

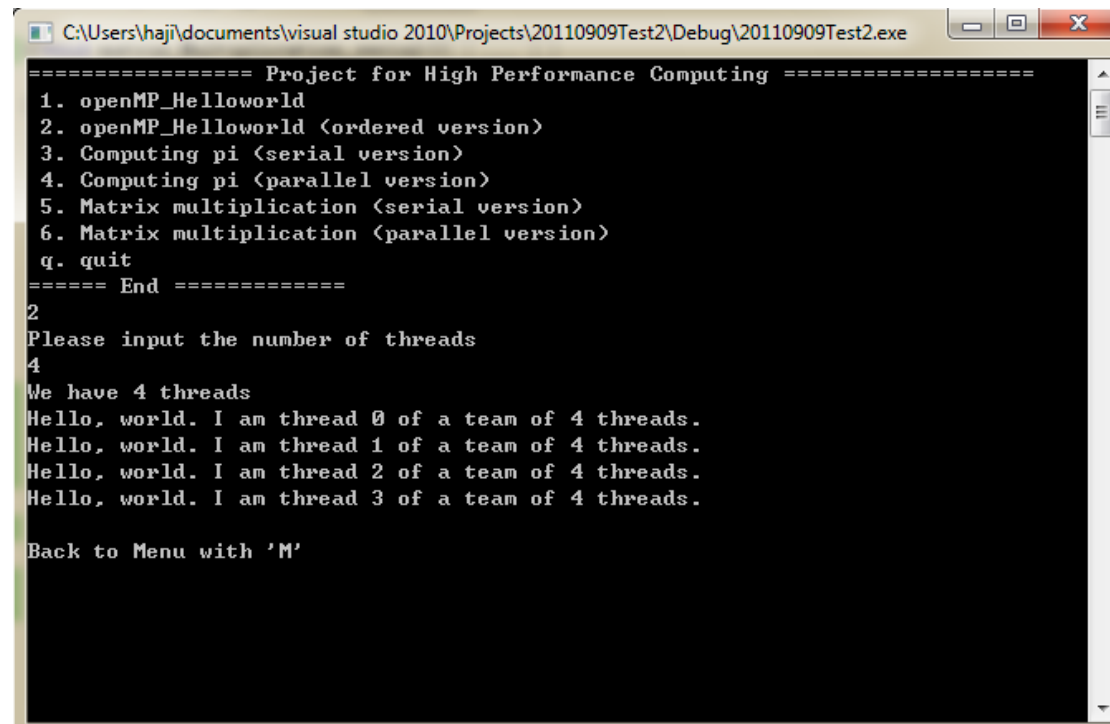
Report for HPC Project 1

Name: Haojian Jin.

Summery

Since Visual Studio has a good integration with OpenMP, I started my work on windows, and then adjusted the code to POSIX system later. The screenshot is captured on windows system, but the final executive application works on POSIX-friendly system as well.

The application I handed in is a console application:



```
C:\Users\haji\documents\visual studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe
===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
2
Please input the number of threads
4
We have 4 threads
Hello, world. I am thread 0 of a team of 4 threads.
Hello, world. I am thread 1 of a team of 4 threads.
Hello, world. I am thread 2 of a team of 4 threads.
Hello, world. I am thread 3 of a team of 4 threads.

Back to Menu with 'M'
```

