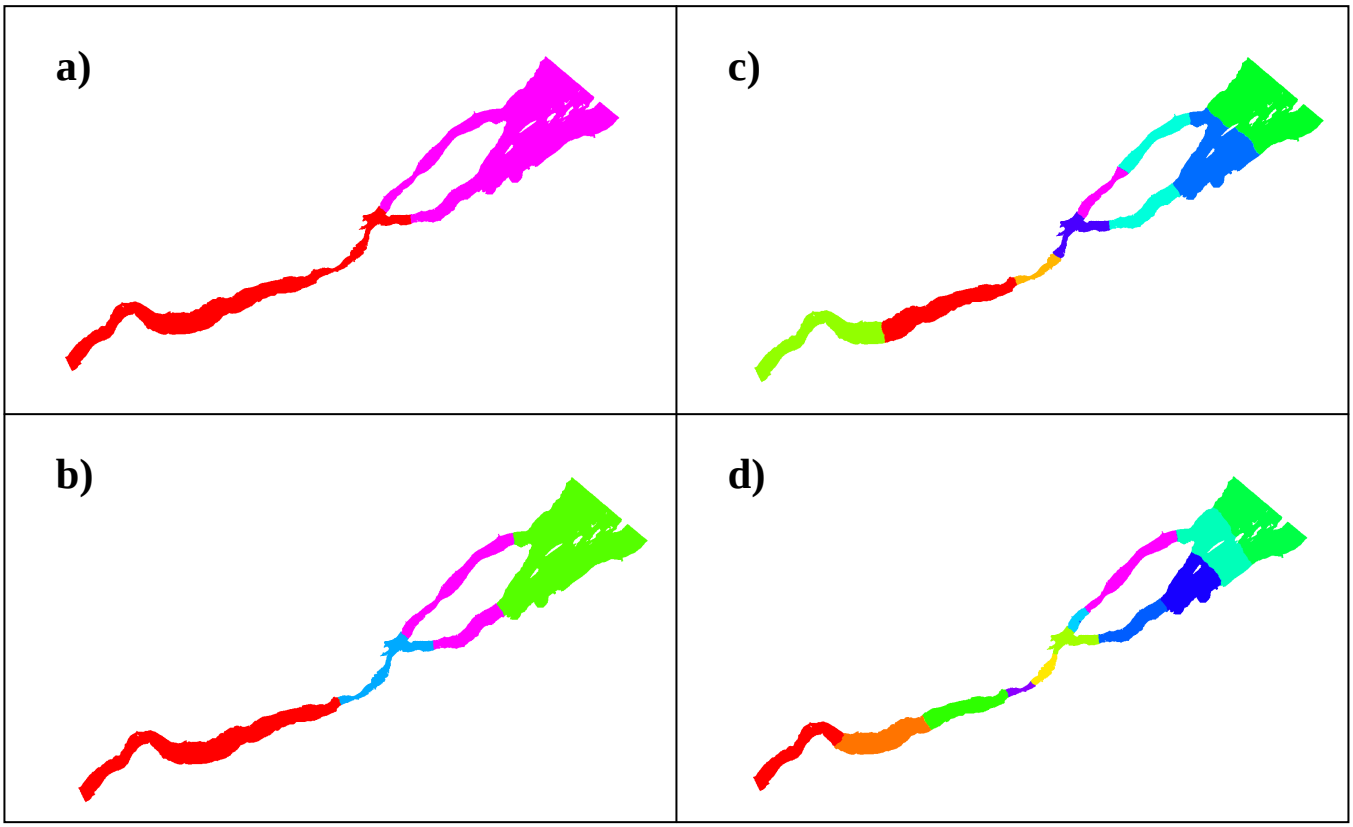

Figure 11 – Exemples de division du maillage en sous-domaines de calcul avec le module Scotch, chaque couleur correspondant à un nœud de calcul : a) 2 nœuds; b) 4 nœuds; c) 8 nœuds; d) 12 nœuds.

3.5.2. Temps de calcul

La Figure 12 présente, pour chaque configuration MPI testée, les temps relatifs consacrés à chacune des tâches principales de la simulation : la résolution (incluant l'assemblage; h2d2.reso), les échanges d'information (h2d2.io) et la construction du maillage partitionné (h2d2.grid). On remarque que plus le nombre de nœuds de calcul est grand, plus les temps associés à la portion h2d2.io augmentent, en raison des entrées sorties qui demandent beaucoup d'échanges réseau lorsque le nombre de processus se multiplie. De plus, les temps liés à la construction du maillage partitionné sont minimes en comparaison avec les temps totaux.

