Csc-387 Parallel Programming with MPI
Instructor : Fikret Ercal

Programming Project 3
PARALLEL BITONIC SORT

Write a program to sort N arbitrary numbers in parallel on P processors ($N \gg P$) using a *bitonic sort* algorithm as explained in this handout and the slides numbered 18-20 in **SLIDES-10-sorting.ppt**.

I. (25 pts) Run your program on MST Cluster and obtain timing results for N= 16M, 32M, 64M, and 128M; P=1, 2, 4, 8, 16, and 32 (a total of 24 timing results). Use 4-byte integers (randomly generated) as the numbers to sort. Tabulate your findings.

II. (20 pts) Run your program for N=400K using the input file at *http://web.mst.edu/~ercal/387/input.400K* and print the following for the sorted list:
(i) 10 numbers starting at index (position) 100,000 and
(ii) 10 numbers starting at index (position) 200,000.

III. (15 pts) Try to obtain an approximate formula for the *speedup* and *efficiency* in terms of N and P. Comment briefly on the asymptotic behaviour of your formula for the following cases:

i) $N = P$ ii) $N \gg P$

IV. (30 pts) Answer the following questions with respect to your timing measurements:

1. How does the speedup change when you increase N for a fixed P ? Is your answer consistent with the formula you derived in Part III ? Why or why not ?
2. How does the efficiency change when you increase N for a fixed P ? Is your answer consistent with the formula you derived in Part III ? Why or why not ?
3. How does the efficiency change when you increase P for a fixed N ? Is your answer consistent with the formula you derived in Part III ? Why or why not ?

V. (5 pts) Electronic copy of your program. Your program must: (i) contain a program overview/summary, your id/name etc. in the header, (ii) execute correctly, (iii) be well documented (Must include comments before every major statement, function/subroutine calls, and every MPI call explaining what the call is for. Must include specs for every function/subroutine)

VI. (5 pts)
Read the file "How to submit projects?" on the class website and follow all the instructions in there. You must store the requested files in your subdirectories.

BONUS: The best running time for P=16 and N= 128 Million will get bonus points as described in the syllabus. Therefore, do not forget to report your running time for P=16 and N= 128 Million.

IMPORTANT NOTE:

1) Time only the sorting part of your program. Do not time the portion where the random numbers are generated.

2) To be more accurate, take the average of 4-5 runs for the same program and report it as one timing result.

Parallel Bitonic Sort Algorithm

The input data need to be partitioned and distributed to the processors first, and then a standard sequential sorting method (e.g., Quicksort) must be used to locally sort the data. Once the processors complete local sorting, they communicate and exchange data in a certain fashion to merge the local subsequences. **Bitonic Sort** uses a predetermined order to merge the locally sorted sequences and to obtain a totally sorted sequence, as depicted in Figure 2. The first $(d^2 - d)/2$ steps[1] are for obtaining a full bitonic sequence and the last d steps are for sorting this Bitonic sequence.

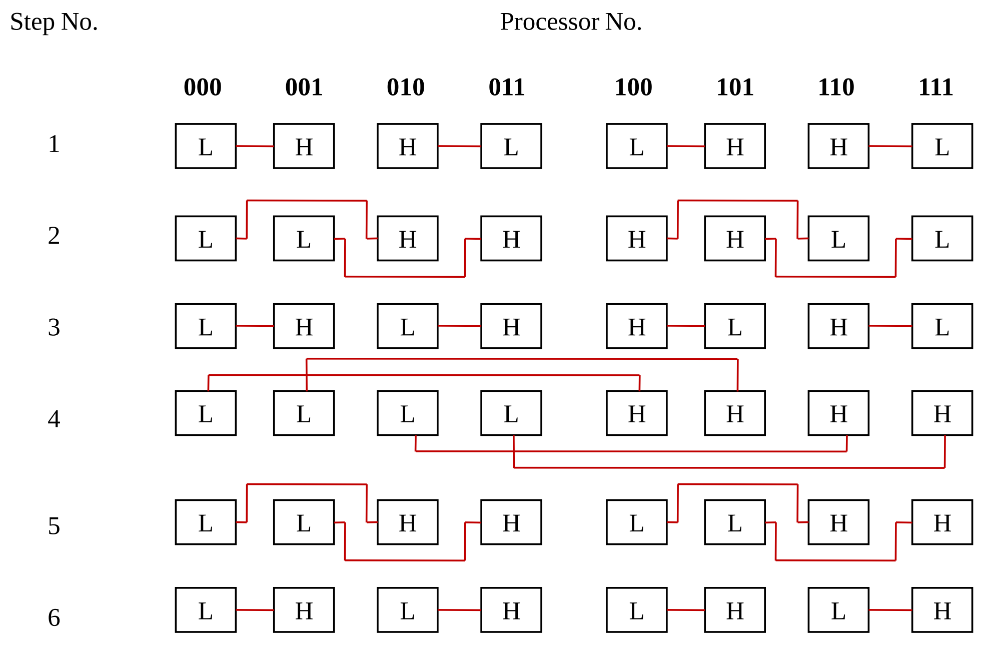

Figure 2: Six phases of Bitonic Sort on a hypercube of dimension 3

There are two main routines, called *CompareHigh* and *CompareLow*, in the algorithm for Bitonic Sort given below. During merging, some processors call the first routine while their neighbors call the second one, so that the lower and the higher subsequences of a sequence are split apart between the two processors. The lower part is gathered on the processor which calls *CompareLow* and the upper part on the processor which calls *CompareHigh*. A *CompareLow*(j) or *CompareHigh*(j) means that a processor will compare and exchange with a neighbor whose (d-bit binary) processor number differs only at the j^{th} bit.

```
Parallel Bitonic Sort Algorithm for processor $P_k$ (for $k := 0 \ldots P-1$)
d:= $\log P$      /* cube dimension */
sort($local - data_k$)     /* sequential sort */
/* Bitonic Sort follows */
for i:=1 to d do
    window-id = Most Significant (d-i) bits of $P_k$
    for j:=(i-1) down to 0 do
        if((window-id is even AND $j^{th}$ $bit\ of\ P_k = 0$)
           OR (window-id is odd AND $j^{th}$ $bit\ of\ P_k = 1$))
          then   call  CompareLow(j)
          else   call  CompareHigh(j)
        endif
    endfor
endfor
```

[1]d is the cube dimension and d = $\log_2 P$ where P is the number of processors.