ضرب المصفوفات باستخدام النياسبPthread

الجلسة الخامسة

Hasan Hasan

أولا: الضرب على التسلسل

```
#define DIM 1000

long matrix_a[DIM][DIM];
long matrix_b[DIM][DIM];
long matrix_c[DIM][DIM];

void init() {
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            matrix_a[i][j] = i+j;
            matrix_b[i][j] = i-j;
            matrix_c[i][j] = 0;
        }
    }
}

void multiply() {
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            for(int k = 0; k < DIM; k++) {
                matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
            }
        }
    }
}

void print() {
    FILE *fp = fopen("serial.txt","w");
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            fprintf(fp, "%ld\n", matrix_c[i][j]);
        }
    }
    fclose(fp);
}

int main(void) {
    init();
    std::cout << "here it got initiated" << std::endl;
    multiply();
    std::cout << "here it got multiplied" << std::endl;
    print();
    std::cout << "here it got printed" << std::endl;
    return 0;
}
```

بعد نسخ الكود التالي وإضافة مكتبة iostreamوبعض رسائل بعد تنفيذ كل تابع.

باستخدام تابع التهيئة وتابع الضرب وتابع الطباعة (على ملف).

واستدعائهم في التابع الأساسي(main).

حصلنا على الخرج التالي عند تنفيذ (time ./serial) وهو زمن التنفيذ:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                bash - hasan2324

hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o serial
hasan2324@LOQ-HASAN:~$ ./serial
here it got initiated
here it got multiplied
here it got printed
hasan2324@LOQ-HASAN:~$ time ./serial
here it got initiated
here it got multiplied
here it got printed

real    0m4.070s
user    0m3.971s
sys     0m0.020s
hasan2324@LOQ-HASAN:~$
```

ثانيا: الجزء ذو الاستهلاك الزمني الأطول

ممكن التنبؤ بنسبه كبيرة ان تابع الضرب هو التابع الذي يحتاج وقت أكثر من الطباعة والتهيئة كون هناك 3 حلقات تكرارية مقارنة ب التوابع الأخرى يوجد فقط حلقتان.

وهو ما يعطي تعقيد جداء $o(n^3)$ الذي هو نفسه تعقيد البرنامج الكلي.

```cpp
int main(void) {
    auto start_init = std::chrono::high_resolution_clock::now();
    init();
    auto end_init = std::chrono::high_resolution_clock::now();
    auto duration_init = std::chrono::duration_cast<std::chrono::microseconds>(end_init - start_init);
    std::cout << "Init time: " << duration_init.count() << " us" << std::endl;
    std::cout << "here it got initiated" << std::endl;

    auto start_multiply = std::chrono::high_resolution_clock::now();
    multiply();
    auto end_multiply = std::chrono::high_resolution_clock::now();
    auto duration_multiply = std::chrono::duration_cast<std::chrono::microseconds>(end_multiply - start_multiply);
    std::cout << "Multiply time: " << duration_multiply.count() << " us" << std::endl;
    std::cout << "here it got multiplied" << std::endl;

    auto start_print = std::chrono::high_resolution_clock::now();

    print();
    auto end_print = std::chrono::high_resolution_clock::now();
    auto duration_print = std::chrono::duration_cast<std::chrono::microseconds>(end_print - start_print);
    std::cout << "Print time: " << duration_print.count() << " us" << std::endl;
    std::cout << "here it got printed" << std::endl;
    return 0;
}
```

بالاستعانة ب agentتم التعديل

اضفنا مكتبة <chrono>

ف هنا فقط عرفنا متحول يقيس الزمن قبل واخر بعد الانتهاء من كل تابع وبطرحهما نحصل على الزمن المستهلك لكل تابع على حدا.

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o serial
hasan2324@LOQ-HASAN:~$ ./serial
Init time: 10287 us
here it got initiated
Multiply time: 3874023 us
here it got multiplied
Print time: 51889 us
here it got printed
```

وبهذا نتأكد من أن زمن الضرب هو الأكبر بين التوابع الأخرى، بعد الحساب تقريبا 98.42% من الزمن الكلي يكون اثناء الضرب.

