PROJET PPAR: Heat diffusion

Assia MASTOR
Amel MOKDADI

Dans ce projet, notre objectif principal fut d'améliorer les performances du code fourni et ainsi le rendre plus rapide et plus efficace pour ainsi pouvoir exécuter le mode challenge rapidement. Nous détaillerons au fur et à mesure du rapport les résultats obtenus pour chacun des codes implémentés.

Voici les commandes utilisées pour la connexion:
ssh amastor@access.grid5000.fr -l nancy
ssh nancy

Celle utilisée pour le transfert de fichier:
scp "nom du fichier".c amastor@access.grid5000.fr:nancy/

La commande de réservation de ressources sur Grid 5000:
oarsub -I -l nodes=6 -t allow_classic_ssh -p "cluster='gros'"
Nous ajustons le nombre de nodes lorsque nous voulons plus de coeurs, notamment pour le mode NORMAL

Les commandes de compilaiton et d'exécution:
mpicc "nom du fichier".c -o "nom de l'executable" -lm -O3 -Wall -lm
mpiexec -n "nombre de processus" -hostfile $OAR_NODEFILE ./"nom de l'executable"

Une fois executé on retrouve les résultats dans le fichier res.txt

Pour commencer, évaluons les temps d'exécution du code séquentiel afin de pouvoir le comparer avec les codes parallélisés.

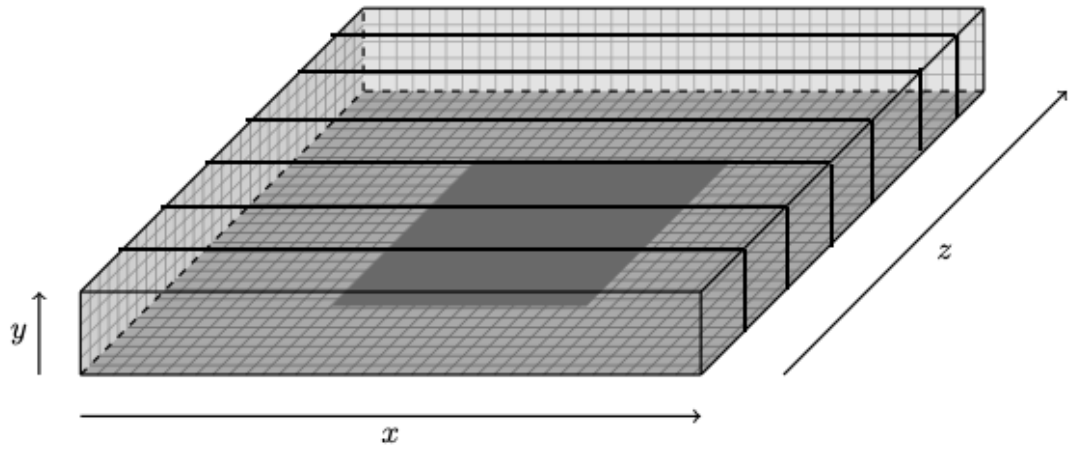
Nous commençons par une simple comparaison des temps d'exécution pour les paramètres ALUMINIUM.

	Fast maxproc= 30	Medium maxproc= 60	Normal maxproc=120	Challenge maxproc =220
Code séquentiel	6 sec	2 min 30	trop long	Trop long
Division en tranches	2 proc→ 3.71 sec 15 proc→ 0.78 sec 30 proc→ 0.41 sec	4 proc→30sec/28 15proc→11sec/16 60 proc→ 2,1 sec/	15proc →157.22 sec 40 proc→ 29 sec 120→ 19,8 sec	Trop long
Division en tranches avec communications non blocantes	2 proc→3.72 sec 15 proc → 0.82 sec 30 proc→ 0.5 sec	4 proc→ 30.1 sec 15 proc → 11.01 sec 60proc→ 3,2 sec	15proc→ 158.79 sec 40 proc→ 24,4 sec 120 proc-_>30.1 sec	Trop long
Division en tranches avec communications non blocantes et checkpoint checkpoint_intervalle =100	2proc→3.1sec 15 proc→ 0.69 sec 30proc→ 1.8 sec	4 proc→ 21 sec 15 proc→6.5 sec 60proc→ 6.1sec	15 prc→ 127sec 40 proc→ 52 sec 120proc→36,2 sec	Trop long

