TDP 3 - PRCD
Problème à N corps avec MPI

Etienne THIERY - Youssef SEDRATI

24 novembre 2016

Dans ce projet nous implémentons un produit de matrice distribué.

1 Le produit distribué Fox

Pour distribuer le produit de matrice, on remanie la formule habituelle :
$C_{i,j} = \sum_{k=0}^{N-1} A_{i,k}B_{k,j}$

en
$C_{i,j} = \sum_{k=0}^{N-1} A_{i,(i+k)\%N}B_{(i+k)\%N,j}$

La raison est simple :

— Avec la formule habituelle, pour le calcul d'un membre de la somme, tous les processeurs en charge des blocs d'une même ligne ont besoin du même bloc $A_{i,k}$, et tout ceux en charge des blocs d'une même colonne ont besoin du même bloc $B_{k,j}$. Pour chaque k, Un broadcast par ligne et un broadcast par colonne sont donc nécessaires, grossièrement de coût $O(log(B))$ chacun, avec B nombre de blocs sur une ligne / une colonne.

— Avec la formule Fox, pour le calcul d'un membre de la somme, tous les processeurs en charge des blocs d'une même ligne ont besoin du même bloc $A_{i,(i+k)\%N}$, mais tout ceux en charge des blocs d'une même colonne ont besoin de blocs différents $B_{(i+k)\%N,j}$. Le broadcast par colonne peut ainsi être remplacé par un "shift" en temps constant pour une taille de bloc donnée, réduisant le coût de communication.

2 Utilisation

2.1 Compilation

La compilation du projet nécessite la présence d'Intel MKL et d'une implémentation d'MPI sur la machine (modules `intel/mkl/64/11.2/2016.0.0`, `intel-tbb-oss/intel64/43_20150424oss` et `mpi/openmpi/gcc/1.10.0-tm` sur PlaFRIM). L'ensemble du projet peut être compilé via la commande :

```
make
```

2.2 Exécution

Une fois compilé, l'exécutable principal (`distributedGemm`) doit être appelé avec 2 arguments, qui sont les chemins vers les fichiers contenant les 2 opérandes du produit de matrice.

```
mpiexec -n 4 ./distributedGemm leftOpFile rightOpFile
```

Le nombre de processus à lancer par MPI doit être un carré (4, 9, 16, ...). Les fichiers contenant les opérandes doivent avoir le format suivant, avec $M = N$ et M multiple de la racine du nombre de processus :

MN
$a_{1,1}\ a_{1,2}\ \ldots\ a_{1,N}$
$\ldots$
$a_{M,1}\ a_{M,2}\ \ldots\ aM,N$

De tels fichiers peuvent être générés aléatoirement grâce à l'utilitaire fourni `randomMatrixGenerator`. Pour générer une matrice aléatoire à coefficient entre -100 et 100, de taille $M \times N$, utiliser la commande :

```
./randomMatrixGenerator M N inputs/Mat_M_N
```

2.3 Tests

Un troisième argument facultatif peut être fourni à `distributedGemm` pour écrire la matrice résultante dans un fichier.

```
mpiexec -n 4 ./distributedGemm leftOpFile rightOpFile output/file
```

Le projet inclut également une version non distribué faisant simplement appel au produit de matrice de la MKL utilisable exactement de la même manière que `distributedGemm`.

On peut utiliser ces 2 fonctionnalités pour vérifier les résultats générés par notre implémentation du produit de matrice distribué de la manière suivante :

```
mpiexec -n 4 ./distributedGemm leftOpFile rightOpFile output/file1
./localGemm leftOpFile rightOpFile output/file2
diff outputFile1 outputFile2
```

2.4 Reproduction de l'étude de scalabilité

Les études de scalabilité dont les résultats sont représentés en section 4 sont reproducibles une fois connecté sur PlaFRIM en programmant les jobs du répertoire `jobs` :

```
sbatch funWithMPI2/jobs/job.sh
sbatch funWithMPI2/jobs/job2.sh
```

Il faut compter environ 1h30 pour chaque job. Une fois les jobs terminés, les courbes sont tracables avec gnuplot et les scripts du répertoire `figures` :

```
gnuplot figures/fig1.p
gnuplot figures/fig2.p
gnuplot figures/fig3.p
gnuplot figures/fig4.p
```

3 Architecture du projet

Les fichiers `util.c|h` contiennent quelques fonctions utilitaires pour lire et écrire des fichiers matrices.

