Stratégies de Parallélisation pour la Simulation Numérique du Transfert de Chaleur dans les Dissipateurs Thermiques de CPU

FOSSI Cedric

18/11/2023

Cette étude se concentre sur la parallélisation d'une simulation numérique visant à déterminer la dynamique de température d'un modèle récent de CPU, spécifiquement l'AMD EPYC "Rome", en utilisant un dissipateur thermique. Nous explorons différentes méthodologies de parallélisation afin d'optimiser le temps de simulation et l'utilisation des ressources computationnelles.

Microprocesseurs et Dissipateur Thermique Métallique

Dans le domaine de l'informatique, l'évolution des microprocesseurs a été marquée par des avancées significatives en termes de puissance de traitement et d'efficacité énergétique. Cependant, ce progrès s'accompagne de ses propres défis, parmi lesquels la surchauffe reste un problème critique. En 2023, la quête de capacités de calcul plus puissantes a conduit au développement de microprocesseurs consommant plus de 200 Watts de puissance. Cette consommation énergétique substantielle se traduit presque entièrement en chaleur, posant une menace importante à la fiabilité et aux performances de ces dispositifs. Le phénomène de surchauffe met non seulement en péril la stabilité opérationnelle des microprocesseurs, mais limite également la portée des avancées futures dans la conception et les fonctionnalités des processeurs. La gestion et la mitigation de la génération de chaleur dans les microprocesseurs sont donc devenues des préoccupations majeures pour les chercheurs et les ingénieurs.

Les dissipateurs thermiques métalliques constituent un composant fondamental dans les systèmes de gestion thermique des microprocesseurs. Leur fonction principale est de transférer efficacement la chaleur loin du processeur vers l'environnement externe. Ce processus est crucial pour maintenir des températures de fonctionnement optimales.

Contents

1 Introduction

Notre recherche se focalise sur la parallélisation d'une simulation numérique afin d'analyser la dynamique de température du CPU AMD EPYC "Rome", avec l'ajout d'un dissipateur thermique. Cette simulation nécessite un processus de discrétisation spatiale où le dissipateur est divisé en une grille finie tridimensionnelle de cellules cuboïdales, permettant une modélisation précise de son comportement thermique.

La granularité du maillage est cruciale pour la précision de la simulation. Bien que des maillages plus fins fournissent des informations plus détaillées, ils augmentent significativement les exigences en ressources computationnelles.

Pour faciliter notre étude, nous utilisons Grid5000, une infrastructure de recherche dédiée aux expériences de calcul parallèle et distribué à grande échelle. Il agit comme un superordinateur, hébergeant de nombreux CPU, ce qui nous permet d'employer des maillages plus fins pour une modélisation thermique plus précise tout en parallélisant le processus pour minimiser le temps d'exécution.

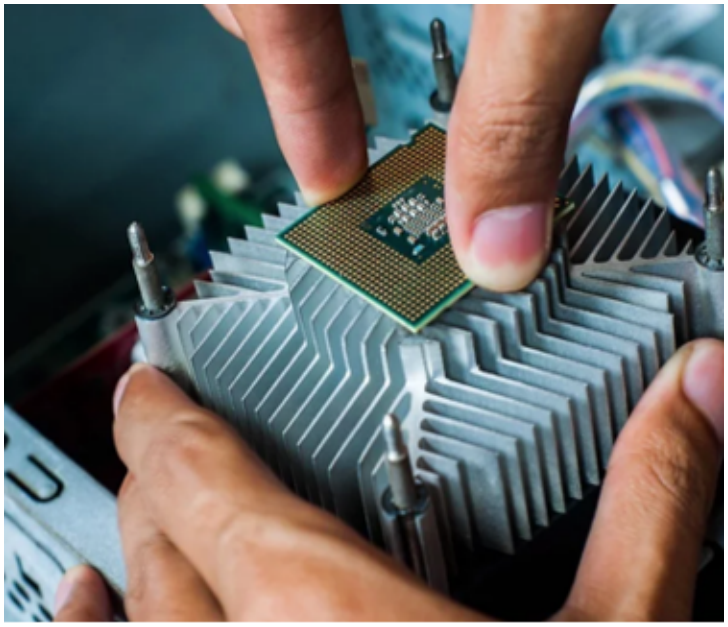


Figure 1: Un dissipateur thermique refroidi par ventilateur sur un processeur.