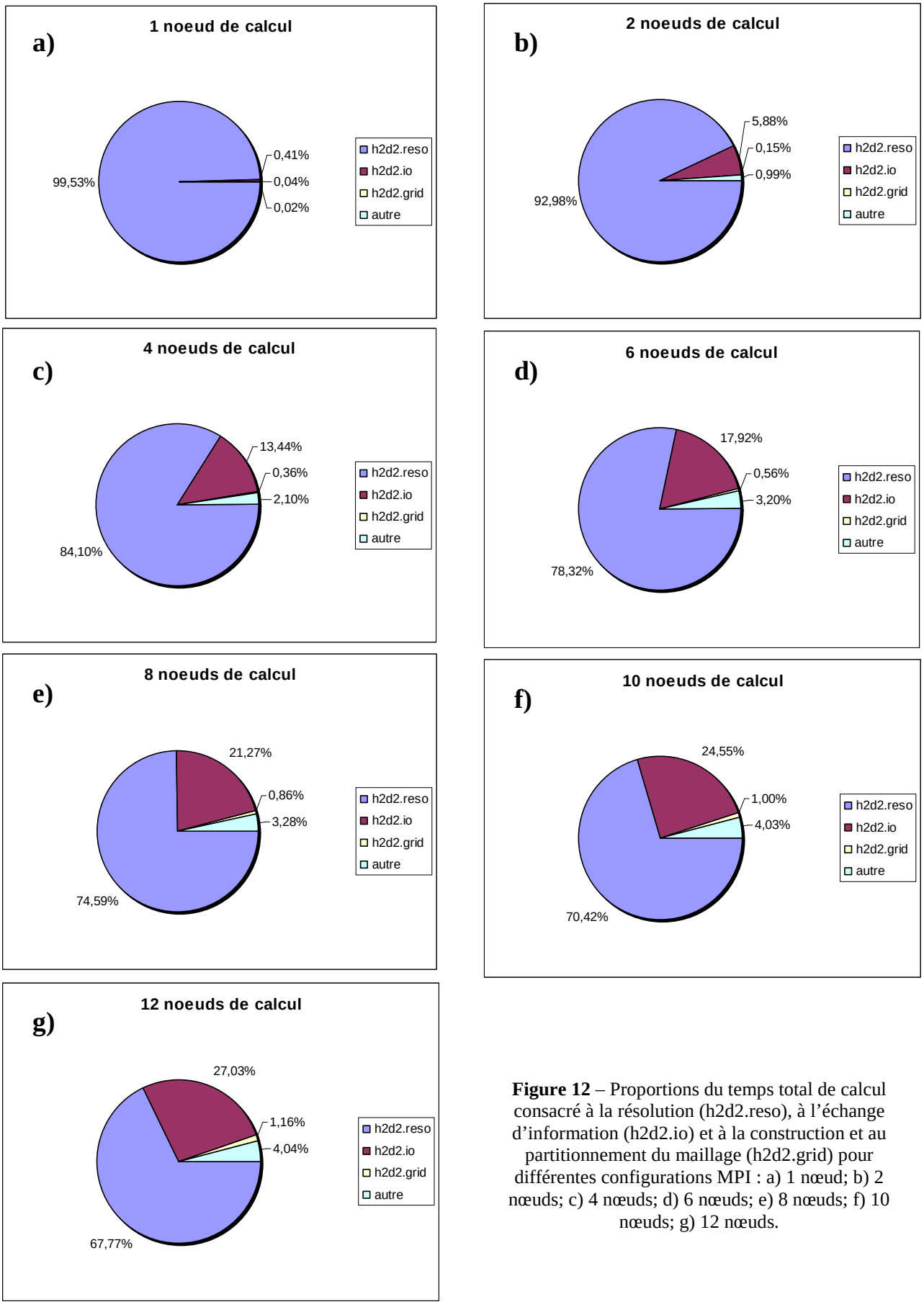

Figure 12 – Proportions du temps total de calcul consacré à la résolution (h2d2.reso), à l'échange d'information (h2d2.io) et à la construction et au partitionnement du maillage (h2d2.grid) pour différentes configurations MPI : a) 1 nœud; b) 2 nœuds; c) 4 nœuds; d) 6 nœuds; e) 8 nœuds; f) 10 nœuds; g) 12 nœuds.

Les Figure 13 et Figure 14 présentent, sur le cluster et l'ordinateur local, respectivement, les temps de calcul totaux et les temps consacrés à chacune des étapes du calcul, pour chaque configuration MPI testée. La Figure 15 présente, pour différentes configurations MPI, une comparaison des temps calcul CPU et Wall time. La Figure 16 présente, quant à elle, les temps de résolution totaux et les temps consacrés à la résolution et à l'assemblage du résidu et de la matrice. Finalement, la Figure 17 présente les temps totaux de création du maillage partitionné et les temps consacrés à la renumérotation et au partitionnement du maillage.

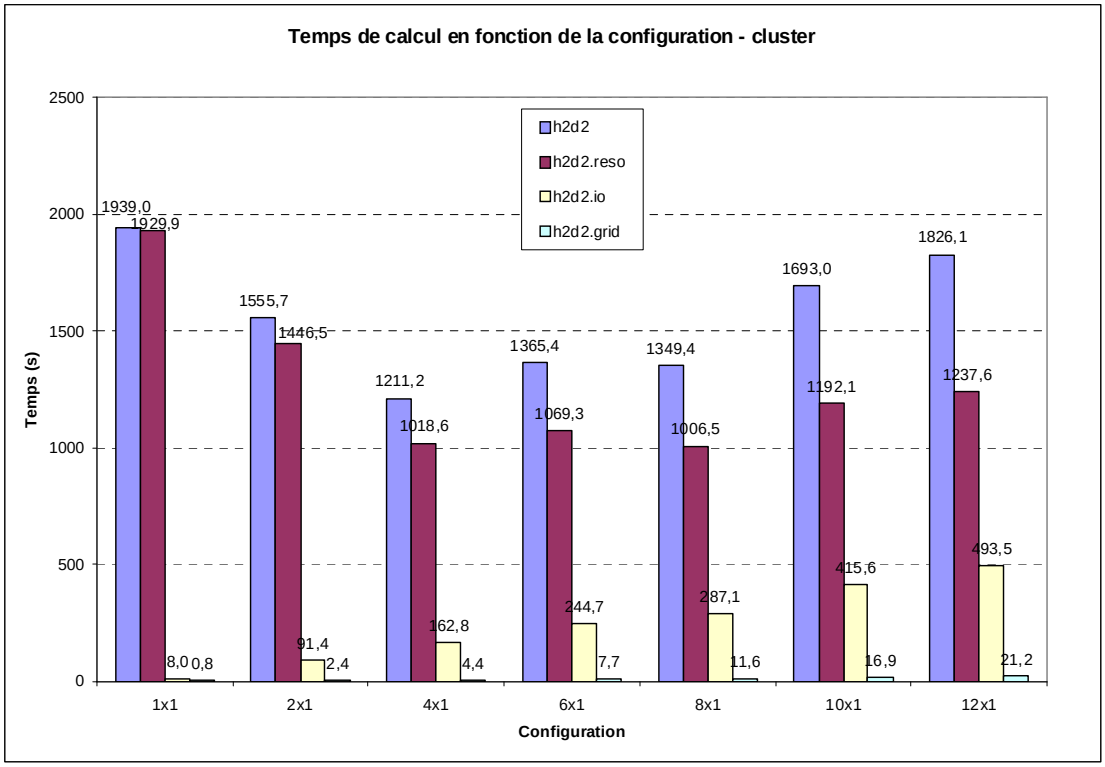

Figure 13 – Sur le cluster, temps de calcul totaux (h2d2) et consacrés à la résolution (h2d2.reso), à l'échange d'information (h2d2.io) et à la construction et au partitionnement du maillage (h2d2.grid) pour différentes configurations MPI.

