Programming with MPI

Abstract: In this course, we will compute matrix multiplications using the outer product algorithm explained last week. To restrain the broadcasts to the targeted processors, we will need to create new groups of processes (`communicators` in MPI) in order to use the `MPI_Bcast` function.

1 Introduction to MPI

1.1 Initializing and terminating a programme

An MPI program generally starts with a call to the `MPI_Init` function:

```
int MPI_Init(int *argc, char ***argv)
```

This function initializes internal variables and connections between nodes based on the program's arguments. MPI will modify `argc` and `argv` to remove its arguments (hence the pointers to these variables). Your program will only see its own arguments after this call with `argc` and `argv`. `MPI_Init` *must be called before* any other MPI functions.

An MPI program generally ends with a call to the `MPI_Finalize` function. All processes must call this function before their ending. This function will block until all remaining communications are finished.

Two other MPI functions can be useful at initialization time:

`MPI_Comm_size:` returns the number of MPI processes;

`MPI_Comm_rank:` returns the rank of the current MPI process (from 0 to `MPI_Comm_size(...)` − 1.

In fact, these functions works for any group (i.e. MPI communicators) of MPI processes. At the begining, you can use the predefined `MPI_COMM_WORLD` communicator that represents all the MPI processes.

```
int MPI_Comm_size ( MPI_Comm comm, int *size )
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

More detailed documentation can be found on these websites:

- http://www-unix.mcs.anl.gov/mpi/www/
- http://mpi.deino.net/mpi_functions/index.htm

1.2 Collective communications with MPI

MPI offers a wide choices of collective operations. With some specialized hardware (such as Quadrics networks for example), some of these collective operations can be managed by the hardware itself and so be very efficient. In case the underlining hardware does not have the required support, the MPI library handle these collective operations for the program. So, it is always better to use these collective primitives and to never (re)program them with point-to-point communication primitives.

For example, to broadcast some information, you can use:

```
int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root,
               MPI_Comm comm )
```

`buffer:` buffer address;

`count:` number of elements in the buffer;

`datatype:` MPI type of the elements in the buffer. There exists some predefined types (`MPI_INT`, `MPI_FLOAT`, ...) but it is also possible to dynamically create user defined (complex) types;

`root:` rank of the processor that initiate the broadcast;

`comm:` group of processes (also called *communicator* in MPI) in which the broadcast is done.

Collective communication operations always imply a communicator and they are blocking operations for this group of processes. So, all processes involved in the broadcast must call `MPI_Bcast` even if the effect will be different from a process to an other (depending whether they are the root or not, etc.). For processes that are not senders, `buffer` is used to tell where data must be stored. For processes that are senders, `buffer` has the data to be sent. Note that, as collective operations are blocking communications, no overlap of communications by computations is possible.

We already see the `MPI_COMM_WORLD` communicator that embrace all running MPI processes. Other communicators can be created for example using the `MPI_Comm_split` function:

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

`comm:` communicator to split. All processes of this communicator must call this function;

`color:` positive integer that determine in which set the calling process will be. The new communicator will regroup all processes with the same color as the caller;

`key:` integer that can be used to force a reorder of the processes within the new sets. It is common to use `MPI_rank()` to get the same order or 0 to let the implementation choose the order;

`newcomm:` new created communicator (the subset of `com` with all processes with the same color as the calling process).

A more detailed description can be found here:
http://mpi.deino.net/mpi_functions/MPI_Comm_split.html

2 Parallel matrix multiplications

2.1 Back on the outer product algorithm

We want to compute $C_{ij} = (A.B)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$ for all i, j in $[[1, n]]$. the algorithm 1 describe the natural way to compute this calculus, however it is not really adapted to a direct parallelization.

Algorithm 1: Classical standard sequential algorithm for matrix multiplication

input : Matrix A, B
output: Matrix C

```
1 function MatMult(A, B, C)
2     for i = 1 to n do
3         for j = 1 to n do
4             for k = 1 to n do
5                 C_ij ← C_ij + A_ik B_kj
6
```

One can see that loops on lines 2 and 3 are parallel. The loop at line 4 can be see as a reduction (that can be done with a tree) and limit the parallelism. Generally, parts of these loops are partially sequentialized: calculus are ordered and more regular (avoid massive and unordered data migrations), control is simplified and, most of all, the grain is adapted to the targeted machines.

Algorithm 2: Parallel algorithm for matrix multiplication

input : Matrix A, B
output: Matrix C

```
1 function MatMult(A, B, C)
2     for k = 1 to n do
3         forall i in parallel do
4             forall j in parallel do
5                 C_ij ← C_ij + A_ik B_kj
6
```

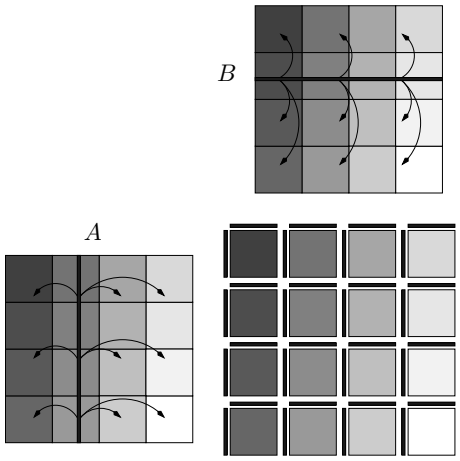

Figure 1: Contiguous homogeneous distribution for a 4×4 tore

So, it is better to permute the loops and implement the algorithm 2. The external loop (line 2) will be computed sequentially and inner loops (lines 3 and 4) will be computed in parallel. If each C_{ij} is allocated to a fixed processor, then A_{ik} and B_{kj} must shift between each step of the computation. However, this does is not a problem for the parallelism and this allows communications to be overlapped by computations.

In case we have $p.q$ identical processors connected with a tore (see Figure 1), each processor is in charge of sub-matrixes of size $\frac{n}{p} \times \frac{n}{q}$ for A, B and C.

The algorithm 2 runs in n steps. At step k, the k^{th} column of A and the k^{th} line of B are broadcasted horizontally (for the column) and vertically (for the line). Each processor receive a part of a column of A and a part of a line of B. Their size are, respectively, n/p and n/q. Each processor updates the part of C for which it is responsible (see Figure 1).

2.2 Writing the outer product algorithm with MPI

You will find in the archive a template for this matrix multiplication algorithm (`matmult_template.c`). The program has comments and only blank parts must be filled in. The provided makefile should help you to compile your program.

To run your program with 4 processes, you need to run the following commands:

```
cleanipcs.mpich-shmem
mpirun.mpich-shmem -np 4 ./matmult 12
```

Note: the first command is only needed to cleanup an hypothetical previously bad run not ending correctly (SEGV, ...).

3 Once you have a first working MPI program...

Once your program is working, try to increase the granularity of the communication (transfert parts of several columns and lines at each braodcast). What does this imply on p and q ?

You can also try to make the computations overlap the communications. How can you do it ?

How to avoid unneeded memory copies when sending parts of lines and columns ?

4 The programming environment

These machines are dedicated to the crypto MASTER. This means that students can have root access on these machines. So, your teacher will give you a user login/password and you will also be given the root password in order to install missing software.

So, as root, you will need to type:

```
apt-get install mpich-bin  mpich-mpd-bin mpich-shmem-bin mpi-doc
apt-get install libmpich-mpd1.0-dev libmpich-shmem1.0-dev libmpich1.0-dev
```

If you are missing your favorite development application/editor, you can also install it... (try: `apt-get install` *application*)