OpenMP-based Matrix Computation Study

2016 spring big-data processing homework I

5130309194 Shen Jiyuan

Abstract—OpenMP(Open Multi-Processing), which packs compiler directives, library routines, and environment variables, is used here to supports multi-platform shared memory multiprocessing programming in C/C++ for Matrix computations(addition, subtraction, and multiplication) .

Index Terms—OpenMP, parallel processing, performance, serial processing, matrix, add, subtract, multiply.

I. Introduction *of OpenMP*

OpenMP(open multi-processing), is an application programming interface(API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, processor architectures and operating systems, including Solaris, AIX, HP-UX, Linux, OS X, and Windows. It consists of a set of comppiler directives, library routines, and environment variables that influence run-time behavior.

It uses a portable, scalar model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from standard desktop computer to the supercomputer. An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface, or more transparently through the use of OpenMP extensions for non-shared memory systems.

An implementation of multi-threading, a method of paralleling whereby a master thread(a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.

The section of code that is meant to run in parallel is marked accordingly, with a preprocessor directive that will cause the threads to form before the section is executed. Each thread has an id attached to it which can be obtained using a function(called omp_get_thread_num()). The thread id is an integer, and the master thread has an id of 0. After the execution of the paralleled code, the threads join back into the master thread, which continues onward to the end of the program. By default, each thread executes the paralleled section independently. Work-sharing constructs can be used to divide a task among the threads.

II. Environment settings and preparations

A. Hardware Environment

- Intel(R) Core(TM) i5-4200U CPU @ 1.60GHZ
- Installed RAM: 4.00 GB
- x64 Processor

B. Software Environment

- Windows 10 Professional
- GCC
- Visual Studio 2013
- Setting Advice on OpenMP programming environment:

Open 'Visual Studio 2013', and new a 'C++ console application program'. Right-click the new program and choose 'property'. Choose 'configuration property' - 'C/C++' - 'Open MP support', you will change the block into 'yes(/openmp)'.

III. Problem Descriptions

This problem introduces simple mathematical computations on matrix, including addition, subtraction and multiplication. And the computation program is required to designed both serially and parallel for the parallel programming practice and performance comparison between serial and parallel processing regarding simple matrix computations.

IV. Computing Algorithms

A. Serial computations

(for this algorithm, not for our program)

Input: two matrix with exactly a same dimension.

Output: a matrix with the same dimension after computing addition, or subtraction, or multiplication.

Algorithm: All the three computations are in completely simple traversal through the two input matrix, and all data in the output matrix are computed one by one.

B. Parallel computations

(for this algorithm, not for our program)

Input: two matrix with exactly a same dimension.

Output: a matrix with the same dimension after computing addition, or subtraction, or multiplication.

Algorithm: divide each computation into small threads that fit into the smallest computing process.

—addition and subtraction: total (dimension)^2 threads, and each thread computes one simple addition or subtraction.

—multiplication: total (dimension)^2 threads as well, and each thread computes one for-loop for one final result.

C. More information about our parallel programming

Key instructions involved in this experiment:

- #include "omp.h"
- #pragma omp parallel num_threads(Thread_num)
- #pragma omp for private(i,j,k)
- omp_get_wtime();

Key points involved in this implementation:

- the integer array(int arr) is set to have 8 numbers(2,4,8, 16,32,64,100,200), and a for-loop in the main function will do a traversal to calculate each kind of thread allocating.
- performance are measured according to the time period.

V. Results and Analysis

(Differences can exist in every execution.)

A. Matrix Dimension = 1

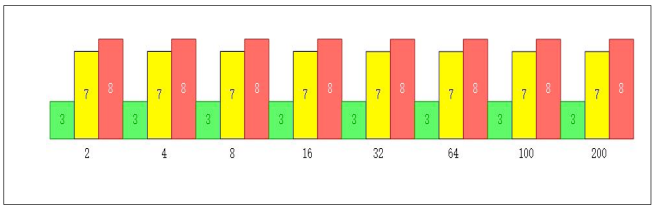
Table I. Serial Computing Time (s)

addition	subtraction	multiplication
0.0000	0.0000	0.0000

Table II. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0000	3.000	0.0000	1#inf	0.0000	1#inf
4	0.0000	3.000	0.0000	1#inf	0.0000	8.000
8	0.0000	3.000	0.0000	1#inf	0.0000	8.000
16	0.0000	3.000	0.0000	1#inf	0.0000	8.000
32	0.0000	1#inf	0.0000	7.000	0.0000	8.000
64	0.0000	3.000	0.0000	1#inf	0.0000	8.000
100	0.0000	3.000	0.0000	7.000	0.0000	1#inf
200	0.0000	3.000	0.0000	1#inf	0.0000	8.000

Figure I. Speedup-ThreadNum(conclusion)