2 Résultats Obtenus

En employant des techniques de parallélisation, nous avons significativement réduit le temps d'exécution. En mode Normal avec des maillages plus fins, en utilisant une méthode de découpage 2D, nous avons observé une réduction de 760 secondes à 154 secondes, résultant en un facteur d'accélération d'environ 4,94x.

De plus, la mise en œuvre de communications non bloquantes a encore diminué le temps d'exécution à 135 secondes, atteignant un facteur d'accélération d'environ 5,63x.

3 Techniques de Parallélisation

Le dissipateur thermique est partitionné en parallélépipèdes, qui sont ensuite distribués sur plusieurs processeurs. Cette approche parallèle permet le traitement simultané de sections distinctes du dissipateur.

Un aspect crucial de notre simulation est le Mécanisme de Mise à Jour de la Température, responsable du calcul des changements de température dans chaque cellule en fonction des températures de ses cellules voisines. Pour les cellules situées aux bords de chaque parallélépipède, une communication avec les parallélépipèdes adjacents est nécessaire pour obtenir les données requises. Cette communication est facilitée par l'Interface de Passage de Messages (MPI).

Pour déterminer la convergence, indiquant si l'état d'équilibre a été atteint, nous calculons la somme des différences de température à travers toutes les cellules entre le temps t et $t+1$. Avec notre approche de parallélisation, cette opération est effectuée sur chaque processeur responsable d'une portion du dissipateur thermique.

Enfin, MPI est utilisé pour rassembler ces valeurs sur un processeur choisi, qui détermine si l'état d'équilibre a été atteint.

3.1 Première Technique : Découpage le long de l'Axe Z

La première technique de parallélisation que nous avons employée consiste à découper le dissipateur thermique le long de l'axe Z. Cette méthode divise le dissipateur en plusieurs parallélépipèdes rectangulaires, qui sont ensuite distribués sur différents processus.
Cette stratégie bénéficie de la contiguïté de la mémoire à travers les plans, facilitant l'agrégation des résultats à la fin des calculs.

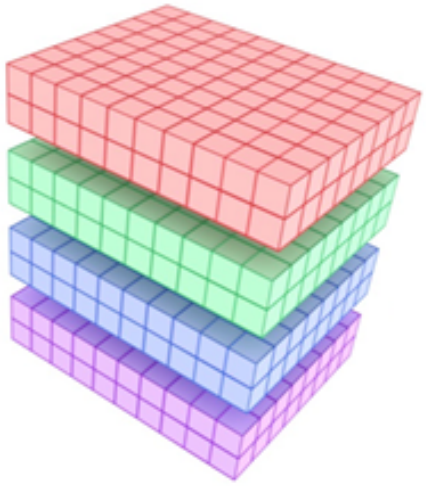


Figure 2: Découpage le long de l'axe Z.

3.2 Deuxième Technique : Découpage 2D des Données le long des Axes Z et Y

Notre deuxième approche de parallélisation intègre un découpage 2D des données le long des axes Z et Y. Cette technique décompose davantage le dissipateur en un plus grand nombre de parallélépipèdes, chacun assigné à un processeur spécifique, tout en maintenant la contiguïté le long de l'axe X. Comme pour la première méthode, cette approche nécessite une augmentation des communications entre les processus pour calculer avec précision les transferts thermiques des cellules sur les bords.

