Modèles et techniques en programmation parallèle hybride et multi-cœurs - TP1

Pierrick Guichard

Novembre 2020

1 Introduction

On se propose dans ce projet de résoudre le problème de la chaleur en 3 dimensions :

$$\begin{aligned} &\textit{Trouver } u : (x,t) \mapsto u(x,t) \textit{ avec } x \in \mathbb{R}^3 \textit{ et } t \geq 0 \textit{ tel que} \\ &\qquad \frac{\partial u}{\partial t} = \Delta u + f(x,t) \qquad \textit{dans } \Omega \\ &\qquad u(x,0) = g(x) \qquad \textit{dans } \Omega \\ &\qquad u(x,t) = g(x) \qquad \textit{sur } \partial\Omega,\ t \geq 0 \end{aligned}$$

Un code séquentiel est fourni et nous nous proposons de le paralléliser en OpenMP avec la technique de "grain fin" (parallélisation des boucles de l'algorithme sans modifications importantes de l'algorithme séquentiel) et de "grain grossier" (découpage du domaine de calcul en plusieurs sous-domaines, chaque sous-domaine étant affecté à un thread).

2 Parallélisme de type "grain fin"

Pour réaliser cette stratégie de parallélisme on procède de la manière suivante.

- On repère les boucles comptant un nombre d'itérations suffisantes. Lorsque plusieurs boucles sont imbriquées, on privilégie la parallélisation de la boucle la plus externe.
- On repère les variables privées de ces boucles.
- On parallélise en ajoutant des pragmas aux bons endroits.

Cette méthode a pour avantage qu'elle est facile à mettre en oeuvre. Il suffit de bien prêter attention aux variables qui doivent rester privées et aux éventuelles réductions. Dans ce cas précis, on a utilisé une réduction dans le fichier *scheme.cxx* pour s'assurer que la variable qui permet de stocker la somme est incrémentée par un thread unique à la fois.

Néanmoins, cette méthode a pour désavantage qu'elle multiplie les régions parallèles ce qui diminue le gain de temps en exécutant sur plusieurs threads. En effet, le temps d'activation/ désactivation des threads n'est pas négligeable donc il est important de réduire autant que possible le nombre de régions parallèles et d'augmenter la taille de chaque région parallèle.

3 Parallélisme de type "grain grossier"

Pour mettre en oeuvre cette méthode il suffit de subdiviser le domaine de calcul en sous-domaines que l'on répartie entre les différents threads disponibles. On procède de la manière suivante :

- Création d'une région parallèle autour de la boucle en temps. Ce sera la seule région parallèle du code.
- Dans la boucle temporelle, la mise à jour de la solution ($C.iteration()$) se parallélise. Les autres instructions ne s'éxécutente pas en parallèle et nécessitent d'être prises en charge par un seul thread ($\#pragma\ omp\ single$).

- Il faut ensuite veiller à répartir correctement les points du sous domaine en fonction du numéro du thread courant obtenu avec l'instruction *omp_get_thread_num*().
- Il faut enfin mettre en oeuvre des barrières et des régions critiques de manière à ce que d'une part, tous les threads mettent à jour la solution nouvelle à partir de l'ancienne solution (et non de la nouvelle) et que d'autre part la variation soit calculée de manière correcte.