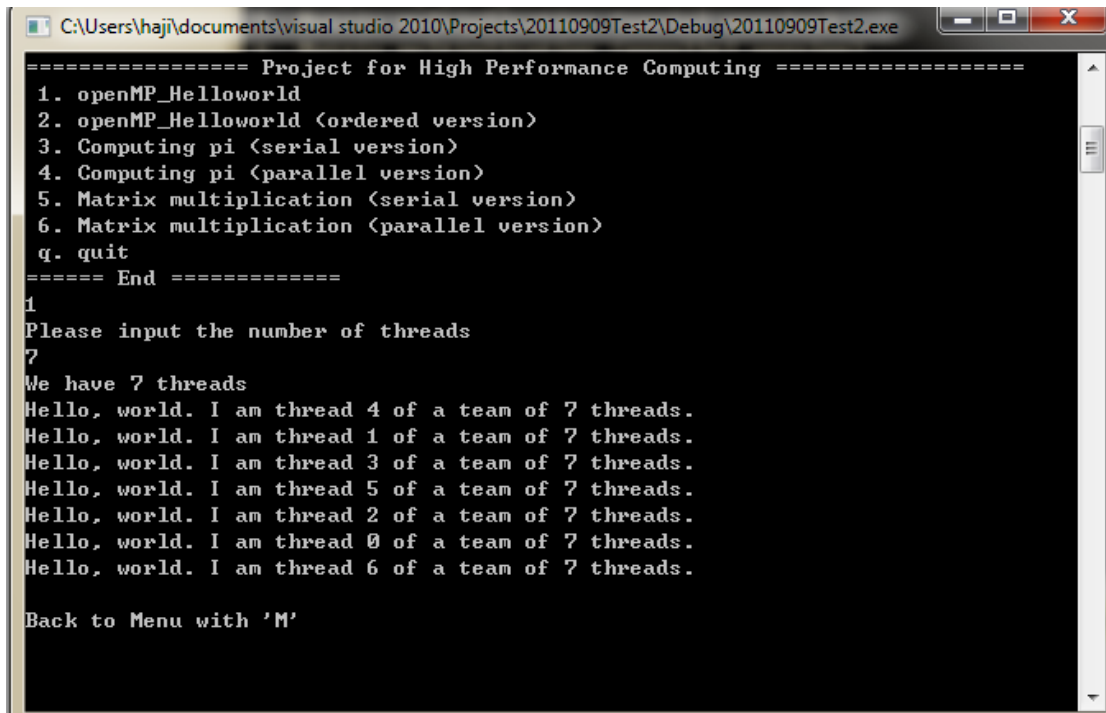
Choose the activity with number key → Input the number of threads → Special parameters may be needed in several activities.

Report for Question 1: Hello World

1. Serial version of “Hello, World”.

```
cin >> thread_num;
cout << "We have " << thread_num << " threads" << endl;
omp_set_num_threads(thread_num);
#pragma omp parallel firstprivate(thread_num)
{
    //std::cout<<"Hello,World! ThreadId="<< omp_get_thread_num()<<std::endl;
    printf("Hello, world. I am thread %d of a team of %d threads. \n",omp_get_thread_num(), omp_get_num_threads());
}
```

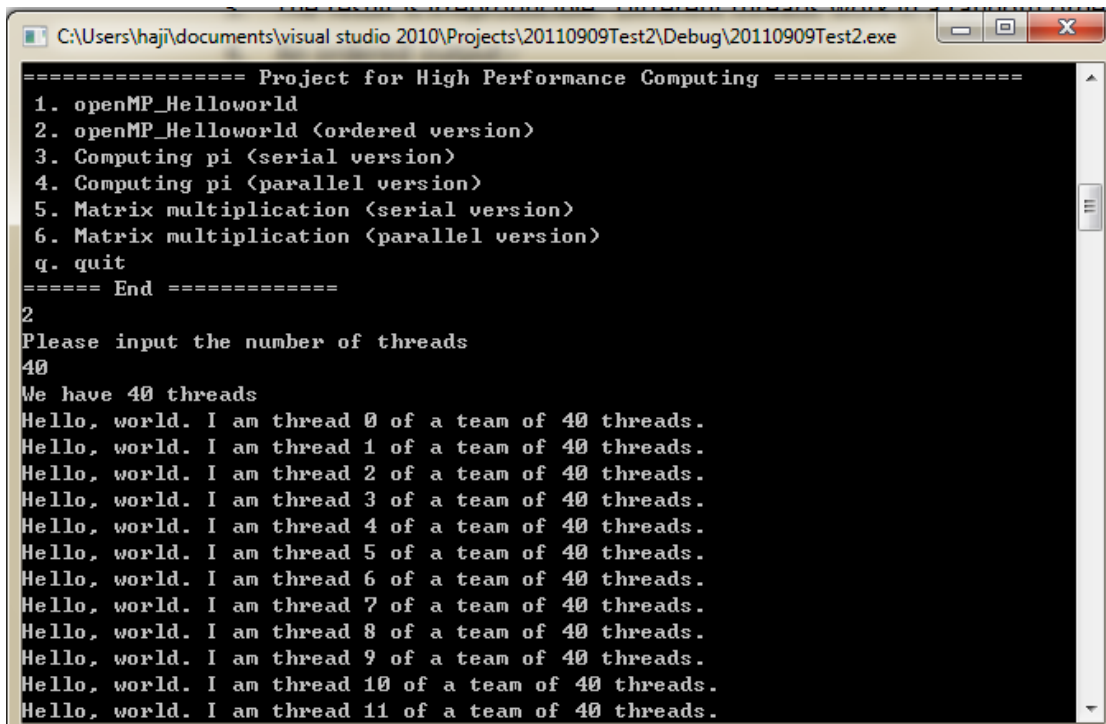
2. Runtime screenshot.



```
==== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
1
Please input the number of threads
7
We have 7 threads
Hello, world. I am thread 4 of a team of 7 threads.
Hello, world. I am thread 1 of a team of 7 threads.
Hello, world. I am thread 3 of a team of 7 threads.
Hello, world. I am thread 5 of a team of 7 threads.
Hello, world. I am thread 2 of a team of 7 threads.
Hello, world. I am thread 0 of a team of 7 threads.
Hello, world. I am thread 6 of a team of 7 threads.

Back to Menu with 'M'
```

