Simulation d'une colonisation d'une galaxie
IN203 Programmation parallèle
Projet

Yufei HU
18 janvier 2021

CONTENU

sommaire

01 Parallélisation en mémoire partagée

OpenMp

OpenMp

J'ai utilisé OpenMp pour effectuer deux parties d'accélération. Un partie est l'affichage du résultat. l'autre est calculation d'image.

- J'ai ajouté OpenMp dans la fonction de mise_a_jour. S'on ajoute seulement la commande " #pragma omp parallel for " , cela va provoquer des conflit de donnés. Après avoir partagé les variables **galaxie_previous** et **galaxie_next**, plusieurs threads peuvent fonctionner à différentes adresse en même temps. Il n'y aura donc pas de conflits de données.

```
void mise_a_jour(const parametres& params, int width, int height, const char* galaxie_previous, char* galaxie_next)
{
...
 # pragma omp parallel for private(i,j) shared(galaxie_previous,galaxie_next) schedule(dynamic) m_threads(THREADS)
   for ( i = 0; i < height; ++i )
    {
      for ( j = 0; j < width; ++j )
      {
...
}
```

- THREADS=16

01 Parallélisation en mémoire partagée

OpenMp

OpenMp

J'ai utilisé OpenMp pour effectuer deux parties d'accélération. Un partie est l'affichage du résultat. l'autre est calculation d'image.

- J'ai ajouté OpenMp dans la fonction de mise_a_jour. S'on ajoute seulement la commande " #pragma omp parallel for " , cela va provoquer des conflit de donnés. Après avoir partagé les variables **galaxie_previous** et **galaxie_next**, plusieurs threads peuvent fonctionner à différentes adresse en même temps. Il n'y aura donc pas de conflits de données.

num_threads = 16	CPU(ms) : calcul
sans OpenMP	46.521
avec OpenMp	5.291
speedup=8.79	

- THREADS=16

01 Parallélisation en mémoire partagée

OpenMp

OpenMp

J'ai utilisé OpenMp pour effectuer deux parties d'accélération. Un partie est l'affichage du résultat. l'autre est calculation d'image.

- J'ai ajouté OpenMp dans la fonction de galaxie_renderer::render. S'on ajoute seulement la commande " # pragma omp parallel for private(i,j) shared(data) um_threads(THREADS) ", cela va provoquer des conflit de donnés. La concurrence des données se produit lors de l'accès à la fonction de rend_planete_habitee et rend_planete_inhabitable. Ils doivent être accédés par une fonction en même temps.

```
galaxie_renderer::render(const galaxie& g){
...
 # pragma omp parallel for private(i,j) shared(data) num_threads(THREADS)
   for (i = 0; i < height; ++i )
     for (j = 0; j < width; ++j ){
     ...
     #pragma omp critical
     rend_planete_habitee(j, i);
     #pragma omp critical
     rend_planete_inhabitable(j, i);
...
}
```

- THREADS=16

01 Parallélisation en mémoire partagée

OpenMp

OpenMp

J'ai utilisé OpenMp pour effectuer deux parties d'accélération. Un partie est l'affichage du résultat. l'autre est calculation d'image.

- J'ai ajouté OpenMp dans la fonction de galaxie_renderer::render. S'on ajoute seulement la commande " # pragma omp parallel for private(i,j) shared(data) um_threads(THREADS) ", cela va provoquer des conflit de donnés. La concurrence des données se produit lors de l'accès à la fonction de rend_planete_habitee et rend_planete_inhabitable. Ils doivent être accédés par une fonction en même temps.

