

OS Project 3 - Matrix Multiplication

Project for Computer Architecture & Operating Systems by Chentao Wu, 2016 Autumn Semester

1. Project Introduction:

Given two matrices A and B, where A is a matrix with M rows and K columns and matrix B contains K rows and N columns, the matrix product of A and B is matrix C, where C contains M rows and N columns. The entry in matrix C for row i column j ($C_{i,j}$) is the sum of the products of the elements for row i in matrix A and column j in matrix B. That is,

$$C_{i,j} = \sum_{n=1}^K A_{i,n} B_{n,j}$$

It is consisted by two steps:

1. Passing Parameters to Each Thread
2. Waiting for Threads to Complete

2. Project Environment:

VirtualBox with Linux Ubuntu 16.04.
Windows 10

3. Project Realization:

There're two steps: Passing Parameters to Each Thread and Waiting for Threads to Complete.

Part1 Passing Parameters to Each Thread

The parent thread will create $M \times N$ worker threads, passing each worker the values of row i and column / that it is to use in calculating the matrix product. This requires passing two parameters to each thread. The easiest approach with Pthreads and Win32 is to create a data structure using a struct. The members of this structure are i and j, and the structure appears as follows:

```
1 //struct for passing data to threads
2 struct v
3 {
4     int i; //row
5     int j; //column
6 };
```

Both the Pthreads and Win32 programs will create the worker threads using a strategy similar. The data pointer will be passed to either the pthread.create() (Pthreads) function or the CreateThreadO (Win32) function, which in turn will pass it as a parameter to the function that is to run as a separate thread.

```
1 //for (int i = 0; i < 1, i + + )
2     for(int j=0;j<NUM_THREADS;++j)
3     {
4         struct v *data=(struct v*)malloc(sizeof(struct v));
5         data->i=0;
6         data->j=j;
7         pthread_create(&workers[tmp],&attr,run,data);
```

```

8      //create thread, run is the start_routine and data is the
      arg
9      ++tmp;
10     if (tmp==NUM_THREADS)
11     {
12         for (int x=0;x<NUM_THREADS;++x)
13         {
14             pthread_join(workers[x],NULL);
15         }
16     }
17 }
18 //}

```

And here I didn't create $M \times N$ threads, and I will explain it now:

The thread is resource-expending, and a common personal computer can only run approximately 10 threads synchronously, and I have tried creating 100 threads to run, the result is that the speed up of computing didn't improve. So the performance will not improve infinitely as the threads increasing, when the threads' number is large, only part of them will execute, and the rest will wait in the queue. What's more, when I use the Linux in VirtualBox, I found that the time will not change, I will talk later at the **The problem I have met**. So I only created ten threads and divided the tasks equally to them.

```

1 inline void *run(void *data)
2 {
3     int sum=0;
4     struct v *d=(struct v*)data;
5     // redefine data
6     int j=d->j;
7     // i=d->i i = 0; To allocate the thread
8     for (int m = j; m<M ; m+=NUM_THREADS) //row
9     {
10         for (int n = 0; n < N; ++n)
11         {
12             sum=0;
13             for (int e = 0; e < K; ++e)
14             {
15                 sum+=A[m][e]*B[e][n];
16             }
17             C[m][n]=sum;
18         }
19     }
20     pthread_exit(0);
21 }

