American University of Beirut
الجامعة الأميركية في بيروت

Faculty of Arts and Sciences *Department of Computer Science*

CMPS 297M – Parallel Computing

Assignment 01

Spring 2013-14

Due Date

February 20th, 5:00 pm

Purpose

The purpose of this assignment is to

1. Run existing BSP and MPI programs in both shared and distributed mode.
2. Bench-mark your parallel computer in both shared and distributed mode.
3. Implement an adaptation of the parallel inner product algorithm for computing sums of squares
4. Measure the scalability of the parallel performance whenever applicable.
5. Design and Analyse a number of basic algorithms in the BSP model.

Important Notes

- All existing BSP and MPI programs can be found on Moodle in zipped folders: `bsptest-inprod-bench.zip` and `bsptest-inprod-bench.zip` respectively.
- All programs should be written in C and supported by the BSPlib library.
- You should work individually on this assignment and not in groups.
- You should submit the source files as well as an accompanying text file (describing your documentation and the answers to some of the non-programming questions below). No handwritten documents will be accepted.
- Note that in some cases obtaining a number of 16 processors will not be possible. Please note this down in your documentation whenever it is the case.

Grade distribution

40% correctness of output, 40% good parallel performance, 20% good programming style.

Question 1 (25%)

Enclosed in this assignment is the file `bspbench.c (mpibench.c)`. This program contains subroutines to benchmark the performance of a parallel computer by estimating the sequential computing rate r of a single processor using a so-called DAXPY operation, the communication parameter g using a full scale h relation, and the synchronisation parameter l.

Run the programs `bspbench.c` (and mpibench.c) on your parallel computer. Measure the values of g and l for a number of processors ranging from 1-16 (*you may for instance choose to benchmark your program for p = 1, 2, 4, 6, 8, 12, and 16).* In each of the runs, do the following:

I. Increase MAXH from 256 to 800 (in strides of 100 for example).

II. Increase MAXN from 512 to 1500 (in strides of 100 for example).

For each value of p (fixed number of processors), note down your results before coming up with a good estimate.

Try benchmarking all cores in one node (shared memory mode). Then try benchmarking one core per every available node (distributed memory mode). Produce your results for the two batches.

Question 2 (25%)

The program `bspinprod.c (mpiinprod.c)` computes the sum of integral squares between 1 and n in parallel using an adaptation of the parallel inner product algorithm described in class. In the following, let T_s denote the sequential time and T_p denote the parallel time in seconds.

(a) Develop a sequential program that performs the above algorithm. Obtain T_s.

(b) Run the parallel program `bspinprod.c` (and `mpiinprod.c`) for a number p of processors (say $p = 1, 2, 4, 6, 8, 12, 16$). For values of n ranging from 100,000-1,000,000 (in strides of 50000), do the following:

I. For each run, time your program and plot the results in a graph of p versus E, where E denotes the absolute efficiency factor $T_s/(pT_p)$.

Make sure to run your MPI programs in both shared and distributed mode.

(c) Explain why E is theoretically a number between 0 and 1 (although in practice and under specific circumstances E can achieve values greater than 1, in a so called *super-linear* performance). Ideally, one wishes to obtain an efficiency that

is as close to 1 as possible. How does the program scale as the input size increases?

Question 3 (25%)

The above inner product parallel algorithm can be modified to combine the partial sums into one global sum by a different method. Assume *p* is a power of 2.

(a) Modify the algorithm to combine the partial sums by repeated pairing of processors. Take care that every processor obtains the final result.

(b) Formulate the modified algorithm exactly in pseudo-code, and compare the BSP cost of the two algorithms.

(c) Perform the necessary changes in `bspinprod.c` and plot the efficiency graph using the set of parameters described in part b above. Do your empirical results confirm your theoretical predictions in any way?

(d) For which ratio *l/g* is the pairwise algorithm faster?

Question 4 (25%)

Analyse the following operations and derive the BSP cost for a parallel algorithm. Let **x** be the input vector (of size *n*) of the operation and **y** the output vector. Assume that these vectors are block distributed over *p* processors, with *p* less than or equal to *n*. Furthermore, *k* is an integer in the range *1,...,n*. The operations are:

(a) Minimum finding: determine the index *j* of the component with the minimum value and subtract this value from every component.

(b) Shifting (to the right): assign $y_{(i+k) \bmod n} = x_i$

(c) Partial summing: compute $y_i = \sum_{j=0}^{i} x_j$, for all *i*. (This problem is an instance of the parallel prefix problem).

(d) Sorting by counting: sort **x** by increasing value and place the result in **y**. Each component x_i is an integer in the range *0 to k*, where .