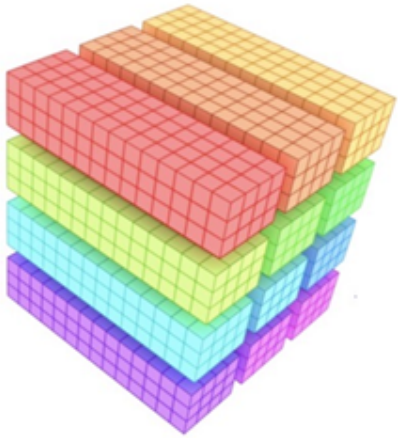


Figure 3: Découpage 2D des données le long des axes Z et Y.

4 Analyse des Résultats

Dans notre étude de l'efficacité de la parallélisation, nous avons utilisé à la fois des techniques de partitionnement 1D et 2D sur deux modes opérationnels distincts : Rapide et Normal. Le mode "Rapide" se caractérise par des tranches comparativement plus grandes, conduisant à une charge computationnelle réduite et, par conséquent, à des temps d'exécution plus rapides. En revanche, le mode "Normal" utilise des maillages plus fins pour le découpage, visant une plus grande précision au coût d'une demande computationnelle accrue. Cette distinction est cruciale, car des maillages plus fins, malgré leur potentiel pour fournir des résultats plus détaillés, élèvent significativement les ressources computationnelles requises, prolongeant ainsi le temps d'exécution.

4.1 Performance en Mode Rapide

Une performance optimale a été atteinte en utilisant une partition 1D avec quatre processeurs, résultant en un temps d'exécution de 2 secondes, représentant un facteur d'accélération de 2x comparé au code séquentiel. Cependant, un partitionnement supplémentaire de petits ensembles sur plus de processeurs a conduit à des communications excessives et chronophages, causant une baisse de performance au-delà de cette configuration.
En revanche, le partitionnement 2D a fourni des améliorations de performance acceptables par rapport au code séquentiel, avec moins de variations comparativement au découpage 1D.

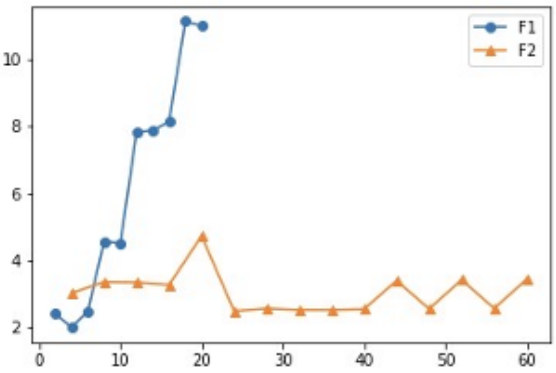


Figure 4: Performance du partitionnement 1D et 2D en mode Rapide.

4.2 Performance en Mode Normal

Le partitionnement 2D démontre une efficacité supérieure, atteignant un temps d'exécution de 154 secondes et accélérant l'exécution du code séquentiel par un facteur de 4,94x. De plus, le partitionnement 2D maintient une cohérence, atteignant des résultats optimaux avec 32 processeurs et montrant une variance minimale comparativement au découpage 1D.
D'autre part, le partitionnement 1D atteint sa meilleure performance avec 12 processeurs, résultant en un temps d'exécution de 168 secondes, représentant un facteur d'accélération de 4,52x comparativement au code séquentiel. Cependant, la performance commence à décliner au-delà de 20 processeurs.

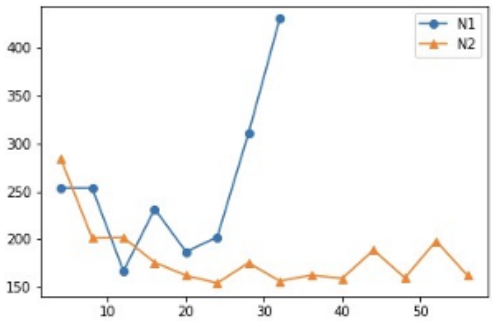


Figure 5: Performance du partitionnement 1D et 2D en mode Normal.