ثالثا: تنفيذ تابع الضرب عن طريق نيسب مختلف عن النيسب الأساسي

أضفنا مكتبة thread

والتعديل أثناء الاستدعاء:

```
auto start_multiply = std::chrono::high_resolution_clock::now();
std::thread multiply_thread(multiply);
multiply_thread.join();
auto end_multiply = std::chrono::high_resolution_clock::now();
auto duration_multiply = std::chrono::duration_cast<std::chrono::microseconds>(end_multiply - start_multiply);
std::cout << "Multiply time: " << duration_multiply.count() << " us" << std::endl;
std::cout << "here it got multiplied" << std::endl;
```

وكان الزمن

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o para -pthread
hasan2324@LOQ-HASAN:~$ time ./para
Init time: 14734 us
here it got initiated
Multiply time: 4290851 us
here it got multiplied
Print time: 55423 us
here it got printed

real    0m4.372s
user    0m4.342s
sys     0m0.024s
hasan2324@LOQ-HASAN:~$ diff Para1.txt serial.txt | wc -l
diff: Para1.txt: No such file or directory
0
hasan2324@LOQ-HASAN:~$
```

نلاحظ عدم حدوث فرق كثير في الزمن نتيجة تابع الضرب الذي ينفذ على التسلسل ضمن النيسب الجديد.

باستخدام Pthread.h

```
void multiply() {
void* multiply(void* arg) {
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            for(int k = 0; k < DIM; k++) {
                matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
            }
        }
    }
    return NULL;
}
```

وفي الاستدعاء

```
auto start_multiply = std::chrono::high_resolution_clock::now();
std::thread multiply_thread(multiply);
multiply_thread.join();
pthread_t multiply_thread;
pthread_create(&multiply_thread, NULL, multiply, NULL);
pthread_join(multiply_thread, NULL);
auto end_multiply = std::chrono::high_resolution_clock::now();
auto duration_multiply = std::chrono::duration_cast<std::chrono::microseconds>(end_multiply - start_multiply);
std::cout << "Multiply time: " << duration_multiply.count() << " us" << std::endl;
std::cout << "here it got multiplied" << std::endl;
```

Keep Undo

كما أن النيسب الأساسي ينتظر النيسب الجديد لينتهي بسبب pthread_join.

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o para -pthread
hasan2324@LOQ-HASAN:~$ time ./para
Init time: 10730 us
here it got initiated
Multiply time: 3972845 us
here it got multiplied
Print time: 55293 us
here it got printed

real    0m4.046s
user    0m3.974s
sys     0m0.012s
hasan2324@LOQ-HASAN:~$
```

وهنا يظهر الزمن باستخدام Pthread.h

قريب جدا نتيجة استخدام thread واحد فقط ف سينفذه حتى ينتهي ثم سيكمل أي تسلسل واضح لن يكون هنا تغيير كبير.

رابعا: تسريع البرنامج عن طريق توزيع العمل على عدة نياسب واختبار صحة العمل المنفذ

تم التعديل على الكود السابق:

نعرف عدد الthreads التي سنستخدمها و ال thread_arg

جربت العدد 4 أولا:

نعدل في تابع الضرب حيث عرفنا ال range ك argument للthread (هنا نكتشف أهمية عملية تعريب المصطلحات الأجنبية)

وجعلنا ضرب المصفوفة يمتد على عدد ال threads من بدايته (start) حتى النهاية (end)

```c
#define DIM 1000

#define NUM_THREADS 4

long matrix_a[DIM][DIM];
long matrix_b[DIM][DIM];
long matrix_c[DIM][DIM];

struct thread_args{
    int start;
    int end;
};

void init() {
```

```c
void* multiply(void* arg) {
    struct thread_args * range = (struct thread_args *) arg;
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            for(int k = 0; k < DIM; k++) {
            for(int k = range->start; k < range->end; k++) {
                matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
            }
        }
    }
    return NULL;
}
```

أما داخل ال main:

عرفنا بدايات مناطق(range) كل thread وتأكدنا من تلاصق نهاية كل واحد مع ببداية يلي بعدو

ثم انشأنا كل threadليقوم بجزء من عملية الضرب مباشرة و pthread_join ضمن انتظار ال thread الأساسي ل كل threadحتى ينتهي

هذا كان التعديل

أما النتيجة

```
auto start_multiply = std::chrono::high_resolution_clock::now();
pthread_t multiply_thread;
pthread_create(&multiply_thread, NULL, multiply, NULL);
pthread_join(multiply_thread, NULL);
pthread_t threads[NUM_THREADS];
struct thread_args work_ranges[NUM_THREADS];
int current_start, range;
current_start = 0;
range = DIM / NUM_THREADS;
for(int i = 0; i < NUM_THREADS; i++) {
    work_ranges[i].start = current_start;
    work_ranges[i].end = current_start + range;
    current_start += range;
}
work_ranges[NUM_THREADS-1].end = DIM;
for(int t = 0; t < NUM_THREADS; t++) {
    pthread_create(&threads[t], NULL, multiply, &work_ranges[t]);
}
for(int t = 0; t < NUM_THREADS; t++) {
    pthread_join(threads[t], NULL);
}
auto end_multiply = std::chrono::high_resolution_clock::now();
```

```
hasan2324@LOQ-HASAN:~$ time ./para4
Init time: 12994 us
here it got initiated
Multiply time: 442359 us
here it got multiplied
Print time: 64948 us
here it got printed

real    0m0.531s
user    0m3.964s
sys     0m0.027s
hasan2324@LOQ-HASAN:~$
```

نلاحظ أن الزمن قل بشكل كبير تقريبا من ناحية استهلاك تابع الضرب للزمن

```
hasan2324@LOQ-HASAN:~$ time ./para4
Init time: 12994 us
here it got initiated
Multiply time: 442359 us
here it got multiplied
Print time: 64948 us
here it got printed

real    0m0.531s
user    0m3.964s
sys     0m0.027s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
2830
hasan2324@LOQ-HASAN:~$
```

تقريبا صار أسرع ب 7.5 مرات، لكن على حساب الدقة وصحة عمليات الضرب والاسناد.

حيث أصبح لدينا حالة سباق بين ال threads على المتحول c[i][j] (عنصر من المصفوفة التي ستشمل الناتج) الذي سنسند اليه مجموع قيمة الجداءات كونه critical section

خامسا: تحقيق تزامن النياسب على الجزء المشترك بينهم واختبار صحة العمل المنفذ

عرفنا lock ك متغير عام من نوع pthread_mutex_t

أما في تابع الضرب

```
31  void *multiply(void *arg) {
32      struct thread_args * range = (struct thread_args *) arg;
33      for(int i = 0; i < DIM; i++) {
34          for(int j = 0; j < DIM; j++) {
35              for(int k = range->start; k < range->end; k++) {
36                  pthread_mutex_lock(&lock);
37                  matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
38                  pthread_mutex_unlock(&lock);
39              }
40          }
41      }
42      return NULL;
43  }
```

أصبحت عملية الجمع تتم مع قفل كي لا يستطيع أكثر من thread الوصول الى matrix_c[i][j] بنفس اللحظة.

```
hasan2324@LOQ-HASAN:~$ g++ paralelll.cpp -o para -pthread
hasan2324@LOQ-HASAN:~$ time ./para

real    1m3.945s
user    1m17.600s
sys     8m4.885s
hasan2324@LOQ-HASAN:~$ g++ true_serial.cpp -o serial
hasan2324@LOQ-HASAN:~$ time ./serial
Init time: 12638 us
Multiply time: 4170917 us
Print time: 57161 us

real    0m4.247s
user    0m4.225s
sys     0m0.021s
hasan2324@LOQ-HASAN:~$ diff para.txt serial.txt | wc -l
0
```

اول مرة هو زمن العملية بعد إضافة القفل وثاني مرة هي بالتسلسل (الطرقة الأولى)

النتيجة صحيحة كون الفرق صفر بين محتوى الملفين لكن نلاحظ أن الوقت مع قفل كان أكبر بكثير.