B. Matrix Dimension = 10

Table III. Serial Computing Time (s)

addition	subtraction	multiplication
0.0000	0.0000	0.0000

Table IV. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0000	10.49	0.0000	23.25	0.0000	2.682
4	0.0000	10.49	0.0000	31.00	0.0000	1.283
8	0.0000	10.49	0.0000	23.25	0.0000	1.553
16	0.0000	10.50	0.0000	15.50	0.0000	2.107
32	0.0000	8.400	0.0000	15.50	0.0000	2.107
64	0.0000	7.000	0.0000	18.60	0.0000	2.185
100	0.0000	8.400	0.0000	18.60	0.0000	2.034
200	0.0000	8.400	0.0000	15.50	0.0000	1.788

Figure II.I Speedup-ThreadNum(add)(2-200)

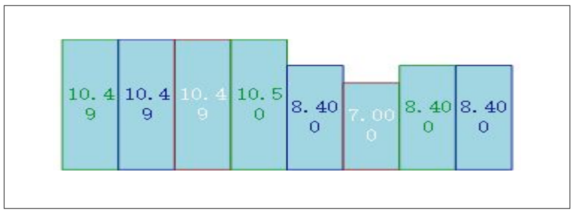


Figure II.II Speedup-ThreadNum(sub)(2-200)

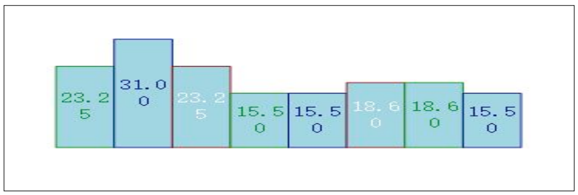
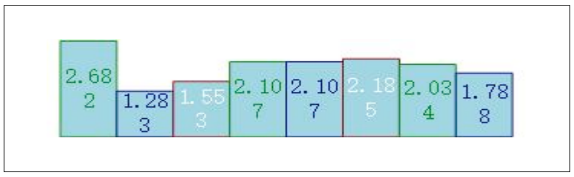


Figure II.III Speedup-ThreadNum(mul)(2-200)



C. Matrix Dimension = 100

Table V. Serial Computing Time (s)

addition	subtraction	multiplication
0.0003	0.0002	0.0060

Table VI. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0002	2.676	0.0001	1.798	0.0065	0.926
4	0.0001	2.437	0.0001	1.804	0.0067	0.893
8	0.0001	2.712	0.0001	1.946	0.0066	0.904
16	0.0002	1.317	0.0001	1.804	0.0068	0.874
32	0.0003	1.043	0.0001	1.686	0.0070	0.850
64	0.0001	1.893	0.0001	1.640	0.0098	0.610
100	0.0001	2.272	0.0002	1.592	0.0059	1.005
200	0.0001	2.000	0.0001	1.798	0.0059	1.017

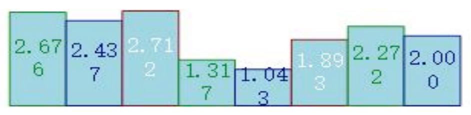


Figure III.I Speedup-ThreadNum(add)(2-200)

Figure III.II Speedup-ThreadNum(sub)(2-200)

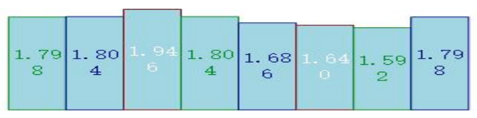
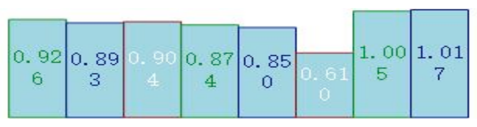


Figure III.III Speedup-ThreadNum(mul)(2-200)



D. Matrix Dimension = 1000

Table VII. Serial Computing Time (s)

addition	subtraction	multiplication
0.0167	0.0326	13.5668

Table VIII. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0148	1.125	0.0160	2.033	13.143	1.032
4	0.0100	1.673	0.0095	3.417	12.297	1.103
8	0.0087	1.921	0.0097	3.367	12.141	1.117
16	0.0097	1.721	0.0099	3.277	12.205	1.112
32	0.0102	1.628	0.0096	3.393	12.263	1.106
64	0.0099	1.688	0.0093	3.517	12.340	1.099
100	0.0108	1.544	0.0138	2.370	12.308	1.102
200	0.0121	1.379	0.0105	3.095	12.476	1.087

Figure IV.I Speedup-ThreadNum(add)(2-200)

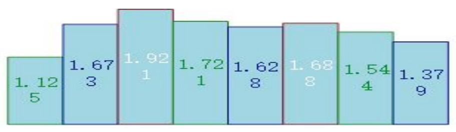


Figure IV.II Speedup-ThreadNum(sub)(2-200)

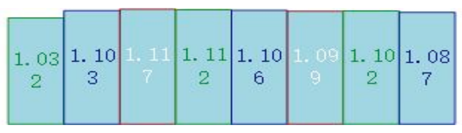


Figure IV.III Speedup-ThreadNum(mul)(2-200)

