

CS 292 Introduction to Parallel Computing

Spring 2008

Programming Assignment #2

Due: April 4, 2008

Objectives

- Practice parallel programming with OpenMP

Collaboration

- You may discuss all aspects of the assignment with your classmates. However, you should never show any of your code to another student.

Description

In the previous programming assignment, you implemented parallel matrix multiplication using Pthread. In this assignment, you will implement two versions of matrix multiplication with OpenMP. You should parallelize the loop structures using either parallel loop constructs (`for`) or parallel region constructs (`sections`). You may use any of these OpenMP methods to parallelize these 2 algorithms.

- Algorithm 1

This algorithm is basically the same as what you implemented in Programming Assignment 1.

```
for(i=0; i<row_c; i++)
  for(j=0; j<col_c; j++)
    for(k=0; k<col_a; k++)
      C[i][j] += A[i][k]*B[k][j];
```

- Algorithm 2

This algorithm is similar to Algorithm 1, but it holds a constant for better data reuse.

```
for(i=0; i<row_c; i++)
  for(k=0; k<col_a; k++)
    temp = A[i][k];
    for(j=0; j<col_c; j++)
      C[i][j] += temp*B[k][j];
```

Verifying results

In this homework you need to verify that your matrix multiply algorithms produced correct results but that verification need not run in parallel. To compare two matrices M_1 and M_2 , you can compute the Frobenius

norm of the difference matrix $M_1 - M_2$. This norm should be very close to 0, but will not be 0 due to round errors. The Frobenius norm of matrix M ($s \times t$) is defined as:

$$\|M\|_2 = \sqrt{\sum_{i=0}^{s-1} \sum_{j=0}^{t-1} m_{i,j}^2}$$

Performance measurement

You should measure the time it takes to complete the actual multiplications, not including I/O (reading input and writing output files) and checking the results. You can use `gettimeofday()` to get better timing resolutions. Run your programs on one of the dual-dual gateways (`vmps01[vmps02].accre.vanderbilt.edu`). Since those gateways have only 4 CPUs, using more than 4 threads should show little or no performance improvement. You can set `OMP_NUM_THREADS` to more than 4 to confirm that your code works. You have to be sure of your analysis of the parallel OpenMP constructs. Odd prime numbers of threads often generate errors if your code is not quite right. Setting the number of threads larger than the physical processors often uncovers communication events among threads that you have not thought about e.g., something that you should have set private but forgot to do so. *Run each program multiple times and take the average as its timing.*

Compiling and running code

You can use either Intel or GCC compilers. Set the packages properly with `setpkgs -a [package_name]`. For Intel compiler `icc`, use `-openmp` to enable OpenMP compilation. For GCC compiler `gcc`, only the latest 4.2 version has the ability to compile OpenMP code with the flag `-fopenmp`. Your programs should take 3 parameters (The number of threads to be used MUST be set using the `OMP_NUM_THREADS` environment variable). E.g.

```
matmul a_file b_file c_file
```

where `a_file` and `b_file` are input files generated with the program you wrote for Programming Assignment 1.

Problem size and thread counts

Use the following parameter set to collect the timings.

case	row_A	col_A (row_B)	col_B	# of threads
1	500	500	500	1,2,4,8
2	1000	1000	1000	1,2,4,8
3	843	1379	1107	1,2,4,8

What to submit

Submit all the source code, compiled executable etc. as a `tar` file. Also put the code in the proper directory on the cluster so that we can run it. Along with the tar file, you should also submit a `README` file (preferably in PDF format) describing how to compile and run your program, the known bugs (if any) in your program and any comments. *Include a plot of the execution time vs. # of the threads (one line per each size) for each algorithm.* Discuss any findings you observed from the performance results.