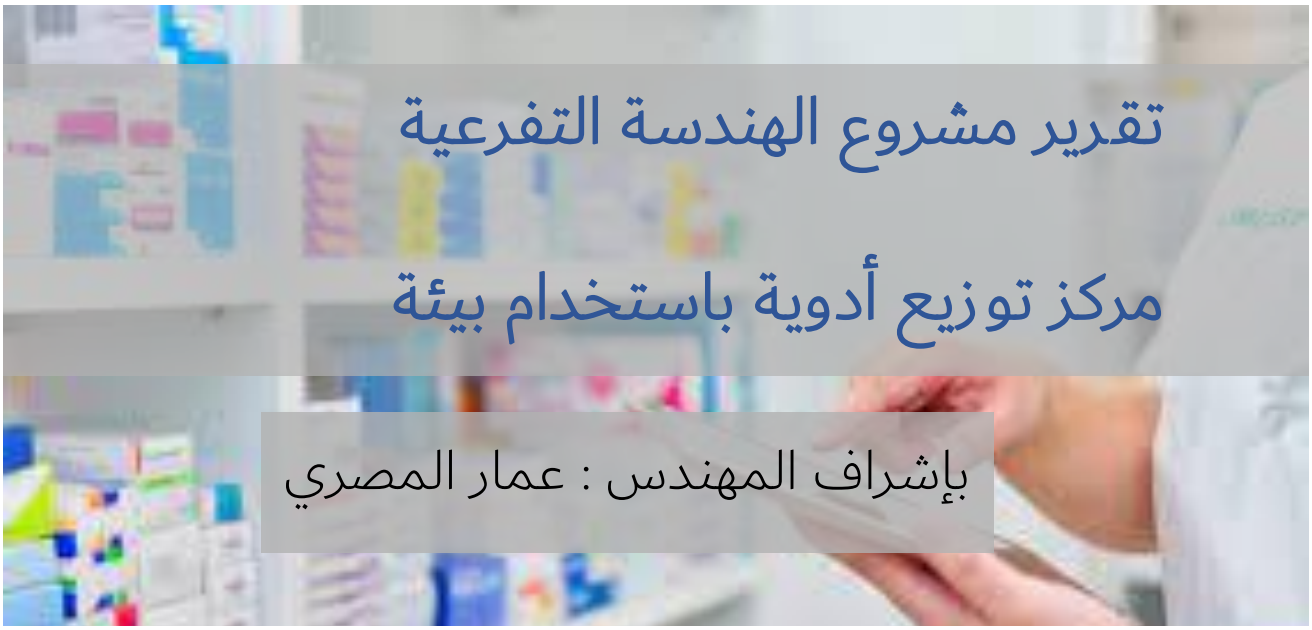
جامعة دمشق

كلية الهندسة المعلوماتية



تقرير مشروع الهندسة التفرعية

مركز توزيع أدوية باستخدام بيئة

بإشراف المهندس : عمار المصري

تقديم الطالبات :

سارة ايمن القاضي

رنيم نوار الأمين

مرام محمد هيثم حصري

نور معتز نابلسي

مرام نعيم الجردي

*ملحوظة : سيتم كتابة الكود بالخلفية المعتمة ليصبح اسهل للقراءة ..

يهدف البرنامج إلى حساب الزمن الكلي اللازم لإنجاز عملية التوزيع عند تنفيذها بشكل متوازي، من خلال توزيع المهام على عدة موزعين يعملون بشكل متزامن.

التقسيم المقترح :

1. ان يتم تقسيم النقاط (صيدليات + عيادات) على عدد من الموزعين ليعملوا بالتوازي.

2. ثم يتم احتساب الزمن اللازم لكل موزع لإنجاز مهمته بناءً على عدد النقاط الموكلة إليه والزمن الوسطي المطلوب لتوزيع الأدوية على كل نقطة.

3. ثم يُحسب الزمن الكلي للتوزيع التفرعي باعتباره أطول زمن بين جميع الموزعين (لأن التنفيذ يتم بالتوازي).

- ال input

1. عدد الصيدليات في المحافظة.
2. عدد العيادات في المحافظة.
3. عدد الموزعين في المركز.
4. الزمن الوسطي لتوزيع الأدوية على نقطة واحدة.

- ال output

الزمن الكلي للعملية التفرعية.

الحل في بيئة pvm

كود الأب (master):

1. يستقبل المدخلات من المستخدم.

```c
#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"

int main()
{
    int num_pharmacies, num_clinics, num_distributors, avg_time;
    int total_points, points_per_distributor;
    int *tids;
    int mytid, spawned, i, result;
    int task_input, max_time = 0;


    printf("Enter number of pharmacies: ");
    scanf("%d", &num_pharmacies);
    printf("Enter number of clinics: ");
    scanf("%d", &num_clinics);
    printf("Enter number of distributors: ");
    scanf("%d", &num_distributors);
    printf("Enter average time per point: ");
    scanf("%d", &avg_time);
```

2. يحسب العدد الكلي للنقاط (صيدليات + عيادات).

