Problem descriptions

在這次作業中，請使用 OpenMP 撰寫一個程式，該程式需要宣告 4 個 Threads，並使用 for loop 執行 1000000 次以計算 pi。這次有 4 個需要實作的 programs，每個必須按照以下要求完成：

1. 使用 parallel 指令和 critical section 指令
2. 使用 parallel for 指令和 critical 指令
3. 移除第二題的 critical 指令，並觀察結果是否有錯誤
4. 使用第三題的程式碼，並結合 reduction 子句來計算出正確的 pi

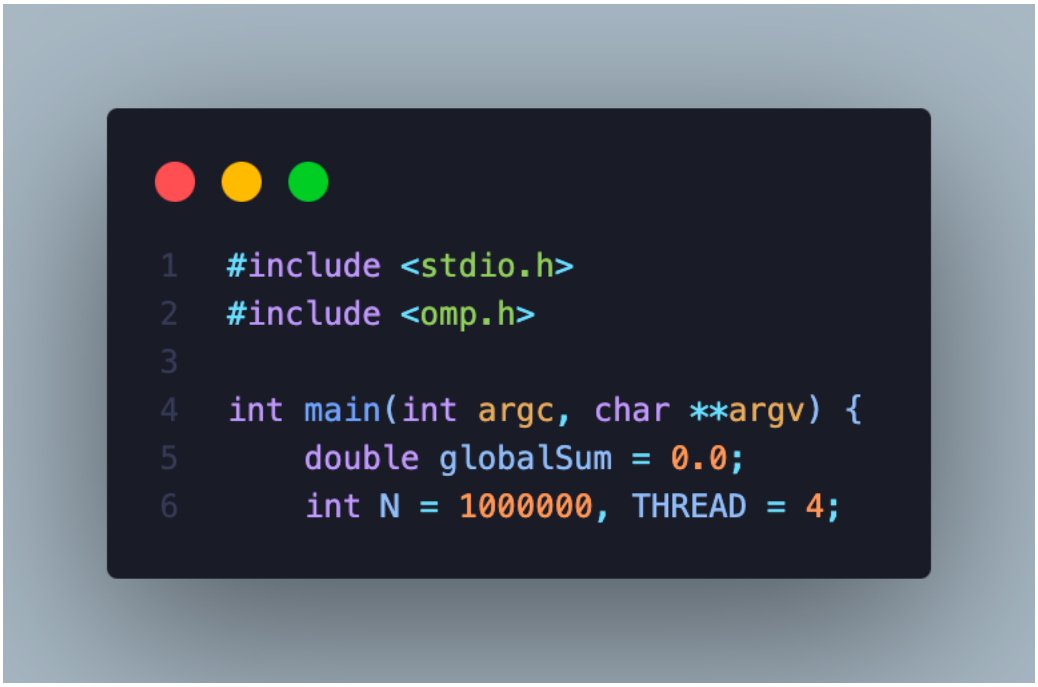
Code and explanations

PPD_PA2_B0829002_1.c

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    double globalSum = 0.0;
    int N = 1000000, THREAD = 4;
    omp_set_num_threads(THREAD);
    double start = omp_get_wtime();
    #pragma omp parallel num_threads(THREAD)
    {
        double localSum = 0.0;
        int localHead = (N / THREAD) * omp_get_thread_num();
        int localTail = (N / THREAD) * (omp_get_thread_num() + 1);
        double factor = 1.0;
        for (int i = localHead; i < localTail; i++) {
            localSum += factor / (2 * i + 1);
            factor = -factor;
        }
        localSum *= 4.0;
        #pragma omp critical
        {
            globalSum += localSum;
        }
        printf("Processor %d out of %d sum: %f\n", omp_get_thread_num(), THREAD, localSum);
    }
    double end = omp_get_wtime() - start;
    printf("Final estimated results with n = %d : %f\n", N, globalSum);
    printf("Time: %f\n", end);
    printf("Author: B0829002 廖洺玄(Ming-Hsuan Liao)\n");
    return 0;
}
```

Import library 和變數宣告

這部分用於印入 C 的 Standard library 和本次作業用到最重要的 OpenMP library，然後 Main function 會把使用者執行程式後面連帶的 arguments 引入讓 MPI 在 initialize parallel program 的時候可以使用到。globalSum 是用來儲存全域的 summary、N 是 sigma 的次數、THREAD 是 threads 數目



