高性能并行计算第 2 次作业

姓名：姚虎成 学号：2020317110033

代码地址：

矩阵相乘的 C 语言程序路径：

/home2/2020317110033/matrix_2/multi_mat.c

C 语言条件编译代码路径：

/home2/2020317110033/openmp_3/hello_con.c

OpenMP hello world 以及条件编译代码路径：

/home2/2020317110033/openmp_3/openmp_hello_con.c

OpenMP 环境变量代码路径：

/home2/2020317110033/openmp_3/openmp_hello_env.c

实验结果：

（一）、N 维矩阵相乘

程序的运行逻辑是，使用 ./multi_mat 命令运行 multi_mat 的可执行程序，出现提示语 Please input size of matrix N: ，每次输入不同的数组维度 N，即可输出运行时间，如下：

```
the matrix_multiply program running for    0.570s。
```

具体代码见附录 1。

表 1 是改变维度的 N，用 C 语言中的 clock()函数计算了程序运行的时间。图 1 是将运行时间和数组维度绘制散点图，并进行了三次方拟合。随着维度的增加，时间呈三次幂增长。

表 1 N-Time

N（维）	200	400	600	800	1000	1500	1800	2000
Time（秒）	0.030	0.270	1.060	2.550	4.980	21.190	48.140	71.840

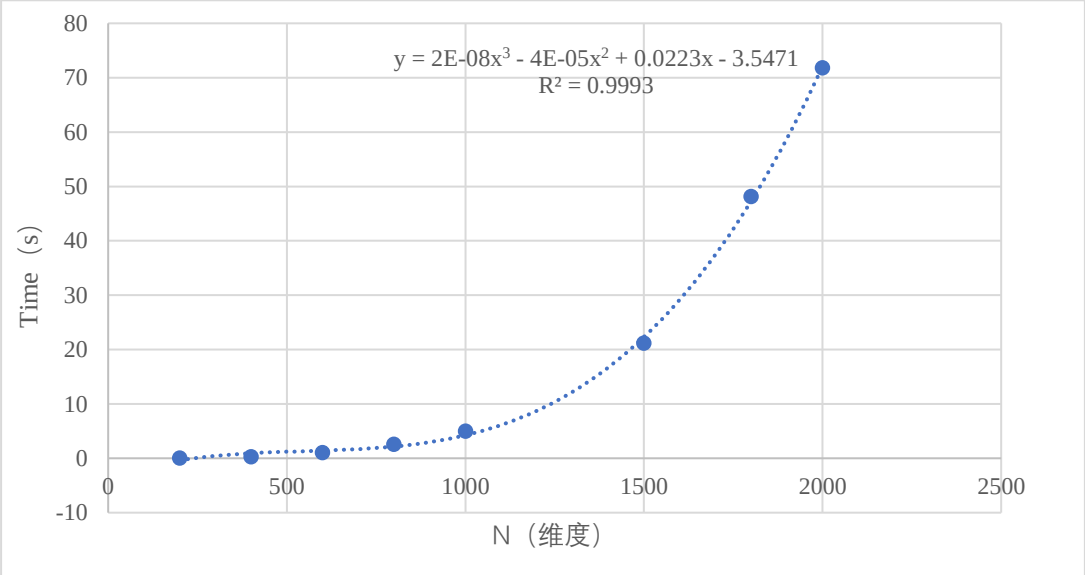

图 1 N-Time 图

（二）、C 语言条件编译

通过使用 gcc -D 参数进行条件编译，使用不同的命令参数，可以编译不同的 hello world 语言版本。具体代码见附录 2。

编译中文版本命令：

```
gcc hello_con.c -o hello_con -D cn
```

中文版本 hello world 输出如下：

```
你好，世界
```

编译英文版本命令：

```
gcc hello_con.c -o hello_con
```

英文版 hello world 输出如下：

```
Hello World
```

（三）、OpenMP hello world 以及条件编译

同上 hello world 不同语言版本的条件编译一样，OpenMP hello world 也是使用 gcc -D 参数进行条件编译，在编译前，需要设置 omp 的线程数目，命令 export OMP_NUM_THREADS=8，具体代码见附录 3。

在运行编译好的程序之前，我们需要设置 OpenMP 的并行线程数，可以直接在命令行下 export 设置即可，代码如下：

```
export OMP_NUM_THREADS=8
```

编译中文版本的编译代码：

```
gcc -fopenmp openmp_hello_con.c -o openmp_hello_con -D cn
```

中文版本 OpenMP hello world 输出如下：

```
你好(4)世界(4)
你好(6)世界(6)
你好(1)世界(1)
你好(2)世界(2)
你好(0)你好(5)世界(0)
你好(3)世界(3)
世界(5)
你好(7)世界(7)
```

