Programmation Parallèle TP-3
Multiplication des Matrices en Parallèle avec l'Algorithme SUMMA

Oguz Kaya
oguz.kaya@lri.fr

23/12/2018

Scalable Universal Matrix Multiplication Algorithm (SUMMA) est l'un des algorithmes les plus populaires pour multiplier deux matrices en parallèle en mémoire distribuée. Pour simplifier les choses, on ne va multiplier que des matrices A et B de taille $N \times N$ afin d'obtenir une autre matrice $C = A \times B$ de la même taille. On va utiliser $P = p \times p$ processus pour $p \in \mathcal{Z}^+$ et supposer que p divise N. Dans l'algorithme SUMMA, les matrices A, B et C sont partitionnées en $p \times p$ sous-matrices de taille $(N/p) \times (N/p)$ chacune. Le processus ayant le rang r possède les sous-matrices correspondantes A_{ij}, B_{ij} et C_{ij} où $i = r/p$ et $j = r \mod p$ et est responsable pour effectuer le calcul

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

Par exemple, pour $P = 9$ (donc $p = 3$), C est partitionnée en des sous-matrices C_{00}, C_{01}, C_{02}, C_{10}, C_{11}, C_{12}, C_{20}, C_{21}, et C_{22} de taille $(N/3) \times (N/3)$ (idem pour A et B). Cette fois-ci, le processus P_5 contiendra A_{12} et B_{12} au départ et aura calculé le morceau C_{12} au final.

Algorithm 1 La méthode SUMMA pour la multiplication des matrices

Input: Matrices A, B, C de taille $N \times N$
$P = p \times p$ processus
Output: Calculer $C = AB$.

1: (Facultatif) Distribuer les matrices du processus racine (avec rang 0) tel que le processus de rang r possède les matrices A_{ij} et B_{ij} où $i = r/p$ et $j = r \mod p$.
2: **for** $k = 1 \dots p$ **do** ▶ L'étape k de communication et de calcul
3: Dans chaque ligne i, $1 \leq i \leq p$, broadcaster la matrice A_{ik} à tous les processus dans la meme ligne (c'est-à-dire ayant le rang r tel que $i = r/p$).
4: Dans chaque colonne j, $1 \leq j \leq p$, broadcaster la matrice B_{kj} à tous les processus dans la meme colonne (c'est-à-dire ayant le rang r tel que $j = r \mod p$).
5: Dans le processus r, effectuer le calcul locale $C_{ij} = C_{ij} + A_{ik} B_{kj}$ où $i = r/p$ et $j = r \mod p$.
6: (Facultatif) Recueillir la matrice C dans le processus racine.

L'algorithme SUMMA pour effectuer ce calcul est détaillé dans Algorithm 1. Cet algorithme s'effectue en p étapes. Dans l'étape k, les matrices A_{ik} et B_{kj} sont envoyé au processus responsable pour calculer C_{ij} afin de faire la mise-à-jour locale $C_{ij} = C_{ij} + A_{ik} B_{kj}$ dû à ces blocs dans ce processus. Cette communication est faite à l'aide d'un broadcast, car les processus ayant la même i (respectivement j) recevront le même bloc A_{ik} (respectivement B_{kj}). Ceci nécessite de créer des sous-communicateurs pour les processus avec le même valeur de i ou j (ou on pourrait également dire, dans la même ligne ou la même colonne de la grille des processus detaille $p \times p$). Créer un nouveau communicateur est assez simple avec la commande suivante:

```
int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)
```

- `comm` est le communicateur que l'on veux diviser.
- `color` est l'entier qui definit la couleur de chaque processus. A la fin, les processus ayant la même couleur seront dans le même communicateur `newcomm`.
- `key` est l'entier qui sert à déterminer le nouveau rang de chaque processus dans le nouveau communicateur. Si la `key` fournie par un processus est inferieure à celle d'un autre, son rang sera inferieur au rang de l'autre processus. En cas d'égalite de `key`, l'ordre des rangs dans `comm` sera retenu.

- `newcomm` est l'adresse du nouveau communicateur qui sera crée.

Noter que le communicateur de départ est toujours disponible à utiliser après l'avoir divisé en de plusieurs communicateurs (en l'occurrence, on va partitionner le `MPI_COMM_WORLD`).

Part 1

Multiplication des matrices avec des communicateurs de ligne et de colonne

Dans cette section, on va supposé que les sous-matrices A_{ij} et B_{ij} sont déjà distribuées qux processus.

Question 1

a) Comme mentionné, implanter Algorithm 1 nécessite de former des communicateurs pour chaque ligne et colonne de la grille des processus. On peut ce faire en divisant le communicateur `MPI_COMM_WORLD` adéquatement. Comment ce faire? Quelles valeurs de `color` et `key` faudrait-il fournir? Trouver-le et former un communicateur de ligne et un communicateur de colonne.

b) Pour simplifier les choses, on va eviter la distribution de la matrice globale en créant les matrices locales de taille $(N/p) \times (N/p)$ dans chaque processus directement à l'aide de la fonction suivante:

```
createMatrix(double **pmat, int nrows, int ncols, char *init).
```

On fournit un pointeur vers le pointeur qui contiendra l'adresse de la matrice créée. On précise également le nombre de lignes et de colonnes de la matrice. Il y a deux possibilités d'initialisation de la matrice. On pourrait fournir le mot "random" dans `init` pour remplir la matrice aléatoirement ou passer le mot "zero" pour mettre toutes les valeurs à zero. En utilisant cette fonction, on crée les matrices `Aloc`, `Bloc` et `Cloc` de taille $(N/p) \times (N/p)$ dans chaque processus. Veiller à ce que `Aloc` and `Bloc` soit initialisé aléatoirement et `Cloc` soit mise à zero.

c) On a la fonction

```
multiplyMatrix(double *a, double *b, double *c, int M)
```

afin de multiplier deux matrices et additioner ce résultat dans une troisième matrice. Particulièrement, on effectue l'operation $C \leftarrow C + AB$ où A, B et C sont des matrices de taille $M \times M$. A l'aide de cette fonction et les communicateurs de ligne et de colonne, réaliser l'algorithme SUMMA fournit dans l'Algorithm 1.

d) Maintenant, on va s'occuper de la distribution le rassamblement des matrices. On va modifier le code tel que les matrices globale de taille $N \times N$ sont crée dans le processus 0, puis les morceaux correspondant sont envoyé à chaque processus avec `MPI_Scatter`. Du même, une fois que C locale est calculée dans chaque processus, on va les mettre ensemble dans le processus 0 à l'aide de `MPI_Gather`.

N'oublions pas de sauvegarder nos codes qui pourront être utile après!