

# Experiment Report Three

Course Name: Operating System

Experiment Name: Multi-thread Matrix Multiplication

Student Name: 李添豪

Student ID: 5140309349

Class Number: F1403015

Catalogue

[Experiment Report Three 1](#_Toc450385624)

[Experiment Environment 3](#_Toc450385625)

[Experiment Target 3](#_Toc450385626)

[Experiment Design 3](#_Toc450385627)

[Experiment Code 3](#_Toc450385628)

[Multi-Thread 3](#_Toc450385629)

[Single Thread: 6](#_Toc450385630)

[Experiment Data Analysis: 9](#_Toc450385631)

## Experiment Environment

Operating System: Ubuntu LTS 16.04 (based on VMware)

Kernel Version: 4.4.6

CPU: 4 processors, 8 cores

## Experiment Target

1. Using multi-thread method to do a matrix multiplication, record the time of the program to get the answer comparing with single-thread method.
2. Change the parameters (referring to M N K), do (1) again.
3. Analyze the result.

## Experiment Design

1. Create Three matrix:

A[M][N], B[N][K], C[M][K]

2.

Multi-Thread: Creating M threads to do the calculation.

Single-Thread: do calculation with only one thread.

## Experiment Code

### Multi-Thread

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <pthread.h>

#define M 2

#define N 10000

#define K 10000

int **\*\***a**;**

int **\*\***b**;**

int **\*\***c**;**

struct position

**{**

int i**;**

int j**;**

**};**

void**\*** func**(**void **\*** arg**)**

**{**

struct position **\***p**=(**struct position **\*)**arg**;**

int sum**=**0**;**

**for(**int j**=**0**;**j**<**K**;++**j**){**

sum**=**0**;**

**for(**int i**=**0**;**i**<**N**;++**i**){**

sum**+=**a**[**p**->**i**][**i**]\***b**[**i**][**j**];**

**}**

c**[**p**->**i**][**j**]=**sum**;**

**}**

printf**(**"Thread %d exits!\n"**,**p**->**i**);**

pthread\_exit**(NULL);**

**}**

int main**()**

**{**

printf**(**"Thread:\n"**);**

a**=(**int**\*\*)**calloc**(**M**,sizeof(**int**\*));**

b**=(**int**\*\*)**calloc**(**N**,sizeof(**int**\*));**

c**=(**int**\*\*)**calloc**(**M**,sizeof(**int**\*));**

**for(**int i**=**0**;**i**<**M**;++**i**)** a**[**i**]=(**int**\*)**calloc**(**N**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**N**;++**i**)** b**[**i**]=(**int**\*)**calloc**(**K**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**M**;++**i**)**c**[**i**]=(**int**\*)**calloc**(**K**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**M**;++**i**){**

**for(**int j**=**0**;**j**<**N**;++**j**){**

**if(**random**()%**2**==**1**)** a**[**i**][**j**]=**rand**()%**2**;**

**else** a**[**i**][**j**]=-**rand**()%**2**;**

**}**

**}**

**for(**int i**=**0**;**i**<**N**;++**i**){**

**for(**int j**=**0**;**j**<**K**;++**j**){**

**if(**random**()%**2**==**1**)** b**[**i**][**j**]=**rand**()%**2**;**

**else** b**[**i**][**j**]=-**rand**()%**2**;**

**}**

**}**

/\*

printf("A:\n");

for(int i=0;i<M;++i){

for(int j=0;j<N;++j){

printf("%d\t",a[i][j]);

}

printf("\n");

}

printf("B:\n");

for(int i=0;i<N;++i){

for(int j=0;j<K;++j){

printf("%d\t",b[i][j]);

}

printf("\n");

}

\*/

int err**;**

struct position**\***p**[**M**];**

**for(**int i**=**0**;**i**<**M**;++**i**)** p**[**i**]=(**struct position**\*)**calloc**(**1**,sizeof(**struct position**));**

pthread\_t**\*** pthread**[**M**];**

**for(**int i**=**0**;**i**<**M**;++**i**){**

p**[**i**]->**i**=**i**;**

err**=**pthread\_create**(&**pthread**[**i**],NULL,**func**,(**void **\*)**p**[**i**]);**

**}**

**for(**int i**=**0**;**i**<**M**;++**i**){**

pthread\_join**(**pthread**[**i**],NULL);**

**}**

/\*

printf("C:\n");

for(int i=0;i<M;++i){

for(int j=0;j<K;++j){

printf("%d\t",c[i][j]);

}

printf("\n");