```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    double globalSum = 0.0;
    int N = 1000000, THREAD = 4;
```

設定和初始化 Parallel Program

設定 thread 數目後計算開始平行的時間，然後進入平行的部分就會先計算每個 thread 的頭尾，然後每個 thread 用算出來的頭尾跑 for loop 計算老師給的公式，之後就用 critical section 加入 globalSum 避免有 mutual excecution，之後印出每個 thread 的結果。

```c
        double factor = 1.0;
        double sum = 0.0;
        for (k = 0; k < n; k++) {
            sum += factor/(2*k+1);
            factor = -factor;
        }
        pi_approx = 4.0*sum;
```

```c
omp_set_num_threads(THREAD);
double start = omp_get_wtime();
#pragma omp parallel num_threads(THREAD)
{
    double localSum = 0.0;
    int localHead = (N / THREAD) * omp_get_thread_num();
    int localTail = (N / THREAD) * (omp_get_thread_num() + 1);
    double factor = 1.0;
    for (int i = localHead; i < localTail; i++) {
        localSum += factor / (2 * i + 1);
        factor = -factor;
    }
    localSum *= 4.0;
    #pragma omp critical
    {
        globalSum += localSum;
    }
    printf("Processor %d out of %d sum: %f\n", omp_get_thread_num(), THREAD, localSum);
}
```

結束時間以及印出結果

計算 parallel 的結束時間以及印出作業 requirement 的結果

```c
double end = omp_get_wtime() - start;
printf("Final estimated results with n = %d : %f\n", N, globalSum);
printf("Time: %f\n", end);
printf("Author: B0829002 廖洺玄(Ming-Hsuan Liao)\n");
return 0;
```

PPD_PA2_B0829002_2.c

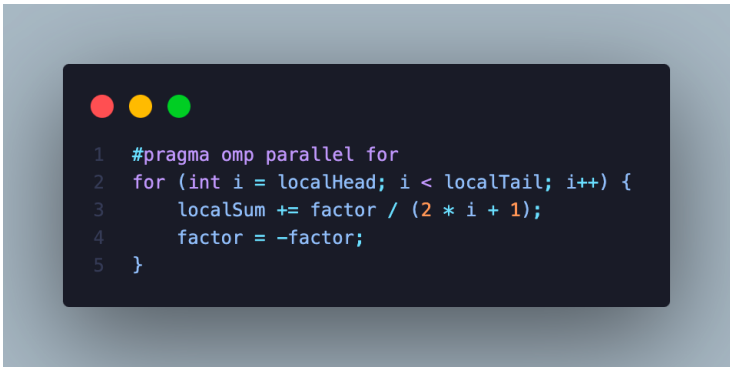
```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    double globalSum = 0.0;
    int N = 1000000, THREAD = 4;
    omp_set_num_threads(THREAD);
    double start = omp_get_wtime();
    #pragma omp parallel num_threads(THREAD)
    {
        double localSum = 0.0, factor = 1.0;
        int localHead = (N / THREAD) * omp_get_thread_num();
        int localTail = (N / THREAD) * (omp_get_thread_num() + 1);
        #pragma omp parallel for
        for (int i = localHead; i < localTail; i++) {
            localSum += factor / (2 * i + 1);
            factor = -factor;
        }
        localSum *= 4.0;
        #pragma omp critical
        {
            globalSum += localSum;
        }
        printf("Processor %d out of %d sum: %f\n", omp_get_thread_num(), THREAD, localSum);
    }
    double end = omp_get_wtime() - start;
    printf("Final estimated results with n = %d : %f\n", N, globalSum);
    printf("Time: %f\n", end);
    printf("Author: B0829002 廖洺玄(Ming-Hsuan Liao)\n");
    return 0;
}
```

Import library 、變數宣告和初始化 Parallel Program

同 PPD_PA2_B0829002_1.c [按我到之前的解釋]

設定和初始化 Parallel Program

這邊和前面的方法不太一樣，因為要用到 pragma omp parallel for 去處理 parallel 的部分所以要改下面這一小部分