```

Part2 Waiting for Threads to Complete

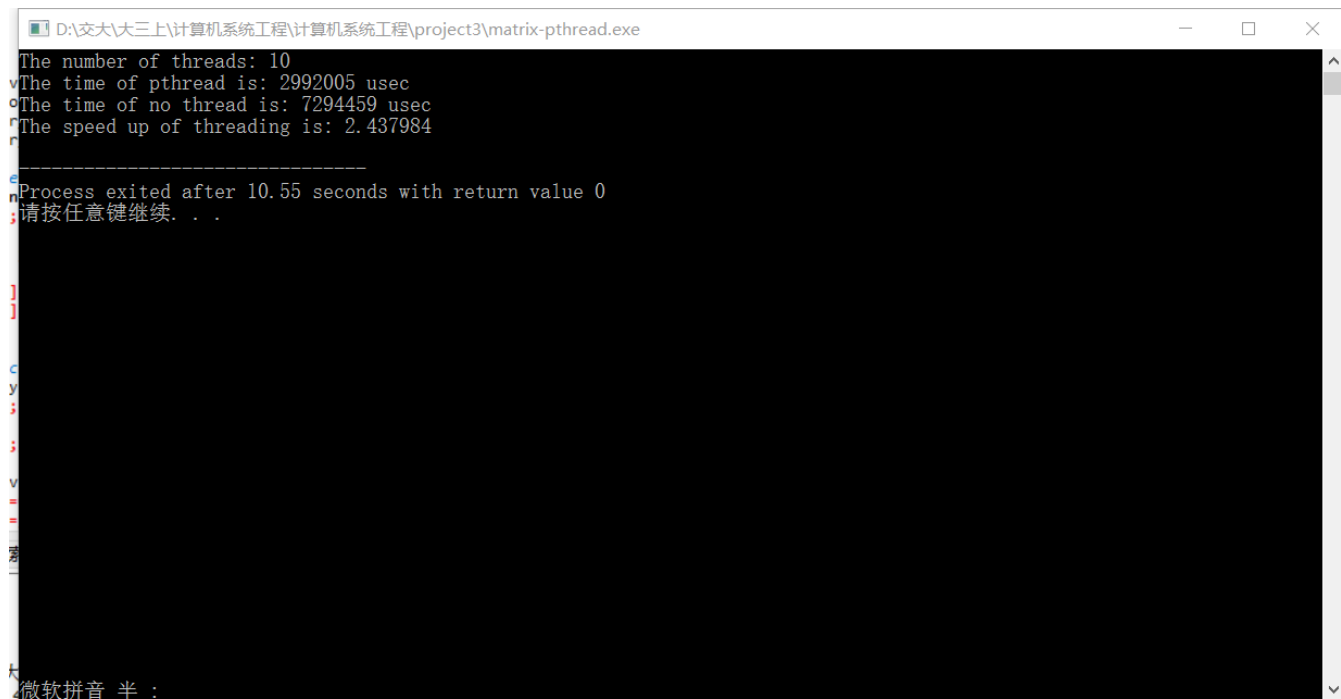
Once all worker threads have completed, the main thread will output the product contained in matrix C. This requires the main thread to wait for all worker threads to finish before it can output the value of the matrix product. Several different strategies can be used to enable a thread to wait for other threads to finish.

As we can see in the former code, I used a thread count to wait for threads, if $tmp(count) ==$

`NUM_THREADS`, the program will use `pthread_join()` to enclose the join operation within a simple for loop.

4. Project Result: I made a simple comparison between the time program running with thread and without thread. Here's the result:

