Programmation Parallèle - MPI + OpenMP

Mouhamadou Mansour GUEYE

Mai 2024

1 MPI + OpenMP

Pour la version MPI, nous avons mis en place la fonction **spin_compute_mpi**. Au sein de cette fonction, nous avons tout d'abord initialisé MPI en utilisant **MPI_THREAD_FUNNELED**, afin que seuls les threads principaux de chaque processus fassent appel à MPI.

Ici, chaque processus doit traiter une partie de l'image de taille `DIM / size`. Nous avons une double boucle `for` qui calcule la partie locale de l'image. Ensuite, nous avons les échanges de bordures qui se font via une communication entre processus MPI. L'idée est la suivante : nous allouons des buffers pour les zones fantômes et de débordement, nous échangeons les zones fantômes entre les processus voisins. L'image est ensuite tournée dans le processus maître, cette rotation est diffusée aux autres processus via `MPI_Bcast`.

```
unsigned spin_compute_mpi(unsigned nb_iter)
{
    int rank, size;
    int provided;
    MPI_Init_thread(NULL, NULL, MPI_THREAD_FUNNELED, &provided);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int row_count = DIM / size;
    int row_start = rank * row_count;
    int row_end = (rank + 1) * row_count;

    unsigned* recv_top = (unsigned*)malloc(DIM * sizeof(unsigned));
    unsigned* recv_bottom = (unsigned*)malloc(DIM * sizeof(unsigned));

    for (unsigned it = 1; it <= nb_iter; it++)
    {
        #pragma omp parallel for collapse(2)
        for (int i = row_start; i < row_end; i++)
        {
            for (int j = 0; j < DIM; j++)
```

```
            {
                cur_img(i, j) = compute_color(i, j);
            }
        }

        if (size > 1)
        {
            if (rank < size - 1)
            {
                MPI_Send(&cur_img(row_end - 1, 0), DIM, MPI_UNSIGNED, rank + 1, 0, MPI_COMM_
                MPI_Recv(recv_bottom, DIM, MPI_UNSIGNED, rank + 1, 0, MPI_COMM_WORLD, MPI_ST
            }

            if (rank > 0)
            {
                MPI_Send(&cur_img(row_start, 0), DIM, MPI_UNSIGNED, rank - 1, 0, MPI_COMM_WO
                MPI_Recv(recv_top, DIM, MPI_UNSIGNED, rank - 1, 0, MPI_COMM_WORLD, MPI_STATU
            }
        }

        if (rank == 0)
        {
            rotate();
        }
        MPI_Bcast(&base_angle, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
    }

    free(recv_top);
    free(recv_bottom);

    MPI_Finalize();

    return 0;
}
```

Nous avons parallélisé la double boucle avec un `#pragma omp parallel for`, ce qui a permis de réduire le temps d'exécution de $\pm$34615.712 ms à $\pm$19306.836 ms.

Cependant, un problème spécifique a été rencontré : l'image obtenue est incomplète et la partie obtenue est proportionnelle au nombre de processus. C'est-à-dire que si nous utilisons `-np N`, nous obtenons seulement `1/N` de l'image complète. Cela indique qu'il y a probablement un problème de communication ou de synchronisation entre les processus, ou une mauvaise gestion des indices de début et de fin des parties d'image traitées par chaque processus.

Pour résoudre ce problème, il est crucial de vérifier la logique de décomposition de l'image entre les processus et de s'assurer que les données échangées entre

les processus voisins sont correctement synchronisées et intégrées dans l'image globale.