3. The result is irreproducible. Different threads work in a random order.
4. An ordered output:



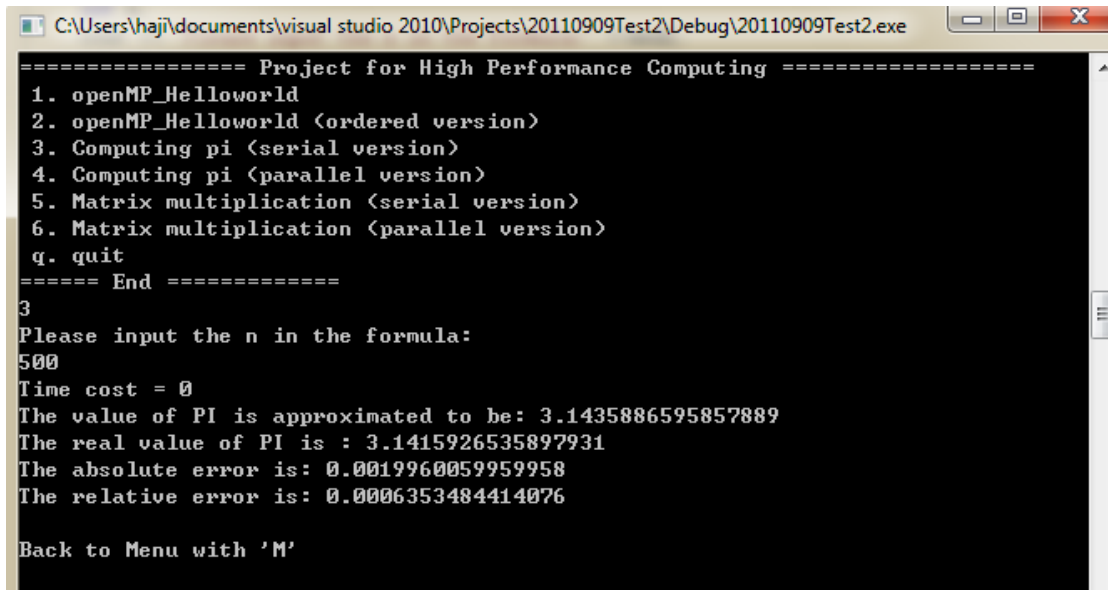
```
==== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
2
Please input the number of threads
40
We have 40 threads
Hello, world. I am thread 0 of a team of 40 threads.
Hello, world. I am thread 1 of a team of 40 threads.
Hello, world. I am thread 2 of a team of 40 threads.
Hello, world. I am thread 3 of a team of 40 threads.
Hello, world. I am thread 4 of a team of 40 threads.
Hello, world. I am thread 5 of a team of 40 threads.
Hello, world. I am thread 6 of a team of 40 threads.
Hello, world. I am thread 7 of a team of 40 threads.
Hello, world. I am thread 8 of a team of 40 threads.
Hello, world. I am thread 9 of a team of 40 threads.
Hello, world. I am thread 10 of a team of 40 threads.
Hello, world. I am thread 11 of a team of 40 threads.
```

5. The code of ordered output:

```
#pragma omp parallel firstprivate(thread_num), shared(cur_index)
{
    while(omp_get_thread_num() > cur_index)
    {
        sleep(100);
    }
    printf("Hello, world. I am thread %d of a team of %d threads. \n", omp_get_thread_num(), omp_get_num_threads());
    cur_index ++;
}
```