编译英文版本的编译代码：

```
gcc -fopenmp openmp_hello_con.c -o openmp_hello_con
```

英文版本 OpenMP hello world 输出如下：

```
hello(7)world(7)
hello(5)world(5)
hello(2)world(2)
hello(6)world(6)
hello(0)hello(4)world(4)
world(0)
hello(1)world(1)
hello(3)world(3)
```

（四）、OpenMP 环境变量

通过编写 OpenMP 程序输出 OpenMP 所有环境变量，具体代码见附录 4。通过 gcc -fopenmp 来编译写好的 openmp_hello_env.c 文件，具体代码如下：

```
gcc -fopenmp openmp_hello_env.c -o    openmp_hello_env
```

运行 ./openmp_hello_env 可执行程序得到输出如下：

```
Hello World from OMP thread 5
Hello World from OMP thread 4
Hello World from OMP thread 6
Hello World from OMP thread 3
Hello World from OMP thread 7
Hello World from OMP thread 1
Hello World from OMP thread 0
The environment infomation about thread 0
Number of thread 8
Number of processsors = 64
Max threads =8
Dynamic threads =0
Whether activate parallel region=1
Nested parallelism =0
Hello World from OMP thread 2
```

实验分析：

（一）、N 维矩阵相乘

需要将计算 N 维数组相乘的 C 语言程序 multi_mat.c 编译成可执行程序 multi_mat。

编译 C 语言程序的命令为：gcc -o multi_mat.c multi_mat。

在 multi_mat.c 文件中使用 C 语言中的 clock 函数计算矩阵相乘的时间，clock_t 是一个长整形数。在 time.h 文件中，还定义了一个常量 CLOCKS_PER_SEC，它用来表示一秒钟会有多少个时钟计时单元。

最终的图表结果表明，随着数组维度 N 的增大，时间 time 也逐渐增大，两者的拟合曲线表明，随着矩阵维度的增大，计算时间近 $n^{\wedge 3}$ 增长，与矩阵相乘的时间复杂度有关。经过查看资料发现 N 维矩阵相乘的时间复杂度为 O（$n^{\wedge 3}$）。使用串行程序编写的 N 维矩阵相乘的计算时间比较长，尤其是当 N>1000 维之后，运行时间近指数增长。在有限的时间内很难完成高维数组的计算。

（二）、C 语言条件编译

使用 #ifdef #else # endif 的格式进行不同语言版本的 hello world 的编译。#ifdef 如果宏已经定义，则编译下面代码。具体执行逻辑如下：

```
#ifdef cn   //先测试 cn 是否被宏定义过
程序段 1    //如果 cn 被宏定义过，那么就编译程序段 1
# else
程序段 2 //如果 cn 没有被定义过则编译程序段 2 的语句，“忽视”程序段 1。
#endif //终止 ifdef 指令
```

条件编译指令可以使编译器按不同的条件编译不同的程序部分，因而产生不同的目标代码文件。这对于程序的移植和调试是很有用的，尤其是针对于跨平台程序移植的时候。以及针对不同语言的程序进行编译的时候。

在进行条件编译时，有两种方式可以实现，以本程序为例，第一种是在头文件中，直接加上#define cn 然后正常编译即可输出中文版的“你好，世界”。第二种方法是在 gcc 编译时，使用-D 的参数，以本程序为例，gcc -D cn 即可。

（三）、OpenMP hello world 以及条件编译

条件编译代码部分同上述的 C 语言条件编译，但使用 OpenMP 编写并行程序时，需要在头文件中加入#include<omp.h>，还需要使用 #pragma omp parallel{}标记程序的并行段，表示程序从花括号左边开始并行，在附录 3 中我们在并行段打印了程序调用线程的标识符，右花括号表示程序并行结束。然后并行程序 join 到一起，执行后续部分。

从上面程序 3 的运行结果，可以看出，每个线程执行的速度不一样，有的线程执行 printf(“hello”)和另一些线程执行 printf(“world”)交织在一起，也就是在线程并行过程中，每个线程并行的速度不一样。

（四）、OpenMP 环境变量

在 OpenMP 程序使用# pragma omp parallel 开始并行时，主线程也就是标识符为 0 的主线程在所有使用 OpenMP 进行并行编程的程序中都有，可能标识符为 1、2、3 以及其他数字的线程在不同的程序中不一定会产生。所以我们可以设置一个条件，只让主线程完成某件事。