num_threads = 16	affichage(ms)
sans OpenMP	0.645
avec OpenMp	0.445
speedup=1.45	

- THREADS=16

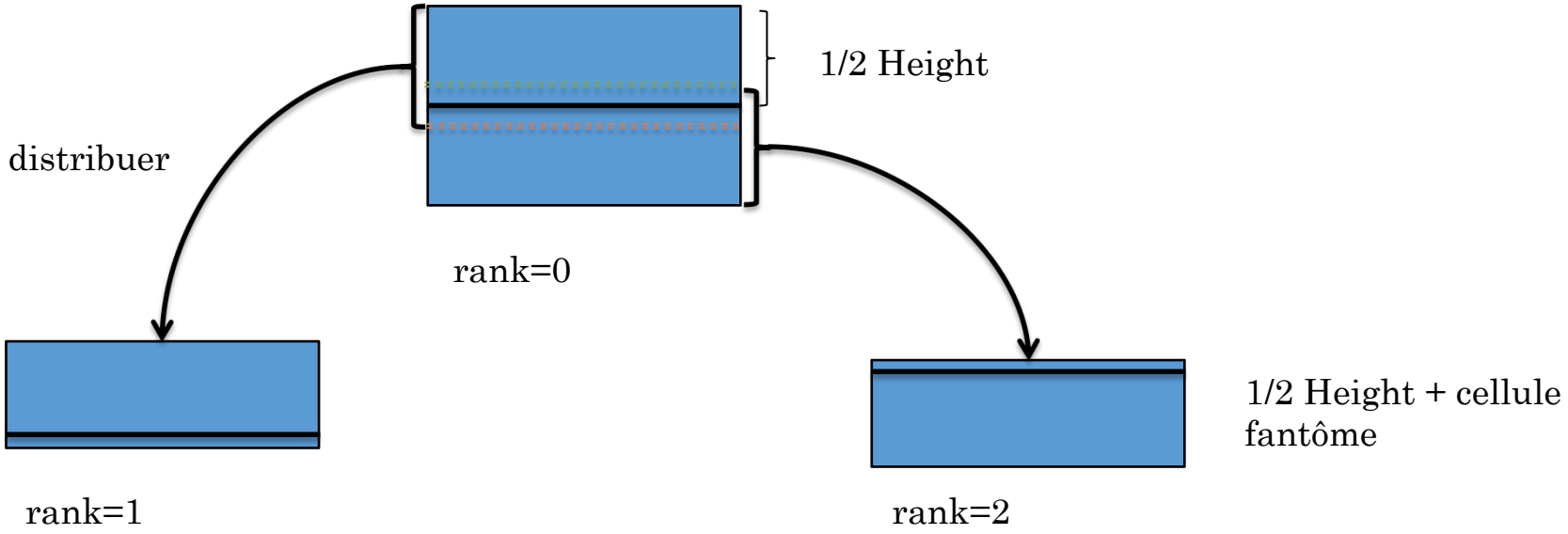
02 Parallélisation en mémoire distribuée

MPI

MPI

La méthode de conception MPI que j'ai utilisée ici est peu différente.

Le thread zéro est responsable de l'envoi et de la réception des données et de l'affichage. Les autres threads sont responsables du calcul de leurs fonctions de mise à jour respectives. Quand les threads sont égal à 3, la distribution des tâches est la suivante.