}

\*/

**return** 0**;**

**}**

### Single Thread:

#include <stdio.h>

#include <stdlib.h>

#include<malloc.h>

#include <pthread.h>

#define M 2

#define N 10000

#define K 10000

int **\*\***a**;**

int **\*\***b**;**

int **\*\***c**;**

struct position

**{**

int i**;**

int j**;**

**};**

void**\*** func**(**void **\*** arg**)**

**{**

struct position **\***p**=(**struct position **\*)**arg**;**

int sum**=**0**;**

**for(**int i**=**0**;**i**<**N**;++**i**){**

sum**+=**a**[**p**->**i**][**i**]\***b**[**i**][**p**->**j**];**

**}**

c**[**p**->**i**][**p**->**j**]=**sum**;**

**return** **NULL;**

**}**

int main**()**

**{**

a**=(**int**\*\*)**calloc**(**M**,sizeof(**int**\*));**

b**=(**int**\*\*)**calloc**(**N**,sizeof(**int**\*));**

c**=(**int**\*\*)**calloc**(**M**,sizeof(**int**\*));**

**for(**int i**=**0**;**i**<**M**;++**i**)** a**[**i**]=(**int**\*)**calloc**(**N**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**N**;++**i**)** b**[**i**]=(**int**\*)**calloc**(**K**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**M**;++**i**)**c**[**i**]=(**int**\*)**calloc**(**K**,sizeof(**int**));**

**for(**int i**=**0**;**i**<**M**;++**i**){**

**for(**int j**=**0**;**j**<**N**;++**j**){**

**if(**random**()%**2**==**1**)** a**[**i**][**j**]=**rand**()%**2**;**

**else** a**[**i**][**j**]=-**rand**()%**2**;**

**}**

**}**

**for(**int i**=**0**;**i**<**N**;++**i**){**

**for(**int j**=**0**;**j**<**K**;++**j**){**

**if(**random**()%**2**==**1**)** b**[**i**][**j**]=**rand**()%**2**;**

**else** b**[**i**][**j**]=-**rand**()%**2**;**

**}**

**}**

/\*

printf("A:\n");

for(int i=0;i<M;++i){

for(int j=0;j<N;++j){

printf("%d\t",a[i][j]);

}

printf("\n");

}

printf("B:\n");

for(int i=0;i<N;++i){

for(int j=0;j<K;++j){

printf("%d\t",b[i][j]);

}

printf("\n");

}

\*/

struct position**\***p**;**

p**=(**struct position**\*)**calloc**(**1**,sizeof(**struct position**));**

**for(**int i**=**0**;**i**<**M**;++**i**){**

**for(**int j**=**0**;**j**<**K**;++**j**){**

p**->**i**=**i**;**

p**->**j**=**j**;**

func**((**void **\*)**p**);**

**}**

**}**

/\*

printf("C:\n");

for(int i=0;i<M;++i){

for(int j=0;j<K;++j) printf("%d\t",c[i][j]);

printf("\n");

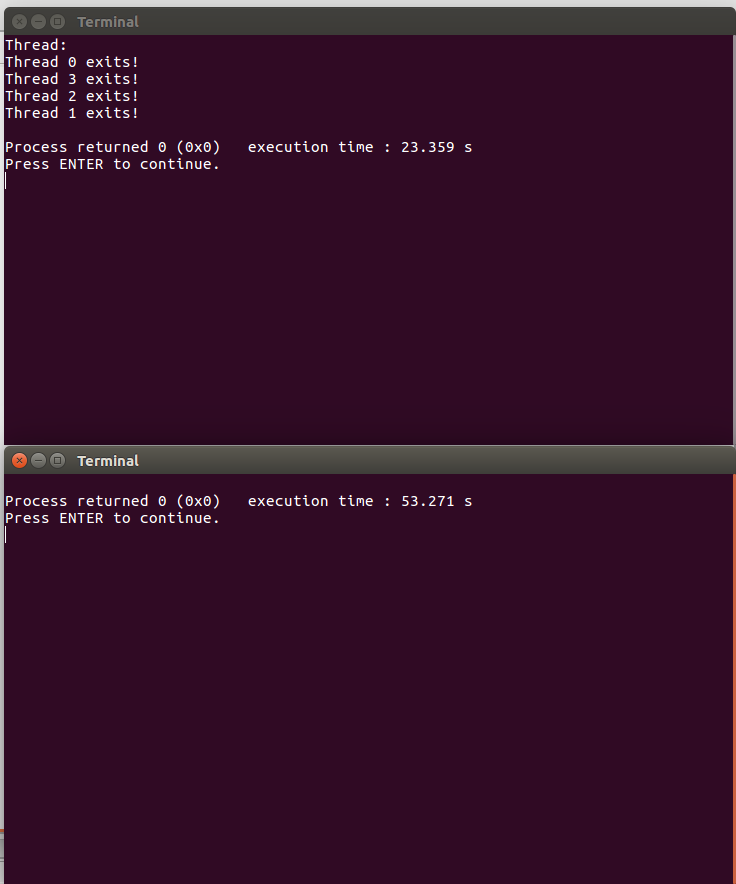
}

\*/

**return** 0**;**

**}**

**Test:**



## Experiment Data Analysis:



As what we can see from the sheet above, when thread increase from 1 to 4, the efficiency of the program is growing, but when the number of thread reaches 4, the efficiency of the program doesn’t grow up anymore.

The reason of this should be the CPU number of the PC we run this program which has 4 process, each process has two cores. So when the number grows more the number of the process, it can never improve the concurrency of the CPU.