1. Windows with 10 threads

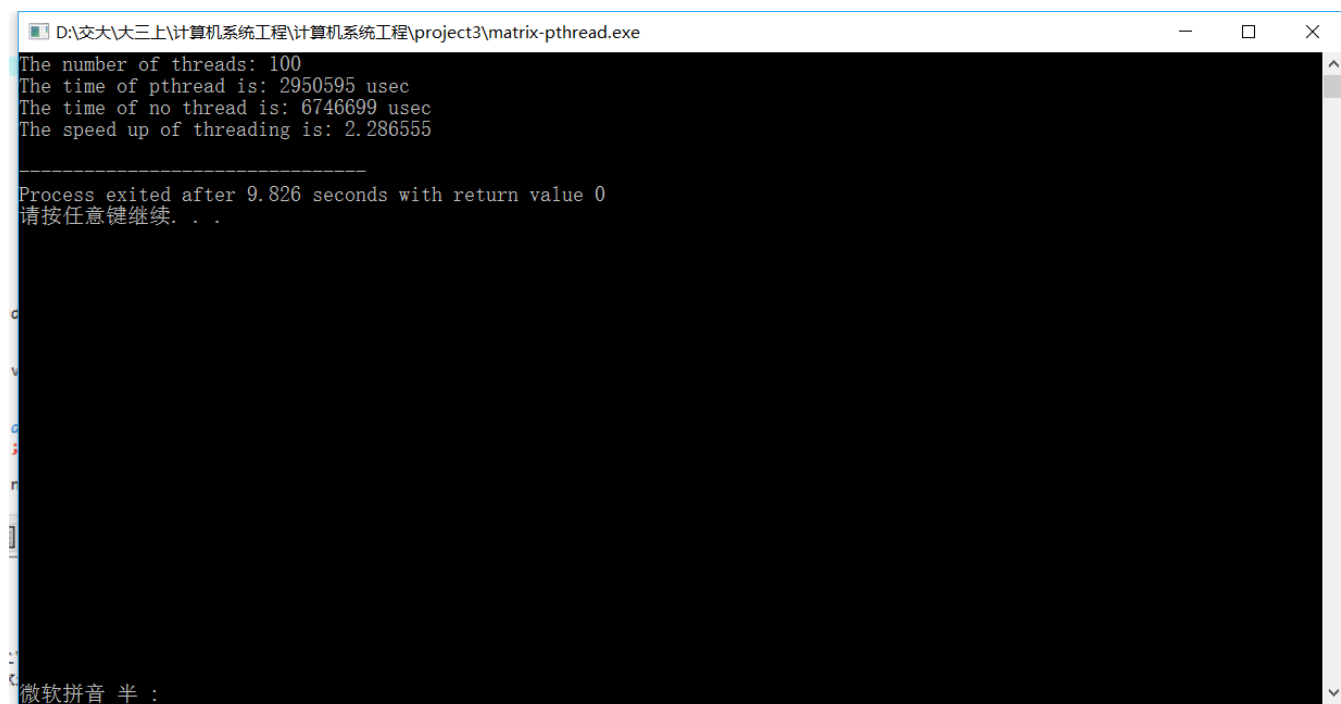


```
D:\交大\大三上\计算机系统工程\计算机系统工程\project3\matrix-pthread.exe
The number of threads: 10
The time of pthread is: 2992005 usec
The time of no thread is: 7294459 usec
The speed up of threading is: 2.437984

-----
Process exited after 10.55 seconds with return value 0
请按任意键继续. . .
```

图 1: Result: Windows with 10 threads

2. Windows with 100 threads



```
D:\交大\大三上\计算机系统工程\计算机系统工程\project3\matrix-pthread.exe
The number of threads: 100
The time of pthread is: 2950595 usec
The time of no thread is: 6746699 usec
The speed up of threading is: 2.286555

-----
Process exited after 9.826 seconds with return value 0
请按任意键继续. . .
```

图 2: Result: Windows with 100 threads

3. Linux with 10 threads(single CPU) in VirtualBox

```
xiaolanchong@xiaolanchong-VirtualBox:~$ g++ matrix-pthread.cpp -lpthread
xiaolanchong@xiaolanchong-VirtualBox:~$ g++ matrix-pthread.cpp -o thread -lpthread
xiaolanchong@xiaolanchong-VirtualBox:~$ ./thread
The number of threads: 10
The time of pthread is: 8114939 usec
The time of no thread is: 8071881 usec
The speed up of threading is: 0.994694
xiaolanchong@xiaolanchong-VirtualBox:~$ ./thread
The number of threads: 10
The time of pthread is: 7748503 usec
The time of no thread is: 7774977 usec
The speed up of threading is: 1.003417
xiaolanchong@xiaolanchong-VirtualBox:~$
```

图 3: Result: Linux with 10 threads(single CPU)