```
#pragma omp parallel for
for (int i = localHead; i < localTail; i++) {
    localSum += factor / (2 * i + 1);
    factor = -factor;
}
```

結束時間以及印出結果

計算 parallel 的結束時間以及印出作業 requirement 的結果
同 PPD_PA2_B0829002_1.c [按我到之前的解釋]

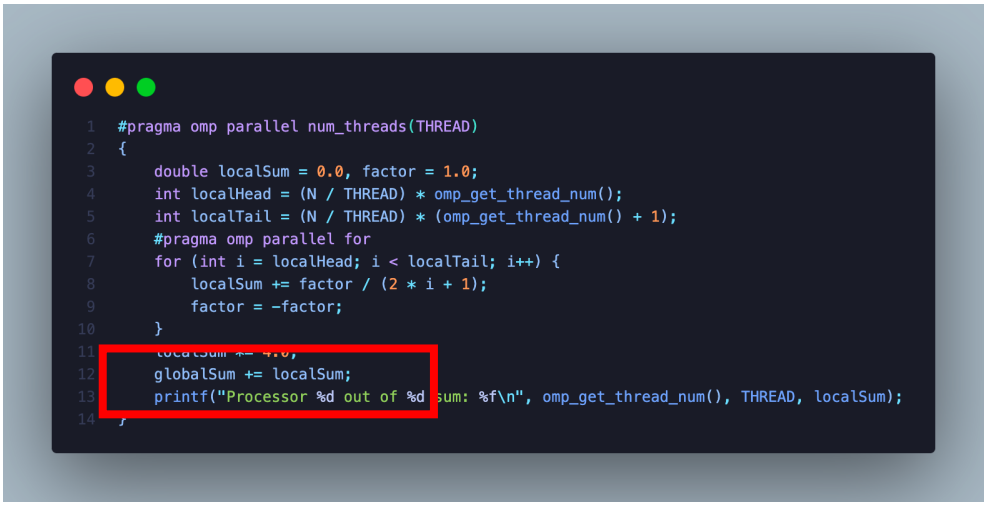
PPD_PA2_B0829002_3.c

Import library 、變數宣告和初始化 Parallel Program

同 PPD_PA1_B0829002_1.c [按我到之前的解釋]

Master Parallel Program

在這裡延續第二題的部分但要把 critical section 刪除來觀察是否會跑出錯誤結果



```
#pragma omp parallel num_threads(THREAD)
{
    double localSum = 0.0, factor = 1.0;
    int localHead = (N / THREAD) * omp_get_thread_num();
    int localTail = (N / THREAD) * (omp_get_thread_num() + 1);
    #pragma omp parallel for
    for (int i = localHead; i < localTail; i++) {
        localSum += factor / (2 * i + 1);
        factor = -factor;
    }
    globalSum += localSum;
    printf("Processor %d out of %d sum: %f\n", omp_get_thread_num(), THREAD, localSum);
}
```

結束時間以及印出結果

計算 parallel 的結束時間以及印出作業 requirement 的結果
同 PPD_PA2_B0829002_1.c [按我到之前的解釋]

PPD_PA4_B0829002_3.c

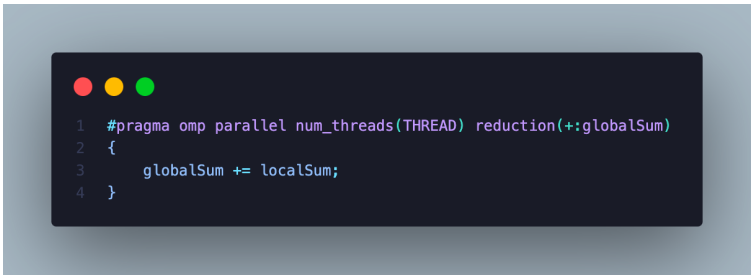
```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    double globalSum = 0.0;
    int N = 1000000, THREAD = 4;
    omp_set_num_threads(THREAD);
    double start = omp_get_wtime();
    #pragma omp parallel num_threads(THREAD)
    {
        double localSum = 0.0, factor = 1.0;
        int localHead = (N / THREAD) * omp_get_thread_num();
        int localTail = (N / THREAD) * (omp_get_thread_num() + 1);
        #pragma omp parallel for
        for (int i = localHead; i < localTail; i++) {
            localSum += factor / (2 * i + 1);
            factor = -factor;
        }
        localSum *= 4.0;

        #pragma omp parallel num_threads(THREAD) reduction(+:globalSum)
        {
            globalSum += localSum;
        }

        printf("Processor %d out of %d sum: %f\n", omp_get_thread_num(), THREAD, localSum);
    }
    double end = omp_get_wtime() - start;
    printf("Final estimated results with n = %d : %f\n", N, globalSum);
    printf("Time: %f\n", end);
    printf("Author: B0829002 廖洺玄(Ming-Hsuan Liao)\n");
    return 0;
}
```

Import library 、變數宣告和初始化 Parallel Program

同 PPD_PA1_B0829002_1.c [按我到之前的解釋]

Master Parallel Program

這裡大致上和前三題一樣但要 base on No.3 加上用 reduction 把 globalSum 加起來。