حساب إجمالي النقاطو توزيعها بالتساوي بين الموزعين (لكل واحد عدد معين من النقاط).

```c
total_points = num_pharmacies + num_clinics;
 points_per_distributor = total_points / num_distributors;
```

3. تخصيص مصفوفة لتخزين معرفات الموزعين.(PVM TIDs) والحصول على معرف المهمة الحالية (الماستر).

```c
tids = (int *)malloc(num_distributors * sizeof(int));
    mytid = pvm_mytid();
```

4. طباعة TID للماستر حيث يستخدم pvm_spawn لتشغيل عدد من عمليات الأبناء (workers).

```
 printf("Master started, my tid = %d\n", mytid);

    spawned = pvm_spawn("hello_other", NULL, PvmTaskDefault, "", num_distributors,
tids);
```

5. فحص ما إذا تم إنشاء المهام الفرعية بنجاح. لو فشل، يتم إنهاء البرنامج.

```
if (spawned < 1) {
        printf("Error spawning worker tasks\n");
        pvm_exit();
        return 1;
    }
```

6. يتم إرسال نفس البيانات لكل موزع وعدد النقاط التي عليه خدمتها و الزمن الوسطي .

```
for (i = 0; i < spawned; i++) {
        task_input = points_per_distributor;

        pvm_initsend(PvmDataDefault);
        pvm_pkint(&task_input, 1, 1);
        pvm_pkint(&avg_time, 1, 1);
        pvm_send(tids[i], 1);
    }
```

7. بعد أن ينتهي كل موزع من عمله، يُرسل زمنه. أما الماستر يستقبل كل النتائج ويحسب الزمن الأعظمي (لأن الموزعين يعملون على التوازي، فالزمن الكلي هو زمن أطول موزع).

```
 for (i = 0; i < spawned; i++) {
        pvm_recv(-1, 2);
        pvm_upkint(&result, 1, 1);
        printf("Received result from worker: %d\n", result);
        if (result > max_time)
            max_time = result;
```

8. طباعة الزمن الكلي للتوزيع المتوازيثم انهاء المهمة .

```
printf("\nTotal parallel distribution time: %d units\n", max_time);

    pvm_exit();
    return 0;
}
```

كود الابن (worker):

1. حصل على **id للمهمة الأب** (الماستر) التي أنشأت هذه المهمة(worker) يحصل على **id المهمة الحالية** الـ worker نفسه

```
  int ptid = pvm_parent();
    int mytid = pvm_mytid();
```

2. تعريف المتحولات الثلاث الأساسية و هي :
    1. عدد النقاط (صيدليات + عيادات) التي يجب على هذا الموزّع خدمتها.
    2. الزمن الوسطي اللازم لتوزيع الدواء على نقطة واحدة.
    3. الزمن الكلي الذي يستغرقه هذا الموزّع

```
    int num_points, avg_time, total_time;
```

3. شرط تحقق إذا لم يكن هناك Parent Task أي لم يتم استدعاء هذا البرنامج من قبل الماستر باستخدام pvm_spawn يتم طباعة خطأ والخروج ثم طباعة Tidهذا الموزّع لأغراض التحقق أنه بدأ بنجاح.

```
if (ptid < 0) {
        printf("Error: This is not a worker task\n");
        exit(1);
    }

    printf("Worker started, my tid = %d\n", mytid);
```

4. يستقبل رسالة من الماستر ويجب أن تكون مرسلة بالـ tag = 1. ثم فك تحزيم البيانات المستلمة(unpacking)

حيث تحوي عدد النقاط و الزمن الوسطي لكل نقطة (هذه البيانات تم إرسالها مسبقًا من قبل الماستر باستخدام .pvm_send).

4. يتم حساب الزمن الإجمالي الذي يحتاجه هذا الموزّع لإتمام التوزيع = عدد النقاط × الزمن الوسطي لكل نقطة.

```
pvm_recv(ptid, 1);
  pvm_upkint(&num_points, 1, 1);
  pvm_upkint(&avg_time, 1, 1);

5.     total_time = num_points * avg_time;
```

5. في هذا السطر تعرض التفاصيل

```
    printf("Worker %d will handle %d points, total time = %d\n", mytid, num_points,
total_time);
```

6. هنا نقوم بتحضير رسالة جديدة :

- تحزيم قيمة total_timeفي الرسالة
- إرسال الرسالة إلى الأب (الماستر) باستخدام tag = 2
- الماستر سيكون بانتظار هذه الرسائل باستخدام .pvm_recv(-1, 2)

```
pvm_initsend(PvmDataDefault);
   pvm_pkint(&total_time, 1, 1);
   pvm_send(ptid, 2);
```

إنهاء الاتصال مع نظام PVM بطريقة نظيفة و إنهاء البرنامج بنجاح

```
pvm_exit();
    return 0;
}
```

نلاحظ أن : عند زيادة عدد الموزعين، يقل الزمن الكلي.

التنفيذ التفرعي يحسّن الأداء إذا تم تقسيم العمل بعدالة.

البرنامج يحاكي بشكل دقيق مفهوم التوازي.