4. Linux with 10 threads(Four CPU) in VirtualBox

```
xiaolanchong@xiaolanchong-VirtualBox: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

xiaolanchong@xiaolanchong-VirtualBox:~$ g++ matrix-pthread.cpp -o thread -lpthread
xiaolanchong@xiaolanchong-VirtualBox:~$ ./thread
The number of threads: 10
The time of pthread is: 1925329 usec
The time of no thread is: 6381541 usec
The speed up of threading is: 3.314520
xiaolanchong@xiaolanchong-VirtualBox:~$
```

图 4: Result: Linux with 10 threads(Four CPU)

5. The problem I have met

As we can see in the **Result**: at first I find that the time is almost same in the VirtualBox's Linux, and I googled it online, finding that the linux creating by VirtualBox will only be allocated one CPU at first, as a result, the thread is a trick in such environment, because the Linux can

only run one thread at a time, which made no difference to calculating without threads. And after I changed the setting:

Then the time speed-up is regained.

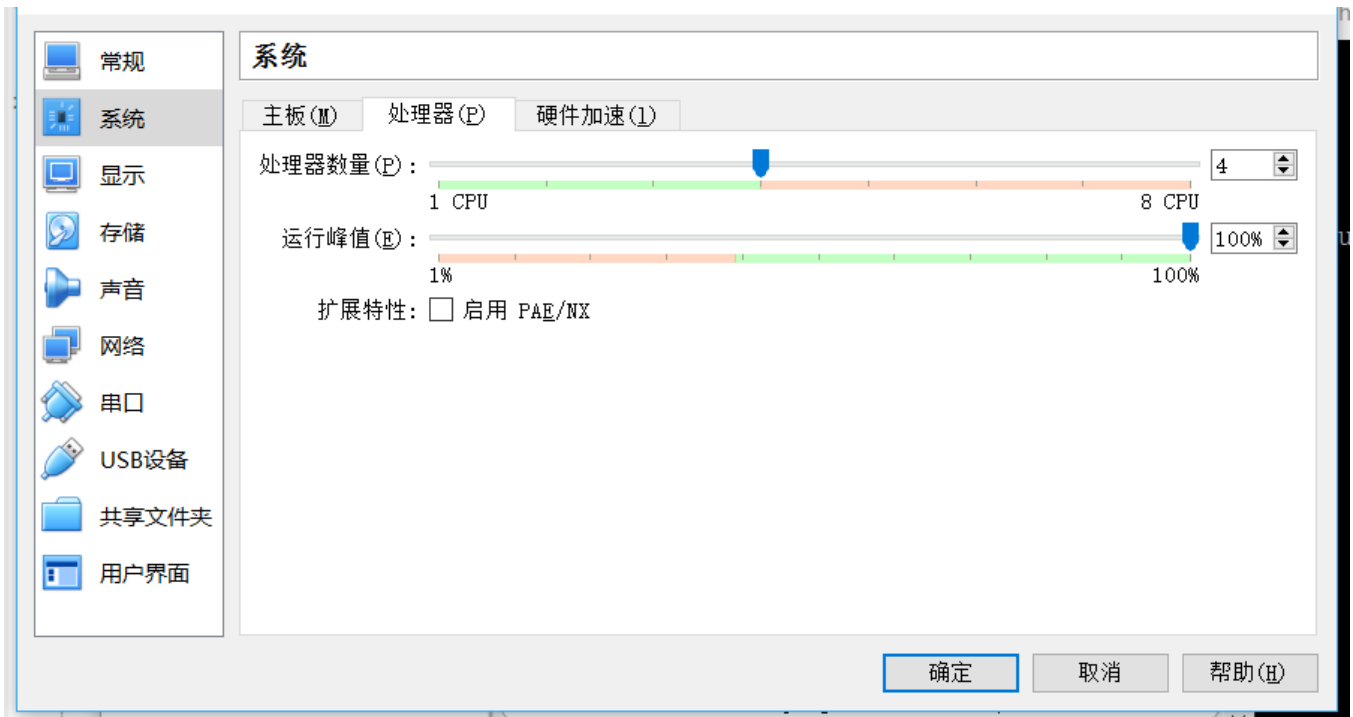


图 5: Result: Setting

6. Harvest

Through this project I learned a lot knowledge about thread and how to initial and create the threads at a program. And dealing with the problem about linux with multiprocessors makes me understand the threads more deeply. The operating system is a very fun thing to manipulate, and solving problems makes me very proudable.

