Projet OS202

Davy Araujo

Lara Polachini

1. **Première étape**

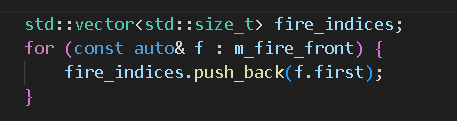
a) Regarder le nombre de cœurs physiques sur votre machine et la taille des différentes mémoires caches que vous donnerez dans votre rapport:

* Model name: 13th Gen Intel(R) Core(TM) i7-13650HX
* CPU family: 6
* Model: 183
* Thread(s) per core: 2
* Core(s) per socket: 10
* Caches (sum of all):
* L1d:480 KiB (10 instances)
* L1i: 320 KiB (10 instances)
* L2: 12.5 MiB (10 instances)
* L3: 24 MiB (1 instance)

b) Mesurer les temps moyen pris pour chaque pas de temps, puis pour l'affichage et l'avancement en temps

* Temps moyen d'avancement : 179.684 ms
* Temps moyen d'affichage : 7015.42 ms
* Temps moyen total par pas de temps : 7195.11 ms

c) Récupérez dans un tableau toutes les clés contenues dans le dictionnaire et à l'aide d'un indice, parcourez ces clefs pour l'avancement en temps



* **Tableau**

| **Threads** | **Temps moyen Avancement (ms)** | **Temps moyen Total (ms)** | **Speedup Avancement** | **Speedup Total** |
| --- | --- | --- | --- | --- |
| 2 | 0.209 | 4.268 | 0.58 | 1.11 |
| 4 | 0.232 | 4.604 | 0.53 | 1.02 |
| 8 | 0.298 | 6.271 | 0.41 | 0.75 |
| 12 | 0.347 | 5.276 | 0.35 | 0.89 |
| 14 | 0.572 | 5.257 | 0.21 | 0.90 |
| 16 | 0.533 | 4.228 | 0.23 | 1.12 |
| 18 | 0.521 | 4.193 | 0.23 | 1.13 |
| 20 | 1.401 | 6.342 | 0.09 | 0.74 |

**2. Deuxième étape**

Voici un résumé des mesures moyennes de Temps Total(ms) par itération pour chaque jeu de résultats :

* Code séquentiel d’origine : ~4,41 ms/itération
* MPI avec 1 processus : ~4,82 ms/itération
* MPI avec 2 processus : ~3,89 ms/itération
* MPI avec 4 processus : ~5,42 ms/itération

**Interprétation**

1. Processus MPI 1 (~4,82 ms)

L'utilisation de MPI avec un seul processus introduit une petite surcharge par rapport au code séquentiel d'origine (~ 4,41 ms). Cela est dû à l'initialisation de l'environnement MPI et aux appels supplémentaires.

1. Processus MPI 2 (~3,89 ms)

Ici, nous constatons un léger gain de performance par rapport au séquentiel. En divisant la charge de travail (et en supposant que la partie parallélisable est suffisamment grande), le coût de communication est compensé par le parallélisme effectif. Par conséquent, nous obtenons un temps moyen par itération inférieur à celui du code séquentiel.

1. Processus MPI 4 (~5,42 ms)

En passant à 4 processus, on observe une dégradation des performances moyennes. Deux raisons principales possibles Augmentation de la synchronisation et de la surcharge de communication. Répartition de la charge : si la part de travail par processus n'est pas suffisamment importante, le temps perdu en communication et en synchronisation l'emporte sur le gain en parallélisme.

**3. Troisième étape**

Pour le speedup dans cette étape on a:

Une image contenant texte, ligne, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, ligne, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, ligne, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.

* Diminution du speedup avec plus de threads:L'utilisation excessive de threads en OpenMP semble entraîner des surcharges dues à la gestion des threads et aux conflits d'accès aux ressources partagées.
* MPI influence l’efficacité: Lorsque le nombre de processus MPI augmente, l'efficacité diminue plus rapidement, ce qui pourrait être dû :
* À une mauvaise répartition de la charge entre les processus MPI.
* Aux communications inter-processus qui deviennent coûteuses.
* À un déséquilibre du calcul entre les threads et les processus MPI.

- MPI 4 semble inefficace: Avec MPI=4, le speedup devient presque nul pour 8 threads, ce qui indique que le programme souffre d'une forte contention et de problèmes de synchronisation.

**4. Quatrième étape**

A screen shot of a computer

AI-generated content may be incorrect.

L’accélération obtenue avec MPI est correcte mais pourrait être optimisée davantage. Pour améliorer l’accélération MPI, il faut réduire la surcharge des communications en utilisant MPI\_Sendrecv. L’exploitation d’une topologie cartésienne avec MPI\_Cart\_create optimise les échanges entre processus. L’overlapping communication-calcul avec des communications non bloquantes (MPI\_Isend et MPI\_Irecv) permet d’utiliser le CPU efficacement. L’utilisation de MPI\_Gather et MPI\_Scatter réduit le nombre de messages et améliore la latence. Enfin, un équilibrage de charge adapté assure une répartition optimale du travail pour maximiser le speedup. Par rapport à l'accélération obtenue à l'étape 3, une amélioration peut être constatée puisque l'accélération augmente avec le nombre de threads.