Programmation comparée

Calcul parallèle : OpenMP vs. Cilk vs. ThreadBuildingBlocks

Réalisé par :

- Boussaad BOUDJEMA

1. L'intérêt du calcul parallèle

Depuis les années 1960, la densité des transistors dans un microprocesseur double tous les 18 à 24 mois, selon la loi de Moore. Mais des limitations physiques et des problèmes liés à la consommation énergétique ont fait que cette loi est de plus en plus difficile à maintenir. C'est pourquoi de nouvelles solutions ont dû être trouvées pour augmenter le vitesse de calcul sans augmenter la fréquence d'un CPU.

D'où l'intérêt du calcul parallèle, qui a pour objectif d'augmenter le nombre de processeurs permet de réduire le temps de calcul sans augmenter la fréquence de chaque processeur. Et parmi ces solutions, on retrouve un nombre de libraires. **OpenMp**, **Cilk**, **ThreadBuildingBlocks** sont parmi les plus connues.

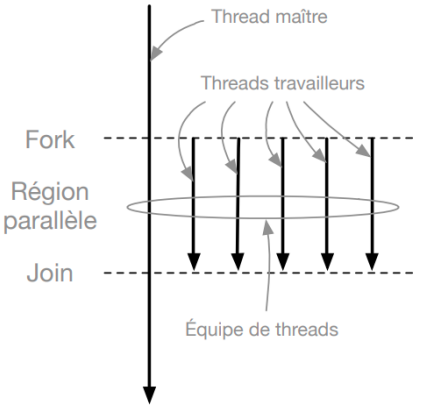
2. OpenMp

OpenMP (Open Multi-Processing) est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Le développement de la spécification OpenMP est géré par le consortium *OpenMP Architecture Review Board*. Cette API est prise en charge par de nombreuses plateformes, incluant GNU/Linux, OS X et Windows. La librairie permet d'accélérer les calculs sans devoir gérer les threads à la main. OpenMP 1.0 pour C/C++ a été publié en octobre 1998. La version actuelle est 5.1 sortie en 2020

2.1. Modèle d'exécution OpenMP

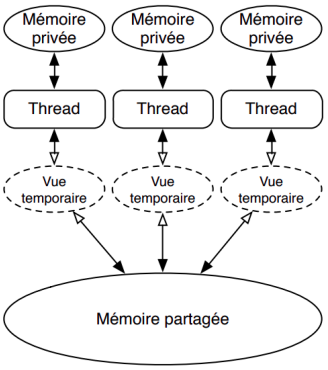
L'utilisateur introduit des directives établissant des régions parallèles. Durant l'exécution, leur Comportement respecte le modèle fork-join :

- Le thread maître crée des threads travailleurs et forme une équipe avec eux
- Les threads travailleurs se terminent avec la région parallèle
- Le thread maître continue son exécution**.1.**



2.2. Modèle mémoire OpenMP

Tous les threads ont accès à la même mémoire partagée. Chaque thread possède sa propre mémoire. Les données privées sont accessibles seulement par le thread correspondant. Les transferts de données sont transparents pour le programmeur.



2.3. Principales directives OpenMP

- **Construction de régions parallèles**
  - **Parallel** : crée une région parallèle sur le modèle fork-join
- **Partage du travail**
  - **for** : partage des itérations d'une boucle parallèle
  - **sections** : définit des blocs à exécuter en parallèle
  - **single** : déclare un bloc à exécuter par un seul thread
- **Synchronisation**
  - **master** : déclare un bloc à exécuter par le thread maître
  - **critical** : bloc à n'exécuter qu'un thread à la fois
  - **atomic** : instruction dont l'écriture mémoire est atomique
  - **barrier** : attente que tous les threads arrivent à ce point
- **Gestion de tâches**
  - **task** : déclaration d'une tâche fille
  - **taskwait** : attente de la fin des tâches filles

2.4. Format des directives OpenMP

#pragma omp directive [clause [clause] . . .]

- La sentinelle : #pragma omp
- Un nom de directive valide
- Une liste de clauses optionnelles (infos supplémentaires)

3. Cilk

Cilk, **Cilk++, Cilk Plus** est une librairie conçue pour le calcul parallèle multithread basées sur les langages de programmation C et C++. Développé à l'origine dans les années 1990 au Massachusetts Institute of Technology (MIT). Cilk a ensuite été commercialisé sous le nom de Cilk++ par une société dérivée, **Cilk Arts**. Cette société a ensuite été acquise par Intel, qui a augmenté la compatibilité avec le code C et C++ existant, appelant le résultat **Cilk Plus**. Intel ait cessé de prendre en charge Cilk Plus en 2017, le MIT développe à nouveau Cilk sous la forme d'**OpenCilk.** OpenCilk fournit toutes les fonctionnalités parallèles aux tâches d'Intel Cilk Plus.