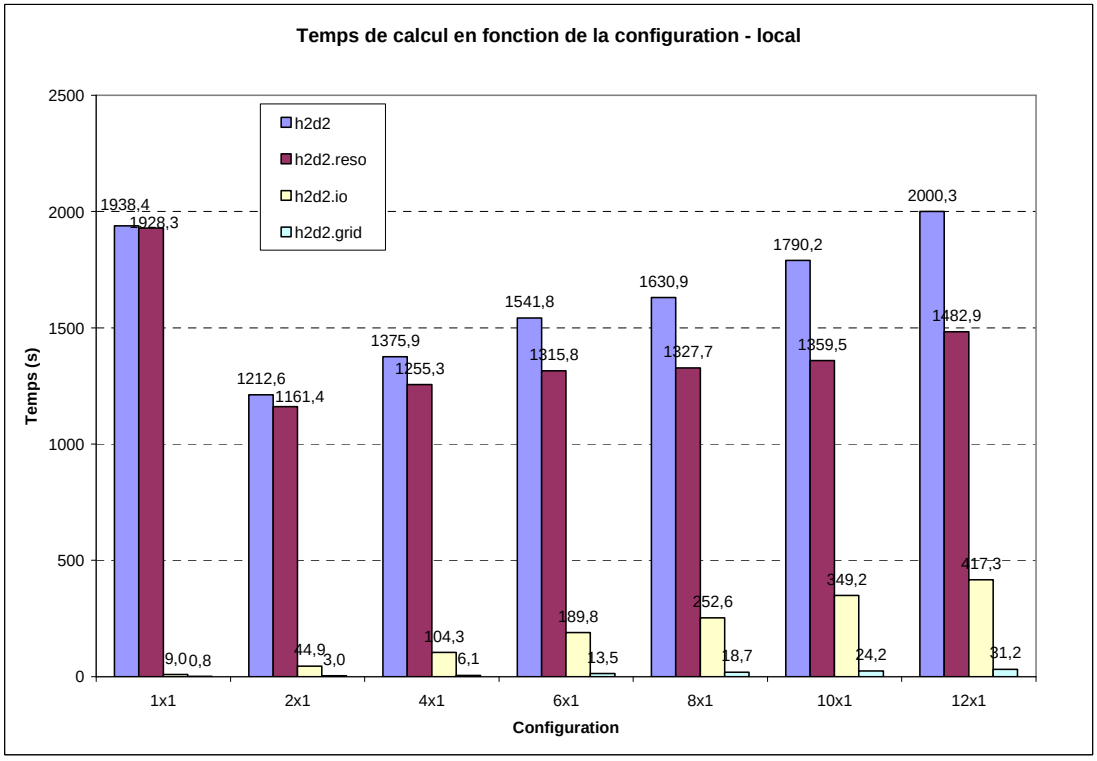

Figure 14 – Sur l'ordinateur local, temps de calcul totaux (h2d2) et consacrés à la résolution (h2d2.reso), à l'échange d'information (h2d2.io) et à la construction et au partitionnement du maillage (h2d2.grid) pour différentes configurations MPI.

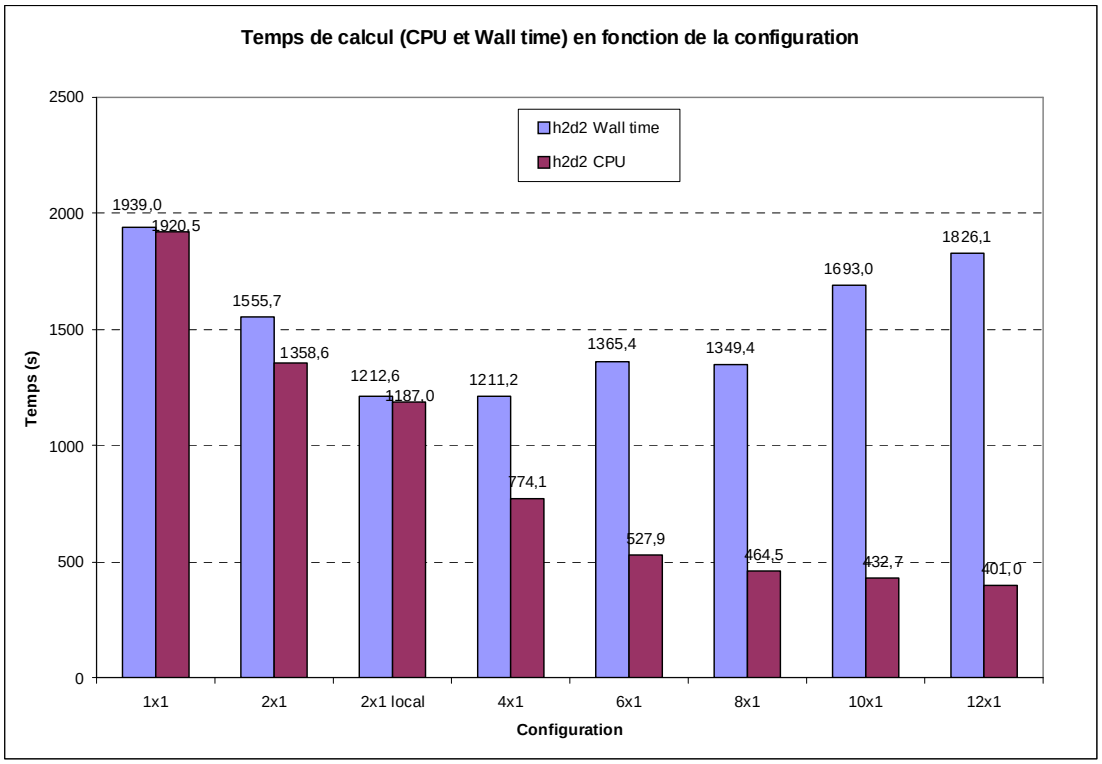

Figure 15 – Comparaison des temps de calcul CPU et Wall time pour différentes configuration MPI.