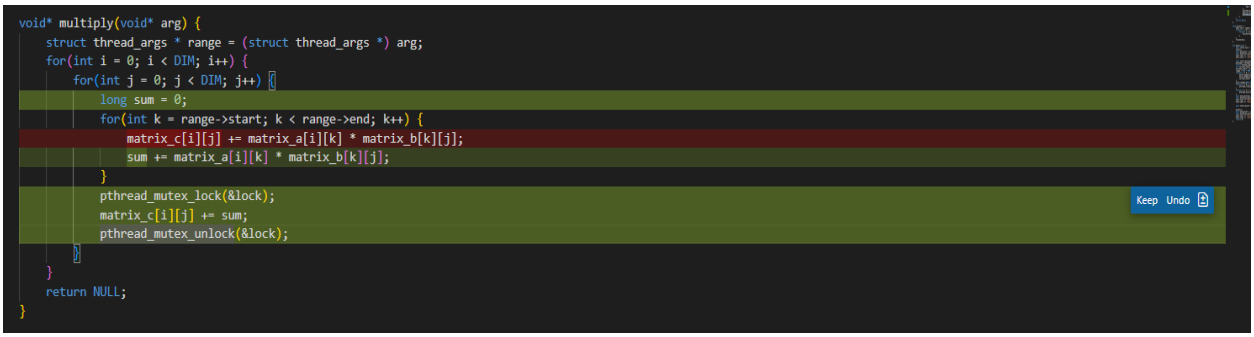
نحن لدينا عملية قفل وفك قفل مع كل thread تقريبا 1000*1000*1000 تقسيم (num of threads)

يعني بالمجمل 10^9 عملية قفل و فك قفل.

وهذا ما سبب التأخير وجعل المهمة تكون على شكل سلسلة واحدة أي عدنا على التسلسل لكن مع موقت القفل وفكه لذلك الزمن أصبح أكبر.

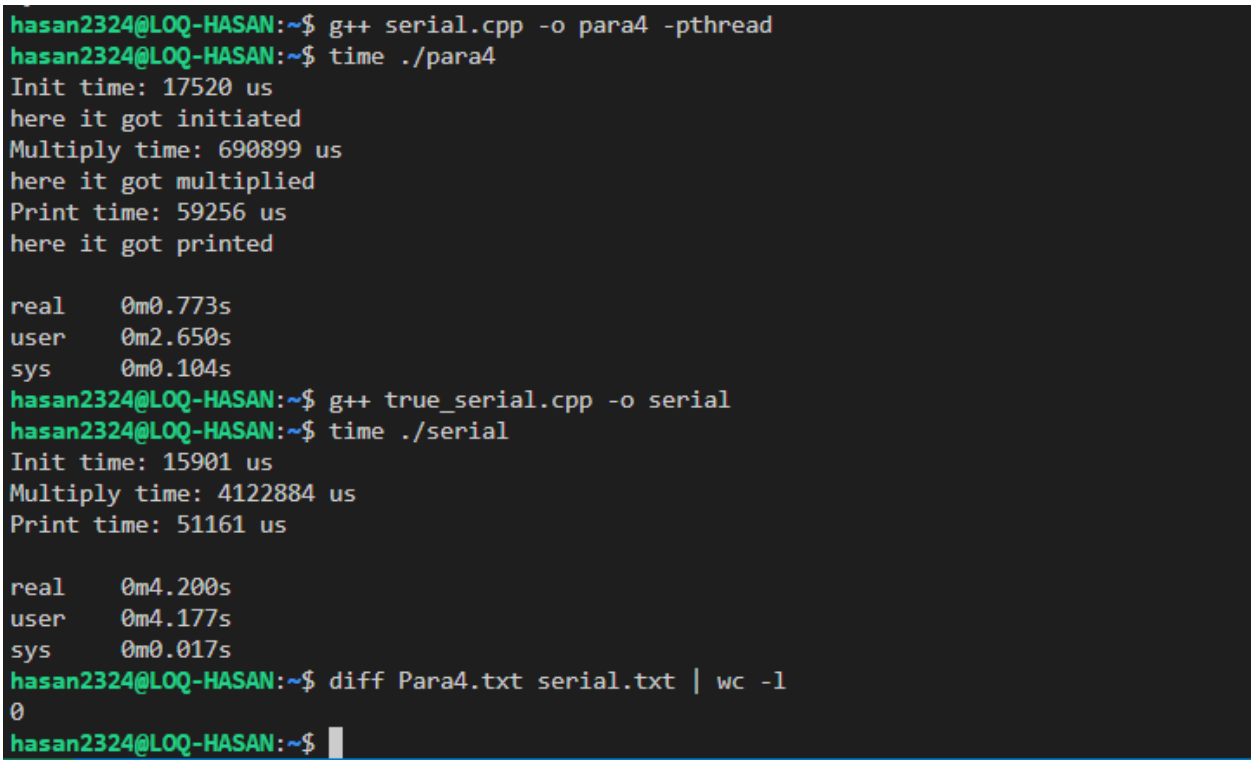
سادسا: تقليل عملية التنافس على القفل لتقتصر على أقل تنفيذ ممكن