5 Communications Non Bloquantes

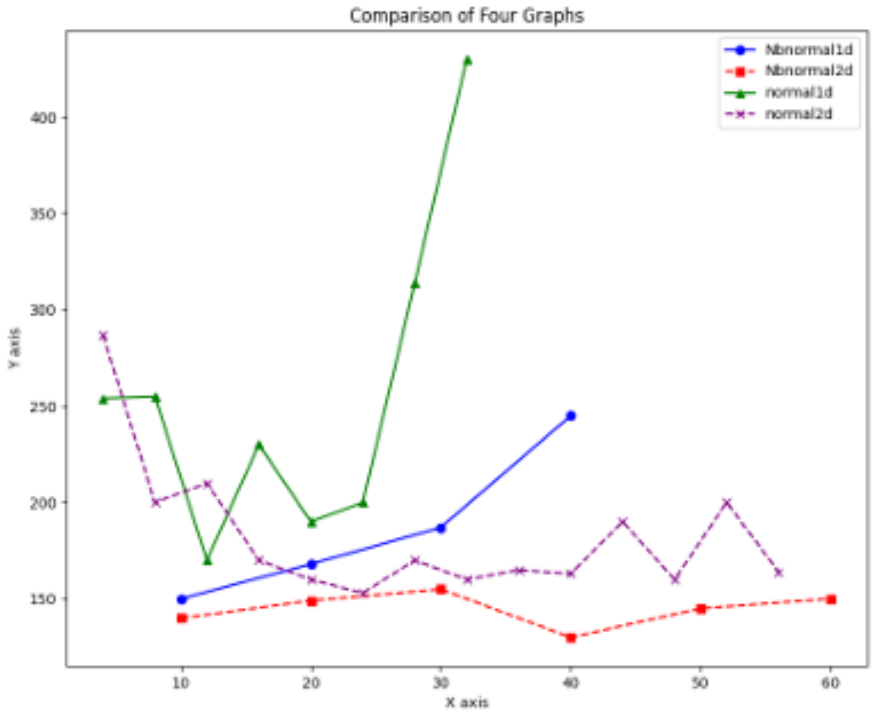


Figure 2: Performance du partitionnement 1D et 2D en mode Normal utilisant des communications non bloquantes.

Poursuivant notre étude, nous avons exploré l'impact de l'utilisation de la communication non bloquante dans nos deux approches de parallélisation. La communication non bloquante permet aux processus d'initier des opérations de transfert de données et de poursuivre leur exécution sans avoir à attendre la fin du transfert. Cette technique peut considérablement améliorer l'efficacité des applications parallèles en réduisant le temps et potentiellement en abaissant le temps d'exécution global dans un environnement de calcul distribué. Dans les méthodes de découpage 1D et 2D, le mode de communication non bloquante a mieux performé, atteignant des résultats optimaux. Spécifiquement, l'utilisation de communications non bloquantes 2D avec plus de 40 processeurs a résulté en une accélération du temps d'exécution du code séquentiel par un facteur de 5,63x.

6 Point de Contrôle pour une Efficacité et une Fiabilité Accrues

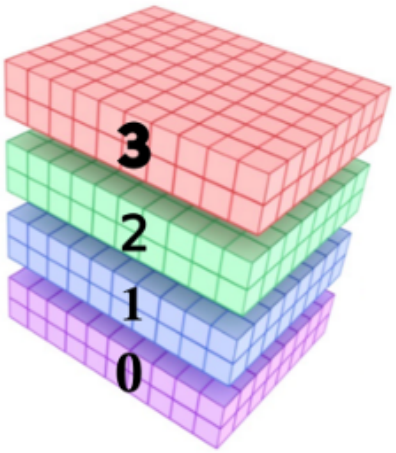
Pour des calculs s'étendant sur de longues périodes, l'intégration d'un système de point de contrôle dans nos codes émerge comme une stratégie hautement bénéfique. Ce système, désigné pour les codes marqués avec le suffixe `*cp.c`, permet au processus de sauvegarder périodiquement son état actuel.