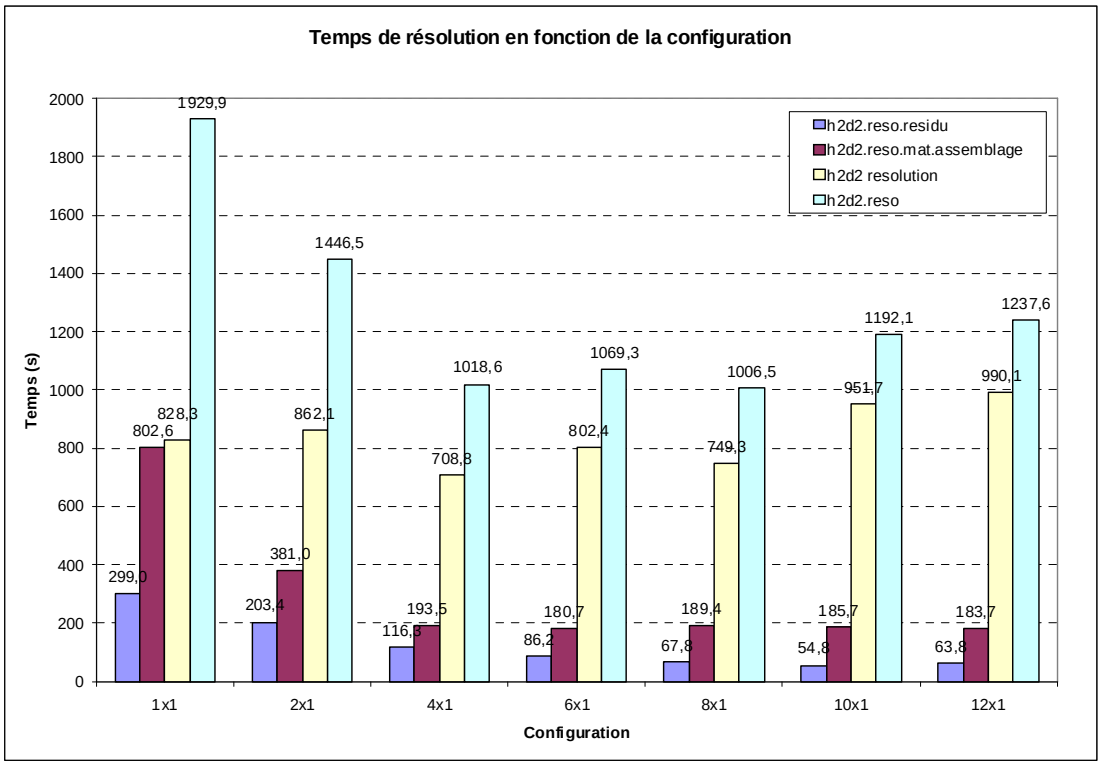

Figure 16 – Temps de résolution totaux (h2d2.reso) et temps consacrés à l'assemblage du résidu (h2d2.reso.residu) et de la matrice (h2d2.reso.mat.assemblage) et à la résolution pour différentes configurations MPI.

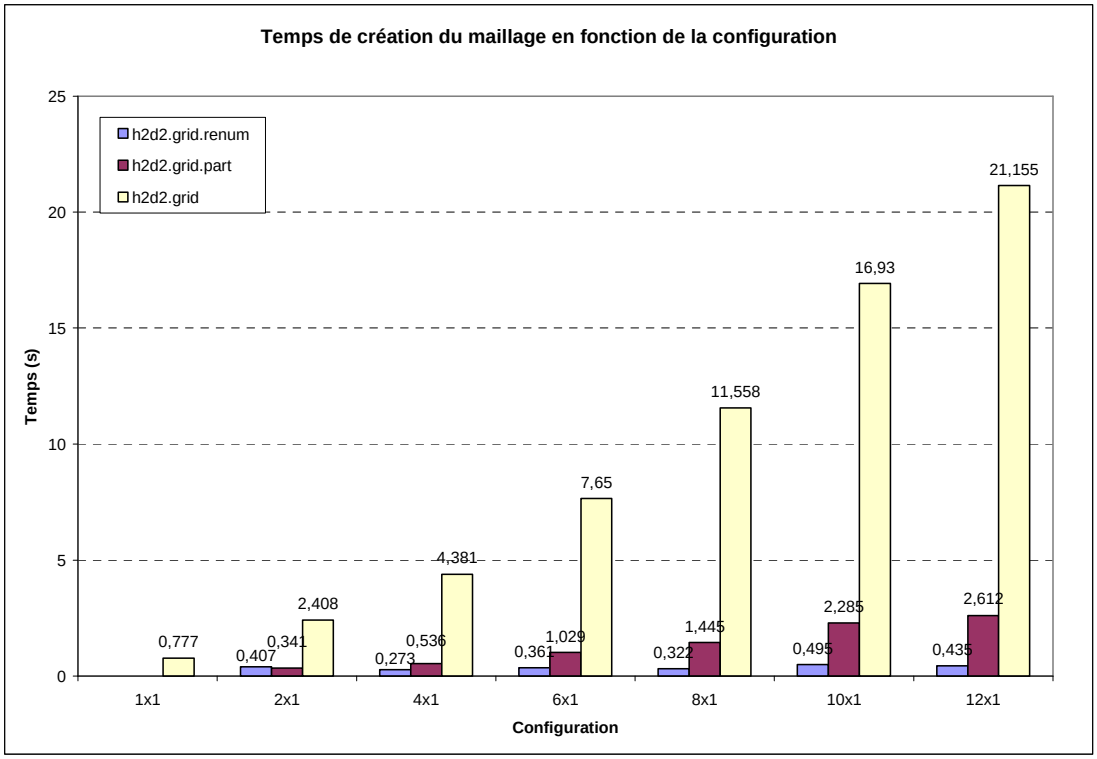

Figure 17 – Temps de création du maillage partitionné (h2d2.grid) et temps consacrés au partitionnement (h2d2.grid.part) et à la renumérotation (h2d2.grid.renum) pour différentes configurations MPI.

Sur le cluster, les temps de calcul totaux diminuent de façon marquée avec le nombre de processus, jusqu'à ce que ce nombre excède les capacités du cluster (2 processeurs double-cœur), soit passé 4x1. De même, sur un ordinateur local (1 processeur double-cœur), les temps diminuent jusqu'à la configuration 2x1, pour ensuite augmenter pour un nombre de processus supérieur à 2. Ces augmentations sont principalement dues aux entrées sorties (h2d2.io) qui demandent beaucoup d'échanges réseau. Il est intéressant de remarquer que les temps totaux pour la configuration 12x1 sont comparables à ceux de la configuration 1x1. Les temps consacrés à la résolution par MUMPS (Figure 16), quant à eux, ne présentent pas de variations significatives, à l'exception d'une légère augmentation pour les configurations 10x1 et 12x1. La taille du problème testé n'est possiblement pas suffisante pour surpasser l'overhead du parallélisme, ce qui en apparence donne des temps de résolution relativement constants (voir section 3.5.3). De plus, les temps d'assemblage semblent tendre asymptotiquement vers une valeur plancher. Les temps nécessaires à la création du maillage partitionné, quant à eux, augmentent avec le nombre de processus; ces temps sont toutefois négligeables en comparaison avec les temps totaux.

La Figure 15 permet de mesurer la part du temps de calcul utilisé par MPI, qui correspond environ à la différence entre les temps CPU et Wall time pour les configurations n'excédant pas 4x1 (au-delà de ce nombre de processus, il devient difficile de mesurer la part utilisée par MPI en raison du swap sur le disque). Contrairement à la configuration 2x1, la configuration 2x1 local présente des temps CPU très près des temps Wall time, ce qui montre que la part de calcul utilisée par MPI est réduite de façon significative lorsque les processus tournent sur le même ordinateur, plutôt qu'un cluster. D'une part, la vitesse du réseau ralentit les échanges effectués par MPI et, d'autre part, H2D2 n'exploite peut-être pas MPI de façon optimale. Par ailleurs, en regardant de plus près les temps Wall time pour la configuration 2x1 sur le cluster et sur un seul ordinateur (local), on peut en déduire les temps consacrés aux échanges réseau, qui correspondent à une augmentation de 28 % des temps de calcul pour cette configuration.

3.5.3. Speed-up

La Figure 18 présente les speed-up calculés à partir des temps totaux (wall time) en fonction du nombre de processus MPI. Les temps obtenus sur un seul ordinateur (local : 1 processeur double-cœur) et sur le cluster y sont comparés.

