Parallel and Distributed Algorithms and Programs
TP n°3 - Matrix multiplication

Hadrien Croubois
hadrien.croubois@ens-lyon.fr

Aurélien Cavelan
aurelien.cavelan@ens-lyon.fr

9/10/2015

All documents are available on my website : `http://hadriencroubois.com/#Teaching`

Part 1

Collective communications and communicators

One of MPI's great strength is the diversity of collective operations available. Yet, collective communications are blocking and must involve all processes in the communicator.

If one was to do a broadcast using `MPI_Bcast`, one process will be sending data to everyone else. While this is useful in some cases, it is often more efficient to organize the processes according to a shape that involves collective operations in a subset of processes. In this case, creating new communicators using `MPI_Comm_split` is the way to go.

As we saw previously, `MPI_COMM_WORLD` is the default communicator, which involves every process spawned my `mpirun`. Generating new communicators using `MPI_Comm_split` is very simple :

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

— `comm` is the communicator we want to split.

— `color` is an integer which determines in which subset of process we are. New communicator will include processes sharing the same color.

— `key` is an integer used to determine the rank of a process in the new communicator. Two processes with the same color will end up in the same communicator and there relative rank will be determined by this key.

— `newcomm` is a pointer to the new communicator being created.

Note that a communicator is still available after you split it. You may split `MPI_COMM_WORD` however you want, it will still be available for any operation (including splitting it differently to create yet another communicator).

Question

a) What communicators would you want to build when dealing with a grid of processors ?

b) How can we use `MPI_Comm_split` to build such communicators ?

Part 2

Distributed matrix multiplication

We want to compute $C_{ij} = (A.B)_{ij} = \sum_{k=1}^{n} A_{ik}B_{k,j}$ for all i, j in $[\![1, M]\!] \times [\![1, N]\!]$. Algorithm 1 describe the "natural" way to do such computation.

Algorithm 1: Matrice multiplication

Data: Matrix A $(M \times K)$, Matrix B $(K \times N)$
Result: Matrix C $(M \times N)$

```
1 for i ← 1 to M do
2 |   for j ← 1 to N do
3 |   |   for k ← 1 to K do
4 |   |   |   C_ij ← C_ij + A_ik B_kj
5 |   |   end
6 |   end
7 end
```

While loops 1 and 2 can be done in parallel, loop 3 is a reduction operation which limits parallelism. To perform this computation in parallel, we can reorder the loops like in algorithm 2. If each C_{ij} is owned by a process, A_{ik} and B_{kj} must go through those processes for the reduction to happen.

Algorithm 2: Parallel matrix multiplication

Data: Matrix A $(M \times K)$, Matrix B $(K \times N)$
Result: Matrix C $(M \times N)$

```
1 for k ← 1 to K do
2 |   for i do in parallel
3 |   |   for j do in parallel
4 |   |   |   C_ij ← C_ij + A_ik B_kj
5 |   |   end
6 |   end
7 end
```

Assuming we have a grid of $p \times q$ processes, we can distribute each matrix among the processes. Each process will own a block of A of size $\frac{N}{p} \times \frac{K}{q}$ and a block of B of size $\frac{K}{p} \times \frac{M}{q}$. Each process objective is to compute a block of C of size $\frac{N}{p} \times \frac{M}{q}$, which can be done using algorithm 2.

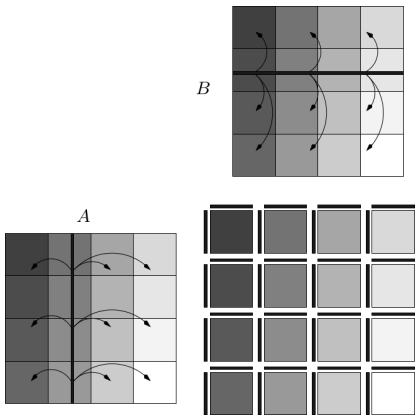

FIGURE 1 – Homogeneous matrix distribution on a 4×4 grid of processes

Algorithm 2 runs for K step. At each step k, the k-th line and the k-th column are broad-casted vertically (for the line) and horizontally (for the column). Each process receives a fragment of column, a fragment of line and updates its own block of C.

Question

a) Implement algorithm 2 using MPI.

Part 3

Collective IO

We would like to be able to import/export matrices from binary file. While it's easy for a single process to read from a binary file (or write to a binary file) using `fread` (`fwrite`) to avoid bottlenecks, doing distributed IO is a real challenge. Fortunatelly, the MPI standard includes IO methods to perform parallel read and write.

Question

a) Using views on `MPI_File`, write distributed methods to load/save matrices from/to binary files.