```c
#pragma omp parallel num_threads(THREAD) reduction(+:globalSum)
{
    globalSum += localSum;
}
```

結束時間以及印出結果

計算 parallel 的結束時間以及印出作業 requirement 的結果
同 PPD_PA2_B0829002_1.c [按我到之前的解釋]

Sampled outputs

PPD_PA2_B0829002_1.c

```
root@ccllab-cmp2:/home/ccllab/Parallel-programe-design/hw02# time ./a.out
Processor 0 out of 4 sum: 3.141589
Processor 3 out of 4 sum: 0.000000
Processor 2 out of 4 sum: 0.000001
Processor 1 out of 4 sum: 0.000002
Final estimated results with n = 1000000 : 3.141592
Time: 0.002581
Author: B0829002 廖洺玄(Ming-Hsuan Liao)

real    0m0.009s
user    0m0.012s
sys     0m0.004s
```

PPD_PA2_B0829002_2.c

```
root@ccllab-cmp2:/home/ccllab/Parallel-programe-design/hw02# make^C
root@ccllab-cmp2:/home/ccllab/Parallel-programe-design/hw02# time ./b.out
Final estimated results with n = 1000000 : 3.141592
Time: 0.002857
Author: B0829002 廖洺玄(Ming-Hsuan Liao)

real    0m0.010s
user    0m0.012s
sys     0m0.005s
```

PPD_PA2_B0829002_3.c

```
Author: B0829002 廖洺玄(Ming-Hsuan Liao)
Final estimated results with n = 1000000 : 3.141592
Time: 0.001376
Author: B0829002 廖洺玄(Ming-Hsuan Liao)
Final estimated results with n = 1000000 : 3.141592
Time: 0.001397
Author: B0829002 廖洺玄(Ming-Hsuan Liao)
Final estimated results with n = 1000000 : 3.141591
Time: 0.001411
Author: B0829002 廖洺玄(Ming-Hsuan Liao)
Final estimated results with n = 1000000 : 3.141592
Time: 0.001422
Author: B0829002 廖洺玄(Ming-Hsuan Liao)
Final estimated results with n = 1000000 : 3.141592
Time: 0.001565
```

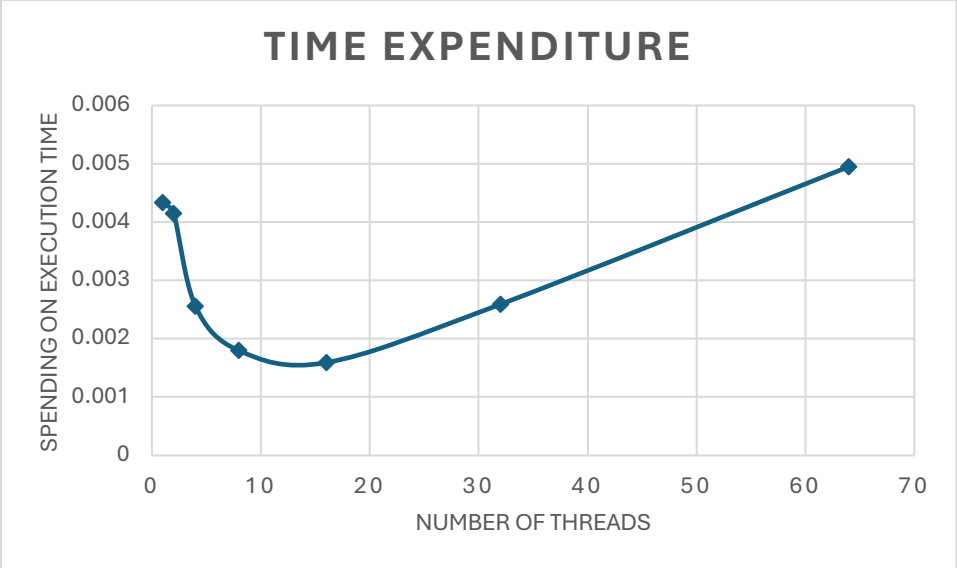
PPD_PA2_B0829002_4.c

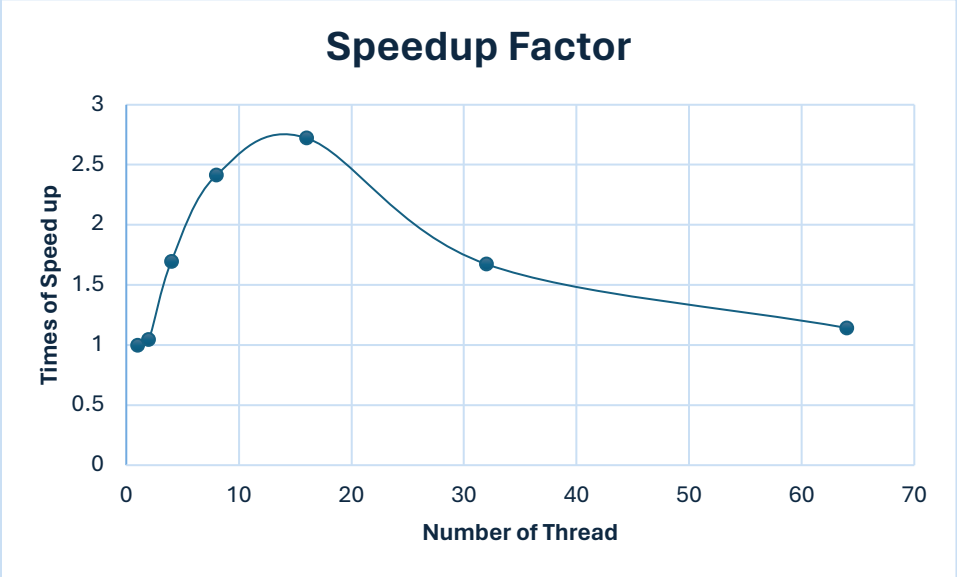
```
sys     0m0.003s
root@ccllab-cmp2:/home/ccllab/Parallel-programe-design/hw02# time ./d.out
Final estimated results with n = 1000000 : 3.141592
Time: 0.002884
Author: B0829002 廖洺玄(Ming-Hsuan Liao)