02 Parallélisation en mémoire distribuée

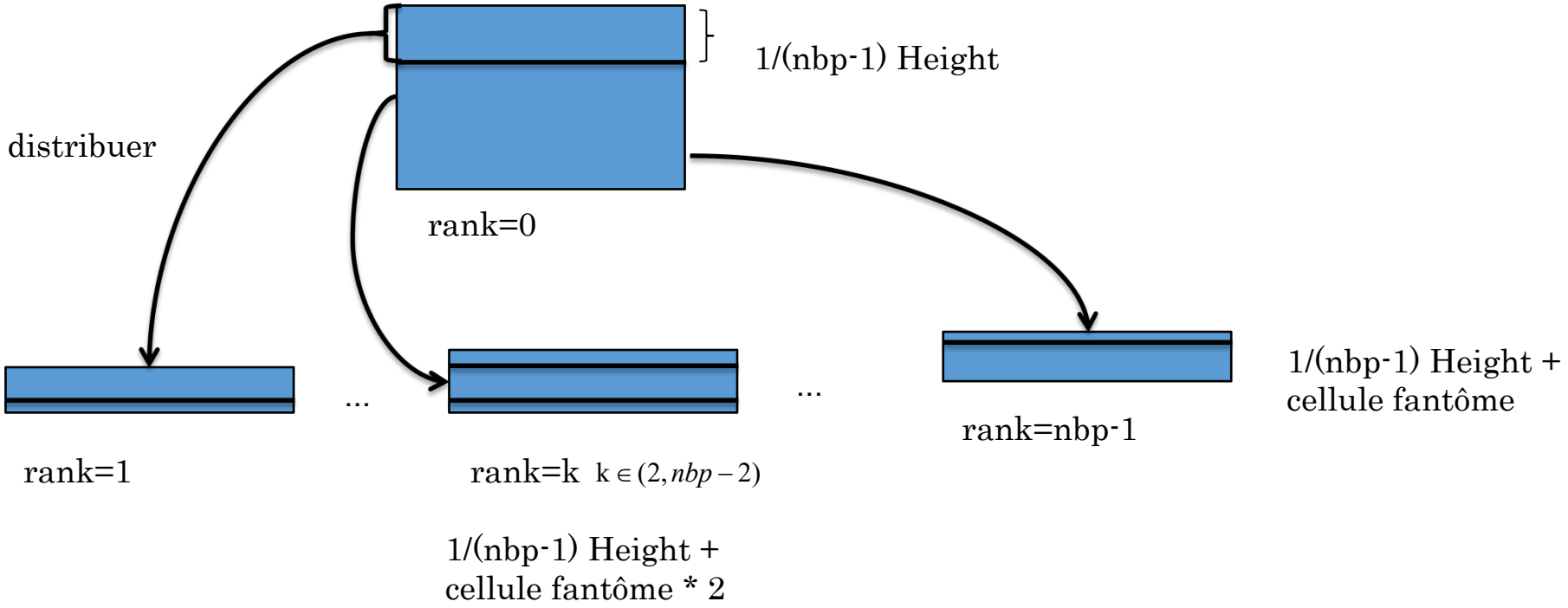
MPI

MPI

La méthode de conception MPI que j'ai utilisée ici est peu différente de l'idées fourni par vous.

Le thread zéro est responsable de l'envoi et de la réception des données et de l'affichage. Les autres threads sont responsables du calcul de leurs fonctions de mise à jour respectives.

Lorsqu'il y a plus de 3 threads, la distribution des tâches est la suivante.



02 Parallélisation en mémoire distribuée

MPI

MPI

La méthode de conception MPI que j'ai utilisée ici est différente de l'idées fourni par l'enseignant.

Le thread zéro est responsable de l'envoi et de la réception des données et de l'affichage. Les autres threads sont responsables du calcul de leurs fonctions de mise à jour respectives.

Les résultats expérimentaux sont présentés dans le tableau suivant.

	CPU(ms) : calcul
sans OpenMP	46.521
MPI+Threads(3)	26.946
speedup=1.73	
MPI+Threads(5)	14.488
speedup=3.21	

03 MPI&OpenMp

MPI&OMP

Quand je considère les deux cas en même temps, le compilateur devient mpic++. Lors de la manipulation des données, j'ai également réalisé OpenMp pour qu'ils accélèrent. Les commandes d'exécution sont les suivantes.

```
export OMP_NUM_THREADS=16
mpirun -np 3 ./colonisation.exe ou
mpirun -np 5 ./colonisation.exe
```

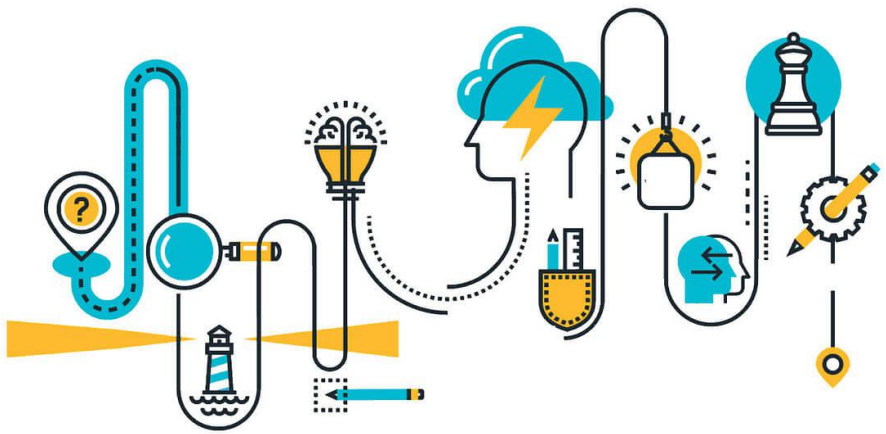
	CPU(ms) : calcul
sans OpenMP et MPI	46.521
MPI+Threads(3)	26.946
speedup =1.73	
MPI+Threads(5)	14.488
speedup=3.21	
MPI+Threads(3)+OMP	7.362
speedup=6.32	
MPI+Threads(5)+OMP	8.597
speedup=5.41	

- THREADS=16

04 Conclusion

En conclusion:

- Lorsque j'utilise MPI ou OMP seulement, je peux obtenir des résultats idéaux.
- Dans le cas de la combinaison de MPI et OMP, l'effet est amélioré par rapport à l'utilisation de MPI seul, mais l'effet est pire que celui de l'utilisation d'OMP uniquement. Je suis curieux de savoir si cela est normal ou si cela est dû à ma mauvaise utilisation.



Configuration

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	39 bits physical, 48 bits virtual
CPU(s):	12
On-line CPU(s) list:	0-11
Thread(s) per core:	2
Core(s) per socket:	6
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	158
Model name:	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
Stepping:	10
CPU MHz:	800.082
CPU max MHz:	4100,0000
CPU min MHz:	800,0000
BogoMIPS:	4399.99
Virtualization:	VT-x
L1d cache:	192 KiB
L1i cache:	192 KiB
L2 cache:	1,5 MiB
L3 cache:	9 MiB
NUMA node0 CPU(s):	0-11