Computing PI

1. Serial code output:



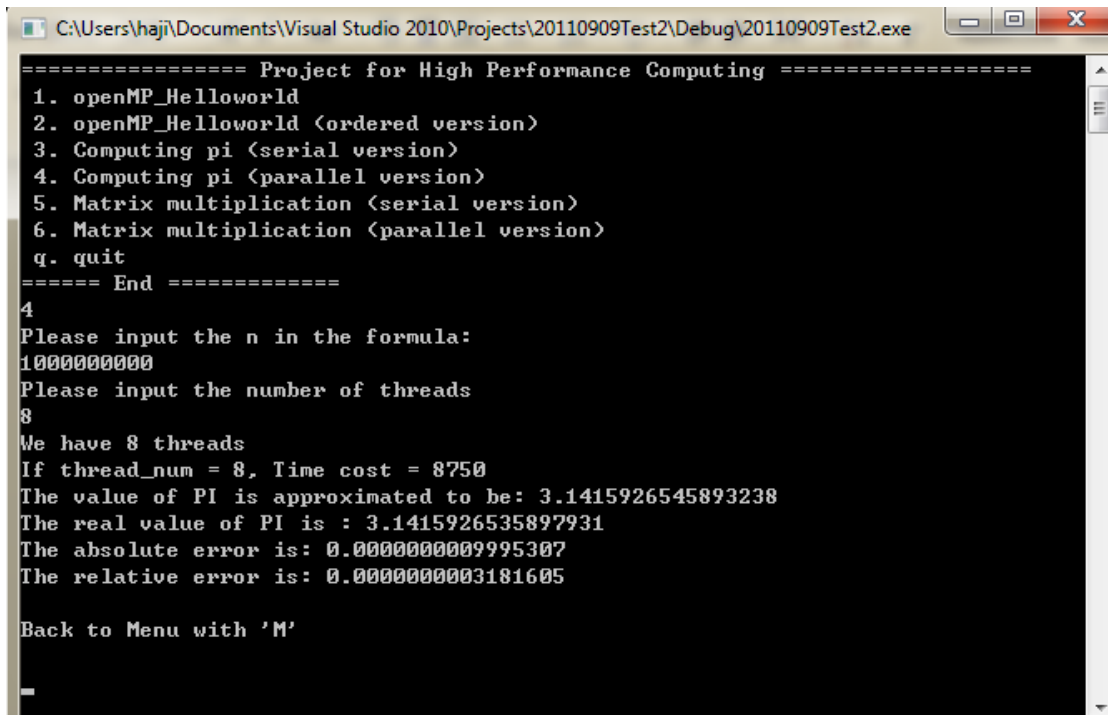
```
C:\Users\haji\documents\visual studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe

===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
3
Please input the n in the formula:
500
Time cost = 0
The value of PI is approximated to be: 3.1435886595857889
The real value of PI is : 3.1415926535897931
The absolute error is: 0.0019960059959958
The relative error is: 0.0006353484414076

Back to Menu with 'M'
```

2. Parallelize output:

While number of threads is 8:



```
C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe

===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
4
Please input the n in the formula:
1000000000
Please input the number of threads
8
We have 8 threads
If thread_num = 8, Time cost = 8750
The value of PI is approximated to be: 3.1415926545893238
The real value of PI is : 3.1415926535897931
The absolute error is: 0.000000009995307
The relative error is: 0.000000003181605

Back to Menu with 'M'
```

While number of threads is 1:

```
C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe
M
===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
4
Please input the n in the formula:
1000000000
Please input the number of threads
1
We have 1 threads
If thread_num = 1, Time cost = 18550
The value of PI is approximated to be: 3.1415926545880506
The real value of PI is : 3.1415926535897931
The absolute error is: 0.0000000009982575
The relative error is: 0.0000000003177552

Back to Menu with 'M'
```

Running in the serial code:

```
C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe
===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
3
Please input the n in the formula:
1000000000
Time cost = 15023
The value of PI is approximated to be: 3.1415926545880506
The real value of PI is : 3.1415926535897931
The absolute error is: 0.0000000009982575
The relative error is: 0.0000000003177552

Back to Menu with 'M'
```

3. Analysis about the output:

This code was running on my personal laptop, with an Intel® Core™2 Duo Processor T7700 (4M Cache, 2.40 GHz, 800 MHz FSB).