En premier lieu nous avons divisé en tranches le Cpu pour ainsi attribuer à chaque processus un bloc comme nous pouvons le voir dans le schéma. Le temps d'exécution est alors bien plus inférieur qu'en séquentiel. Pour un temps d'exécution de 2min30 en médium, on passe alors à environ 30 sec pour 4 coeurs et 1,7 sec pour 60 coeurs. Plus le nombre de processus augmente, plus le temps d'exécution diminue et ce dans toutes les versions de nos codes. En effet, les parties à traiter pour chaque processus sont plus petites et l'étape d'échange des données entre

chaque processus prend moins de temps à s'executer que le calcul des températures pour chacun des blocs. Cependant, on remarque qu'au delà d'un certain nombre de coeurs (appelés max proc dans notre cas) , le code ne s'execute plus et affiche l'erreur de segmentation suivante : mpiexec noticed that process rank 17 with PID 0 on node gros-122 exited on signal 11 (Segmentation fault).

Nous avions alors supposé qu'il s'agissait d'un soucis d'accès à la mémoire en raison de processus concurrents et avons tenté l'utilisation de MPI_Barrier et MPI_Allgather, mais cela n'a pas changé l'erreur et a au contraire augmenté le temps d'execution. Nous avons ensuite supposé une erreur dû à un overflow. En effet, le calcul effectué dans nos boucles dépassait la limite et nous donnait ainsi un résultat infini. Malgré ce changement et l'utilisation de Valgrind pour detecter une quelconque erreur de mémoire, l'erreur de segmentation deumera.



En suivant les indications du sujet nous avons alors implémenté un code permettant d'avoir un checkpoint. Ajoutant des étapes à l'exécution sans modifier l'organisation des processus parallèles, on a pu constater que le temps d'exécution du code est plus long que la simple décomposition en 1D . Également, nous remarquions une différence de temps d'exécution en fonction de l'intervalle du checkpoint. Si on enregistre les donnés à une petite fréquence (avec checkpoint_intervalle=10 par exemple) , on perd alors bien plus de temps que si on les enregistre à une plus grande fréquence (avec checkpoint_intervalle=100), l'enregistrement de checkpoints implique des opérations d'entrée/sortie pour écrire des données sur le stockage dans les fichiers de sorties. En augmentant donc l'intervalle entre les checkpoints, on réduit la fréquence de ces opérations coûteuses, ce qui entraîne une réduction globale de l'overhead. Donc malgré l'utilité de cette option qui permet

de ne pas repartir de 0 à chaque exécution, elle ne permet pas de réduire globalement le temps d'exécution total.

Aussi, nous avons tenté l'utilisation de communications non blocantes avec MPI_Issend et MPI_Irecv. Mais on ne peut pas conclure une nette amélioration, mais plutôt des temps totaux relativement égaux avec en génaral un temps d'exécution inférieur mais on retrouve certaines exceptions notamment pour les valeurs de maxproc. En théorie, l'utilisation des communication non blocantes aurait dû entraîner une baisse du temps total car elle permet normalement l'exécution de tâches en parallèle qui ne sont pas possible avec des communications blocantes, cependant on voit que ce n'est pas le cas dans notre code, on peut supposer que cela soit dû à une complexité supplémentaire dans le code que nous n'avons peut être pas réussi à cibler.

En ce qui concerne le mode Challenge, nous n'avons malheureusement pas pu l'executer jusqu'à convergence totale.Le temps d'exécution étant bien trop long. La division en tranches étant notre premier choix, nous avons bien évidemment tenté la division en 2D puis 3D mais notre code (que vous pouvez trouver dans le dossier) n'aboutit pas au résultat escompté. Par manque de temps et quelques soucis de compréhension nous n'avons donc pas pû résoudre les problèmes du code 3D.c qui s'execute mais n'effectue pas de convergence et reste constant sur la température 20°C.