في تابع الضرب



```
void* multiply(void* arg) {
    struct thread_args * range = (struct thread_args *) arg;
    for(int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            long sum = 0;
            for(int k = range->start; k < range->end; k++) {
                matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            pthread_mutex_lock(&lock);
            matrix_c[i][j] += sum;
            pthread_mutex_unlock(&lock);
        }
    }
    return NULL;
}
```

أصبحت عملية الجمع في متغير sumلكن الاسناد تتم مع قفل كي لا يستطيع أكثر من threadالوصول الى matrix_c[i][j] بنفس اللحظة.



```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o para4 -pthread
hasan2324@LOQ-HASAN:~$ time ./para4
Init time: 17520 us
here it got initiated
Multiply time: 690899 us
here it got multiplied
Print time: 59256 us
here it got printed

real    0m0.773s
user    0m2.650s
sys     0m0.104s
hasan2324@LOQ-HASAN:~$ g++ true_serial.cpp -o serial
hasan2324@LOQ-HASAN:~$ time ./serial
Init time: 15901 us
Multiply time: 4122884 us
Print time: 51161 us

real    0m4.200s
user    0m4.177s
sys     0m0.017s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
hasan2324@LOQ-HASAN:~$
```

اول مرة هو زمن العملية بعد إضافة القفل وثاني مرة هي بالتسلسل (الطريقة الأولى)

النتيجة صحيحة كون الفرق صفر بين محتوى الملفين لكن نلاحظ أن الوقت مع قفل بعد التحسين كان أقل بكثير.

نحن لدينا عملية قفل وفك قفل مع كل threadتقريبا 1000*1000

يعني تقريبا (num of threads)*10^6وهذا أفضل بكثير

في طريقة التحسين السابقة قسمنا

تقسيم حسب البعد k بحيث كل (Thread)يأخذ جزء من البعدk أعمدة المصفوفة A, صفوف المصفوفةB

لكن جميع threads تمر على كل الصفوف والأعمدة في المصفوفة الناتجةC

بما أن أكثر من thread يحاول تحديث نفس العنصرmatrix_c[i][j] لازم نستخدم قفل (Mutex) لتجنب حالة السباق.

حتى مع التحسين (قفل مرة واحدة لكل عنصر (i,j))، ما زال هناك تنافس بين thread على نفس الذاكرة وهذا يضيف تكلفة كبيرة ويبطئ التنفيذ.