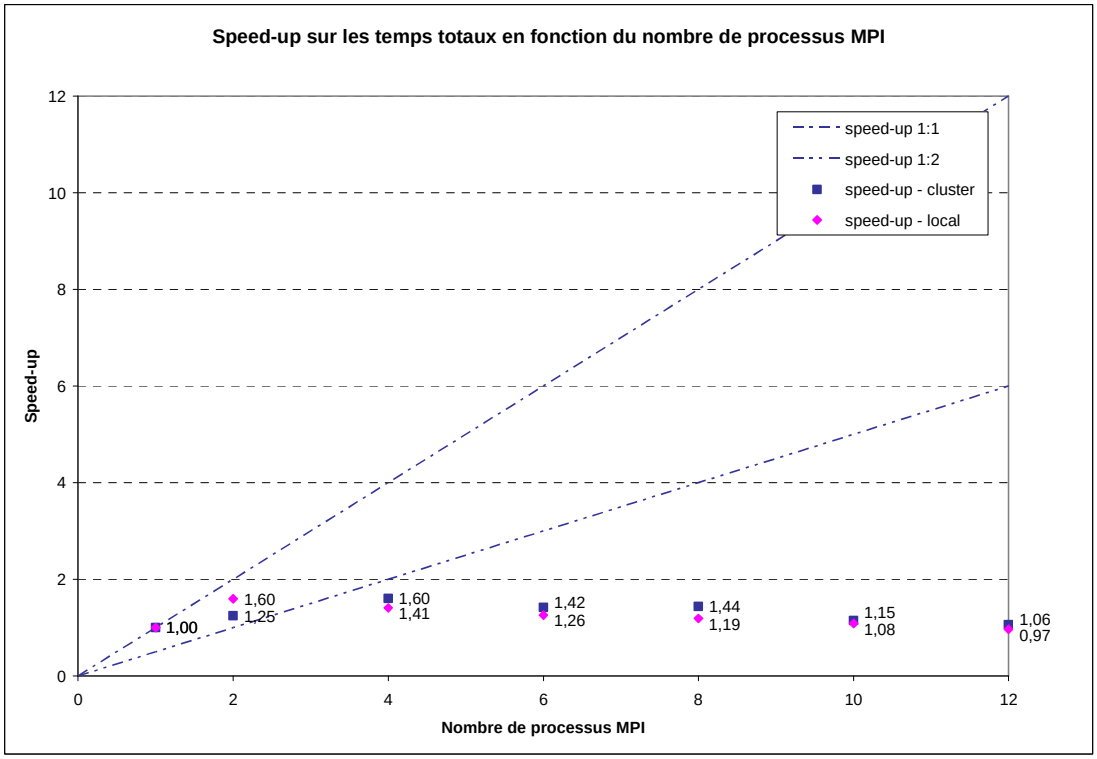

Figure 18 – Speed-up calculés sur les temps totaux en fonction du nombre de processus MPI, pour des calculs effectués sur un cluster et sur un ordinateur local.

On remarque que les speed-up sur le cluster atteignent une valeur maximale de 1,60 sur 4 processus, pour ensuite descendre progressivement suivant l'augmentation du nombre de processus. Il est intéressant de noter que les speed-up sur le cluster sont systématiquement plus élevés que sur un seul ordinateur, à l'exception des résultats à 2 processus, ce qui laisse penser à une limitation mémoire et du swap sur le disque. En effet, les speed-up obtenus localement à 2 processus sont de 1,60, comparativement à 1,25 sur le cluster, dû aux échanges réseau entre les deux ordinateurs du cluster qui viennent ralentir les temps de calcul. Il est surprenant de remarquer que le speed-up local sur 2 processus est égal au speed-up sur le cluster avec 4 processus, ce qui renforce l'idée d'un ralentissement dû aux échanges réseau.

La Figure 19, de son côté, présente les speed-up calculés à partir des temps de résolution totaux et des temps consacrés à l'assemblage et à la résolution par MUMPS.

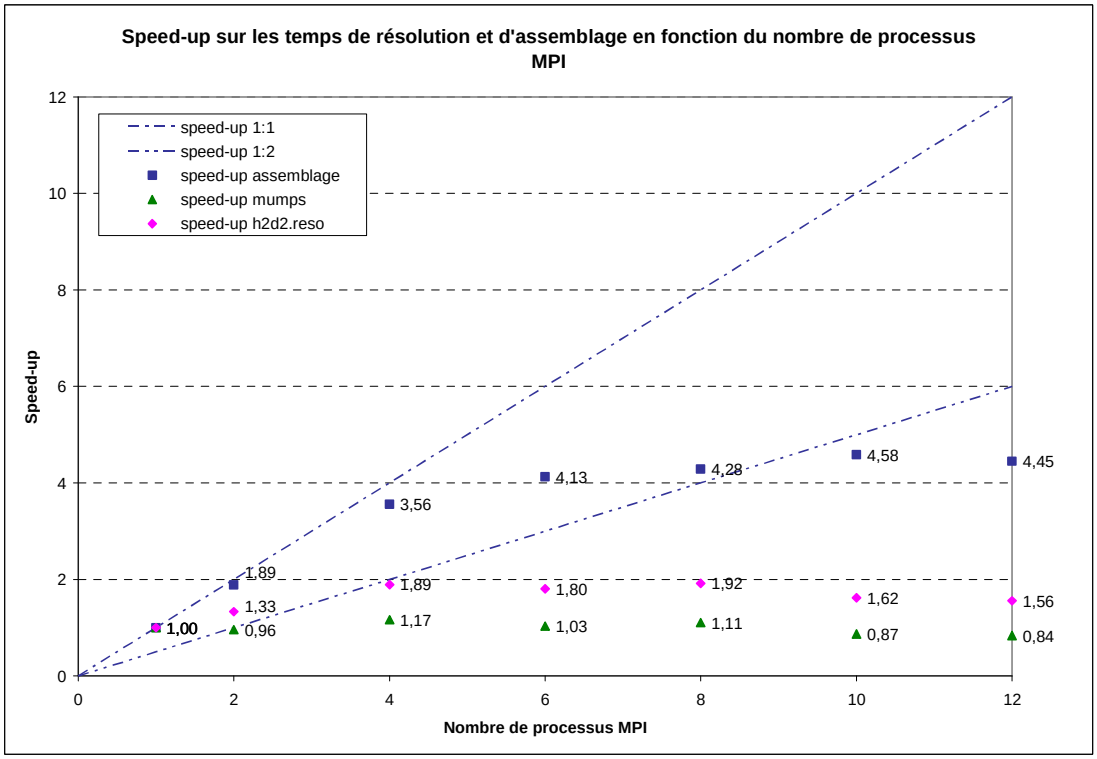

Figure 19 - Speed-up calculés sur les temps de résolution totaux (h2d2.reso) et consacrés à l'assemblage et à la résolution par mumps en fonction du nombre de processus MPI.