4 Performances

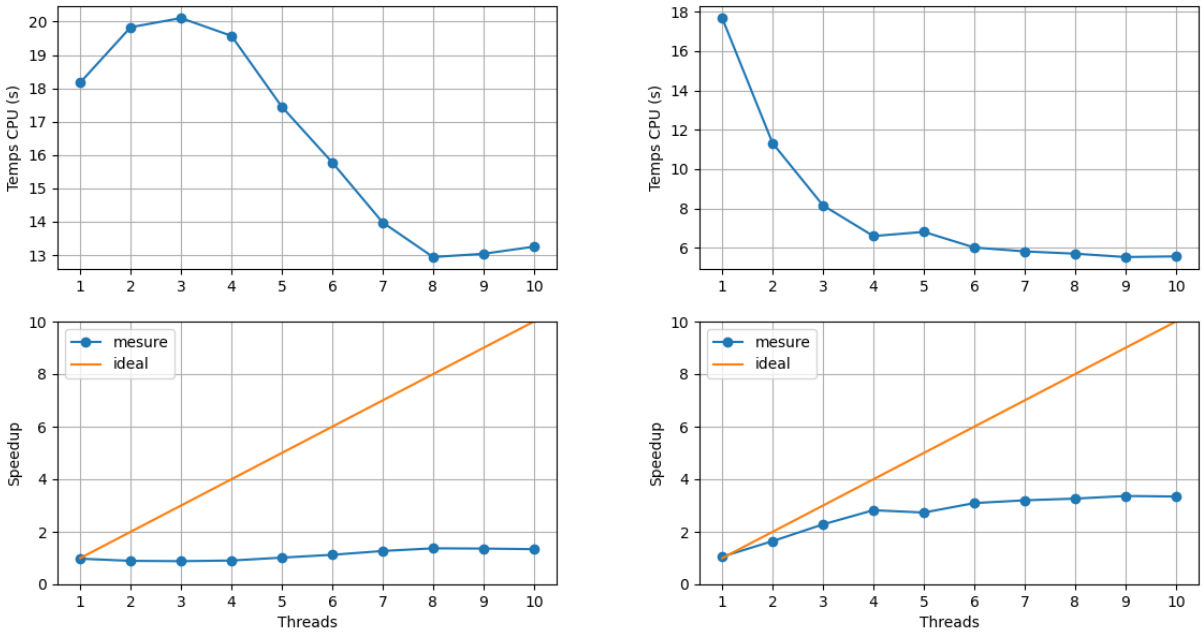

FIGURE 1 – Temps d'exécution et speedup pour le code de type "grain fin" (à gauche) et pour le code de type "grain grossier" (à droite), avec 400 points de calcul dans chaque direction.

On remarque que le parallélisme de type "grain fin" ne permet pas d'obtenir de bonnes performances pour un faible nombre de threads. On peut avancer comme hypothèse que le temps de création/destruction des threads dans les multiples régions parallèles est plus important que le gain de temps obtenu par la répartition des calculs des boucles. D'autre part, on observe un maximum de speedup pour 8 threads (égal au nombre de threads virtuels du processeur), ce qui est conforme à ce qui était attendu (chaque thread supplémentaire sera exécuté en série des 8 premiers threads).

En revanche le parallélisme de type "grain grossier" permet d'obtenir un meilleur speedup, qui plafonne cependant pour un nombre de threads élevés. Cette parallélisation est meilleure que la précédente car il n'y a qu'une région parallèle, les threads sont créés et détruits une seule fois. Le plafonnement des performances pour un nombre important de threads et probablement du au fait que le temps d'exécution de la partie séquentielle du code n'est plus négligeable par rapport à la partie parallèle (loi d'Amdahl).

5 Conclusion

Ce projet nous a permis de comparer 2 stratégies de parallélisation. La méthode dite "de grain fin" est plus facile à implémenter que la méthode "de grain grossier", qui lui est cependant préférable d'un point de vue du temps de calcul.

6 Annexe

6.1 Erreurs dans le code ?

Je pense qu'il y a une erreur de frappe dans les code fourni, sans importance pour le projet cependant. Dans le fichier *scheme.cxx*, ligne 63 du code séquentiel, dans la fonction *boolScheme* :: *iteration*(), je pense qu'il faut écrire $m\_P.imin(2), m\_P.imax(2))$ au lieu de $m\_P.imin(2), m\_P.imax(1))$.

- Version actuelle :

```
bool Scheme::iteration()
{

  m_duv = iteration_domaine(
      m_P.imin(0), m_P.imax(0),
      m_P.imin(1), m_P.imax(1),
      m_P.imin(2), m_P.imax(1));

  m_t += m_dt;
  m_u.swap(m_v);

  return true;
}
```

- Version que je pense être correcte :

```
bool Scheme::iteration()
{

  m_duv = iteration_domaine(
      m_P.imin(0), m_P.imax(0),
      m_P.imin(1), m_P.imax(1),
      m_P.imin(2), m_P.imax(2));

  m_t += m_dt;
  m_u.swap(m_v);

  return true;
}
```

Je n'ai pas fait la modification car cela n'affecte pas le projet en lui-même.

6.2 Spécifications techniques de l'ordinateur : retour de la commande *cat/proc/cpuinfo*

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.033
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.037
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 1
cpu cores       : 4
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.123
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 2
cpu cores       : 4
apicid          : 4
initial apicid  : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1098.143
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 6
initial apicid  : 6
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 4
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.053
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 5
vendor_id       : GenuineIntel
cpu family      : 6
```

```
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.024
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 1
cpu cores       : 4
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 6
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.055
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 2
cpu cores       : 4
apicid          : 5
initial apicid  : 5
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
stepping        : 13
microcode       : 0xde
cpu MHz         : 1100.072
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 7
initial apicid  : 7
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit srbds
bogomips        : 4800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```