فينا نحسن بأن نقسم حسب البعد i

كل thread يأخذ مجموعة صفوف منفصلة من المصفوفةC

thread يحسب هذه الصفوف بالكامل كل الأعمدة j وكل القيمk

لا يوجد threads يلمسون نفس العنصر في C ومنه لا حاجة للقفل

threads تعمل بشكل مستقل تماما وبالتالي الأداء يصبح أفضل.

```c
void* multiply(void* arg) {
    thread_args* range = (thread_args*) arg;
    for (int i = range->row_start; i < range->row_end; i++) {
        for (int j = 0; j < DIM; j++) {
            long sum = 0;
            for (int k = 0; k < DIM; k++) {
                sum += matrix_a[i][k] * matrix_b[k][j];
            }
            matrix_c[i][j] = sum; // no mutex: each thread owns its rows
        }
    }
    return NULL;
}
```

أما الخرج

```
hasan2324@LOQ-HASAN:~$ time ./para
Init time: 13045 us
Multiply time: 678979 us
Print time: 67661 us

real    0m0.766s
user    0m2.746s
sys     0m0.050s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
hasan2324@LOQ-HASAN:~$
```

قريب من السابق لكن ارتحنا من عمليات القفل وفك القفل كون لا يمكن أساسا ل threadsأن تشترك بنفس الجزء من الذاكرة ومنه لا يوجد قسم حرج ولا حاجة للقفل.

هنا سأكتب الجدول على الكود المحسن الذي يستخدم طريقة القفل وفك القفل

عدلت على الكود بحيث يطبع عدد ال threads مباشرة بعد جملة here it got printed

أبعاد المصفوفة	الزمن(us)	عدد النياسب	ملاحظة
1000	4248413	تسلسل	الأطول نظريا
1000	1156073	2	تسريع بنسبة 3.67
1000	610453	4	تسريع بنسبة 6.96
1000	832959	6	تسريع بنسبة 5.1
1000	1191787	8	تسريع بنسبة 3.56
1000	1752552	10	تسريع بنسبة 2.42
1000	2338500	16	تسريع بنسبة 1.82
1000	2804338	20	تسريع بنسبة 1.51
1000	3656695	30	تسريع بنسبة 1.16
1000	5075546	50	تسريع بنسبة 0.84(تبطيء)

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 7919 us
here it got initiated
Multiply time: 2338500 us
here it got multiplied
Print time: 53675 us
here it got printed16

real    0m2.406s
user    0m5.844s
sys     0m18.821s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
```

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 8064 us
here it got initiated
Multiply time: 5075546 us
here it got multiplied
Print time: 52043 us
here it got printed50

real    0m5.141s
user    0m8.752s
sys     0m52.038s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
hasan2324@LOQ-HASAN:~$
```

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 8345 us
here it got initiated
Multiply time: 1752552 us
here it got multiplied
Print time: 59955 us
here it got printed10

real    0m1.826s
user    0m5.205s
sys     0m8.829s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
```

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 7794 us
here it got initiated
Multiply time: 3656695 us
here it got multiplied
Print time: 53019 us
here it got printed30

real    0m3.723s
user    0m7.306s
sys     0m35.507s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
```

```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 8147 us
here it got initiated
Multiply time: 1191787 us
here it got multiplied
Print time: 53184 us
here it got printed8

real    0m1.258s
user    0m4.108s
sys     0m3.727s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
```