Les speed-up sur l'assemblage de la matrice et du résidu sont de l'ordre de 1:1 jusqu'à 4 processus, ce qui confirme que le travail d'assemblage est bien parallélisé. Les speed-up se dégradent lorsque le nombre de processus dépasse les capacités du cluster. De même, les speed-up sur la résolution par MUMPS se détériorent au-delà de 4 processus. Ces derniers sont toutefois très bas, soit de l'ordre de 1 peu importe la configuration MPI. De fait, si le problème testé est trop petit (quelques secondes de factorisation matricielle en séquentiel), on peut s'attendre à ce que MUMPS soit plus lent en parallèle qu'en séquentiel et ne présente aucun speed-up, en raison de l'overhead du parallélisme (MUMPS: Frequently Asked Questions, http://mumps.enseeiht.fr/index.php?page=faq). Dans le cas présent, les temps de résolution sont de l'ordre de 800 secondes (Figure 16). En divisant par 24 pas de temps et par 8 itérations, on obtient un temps approximatif de 4 secondes pour la factorisation d'une matrice, ce qui n'est apparemment pas suffisant pour surpasser l'overhead. De plus gros problèmes devront être testés

afin de déterminer les réels speed-up de MUMPS. Globalement, la partie de résolution de H2D2 (h2d2.reso) atteint des speed-up de 1,89 à 4 processus, ce qui est juste en-deçà de la barre 1:2.

3.6. Conclusion

Les éléments suivants se dégagent des tests de parallélisme effectués en mémoire distribuée :

- Plus le nombre de nœuds de calcul est grand, plus les temps associés à la portion h2d2.io augmentent, en raison des transferts d'information vers le processus maître qui est le seul à lire/écrire les fichiers;
- Les différences entre les temps CPU et wall time révèlent, d'une part, que la vitesse du réseau ralentit les échanges effectués par MPI et, d'autre part, que H2D2 n'exploite peut-être pas MPI de façon optimale;
- Les échanges réseau mènent à une augmentation de 28 % sur les temps de calcul pour la configuration 2x1 sur le cluster par rapport à cette même configuration sur un ordinateur local;
- Les speed-up sur les temps totaux atteignent, sur le cluster, une valeur maximale de 1,60 sur 4 processus, cette valeur étant égale au speed-up local sur 2 processus, ce qui renforce l'idée d'un ralentissement dû aux échanges réseau; les temps de calcul augmentent lorsque le nombre de processus excède les capacités du cluster;
- Les speed-up sur l'assemblage de la matrice et du résidu sont de l'ordre de 1:1 jusqu'à 4 processus (3,56), ce qui confirme que le travail d'assemblage est bien parallélisé;
- Les speed-up sur les temps de résolution totaux (h2d2.reso) atteignent 1,89 à 4 processus, ce qui est tout juste en-deçà de la barre 1:2;
- Les speed-up pour MUMPS sont absents en raison de l'overhead du parallélisme qui domine pour des problèmes de trop petite taille;
- Avec les timers inclus dans H2D2, on a un outil très intéressant pour analyser le comportement du logiciel.

4. Tests comparatifs de parallélisme avec résolution complète

Cette section présente les résultats de tests comparatifs de parallélisme effectués en mémoire partagée et en mémoire distribuée, appliquant les configurations optimales à une résolution typique sur un domaine réel de calcul, soit celui du fleuve Saint-Laurent.

4.1. But

Ces tests ont pout but de :

- Déterminer les speed-up pour différentes configurations OMP pour un problème réel;
- Comparer et déterminer les configurations MPI et OMP optimales pour les solveurs MUMPS et Pardiso.

4.2. Protocole d'analyse

Les tests ont été effectués avec la formulation de l'élément de Saint-Venant 2D (SV2D_Conservatif_CDYS_NN) de H2D2. Une résolution par méthodes directes (MUMPS et Pardiso) sur un système fluvial avec une part de résolution réelle a été réalisée afin de déterminer les vrais speed-up associés autant à la résolution qu'à l'assemblage. Le compilateur intel 11.0 et la configuration *color* ont été utilisés.

Différentes configurations OMP ont été testées sur le maillage du fleuve Saint-Laurent présenté à la Figure 10, pour le même scénario d'analyse que celui décrit à la section 3.2. Le solveur Pardiso a été utilisé avec les configurations suivantes :

- MPI=1, OMP=1 (ou 1x1)
- MPI=1, OMP=2 (ou 1x2)
- MPI=1, OMP=4 (ou 1x4)

Le solveur MUMPS a également été testé avec les configurations suivantes :

- MPI=1, OMP=2 (ou 1x2)

- MPI=2, OMP=2 (ou 2x2)

Ces configurations ont finalement été comparées avec certaines configurations testées à la section 3.2 afin de déterminer les configurations optimales en termes de speed-up.

4.3. Résultats

4.3.1. Speed-up

La Figure 20 présente, pour le solveur Pardiso, les speed-up en fonction du nombre de tâches OMP, obtenus à partir des temps totaux de résolution et des temps consacrés à la résolution et à l'assemblage de la matrice et du résidu.