在程序 3 的基础之上，我们打印了其他的 OpenMP 的环境变量，omp_get_num_procs()表示有多少个处理器，omp_get_num_threads()表示获取线程的数目，omp_get_max_threads()表示获取最大线程数，omp_get_dynamic()返回当前程序是否允许在运行时动态调整并行区域的线程数。omp_in_parallel()可以判断当前是否处于并行状态，返回值为 1 表示是。omp_get_nested()返回是否启用嵌套并行，0 表示否。

因为我们使用的是 Docker 容器技术虚拟出来的机器，无法查看机器的真实内核，但根据以上运行的结果，可以初步得到机器有 8 个线程，64 个处理器，最大线程数也为 8 个，当前程序允许在运行时动态调整并行区域的线程数为 0，程序是处于并行状态，并未启用嵌套并行。

附录：

1．计算 N 维数组相乘的 C 语言程序代码 multi_mat.c

```
/*Student ID:2020317110033
date :2020-09-28
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<math.h>
/*define function to prodcue a empty N demension matrix
  for storing result of matrix multiplication
  */
int **produce_empty_mat(int N){
	int **empty_mat;
	empty_mat=(int**)malloc(N*sizeof(int*));
	int k;
	for(k=0;k<N;k++){
		empty_mat[k]=(int*)malloc(sizeof(int)*N);
	}
	return empty_mat;
}
/*define function to produce a random N demension matrix
for matrix multiplication
*/
int** produce_rand_mat(int N ){
	int **rand_mat;
	rand_mat=(int**)malloc(N*sizeof(int*));
	int k;
	for(k=0;k<N;k++){
		rand_mat[k]=(int*)malloc(sizeof(int)*N);
	}
	srand(time(NULL));
	int i,j;
	for(i=0;i<N;i++){
		for(j=0;j<N;j++){
			rand_mat[i][j]= rand()%10;
		}
	}
	return rand_mat;
}

/* define function to free memory
```

```
*/
void free_mat(int*** arr, int N){
	int i;
	for (i = 0; i < N; i ++){
			free ((*arr)[i]);}
	free (*arr);
}

int main(int argc,char *argv[]){
	int N;
	printf("Please input size of matrix N:");
	scanf("%d", &N);
	int **A=produce_rand_mat(N);
	int **B=produce_rand_mat(N);
	int **C=produce_empty_mat(N);
	int i,j,k,sum;
	clock_t start, end;
	start = clock();
	for(i=0;i<N;i++){
		for(j=0;j<N;j++){
			sum=0;
			for(k=0;k<N;k++){
				sum=sum+A[i][k]*B[k][j];
			}
			C[i][j]=sum;
		}
	}
	end = clock();
	printf("the matrix_multiply program running for %.3fs\n", (double)(end-
start)/CLOCKS_PER_SEC);
	printf("finish");
	free_mat(&A,N);
	free_mat(&B,N);
	free_mat(&C,N);
	return 0;
}
```

2. C 语言条件编译代码 hello_con.c

```
#include<stdio.h>
int main()
{
	#ifdef cn
	printf("你好，世界");
	#else
```

```
	printf("Hello World");
	#endif
	return 0;
}
```

3. OpenMP hello world 以及条件编译代码 openmp_hello_con.c

```
#include <omp.h>
#include <stdio.h>
int main(){
# pragma omp parallel
{
		# ifdef cn
		int ID=omp_get_thread_num();
		printf("你好(%d)",ID);
		printf("世界(%d)\n",ID);
		# else
		int ID=omp_get_thread_num();
		printf("hello(%d)",ID);
		printf("world(%d)\n",ID);
		#endif
}
		return 0;
}
```

4. OpenMP 环境变量代码 openmp_hello_env.c

```
/*
Student ID 2020317110033
Date 2020/09/28
*/
#include <omp.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
		#pragma omp parallel
		{
			int nthreads,tid;
			int nproc,maxtd,dynamic,inparll,nested;
			/* get thread id */
			tid=omp_get_thread_num();
			printf("Hello World from OMP thread %d \n",tid);
			/* Only master do this */
			if(tid==0){
```

```
                printf("The environment infomation about thread %d\n",tid);
                /* to get environment information */
                nthreads=omp_get_num_threads();
                nproc=omp_get_num_procs();
                maxtd=omp_get_max_threads();
                dynamic=omp_get_dynamic();
                inparll=omp_in_parallel();
                nested=omp_get_nested();
                /* print environment information*/
                printf("Number of thread %d\n",nthreads);
                printf("Number of processsors = %d\n",nproc);
                printf("Max threads =%d\n",maxtd);
                printf("Dynamic threads =%d\n ",dynamic);
                printf("Whether activate parallel region=%d\n",inparll);
                printf("Nested parallelism =%d\n",nested);
                }
        }
        return 0;
}
```