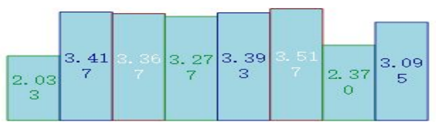
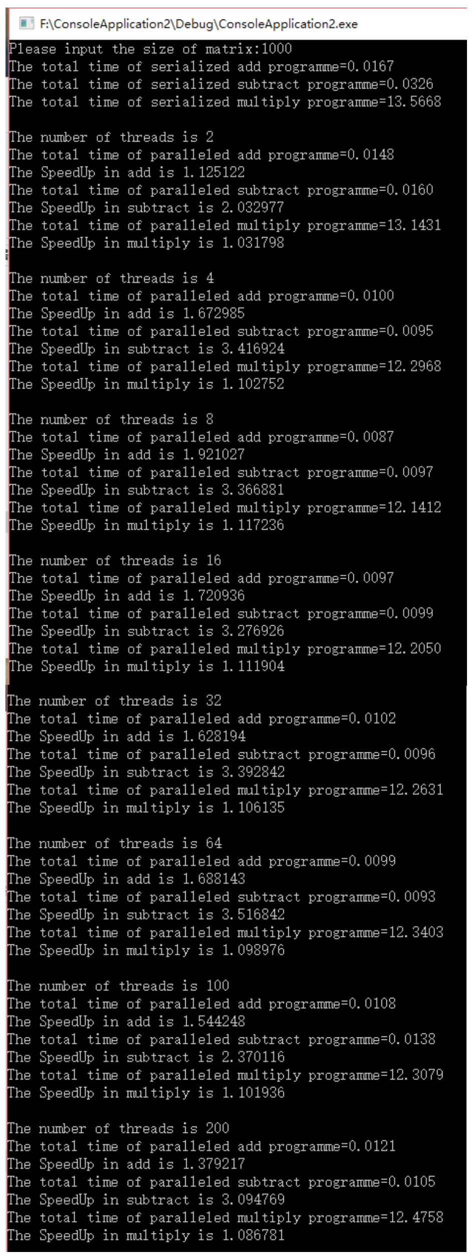


Figure IV.IV Result Screenshot(for 1000_dim)


F:\ConsoleApplication2\Debug\ConsoleApplication2.exe

```
Please input the size of matrix:1000
The total time of serialized add programme=0.0167
The total time of serialized subtract programme=0.0326
The total time of serialized multiply programme=13.5668

The number of threads is 2
The total time of paralleled add programme=0.0148
The SpeedUp in add is 1.125122
The total time of paralleled subtract programme=0.0160
The SpeedUp in subtract is 2.032977
The total time of paralleled multiply programme=13.1431
The SpeedUp in multiply is 1.031798

The number of threads is 4
The total time of paralleled add programme=0.0100
The SpeedUp in add is 1.672985
The total time of paralleled subtract programme=0.0095
The SpeedUp in subtract is 3.416924
The total time of paralleled multiply programme=12.2968
The SpeedUp in multiply is 1.102752

The number of threads is 8
The total time of paralleled add programme=0.0087
The SpeedUp in add is 1.921027
The total time of paralleled subtract programme=0.0097
The SpeedUp in subtract is 3.366881
The total time of paralleled multiply programme=12.1412
The SpeedUp in multiply is 1.117236

The number of threads is 16
The total time of paralleled add programme=0.0097
The SpeedUp in add is 1.720936
The total time of paralleled subtract programme=0.0099
The SpeedUp in subtract is 3.276926
The total time of paralleled multiply programme=12.2050
The SpeedUp in multiply is 1.111904

The number of threads is 32
The total time of paralleled add programme=0.0102
The SpeedUp in add is 1.628194
The total time of paralleled subtract programme=0.0096
The SpeedUp in subtract is 3.392842
The total time of paralleled multiply programme=12.2631
The SpeedUp in multiply is 1.106135

The number of threads is 64
The total time of paralleled add programme=0.0099
The SpeedUp in add is 1.688143
The total time of paralleled subtract programme=0.0093
The SpeedUp in subtract is 3.516842
The total time of paralleled multiply programme=12.3403
The SpeedUp in multiply is 1.098976

The number of threads is 100
The total time of paralleled add programme=0.0108
The SpeedUp in add is 1.544248
The total time of paralleled subtract programme=0.0138
The SpeedUp in subtract is 2.370116
The total time of paralleled multiply programme=12.3079
The SpeedUp in multiply is 1.101936

The number of threads is 200
The total time of paralleled add programme=0.0121
The SpeedUp in add is 1.379217
The total time of paralleled subtract programme=0.0105
The SpeedUp in subtract is 3.094769
The total time of paralleled multiply programme=12.4758
The SpeedUp in multiply is 1.086781
```

E. Key Summary

- speedup = serial_time / parallel_time
- efficiency = speedup / thread_num
- if thread number increase, execution time will decrease. (if thread number double, execution time will decrease to the half)
- if thread number increase, speedup will increase. (if thread number double, speedup will increase to almost double)
- if thread number increase, the efficiency of the program will decrease (relies in the pure time values). However, the efficiency will tend to be stable with the increasing of thread number.