real    0m0.010s
user    0m0.009s
sys     0m0.002s
```

Bonus

1. Recording the execution time of each program
   從執行結果可以看到每個程式的執行時間（including Linux command: time and OpenMP time API: omp_get_wtime()）
2. Recording the speedup in problem 4 over the serial version by varying the number of threads = 1, 2, 4, 8,and 16

Number of threads	Time expenditure	Speedup Factor
1	0.004335	1
2	0.004146	1.045586107
4	0.002556	1.69600939
8	0.001797	2.412353923
16	0.001592	2.72298995
32	0.002591	1.67309919
64	0.004951	1.142099193





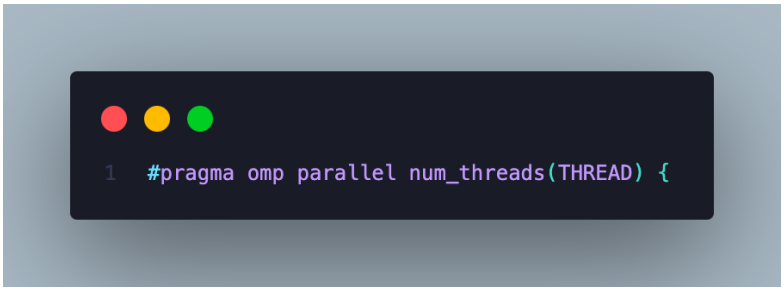
3. Recording the approximated it global sum of problem 4 by varying n = 1000, 10000, 100000, and 1000000

可以看出來 N 越大 PI 值越準

N	Estimation
1000	3.140593
10000	3.141293
100000	3.141583
100000	3.141592
1000000	3.141593

Discussions or what I’ve learned

在這個 Assignment 裡面我學習到了如何利用 OpenMP 提供的 API 開 muti-thread，讓我的 Program 可以平行運算。而且我一開始原本想用自己的 coding style 寫



但發現 compiler 在編譯的時候會直接整段報錯，所以後來才改為 omp.h 定義好到 coding style。除此之外，也深刻體會到使用 threads 進行並行處理所帶來的計算速度提升。與 process 並行相比，這四個 programs 在實作上並沒有太大的困難，只要好好聽課和看講義就會了。

Reference

[1] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," in IEEE Computational Science and Engineering, vol. 5, no. 1, pp. 46-55, Jan.-March 1998, doi: 10.1109/99.660313.keywords: {Message passing;Scalability;Hardware;Computer architecture;Power system modeling;ANSI standards;Parallel processing;Coherence;Software systems;Parallel programming},