Project 3: OpenMP

Jonas Axelsson, Mattias Willemsen

May 21, 2014

0.1 Source code

The source files are included in the submission as .c files, notes on how to compile them are also included.

Note that they are compiled using the MPI command only because we used the MPI timer functions for easy hassle free timing of the project.

0.2 Measurements

0.2.1 Quick Sort

Size: 2048 * 2048 elements

- Sequential Quick Sort: 1.7754 seconds
- Parallel Quick Sort (1 thread): 1.8711 seconds
- Parallel Quick Sort (8 threads): 0.2747 seconds

Speedup: 6.46 times

0.2.2 Gaussian Elimination

Size: 2048 x 2048 matrix

- Sequential Gaussian: 23.9128 seconds
- Parallel Gaussian (1 thread): 27.0419 seconds
- Parallel Gaussian (8 threads): 4.0875 seconds

Speedup: 5.85 times faster

0.3 Implementation

0.3.1 Quick Sort

The parallel quicksort implementation will first check how many threads are available on the system. Each thread will be assigned a fraction of the array to be sorted and will quicksort that fraction.

These fractions will be internally sorted, but the entire array is not yet sorted. To solve this, the fractions are merged on a single thread. After the merge, the entire array will be sorted.

0.3.2 Gaussian Elimination

Firstly, the number of threads are decided by the user (in this case by changing the value of the defined variable THREADS). The specified number of threads will then be created by OpenMP with the for directive (pragma omp parallel for), dividing up the work to be done in the elimination loop of the Work() function.

The in-loop variables i and j are private to each thread, while the outer loop variable k is shared.