# Parallel and Distributed Algorithms and Programs
# TP n°4 - Parallel SUMMA Matrix-Matrix Multiplication

Oguz Kaya
oguz.kaya@ens-lyon.fr

Pierre Pradic
pierre.pradic@ens-lyon.fr

16/11/2016

Scalable Universal Matrix Multiplication Algorithm (SUMMA) est l'un des algorithmes les plus populaires pour multiplier deux matrices en parallèle. 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 $C = AxB$ de la même taille. On va utiliser $P = pxp$ processus et supposer que $p$ divise $N$. Dans l'algorithme SUMMA, les matrices $A$, $B$ et $C$ sont divisées en $pxp$ sous-matrices. Par exemple, pour $p = 2$, $A$ est partitionnée en des sous-matrices $A_{11}$, $A_{12}$, $A_{21}$ et $A_{22}$ de taille $M \times M, M = N/2$. Dans le cas générale, le processus ayant les cordonnées $(i, j)$ possède les sous-matrices correspondantes $A_{ij}$, $B_{ij}$ et $C_{ij}$.

**Algorithm 1** SUMMA matrix-matrix multiplication

**Input:** Matrices $A$, $B$, $C$ of size $N \times N$
$P = p \times p$ processors
**Output:** $C = AB$ is computed.
1: Distribute matrices so that the process $p_{ij}$ owns the matrices $A_{ij}$, $B_{ij}$, and $C_{ij}$.
2: **for** $k = 1 \dots p$ **do**
3: For all $i = 1 \dots p$, broadcast the matrix $A_{ik}$ as $A_{temp}$ to the process row $p_{i1} \dots p_{ip}$.
4: For all $i = 1 \dots p$, broadcast the matrix $B_{ki}$ as $B_{temp}$ to the process column $p_{1i} \dots p_{pi}$.
5: At each process $p_{ij}$, perform the local matrix multiplication update $C_{ij} = C_{ij} + A_{temp}B_{temp}$.

L'algorithmme SUMMA est détaillé dans Algorithm 1. Ici, chaque bloc de $A$ est broadcasté à des processus dans la même ligne des processus et chaque bloc de $B$ est broadcasté à des processus dans la même colonne des processus. Afin d'effectuer ces broadcasts sur les lignes et les colonnes, il faudrait créer des communicateurs pour chaque ligne et colonne de la grille de processus (de taille $pxp$). 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.
- `key` est l'entier qui sert à déterminer le nouveau rang de chaque processus dans le nouveau communicateur. Le plus petit c'est le key d'un processus, le plus petit sera son rang dans le nouveau communicateur.
- `newcomm` est le pointeur au 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

## Parallel SUMMA using splitted communicators

*Question 1*

a) As discussed, implementing Algorithm 1 requires forming a communicator for each row and column of the processor grid. We can do this by splitting the default communicator `MPI_COMM_WORLD` properly. How can this be done? What `color` and `key` values should we use? Figure this out and form the row and the column communicators.

b) Instead of creating $N \times N$ matrices and distributing them, for simplicity, we will create the local submatrices of each process using the provided functi on

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

Here, we provide a pointer to a double pointer (which will point to the created matrix), the number of rows and the columns of the matrix to be created, and the method of initialization of matrix elements. Providing the string "random" as `init` will initialize each matrix element randomly, whereas giving "zero" will initialize each element to 0. At each process, create the local matrices `Aloc`, `Bloc` and `Cloc` of size $(N/p) \times (N/p)$. Make sure to initialize `Aloc` and `Bloc` randomly, and `Cloc` with zeros.

c) We provide the function

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

to perform the multiplication $C \leftarrow C + AB$ where the matrices $A$, $B$, and $C$ are pointed by `a`, `b`, and `c`, and the matrices are of size $M \times M$. Using this function, and the row/column communicators that you created, implement the SUMMA algorithm provided in Algorithm 1.

d) Measure the performance of your implementation using SMPI for $N = 1024$ and $P$ up to 64 (using a $8 \times 8$ processor grid). How well does your algorithm scale? Try to change the network bandwidth, and see when it starts to lose scalability.

**Make sure to backup all your implementations as they might be useful later on!**