Assignment No. 5

Parallel Computing, DM8XX (Fall 2008)

Department of Mathematics and Computer Science
University of Southern Denmark
Daniel Merkle

Due on: Tuesday 10. December, 12:00 p.m. (Department secretaries office (Lone Seidler Petterson) or my office).

Exercise 1

`Bitonic Sort Isoefficiency` (10 points)

Explain the isoefficiency function $\Theta(p^{\log p} \log^2 p)$ for the block-based bitonic sort algorithm on a hypercube as given on page 393. You can take the parallel runtime

$$T_P = \Theta(\frac{n}{p} \log \frac{n}{p}) + \Theta(\frac{n}{p} \log^2 p)$$

as given.

Exercise 2

`Prim's minimum spanning tree` (2+6+2 = 10 points)

In the parallel formulation of Prim's minimum spanning tree algorithm (Section 10.2), the maximum number of processes that can be used efficiently on a hypercube is $O(n/\log n)$. By using $O(n/\log n)$ processes the run time is $O(n \log n)$.

a) What is the run time if you use $O(n)$ processes?

b) What is the minimum parallel run time that can be obtained on a message-passing parallel computer?

c) How does this time compare with the run time obtained when you use $O(n/logn)$ processes?

Exercise 3

`Floyd's All-Pairs Shortest Paths` (20+0 = 20 points)

An alternative way of partitioning the matrix $D^{(k)}$ in Floyd's all-pairs shortest paths algorithm is to use the 1-D block mapping (Section 3.4.1). Each of the p processes is assigned n/p consecutive columns of the $D^{(k)}$ matrix.

a) Compute the parallel run time, speedup, and efficiency of 1-D block mapping on a hypercube-connected parallel computer. What are the advantages and disadvantages of this partitioning over the 2-D block mapping presented in Section 10.4.2?

b) * Compute the parallel run time, speedup, and efficiency of 1-D block mapping on a p-process mesh with store-and-forward routing, a p-process mesh with cut-through routing, and a p-process ring.

Exercise 4

`Floyd's All-Pairs Shortest Paths + MPI` (20 points)

Implement the parallel version of Floyd's algorithm for all-pairs shortest paths using a 2-D block mapping and MPI. Use an adjacent matrix defined as follows:

$$a_{ij} = \begin{cases} 1, & \text{if } (|i-j| = 1) \text{ or } (|i-j| = 13) \\ 0, & \text{if } i = j \\ \infty, & \text{else} \end{cases}$$

The implementation has only to be functional for p processors with p being a square number, and the dimension of the matrix of size $n \times n$ can be expected to be divisible by the number of processors (i.e. $n\%\sqrt{p} = 0$). Consider matrices of size $60 \times k$, where the maximal value of the matrix size and the maximal number of processors should be given as an argument to the program. The maximal number processors to be used should also be given via a parameter. If ($n\%\sqrt{p} = 0$) does not hold, just skip the corresponding experiment. If `floyd` is the name of the binary executable file, then the output of `mpirun -np 100 ./floyd 100 120` should be similar to the following:

```
Dimension         Number of Processes     Avg. Runtime / Deviation (Efficiency) / distance 0->(n-1)
60                1                       XXX / YYY (1.0) / ZZZ
120               1                       XXX / YYY (1.0) / ZZZ
60                4                       XXX / YYY (???) / ZZZ
120               4                       XXX / YYY (???) / ZZZ
..
60                81                      skipped as n is not divisible by sqrt(p)
120               81                      skipped as n is not divisible by sqrt(p)
60                100                     XXX / YYY (???) / ZZZ
120               100                     XXX / YYY (???) / ZZZ
```

Use 10 experiments when determining the average runtime and deviation. Use the average runtime for 1 processor as a reference value for determining the efficiency. The value of `ZZZ` has to be the shortest path length from node 0 to node $(n-1)$ as computed by the algorithm.

Send the source code of the compilable program and an output file (using an IMADA pool machines, maximally 27 nodes, and a maximal matrix size of 600) to `daniel@imada.sdu.dk`.