Assignment No. 4

Parallel Computing, DM8XX (Fall 2008)

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

Due on: Tuesday 25. November, 12:00 p.m. (Department secretaries office (Lone Seidler Petterson) or my office).

Exercise 1

`OpenMP` (15 points)

Implement and test the `OpenMP` program for computing a matrix-matrix product in Example 7.14 of the course book. In a first step, use the `OMP_NUM_THREADS` environment variable to control the number of threads analyze the performance with varying numbers of threads (1,2,. . .,8). Consider three cases in which (i) only the outermost loop is parallelized; (ii) the outer two loops are parallelized; and (iii) all three loops are parallelized.

After controlling the number of threads by an environment variable, use the OpenMP library functions to set the number of threads. Write a program that has an output similar to the following:

```
Dimension         Nested Parallelization         Number of Thread       Avg. Runtime / Deviation (Efficiency)
10                (i)                            1                      XXX / YYY (1.0)
10                (ii)                           1                      XXX / YYY (1.0)
10                (iii)                          1                      XXX / YYY (1.0)
20                (i)                            1                      XXX / YYY (1.0)
20                (ii)                           1                      XXX / YYY (1.0)
20                (iii)                          1                      XXX / YYY (1.0)
...
500               (iii)                          8                      XXX / YYY (???)
```

where `Dimension` refers to the dimension of the matrices, `Nested Parallelization` refers to the cases given above. Use matrix sizes $\{10, 20, 50, 100, 200, 500\}$ with randomly chosen double values between 0 and 1, and a varying number of thread $\{1, 2, \ldots, 8\}$. Compute the average over 10 experiments when determining the average runtime and deviation values. Use the function `omp_get_wtime()` to measure runtimes. (`omp_get_wtime()`) returns a double-precision value equal to the elapsed wallclock time (in seconds) relative to an arbitrary reference time. The efficiency should be computed relative to the corresponding case of using one thread only.

Note: Use the `gcc` compiler option `-fopenmp` to compile your program.

Send the source code of the compilable program, and an output file (using an IMADA pool machines) to `daniel@imada.sdu.dk`.

Exercise 2

Dense Matrix Multiplication and `MPI` (45 points)

Implement the DNS algorithm for dense matrix multiplication using `MPI`. Use the a virtual topology as a mechanism for naming the processes in the communicator (see for example `http://www.mhpcc.edu/training/workshop/mpi/MAIN.html`). The implementation has only to be functional for n^3 processors, with $n \in \{1, 2, 3, 4, 5, 6\}$. 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. If `dns` is the name of the binary executable file, then the output of `mpirun -np 27 ./dns 27 240` should be similar to the following:

```
Dimension         Number of Processes    Avg. Runtime / Deviation (Efficiency)
60                1                      XXX / YYY (1.0)
120               1                      XXX / YYY (1.0)
180               1                      XXX / YYY (1.0)
240               1                      XXX / YYY (1.0)
60                8                      XXX / YYY (???)
...
240               27                     XXX / YYY (???)
```

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.

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`.

Note: your program will be tested on a parallel machine with several thousand nodes!