3.1. Méthodes principales de Cilk

- **cilk_spawn func()** : Permet de créer un thread pour exécuter la fonction func()
- **cilk_sync** : Attend que les threads créés terminent
- **cilk_for** : Permet de paralléliser une boucle for

4. TBB (ThreadBuildingBlocks)

Développée par Intel et utilise uniquement c++.

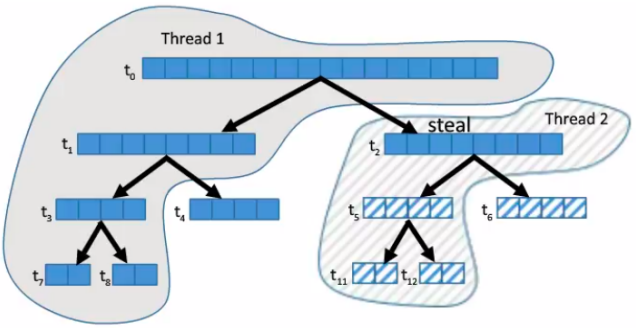
La librairie contient :

- **Algorithmes de bases** : parallel_for, parallel_reduce
- **Algorithmes avancés** : parallel_sort
- **Conteneurs** : concurrent_queue, concurrent_priority_queue, concurrent_vector, concurrent_hash_map
- **Allocation de mémoire** : scalable_malloc
- **Exclusion mutuelle** : mutex
- **Task scheduler (Planificateur de tâches)**: contrôle la création et l'activation des tâches
-

4.1. Work Stealing (mode de fonctionnement de Cilk et TBB)

Pour optimiser leur travail, les processeurs appliquent l'algorithme du Work Stealing:

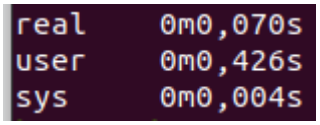
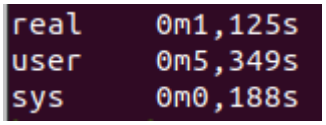
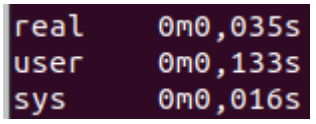
Quand un programme appelle une fonction qui nécessite un travail parallèle. Le processeur maître va découper le thread principal en sous thread et va les mettre dans une file de thread. Quand un autre processeur n'a plus de travail à faire, il vient voler la tâche du processeur maître, l'exécuter et quand il termine, il la réinsère dans à la file des thread. Cela permet de ne pas avoir des processeurs qui attendent à ne rien faire.



Remarques relatives à la présentation :

- Dans l'exemple avec Cilk, un seul appel à **cilk_spawn** aurait suffit vu qu'on a uniquement besoin de deux thread, donc le thread principal et autre thread qu'on aurait créé avec **cilk_spawn.**

Comparaison du temps pris par chaque algorithme en calculant fib(30)

TBB	OpenMp	Cilk
		

- Pour écrire le programme de Fibonacci, TBB était le plus long et le plus complexe parmi les trois. Cependant TBB offre beaucoup plus de fonctionnalités que les autres librairies et serait donc plus utile dans des programmes plus complexes.
- Cilk est la plus facile à utiliser et la moins complexe.

Pour avoir des résultats précis une études a été menée pour comparer la performance de trois libraries (**Comparison of Three Popular Parallel Programming Models on the Intel Xeon Phi Ashkan Tousimojarad and Wim Vanderbauwhede**) en utilisant différent algorithmes et voici un résumé du résultat.

Benchmark	OpenMP	Cilk plus	TBB
Fibonacci	50%	16%	5%
MergeSort	78%	81%	3%
MatMul	22%	6%	1%

- Les résultats montrent que TBB a généralement le plus petit temps. Mais ce n'est pas toujours le cas, cela dépend du programme et sa complexité, comme on a pu le voir dans l'exemple avec FIB.

Références :

Wikipedia

MIT OpenCourseWar

Comparison of Three Popular Parallel Programming Models on the Intel Xeon Phi Ashkan Tousimojarad and Wim Vanderbauwhede - School of Computing Science, University of Glasgow, Glasgow, UK

connect.ed-diamond.com