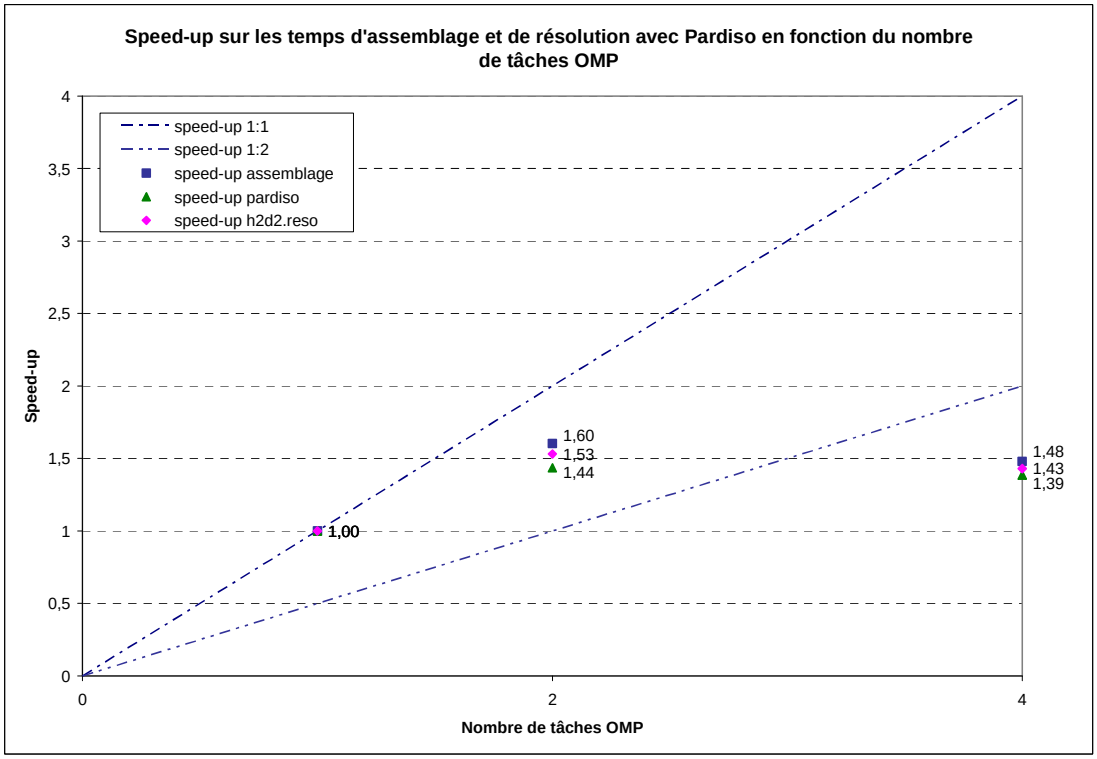

Figure 20 - Speed-up calculés sur les temps de résolution totaux (h2d2.reso) et consacrés à l'assemblage et à la résolution par Pardiso en fonction du nombre de tâches OMP.

Les tests effectués avec le solveur Pardiso donnent des speed-up de 1,53 pour les temps totaux de résolution de H2D2 (h2d2.reso) avec deux tâches OMP. Dans le cas testé, la partie de

résolution correspond à 99 % du temps total de calcul; le speed-up sur les temps totaux de calcul est donc aussi de 1,53. Pour la partie d'assemblage de la matrice et du résidu, le speed-up est de 1,60, alors qu'il est de 1,44 pour la partie de résolution. La configuration OMP à quatre tâches n'est pas optimale, puisque le nombre maximal possible de tâches par processus est 2, vue la configuration du cluster (1 processeur double-cœur par ordinateur). C'est pourquoi les speed-up obtenus avec quatre tâches sont inférieurs à ceux obtenus avec deux tâches.

4.3.2. Comparaison de configurations MPI et OMP

La Figure 21 présente, pour différentes configurations, une comparaison des temps de calcul totaux (temps CPU et wall time) et des temps associés à l'assemblage de la matrice et du résidu et à la résolution par les solveurs MUMPS et Pardiso.

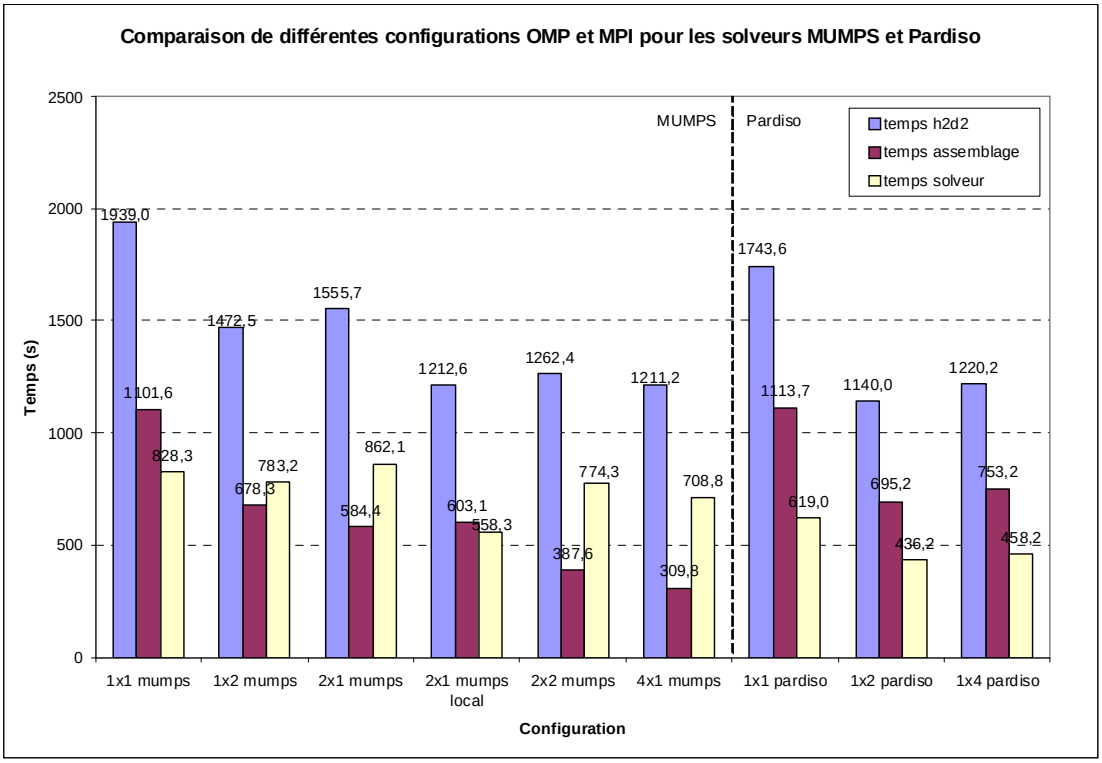

Figure 21 – Temps totaux de calcul et consacrés à l'assemblage et à la résolution pour différentes configurations OMP et MPI avec les solveurs MUMPS et Pardiso.

Dans l'ensemble, la configuration 1x2 de Pardiso semble la plus optimale, avec des temps de calcul inférieurs à toutes les autres configurations; elle est suivie de près par la configuration 4x1

de MUMPS. Le solveur Pardiso est en effet très optimisé et constitue en ce sens une option très avantageuse. En configuration 1x1, Pardiso est d'ailleurs plus performant que MUMPS. On note par ailleurs qu'avec le solveur MUMPS, il est plus avantageux de travailler en multi-process qu'en multi-tâches, les temps obtenus avec la configuration 1x2 de MUMPS étant plus grands que ceux obtenus avec les configurations 2x1 local et 4x1 de MUMPS. De plus, la configuration 2x2 de MUMPS présente des temps de calcul supérieurs à la configuration 2x1 local, en raison des temps de résolution qui augmentent. Les temps d'assemblage, quant à eux, semblent plus faibles en configuration 2x1 qu'en configuration 1x2, indépendamment du solveur. La résolution est toutefois plus rapide avec le solveur Pardiso qu'avec MUMPS.

4.4. Conclusion

Les éléments suivants se dégagent des tests comparatifs de parallélisme effectués avec résolution complète sur le système du Saint-Laurent :

- Les speed-up sur les temps totaux de calcul et sur les temps de résolution sont de 1,53 avec deux tâches OMP;
- La configuration 1x2 avec le solveur Pardiso est la configuration optimale avec le cluster actuel (2 processeurs double-cœur répartis sur 2 ordinateurs) et surpasse la configuration 4x1 avec le solveur MUMPS en termes de temps totaux de calcul;
- Avec le solveur MUMPS, il est plus avantageux de travailler en multi-process qu'en multi-tâches;
- La résolution est plus rapide avec le solveur Pardiso qu'avec MUMPS.