We can find the speed rank is:

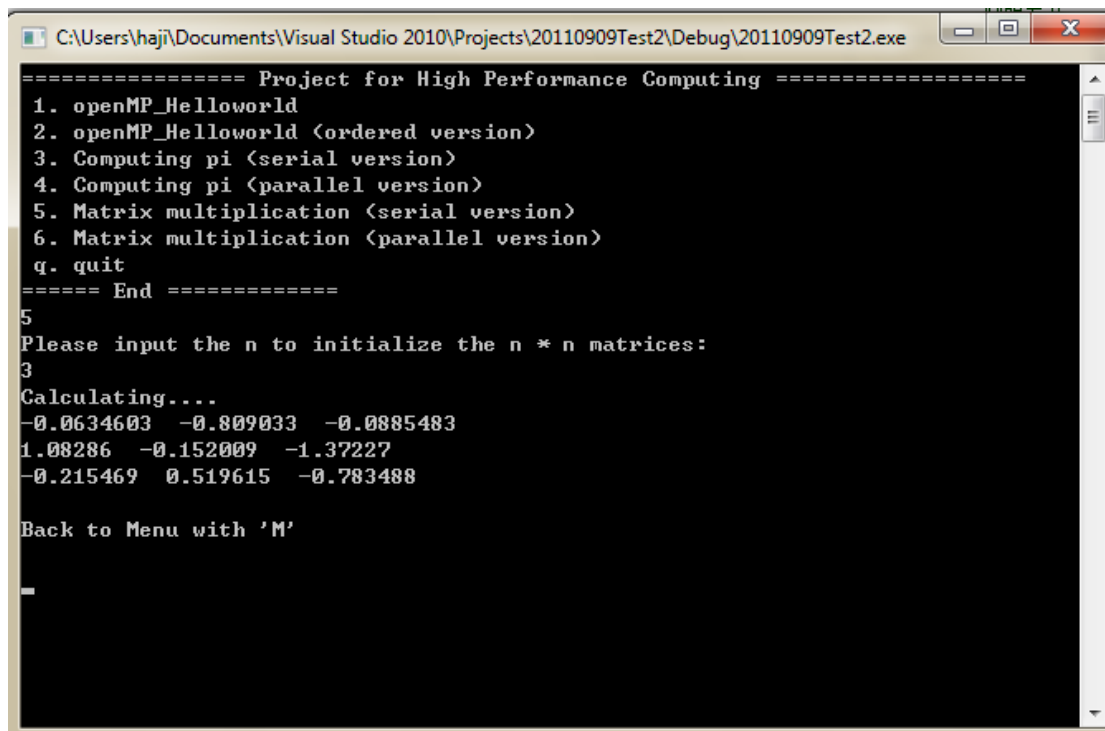
OpenMP code with 8 threads > Serial Code > OpenMP code with 1 thread

That's pretty make sense, since the system has to invest part of the resources on multi-processor computing. Besides, the OpenMP with 8 threads has a nearly double

performance as the OpenMP code with 1 thread. That's because of my laptop has a dual Core processor.

Matrix multiplication

1. The output of serial code while $n = 3$.



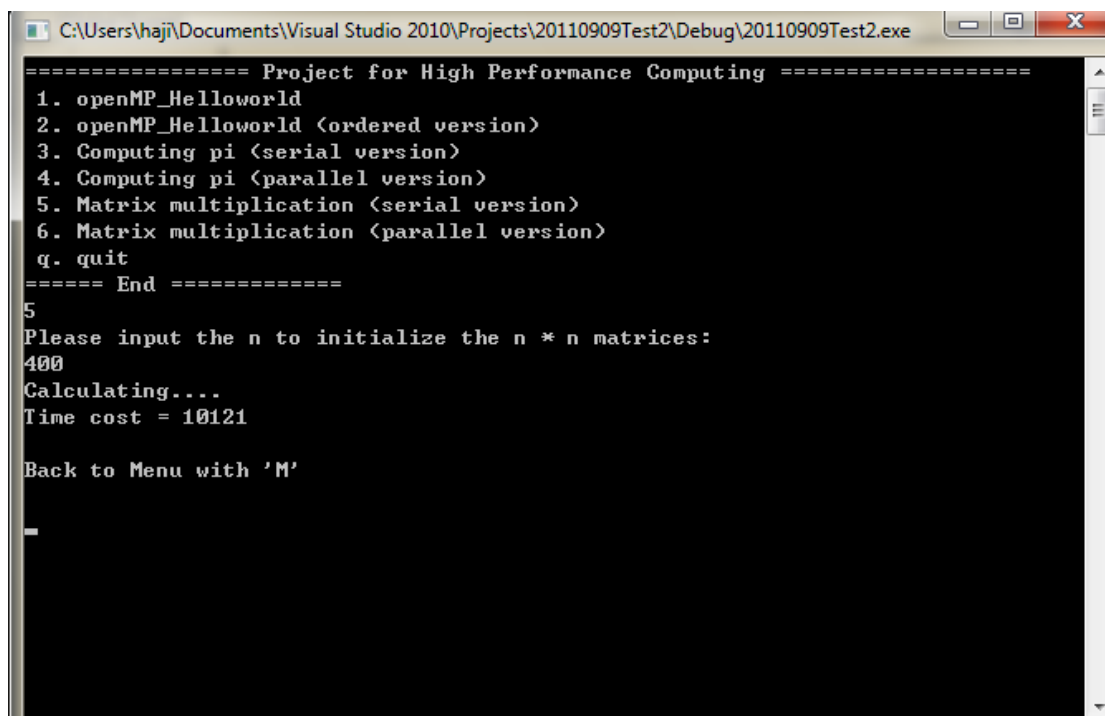
```
C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe

===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
5
Please input the n to initialize the n * n matrices:
3
Calculating....
-0.0634603  -0.809033  -0.0885483
1.08286  -0.152009  -1.37227
-0.215469  0.519615  -0.783488

Back to Menu with 'M'