Le fichier `randomMatrixGenerator.c` contient le code de génération de fichiers matrices aléatoires.

Le fichier `localGemm.c` contient le code de la version référence non distribué du produit de matrice, qui appelle seulement la MKL.

Le fichier `distributedGemm.c` contient notre implémentation du produit de matrice distribué. La méthode `parse` s'occupe de lire les arguments de la ligne de commande et de charger les matrices opérandes sur le processus 0. La structure `ScatterGatherInfo` contient tous les arguments nécessaire à la distribution des opérandes et au rassemblement du résultat, et est initialisé via la fonction `initScatterGatherInfo`. On a séparé la distribution de la matrice, le calcul et le rapatriement du résultat dans 3 fonctions différentes (`scatter`, `gemm` et`gather`) de façon à pouvoir mesurer leur temps d'exécution séparément. Enfin la méthode `writeResult` permet au processus

0 de stocker le résultat du calcul dans un fichier si le 3ème argument facultatif est fourni par l'utilisateur.

4 Étude de scalabilité

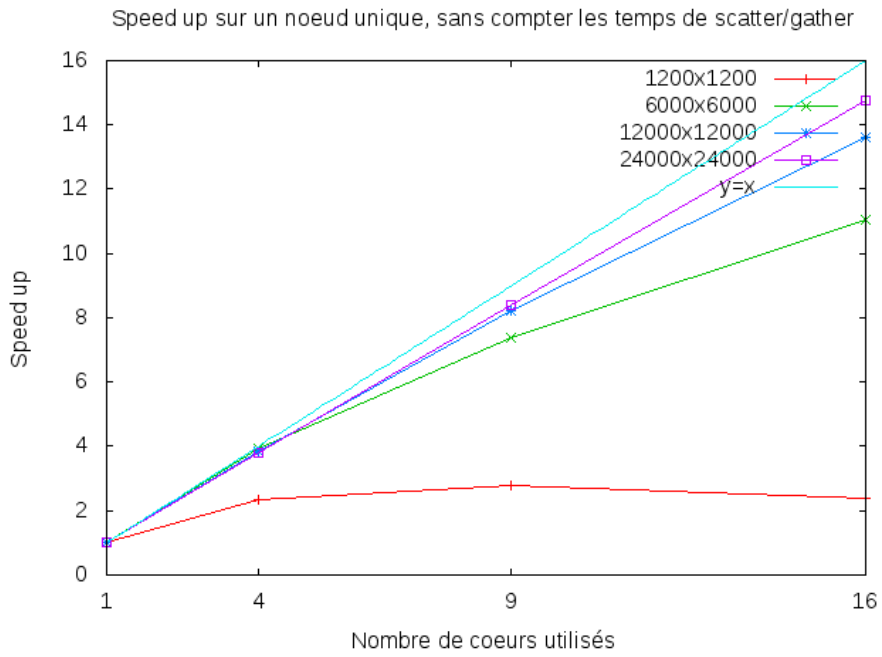
4.1 Sur un même hôte

Nous avons commencé par distribuer des produits de matrices de différentes tailles sur les coeurs d'une même machine. Les résultats obtenus sont représentés dans la figure suivante. Les temps de scatter et de gather n'ont pas été comptés, car ils sont assez faibles face au temps de calcul (qui croit quadratiquement avec le taille de la matrice) pour le nombre de processeurs utilisés. L'extrait de mesures suivant l'illustre pour un produit de matrices de taille 12000×12000.

Nombre de processeurs	temps de scatter (s)	temps de calcul (s)	temps de gather (s)
1	0	183.178410	0
4	1.067798	47.894501	0.266325
9	1.080910	22.225601	0.252229
16	1.076374	13.446110	0.252154

On note tout de même que les temps de scatter/gather restent quasi constants alors que le temps de calcul décroit, et qu'ils peuvent devenir génants pour des produits de matrices de taille trop petite comparée au nombre de processeurs utilisés.

FIGURE 1 – Temps de simulation en utilisant plus ou moins de coeurs d'une même machine



4.2 Sur plusieurs hôtes

On a ensuite voulu vérifier si le fait de distribuer le calcul sur des noeuds différents avait un impact sur le speedup. La courbe suivante montre que quasiment pas, malgré la latence supérieure lors des communications inter noeuds.

FIGURE 2 – Temps de simulation en utilisant plusieurs machines et 1 coeur par machine