5. Conclusion

Des tests ont été effectués en mémoire partagée et en mémoire distribuée afin d'évaluer l'efficacité de la parallélisation de l'assemblage et de la résolution matricielle directe dans H2D2. Ils permettent de tirer les conclusions suivantes :

En mémoire partagée :

- On peut tirer avantage des méthodes de résolution (Pardiso, SuperLU) qui sont des méthodes plus rapides et spécialisées; un plus grand choix de méthodes de résolution est également à disposition;
- Le parallélisme en mémoire partagée est par contre désavantagé par des speed-up de l'ordre de 1,5 pour deux tâches;
- On observe une forte sensibilité de l'assemblage à la perturbation de la renumérotation induite par le coloriage, qui est également dépendante du problème testé (taille du maillage) : même si la numérotation des nœuds est optimale, elle est ensuite détruite par le coloriage.

En mémoire distribuée :

- La méthode de résolution matricielle, MUMPS, a un speed-up de l'ordre de 1, indépendamment du nombre de processus, en raison de l'overhead du parallélisme qui domine pour des problèmes de trop petite taille.
- L'assemblage est par contre très bien parallélisé avec un speed-up proche de *n* (nombre de processus);
- Le parallélisme en mémoire distribuée mène à des speed-up globaux de l'ordre de 1,6 sur 2 processus (local) ou sur 4 processus (cluster); avec un nombre de processus dépassant le nombre de cœurs, les speed-up décroissent;
- Les échanges réseau augmentent les temps de calcul; de plus, la vitesse du réseau ralentit les échanges effectués par MPI;
- Après partitionnement et renumérotation locale, on se trouve à avoir sur chaque processus moins de données et, conséquemment, une meilleure répartition en mémoire et une diminution des cache-miss.

Les règles suivantes peuvent donc être tirées quant au choix de critères de configuration :

- Tant que le problème à résoudre tourne en mémoire sur une machine multi-cœur et/ou multi-processeur, utiliser les méthodes en mémoire partagée (1x2, 1x4, etc.), Pardiso étant le solveur optimal;
- Autrement, passer en mémoire distribuée en utilisant MUMPS pour des configurations *n*x1 (les configurations *n*x*m* n'étant pas optimales).

Ces tests n'ont pas permis de déterminer la taille optimale du maillage pour un nœud de calcul. Ils donnent néanmoins une orientation aux travaux qui seront réalisés dans le futur, lesquels viseront notamment à approfondir les éléments suivants :

En mémoire partagée :

- Investiguer d'autres moyens d'assembler résidu et matrices pour s'affranchir de l'impact du coloriage;
- Investiguer une approche de renumérotation plus globale qui optimise le résultat final;
- Paralléliser les quelques boucles qui ne le sont pas encore.

En mémoire distribuée :

- Réviser l'utilisation de MPI en particulier sur les entrées/sorties (IO) et globalement pour mieux utiliser la librairie;
- Analyser le comportement de MUMPS et la façon dont H2D2 interface cette librairie;
- Tester des problèmes plus volumineux afin d'étudier le comportement de MUMPS au-delà de l'overhead du parallélisme.

Les timers inclus dans H2D2 se sont révélés un outil très intéressant pour analyser le comportement du logiciel. Ils permettent d'établir des stratégies optimales de simulation et d'orienter les développements futurs du logiciel.

6. Glossaire

Cache :

La cache est de la mémoire très rapide, proche du processeur, qui contient une copie des données sur lesquelles le processeur travaille. Tablant sur une localité des données, le gestionnaire de cache, lors d'une demande de données par le processeur, va charger plus de données en un seul accès.

Cache-hit :

Lors d'une demande d'accès à des données par le processeur, on parle de cache-hit si les données sont déjà disponibles dans la cache.

Cache-miss :

Lors d'une demande d'accès à des données par le processeur, on parle de cache-miss si les données ne sont pas disponibles dans la cache et doivent être chargées de la mémoire.

Cœur :

Unité de calcul. Actuellement la majorité des processeurs ont plusieurs cœurs qui se partagent les autres ressources (mémoire, disque) du nœud.

Mémoire distribuée :

Mémoire répartie entre plusieurs processus et qui peut être sur plusieurs machines.

Mémoire partagée :

Mémoire partagée par un processus.

Nœud :

Unité physique (hardware), composante d'une grappe de calcul. Un nœud peut contenir de 1 à 4 processeurs chacun avec de 1 à 4 cœurs.

Overhead :

Coût (en temps) associé à l'utilisation d'une certaine technique par rapport à une référence.

Processus (process) :

Programme roulant dans son propre espace mémoire, séparé des autres processus, sur le même nœud ou sur un autre nœud. Deux processus roulant sur le même nœud partagent les ressources de ce nœud. Un processus peut comprendre plusieurs tâches qui se partagent les ressources du processus comme la mémoire.

Speed-up :

Rapport entre le temps d'exécution multi-tâches et/ou multi-processus et le temps en simple processus. Le speed-up maximum est donc de *n*, *n* étant le nombre de tâches/processus.

Swap :

Mémoire virtuelle, fichiers utilisés par le système d'exploitation pour décharger la section de la mémoire physique non utilisée par un processus sur le disque. L'utilisation d'une mémoire de masse (type disque dur) permet entre autres à des programmes de pouvoir s'exécuter dans un environnement matériel possédant moins de mémoire centrale que nécessaire.

Tâche (task) :

Sous-unité de travail. Portion de code s'exécutant indépendamment mais partageant les ressources du processus parent. Typiquement, lorsqu'on parallélise une boucle de calcul, on lance une tâche sur chacun des cœurs disponibles avec pour chaque tâche une partie des indices de la boucle.

7. Bibliographie

Intel Math Kernel Library 10.1 (MKL). http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm. Intel Corporation. (consulté en mars 2009).

MUMPS: A Multifrontal Massively Parallel Sparse Direct Solver. http://mumps.enseeiht.fr/. (consulté en mars 2009).

OpenMP: The OpenMP API specification for parallel programming. http://openmp.org/wp/. (consulté en mars 2009).

Pardiso Solver Project. http://www.pardiso-project.org/. (consulté en mars 2009).

ParMETIS: Parallel Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview. Karypis Lab. (consulté en mars 2009).

Scotch et PT-Scotch. http://www.labri.fr/perso/pelegrin/scotch/scotch_fr.html. (consulté en mars 2009).

The Message Passing Interface (MPI) standard. http://www.mcs.anl.gov/research/projects/mpi/. (consulté en mars 2009).