PROJECT ASSIGNMENT 3

Multi Processor System (DV 2544)

SUSHEEL SAGAR

suko15@student.bth.se

9303177837

BISWAJEET MOHANTY

bimo15@student.bth.se

9303167754

Parallel Version Of QuickSort :

Implementation :

Aim: To implement a parallel version of Quicksort using OpenMP.

Assumptions: The number of threads (T) is an exponent of 2 (T = 2 x , where x = 0, 1, 2 ...)

Performance: Our implementation of the parallel version of Quicksort has a speedup of atleast 2 if the number of elements greater than 128*1024.

The initialization of the array elements is not parallelized but, we have parallelized all the merge steps of an iteration by using the directive: #pragma omp parallel for.

```
        #pragma omp parallel for private(i)

        for(i = 0; i < threads; i++)

                qsort(list+index[i], index[i+1]-index[i], sizeof(int), Comparision);
```

the above code runs various threads in parallel, where each thread runs qsort function.

TABLE I. MEASUREMENTS OF EXECUTION TIMES OF SEQUENTIAL AND PARALLEL VERSIONS FOR VARIOUS INPUT SIZES

Number Of Elements To be sorted	Sequential Version			Parallel Version (1 CPU)			Parallel Version (8 CPUs)			Speed Up on 8 CPUs
	Execution Time		CPU utilized	Execution Time		CPU Utilized	Execution Time		CPU utilized	
	System	Elapsed		System	Elapsed		System	Elapsed		
1024	0.0	0.00	0%	0.0	0.0	0%	0.0	0.0	420%	-
128*1024	0.0	0.04	87%	0.0	0.03	93%	0.01	0.01	540%	2
256*1024	0.0	0.12	82%	0.0	0.06	98%	0.01	0.02	510%	4
512 *1024	0.0	0.23	87%	0.0	0.12	96%	0.01	0.04	470%	4.6
1024* 1024	0.0	0.4	98%	0.0	0.25	99%	0.02	0.09	454%	4

Parallel version of Gaussian Elimination:

Implementation:

Aim: To implement a parallel version of Gaussian Elimination using OpenMP.

Assumptions: Size of the input matrix size is greater than the number of cores.

Performance: Our implementation of the parallel version of Gaussian Elimination has a speedup of at least 1.5 (on 8 cpus) over the sequential version, for sizes of higher magnitudes (E.g. For input matrix sizes greater than 512).

Data Structure Allocation: The initial matrix is allocated as a 2-dimensional double array in global scope. The b vector and the y vector are allocated as global double arrays.

In our implementation of gaussian elimination, we have parallelized the algorithm in three phases.

1) Division Step:

```
        #pragma omp parallel for num_threads(NUMB_CORES) schedule(dynamic, chunksize)
                for (j = k+1; j < N; j++)
                        A[k][j] = A[k][j] / A[k][k];
```

2) Elimination Step:

```
        #pragma omp parallel for num_threads(NUMB_CORES) schedule(dynamic, chunksize)
collapse(2)
                for (i = k+1; i < N; i++) {

                        for (j = k+1; j < N; j++)
                                A[i][j] = A[i][j] - A[i][k]*A[k][j];
                }
```

3) Back Substitution Step:

```
        #pragma omp parallel for num_threads(NUMB_CORES) schedule(dynamic, chunksize)
                for (i = k+1; i < N; i++) {
                        b[i] = b[i] - A[i][k]*y[k];
                        A[i][k] = 0.0;
        }
```

Apart from these major steps the initialization of the matrices is also parallelized.

TABLE II. MEASUREMENTS OF EXECUTION TIMES OF SEQUENTIAL AND PARALLEL VERSIONS FOR VARIOUS INPUT SIZES

SIZE of the Matrix	Sequential Version			Parallel Version (1 CPU)			Parallel Version (8 CPUs)			Speed Up on 8 CPUs
	Execution Time		CPU utilized	Execution Time		CPU Utilized	Execution Time		CPU utilized	
	System	Elapsed		System	Elapsed		System	Elapsed		
64	0.00	0.00	0%	0.00	0.00	0%	0.00	0.00	600%	-
128	0.00	0.00	98%	0.00	0.00	88%	0.00	0.01	700%	-
256	0.00	0.04	99%	0.00	0.05	98%	0.02	0.04	775%	1
512	0.00	0.37	99%	0.00	0.40	98%	0.01	0.18	779%	1.53
1024	0.01	2.99	99%	0.01	3.35	99%	0.05	1.39	797%	2.18