OMP (OpenMP) Programming
Multiprocessor Systems, DV2544, Project 3

Li Kang, 20121963-0405, Blekinge Institute of Technology
Santosh Bharadwaj Rangavajjula, 940812-4635, Blekinge Institute of Technology

Categories and Subject Descriptors: OpenMP implementations [**Quick Sort**]: Gaussian elimination

General Terms: Design, Algorithms, Performance, Programming

Additional Key Words and Phrases: OpenMP implementations,OpenMP performance, Gaussian elimination, Quick Sort

1. GAUSSIAN ELIMINATION

Gauss elimination is a method to solve linear equations. In this method, the coefficients of the variables in the equation are put in the form of a matrix. The first motive in the gauss elimination method is to make the lower triangular or upper triangular elements of the matrix into zero and the second motive is to make the diagonal elements into one. In order to make it possible, row operations or column operations are to be applied.

1.1. Implementation

Pthread version for Gauss elimination is implemented in the following way:

— Initialised Default values
— Arguments like help, default values, print switch are read.
— Matrix is initialised, vectors are initialised.
— Clock is initialised to measure the time
— Division and Elimination are carried out in the work() method.
— #pragma omp is set to parallel.
— Matrix and vectors are printed using Print_Matrix() method.

1.2. Process

Gaussian elimination is used to solve a pair of linear equations. The process is as follows:

— System of equations are to be converted into a matrix. Such a matrix with co efficients and constants is called as Augmented matrix.
— In order to achieve the echelon form, row operations on the matrix are applied .
— By substituting back the results progressively, we get the solutions for the linear equations.

1.3. Measurements and result

Thread	Sequential Version	OpenMP Version	SpeedUp
1	24.161s		
8		5.60s	4.31

1.3.1. Execution times and speedup. Based on [Rauber and Rnger], the speedup S p (n) of a parallel program with parallel execution time T p (n) is defined as

```
likb13@kraken:~/mp$ gcc Gaussianelimination.c  -o gaussian1
Gaussianelimination.c: In function 'Read_Options':
Gaussianelimination.c:166:3: warning: incompatible implicit declaration of built
-in function 'exit' [enabled by default]
   exit(0);
   ^
likb13@kraken:~/mp$ ./gaussian1

size      = 2048x2048
maxnum    = 15
Init      = rand
Initializing matrix...done


the execution time of the sequential version of the application is   2.4161e+01
secondslikb13@kraken:~/mp$
```

Fig. 1. The execution time of the sequential version of the application

```
likb13@kraken:~/mp$ gcc -fopenmp gaussian_omp.c -o gomp
gaussian_omp.c: In function 'Read_Options':
gaussian_omp.c:170:3: warning: incompatible implicit declaration of built-in fun
ction 'exit' [enabled by default]
   exit(0);
   ^
likb13@kraken:~/mp$ time ./gomp

size      = 2048x2048
maxnum    = 15
Init      = rand
Initializing matrix...done


the execution time of the OpemMP parallel version of the application is 5.597665
 seconds

real    0m5.755s
user    0m44.664s
sys     0m0.096s
likb13@kraken:~/mp$
```

Fig. 2. The execution time of the Parallel version of the application

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

where p is the number of processors used to solve a problem of size n. $T^*(n)$ is the execution time of the best sequential implementation to solve the same problem.

2. QUICK SORT

In Quick sort, input is any group of numbers and output is Sorted array in ascending and descending order. We begin with one number, generally the first number, and discovers its position in sorted array, this number is called a pivot. At that point we partition the array into two sections considering the pivot's position in sorted array.

In every part, independently, we discover the pivot elements. This procedure proceeds until all numbers are handled and sorted array is obtained.

2.1. Implementation

OpenMP version for Quick sort is implemented in the following way:

— Initialised Default values
— Arguments like help, default values, print switch are read.
— Matrix is initialised, vectors are initialised.
— Clock is initialised to measure the time
— Division and Elimination are carried out in the work() method.
— #pragma omp is set to parallel.
— Matrix and vectors are printed using Print_Matrix() method.

2.2. Measurements and result

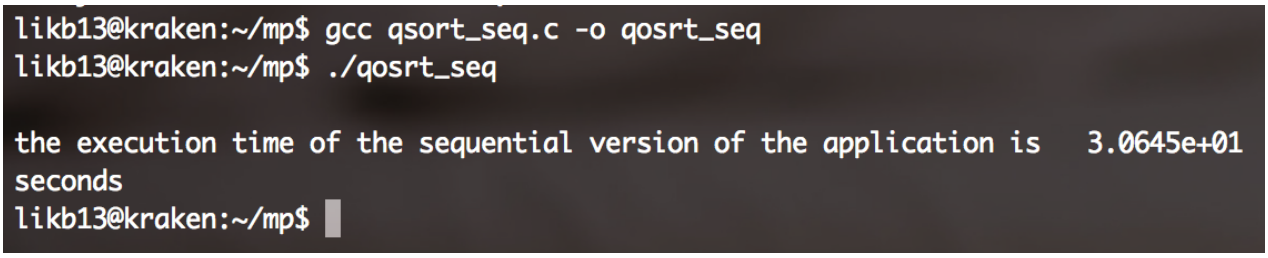

Fig. 3. The execution time of the sequential version of the application

Fig. 4. The execution time of the Parallel version of the application

Thread	Sequential Version	Pthread Version	SpeedUp
1	30.65		
8		12.60s	2.43

2.2.1. Execution times and speedup. Based on [Rauber and Rnger], the speedup S p (n) of a parallel program with parallel execution time T p (n) is defined as

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

where p is the number of processors used to solve a problem of size n. $T^*(n)$ is the execution time of the best sequential implementation to solve the same problem.

REFERENCES

Thomas Rauber and Gudula Rnger. *Parallel programming: For multicore and cluster systems*. Springer Science & Business Media. https://www.google.com/books?hl=en&lr=&id=UbpAAAAAQBAJ&oi=fnd&pg=PR5&dq=Parallel+Programming++for+Multicore+and+Cluster+Systems&ots=9YJB9wnOCD&sig=27rWX3m-JbwIpohfdPLUuQ4dmJw