7. Code

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <malloc.h>
4 #include <sys/time.h>
5 #include <time.h>
6 #include <stdlib.h>
7 #include <ctime>
8
9 #define M 1000
10 #define K 1000
11 #define N 1000
12 #define NUM_THREADS 10
13
14 int A[M][K];
```

```

15 int B[K][N];
16 int C[M][N];
17 int D[M][N];
18
19 //struct for passing data to threads
20 struct v
21 {
22     int i; //row
23     int j; //column
24 };
25
26 inline void *run(void *data)
27 {
28     int sum=0;
29     struct v *d=(struct v*)data;
30     // redefine data
31     int j=d->j;
32     // i=d->i i = 0; To allocate the thread
33     for (int m = j; m<M ; m+=NUM_THREADS) //row
34     {
35         for (int n = 0; n < N; ++n)
36         {
37             sum=0;
38             for (int e = 0; e < K; ++e)
39             {
40                 sum+=A[m][e]*B[e][n];
41             }
42             C[m][n]=sum;
43         }
44     }
45     pthread_exit(0);
46 }
47
48
49
50 int main(){
51     printf("The number of threads: %d\n", NUM_THREADS);
52     struct timeval start, end, start2, end2;
53     pthread_t workers[NUM_THREADS]; //300 pthread
54     pthread_attr_t attr; //To initial the pthread attribute
55     pthread_attr_init(&attr);
56
57     //initialize
58     srand(unsigned(time(0)));
59     for(int i=0;i<M;++i)
60     {
61         for(int j=0;j<N;++j)
62         {
63             A[i][j]=rand()%100;
64             B[i][j]=rand()%100;

```

```

65     }
66 }
67 //parallel calculate
68 gettimeofday(&start, NULL);
69 int tmp = 0;
70 //for (int i = 0; i < 1, i + + )
71 for(int j=0;j<NUM_THREADS;++j)
72 {
73     struct v *data=(struct v*)malloc(sizeof(struct v));
74     data->i=0;
75     data->j=j;
76     pthread_create(&workers[tmp],&attr,run,data);
77     //create thread, run is the start_routine and data is the
        arg
78     ++tmp;
79     if(tmp==NUM_THREADS)
80     {
81         for(int x=0;x<NUM_THREADS;++x)
82         {
83             pthread_join(workers[x],NULL);
84         }
85     }
86 }
87 //}
88 gettimeofday(&end, NULL);
89 int timeuse = 1000000 * (end.tv_sec - start.tv_sec) + end.
        tv_usec - start.tv_usec;
90 printf("The time of pthread is: %d usec\n", timeuse);
91
92 //direct calculate
93 gettimeofday(&start2, NULL);
94 int sum;
95 for(int i=0;i<M;++i)
96 {
97     for(int j=0;j<N;++j)
98     {
99         sum=0;
100         for(int e=0;e<K;++e)
101         {
102             sum+=A[i][e]*B[e][j];
103         }
104         D[i][j]=sum;
105     }
106 }
107 gettimeofday(&end2, NULL);
108 int timeuse2 = 1000000*(end2.tv_sec-start2.tv_sec)+end2.tv_usec -
        start2.tv_usec;
109 printf("The time of no thread is: %d usec\n",timeuse2);
110 double timeuse2_dou = timeuse2;
111 printf("The speed up of threading is: %f\n", timeuse2_dou/

```

```

        timeuse );
112  for(int i=0;i<M;++i)
113  {
114      for(int j=0;j<N;++j)
115      {
116          if(C[i][j]!=D[i][j])
117          {
118              printf("i: %d j: %d C: %d D: %d\n",i,j,C[i][j],D[i][
                  j]);
119          }
120      }
121
122  }
123  return 0;
124 }

```