-
```

While $n = 400$, the output is as follows. The total time cost of serial code is 10121ms.



```
C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe

===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
5
Please input the n to initialize the n * n matrices:
400
Calculating....
Time cost = 10121

Back to Menu with 'M'

-
```

2. The code of Matrix multiplication could be divided into 2 parts: Initialization & Multiplication. A more detail time cost analysis is as follows:

| | N=4 | N=40 | N=100 | N=200 | N=300 | N=400 |
|----------------|-----|------|-------|-------|-------|-------|
| Initialization | 0 | 7 | 24 | 67 | 114 | 197 |
| Multiplication | 1 | 30 | 153 | 1165 | 3999 | 10281 |

```

C:\Users\haji\Documents\Visual Studio 2010\Projects\20110909Test2\Debug\20110909Test2.exe
Part1: Time cost = 114
Time cost = 3999

Back to Menu with 'M'

m
===== Project for High Performance Computing =====
1. openMP_Helloworld
2. openMP_Helloworld <ordered version>
3. Computing pi <serial version>
4. Computing pi <parallel version>
5. Matrix multiplication <serial version>
6. Matrix multiplication <parallel version>
q. quit
===== End =====
5
Please input the n to initialize the n * n matrices:
400
Calculating....
Part1: Time cost = 197
Time cost = 10281

Back to Menu with 'M'

```

From the table above, we can figure out the multiplication part takes up the most time.

3. Since this time analysis in this problem should use the UNIX time command, so I rewrite the problem without a console menu.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|
| N=300 | 3862ms | 2158ms | 2250ms | 2351ms | 2326ms | 2327ms | 2274ms | 2316ms |
| N=400 | 9635ms | 5203ms | 5529ms | 5440ms | 5363ms | 5378ms | 5548ms | 5469ms |
| N=500 | 19419 | 10580 | 10741 | 10562 | 10669 | 10552 | 10751 | 11017 |

We can conclude several trends from the table above:

1. The time complexity of this multiplication is not linear. The speed of increasing the time costing is much faster than the increasing of matrix's size.
2. Since my processor is a dual core CPU, this work performs best when the number of threads is 2. We have more threads for this algorithm, the computer have to spend more resources on Scheduling, leading to a decrease in performance.
3. Comparing the difference between #=1 and #=2, it could be found that double threads application on dual core CPU has a nearly double performance as the serial application.

Program Structure:

```
#include "stdafx.h"
#include <iostream>
#include <string>
#include <omp.h>
#include <math.h>
#include <ctime>
#include <vector>

using namespace std;

//*****Utility fuction*****//

void sleep(unsigned int mseconds) { ... }

void printMainMenu() { ... }

//*****Utility fuction ends*****//

void openMP_Helloworld() { ... }

void openMP_Helloworld_Ordered() { ... }

void computing_PI_serial() { ... }

void computing_PI_parallel() { ... }

void matrix_Multiplication_serial() { ... }

void matrix_Multiplication_paralle() { ... }

void test() { ... }

int _tmain(int argc, _TCHAR* argv[]) { ... }
```