Un tel mécanisme assure que, face à une quelconque défaillance ou interruption, les calculs ne sont pas réinitialisés mais peuvent plutôt reprendre à partir du point de contrôle le plus récent. Cette capacité non seulement améliore significativement l'efficacité en éliminant le besoin de redémarrer les processus depuis le début, mais augmente également substantiellement la fiabilité des opérations de longue durée.

7 Section Questions & Réponses

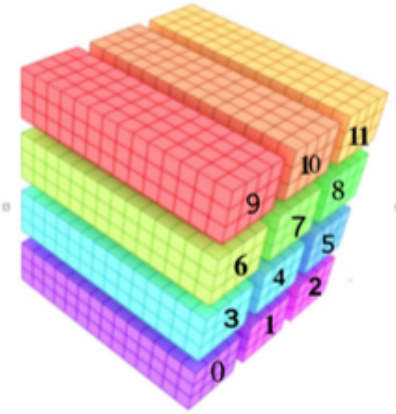
Comment avons-nous distribué les processeurs dans le cas d'un découpage unidimensionnel ?
Nous avons déterminé le nombre de plans XY dans un dissipateur en divisant sa hauteur (Z) par l'épaisseur de découpage ΔZ. Par exemple, avec $Z = 0,15$ et $\Delta Z = 0,01$, nous avons identifié 15 plans XY. Ces plans sont distribués parmi K processeurs en divisant le nombre total de plans XY par K. Si le résultat de la division est inférieur à 1, cela indique plus de processeurs que de plans XY, attribuant un plan par processeur. Sinon, les plans XY sont répartis uniformément, le dernier processeur gérant les éventuels restes.

Comment avons-nous calculé les transferts thermiques pour le découpage 1D ?
Le calcul dépend des cellules voisines. Nous avons introduit deux plans supplémentaires de chaque côté pour chaque processeur, sauf le premier et le dernier, qui en reçoivent un chacun. Ces plans supplémentaires facilitent l'échange d'informations entre les processeurs à chaque itération.



Comment avons-nous facilité les communications dans le cas d'une décomposition 1D ?
La communication entre les processeurs voisins est essentielle. Les processeurs de 1 à $K-2$ effectuent deux communications, tandis que 0 et $K-1$ en effectuent une. Nous avons utilisé la fonction `MPI_Sendrecv` de MPI au sein de structures conditionnelles `if-else` pour gérer cela, répétant le processus à chaque itération.

Comment avons-nous distribué les processeurs dans le cas d'un découpage bidimensionnel ?
Nous avons utilisé une fonction pour diviser k processus en i, j tel que $k = i \times j + r$, puis partitionné l'axe Z par i et l'axe Y par j. Les ij processus ont été distribués en conséquence.



Comment avons-nous calculé les transferts thermiques pour le découpage 2D ?
Une approche similaire au découpage 1D a été utilisée, en introduisant 4 plans supplémentaires pour les informations nécessaires.

Comment avons-nous facilité les communications dans le cas d'une décomposition 2D ?
Nous avons divisé les communications le long des axes Z et Y, permettant aux processus de communiquer avec leurs voisins directs. Cela a été implémenté en utilisant des opérations arithmétiques pour identifier les processeurs d'envoi et de réception.

Qu'est-ce que la communication non bloquante ?
La communication non bloquante permet aux processus de continuer sans attendre que les transferts de données soient terminés, en superposant communication et calcul pour une efficacité améliorée et un temps d'exécution réduit.

Comment avons-nous implémenté la communication non bloquante ?
Nous avons adapté nos codes 1D et 2D pour utiliser des techniques non bloquantes, intégrant `MPI_Wait` et `MPI_Test` pour surveiller la complétion des communications sans perturber les calculs.