INFO-F-404 : Real-Time Operating Systems
Project 2 : Bitonic

Rémy Detobel, Stanislas Gueniffey and Denis Hoornaert

December 19, 2017

1 Implementation choices

Typically, there are two main types of process : the *master node* and the *compute node*. Each node has its own purpose(s) and interact with the other processes using *MPI*.

There is only one *master node* per program instance. Its objective is to oversee and deal *merge tasks* to its *compute nodes*. Typically, the *master node* will, when a non-bitonic list is given, execute multiple successive merge call so that the list will be *bitonic* before performing the sort itself. Afterwards, the sorting —which consists of a merge of the whole list— is performed.

The *compute nodes* have for only mission to compare two values and to determine to which other *compute node* they have to send the outcome of the comparison. The mechanism is as follow :
Initially, each *compute node* receives two values from the master node. Then a first comparison is done. After having complete the comparison, the outcome is send to another *compute node* as mentioned above, a new value (coming from another *compute node* having performed the same mechanism) is received and another comparison is performed.
The number of comparisons and the determination of both the sending *compute node* and the receiving *compute node* are obtained at the runtime. The former is obtained by applying the $\log_2$ of the list size whereas the latter are obtained using the following formula :

$$dest = (id \; xor \; (1 << depth)) + 1$$

Where :

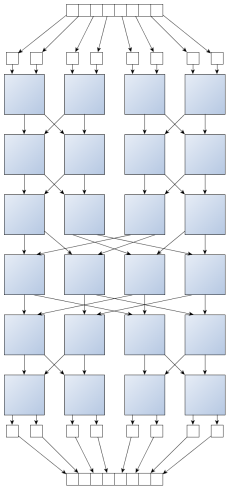
- *id* is the process id (or rank) of the current process
- *depth* is the number of comparisons that the compute node still has to perform
- *dest* is the process id of the destination

2 Project utilisation and configuration

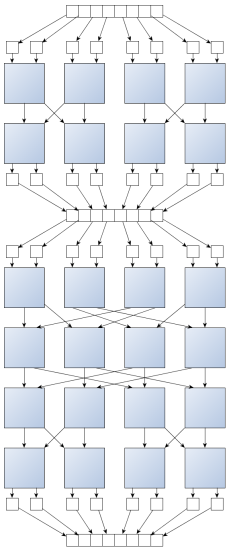
As mentioned in the assignment statements, the source code can either be used with a *bitonic* list as input or with an unsorted list.
In the case of a *bitonic* list input, the user is invited to set the macro `SORT_FIRST` (`main.cpp` line 8) to *false*. Doing so, will make the program to sort the hardcoded list writen line 74. If the user wants to test the project with a fully random list, he only has to set the macro `RANDOM_LIST` to *true*.
Notice that the list generation does not ensure the generation of a *bitonic* list. Hence, it is required to set the macro `SORT_FIRST` to *true*.
Once that the manipulation have been done, the user is invited to compile the project using the provided Makefile.

3 Inter-processes communications

The implemented inter-processes communication protocol (1a) differs a bit from the one suggested in the reference (1b). In fact, in order to ease the implementation of the *parallel bitonic sorting algorithm*, it has been decided that after each merge the *compute nodes* have to send their results to the master node instead of dealing themselves with the communication. Consequently, parallelism is not exploited as far as possible and this has an impact on the expected performances.



(a) State of the art commnucations



(b) Project communications

Figure 1

4 Limitations and performances