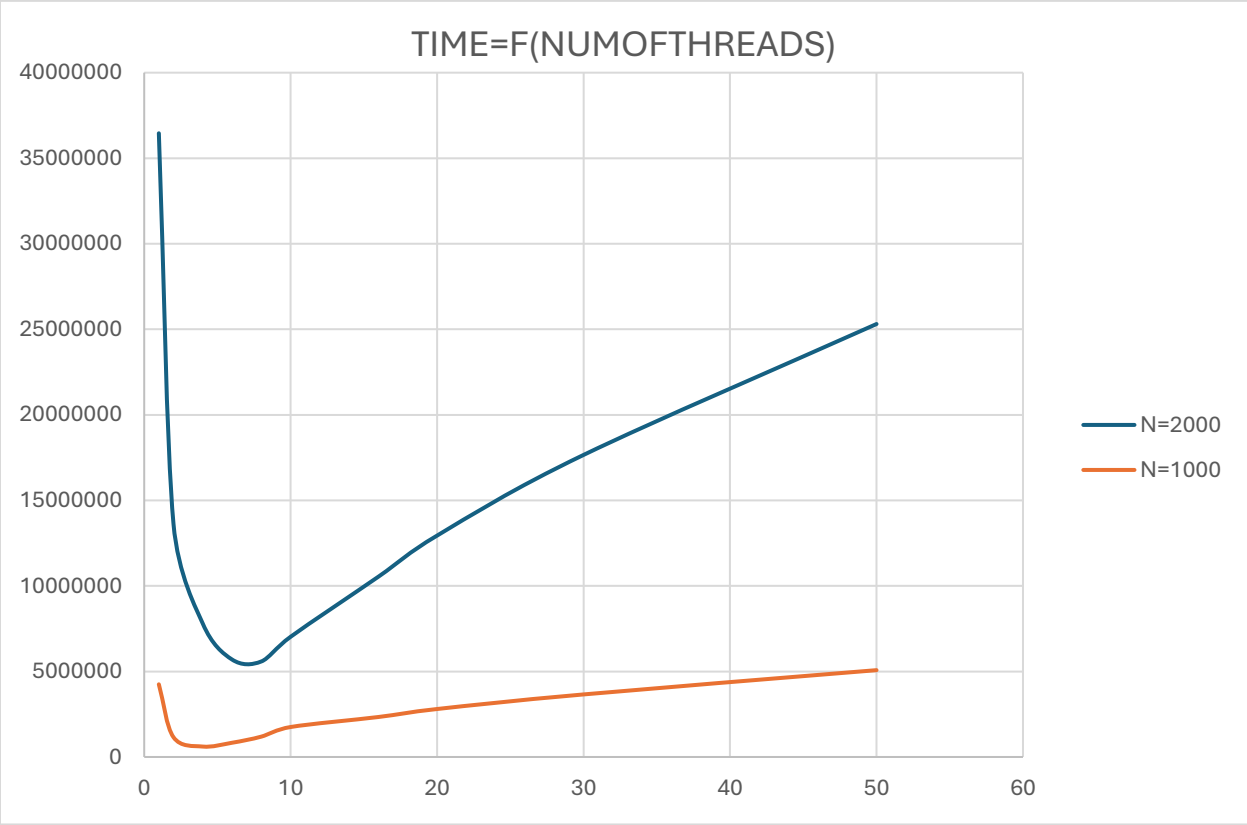
```
hasan2324@LOQ-HASAN:~$ g++ serial.cpp -o paran -pthread
hasan2324@LOQ-HASAN:~$ time ./paran
Init time: 7896 us
here it got initiated
Multiply time: 2804338 us
here it got multiplied
Print time: 59437 us
here it got printed20

real    0m2.877s
user    0m6.773s
sys     0m24.627s
hasan2324@LOQ-HASAN:~$ diff Para4.txt serial.txt | wc -l
0
```

هذه هي الأزمنة الظاهرة لكل حالة

أبعاد المصفوفة	الزمن(us)	عدد النياسب	ملاحظة
2000	36457860	تسلسل	الأطول نظريا
2000	13624004	2	تسريع بنسبة 2.68
2000	7793293	4	تسريع بنسبة 4.68
2000	5696024	6	تسريع بنسبة 6.4
2000	5584588	8	تسريع بنسبة 6.53
2000	7029381	10	تسريع بنسبة 5.19
2000	10545550	16	تسريع بنسبة 3.46
2000	12943136	20	تسريع بنسبة 2.82
2000	17658076	30	تسريع بنسبة 2.06
2000	25307964	50	تسريع بنسبة 1.44

الخرج بنفس الطريقة سيظهر على ال terminal



هنا بواسطة excel تم رسم الزمن بدلالة عدد النياسب في الحالتين.

بحالة 4 نياسب مع n=1000 كان أسرع عملية

بحالة 8 نياسب مع 2n= 000 كان أسرع عملية

التسلسل هو الأبطأ كما هو متوقع والأداء يتحسن تدريجيا حتى يصل إلى الذروة بعد ذلك التوازي الزائد يضر الأداء بدل أن يحسنه بسب كثرة تبديل السياق والقفل وفكه.