PROJET PPAR: Fast Fourier Transform

Assia MASTOR Amel MOKDADI

Introduction

Un programme C séquentiel `fft.c` nous a été fourni et nous offre la possibilité d'effectuer une Transformée de Fourier Rapide (FFT). Ce programme accomplit plusieurs étapes clés, notamment la génération de bruit blanc, le calcul d'une transformée de Fourier, l'application de transformations aux coefficients de Fourier, le calcul de la transformée de Fourier inverse et la production d'une partie du résultat sous forme d'un fichier audio. Le son obtenu est « amorti », évoquant alors les sons que l'on peut entendre sous l'eau.

L'objectif est de créer une version parallèle de ce code qui s'exécute aussi rapidement que possible sur un nœud de calcul unique. Nous avons alors utilisé OpenMp pour la parallélisation.

Compilation

Afin de compiler le code parallélisé nous utilisons Grid 5000 mais seulement sur un nœud, à l'aide des commandes suivantes.

Pour la connexion à Grid 5000:
ssh amastor@access.grid5000.fr -l nancy
ssh nancy

Pour le transfert de fichier:
scp "nom du fichier".c amastor@access.grid5000.fr:nancy/

Pour la commande de réservation de ressources sur Grid 5000:
oarsub -I -l nodes=1

Pour la compilation et l'execution :
gcc -o "nom_de_notre_fichier" "nom_de_notre_fichier".c -0s3 -ftree-vectorize -fopenmp -lm

./ "nom_de_notre_fichier" --size $((2**(taille voulue)))$

Nous ajoutons alors --output bruit.wav à l'execution afin d'avoir un fichier audio en sortie.

Pour faciliter la compilation nous avons alors créé un Makefile

Le code

Pour commencer, les boucles importantes du code ont été parallélisées à l'aide de #pragma omp parallel for notamment la boucle de la fonction FFT_rec, toutes les boucles for de FFT et celles du main.

Egalement, nous avons décidé dans la fonction FFT_rec, d'utiliser la directive #pragma omp parallel sections pour répartir le travail de manière parallèle entre deux sections pour accélérer la fft et diviser le calcul entre les sous-problèmes.

Pour la recherche du maximum dans le processus de normalisation, nous avons estimé que l'utilisation de la clause réduction pourrait être un moyen de réduire les résultats critiques. La valeur maximale est alors determinée de manière efficace, en réduisant les conflits d'accès et en assurant la cohérence des résultats parallèles.

Aussi, nous avons voulu tester la vectorisation sur la partie qui suit en utilisant simd cependant cela ralongeait le temps d'exécution et nous avons donc préféré le retirer.

```
#pragma omp simd
for (u64 i = 0; i < size; i++)
{
   double real = 2 * (PRF(seed, 0, i) * 5.42101086242752217e-20) - 1;
   double imag = 2 * (PRF(seed, 1, i) * 5.42101086242752217e-20) - 1;
   A[i] = real + imag * I;
}
```

Résultats

Size / Code	2 max= 4.62205e-113	2**4 max = 1.45165e-18	2**10 max = 4.01416e-05	2**15 max = 0.0493073	2**20 max= 60.5386
fft.c	0.000107 seconds	0.000116 seconds	0.001193 seconds	0.032193 seconds	1.143001 seconds
fft_paral.c	0.001834 seconds	0.001972 seconds	0.006395 seconds	0.104348 seconds	2.709760 seconds
rapport entre les codes	17.1	17	5.3	3.24	2.37

Analyse des résultats et critiques

Après avoir effectué de nombreuses modifications dans le code séquentiel nous avons alors comparé les différents temps d'executions en fonction de la taille de la fft et de la forme du code (séquentiel ou parallélisé avec OpenMp).
Nous constatons d'emblée que les temps d'executions sont bien plus élevés avec notre code parallélisé avec parfois bien plus du double du temps d'exécution en sequentiel. Ne comprenant pas particulièrement cette augmentation significative de temps , on pourrait sûrement expliquer ceci par un overhead ajouté par la gestion parallèle dans des cas où les calculs sont simples et peut être qu'un surcoût de parallélisation peut être significatif par rapport au travail effectif.
Ou bien simplement une erreur d'implémentation ou de compilation de notre part qui ne donne alors pas le résultat escompté.

Afin de comparer les temps d'executions nous avons également décidé de calculer un rapport entre le temps d'exécution du code séquentiel et le temps d'exécution du

code parallélisé. Nous constatons alors que plus la taille de la fft est elevé moins le rapport l'est. On passe alors d'une différence significative lorsque la taille est petite (par exemple 2^1) à une différence bien plus réduite lorsque la taille arrive à 2^20.

On peut en déduire que l'implémentation parallélisée est plus efficace pour les fft de taille conséquentes. On peut alors supposer qu'à un certain point, avec un 2 à la puissance élevée , la tendance s'inverse et la fft séquentiel deviendrait alors moins rapide que le code avec OpenMp. Nous n'avons pas tenté l'expérience car les temps d'executions étaient trop longs.