الحل في بيئة mpi

كود واحد لا غير لان مفهوم الاب و الابن غير موجود في mpi

1. تهيئة MPI ويضمن إنهاء البيئة تلقائيًا عند الانتهاء

```
using (new MPI.Environment(ref args))
{
```

2. تعريف المتحولات لاساسية : قناة الاتصال بين كل العمليات و رقم العملية (يبدأ من 0) عدد العمليات الكلي) كم عملية MPI تعمل معك)

```
Intracommunicator comm = Communicator.world;
int rank = comm.Rank;
int size = comm.Size;
```

3. طلب المدخلات من المستخدم من عدد صيدليات \موزعين\عيادات\..

```
if (rank == 0)
{
    Console.Write("Enter number of pharmacies: ");
    numPharmacies = int.Parse(Console.ReadLine());

    Console.Write("Enter number of clinics: ");
    numClinics = int.Parse(Console.ReadLine());

    Console.Write("Enter number of distributors (workers): ");
    numDistributors = int.Parse(Console.ReadLine());

    Console.Write("Enter average time per point: ");
    avgTime = int.Parse(Console.ReadLine());
```

4. يتم جمع النقاط ثم التحقق **في حال** أدخل المستخدم عدد موزعين أكبر من العمليات المتاحة (باستثناء الماستر)، يتم إيقاف البرنامج

```
totalPoints = numPharmacies + numClinics;

if (numDistributors != size - 1)
{
```

يقوم بتوزيع المهمات و ارسال المعلومات لكل موزع

```
pointsPerDistributor = totalPoints / numDistributors;

for (int i = 1; i < size; i++)
{
    comm.Send(pointsPerDistributor, i, 1);
    comm.Send(avgTime, i, 1);
}
```

.يستقبل النتائج من كل موزع بعد أن يحسب كل موزع الزمن، يقوم بإرساله الماستر يستقبله ويطبع النتيجة و يحتفظ بأطول وقت و هو الوقت الموازي الفعلي.

```
for (int i = 1; i < size; i++)
{
    totalTime = comm.Receive<int>(i, 2);
    Console.WriteLine($"Received result from worker {i}: {totalTime}");
    if (totalTime > maxTime)
        maxTime = totalTime;
}

Console.WriteLine($"\nTotal parallel distribution time: {maxTime} units");
```

يستقبل عدد النقاط والزمن من الماستر و نحسب الزمن ثم نرسل القيمة للماستر ليقارنها مع maxTime ويطبعها ثم ينهي البرنامج بشكل نظيف

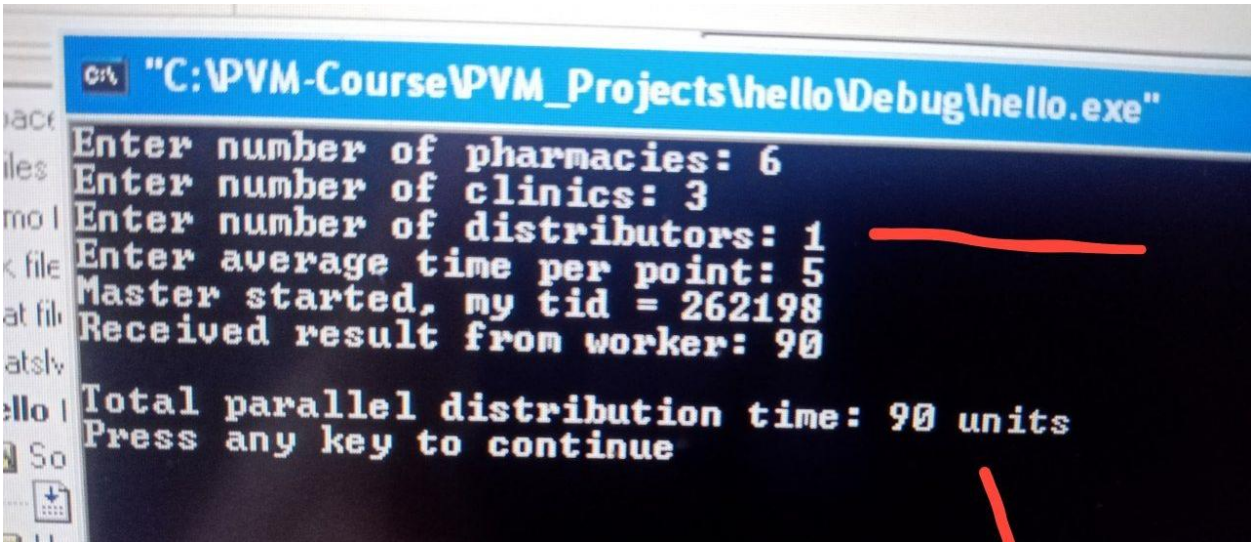
```
int numPoints = comm.Receive<int>(0, 1);
avgTime = comm.Receive<int>(0, 1);

totalTime = numPoints * avgTime;

Console.WriteLine($"Worker {rank} will handle {numPoints} points, total time = {totalTime}");

comm.Send(totalTime, 0, 2);
```

الخرج PVM



الخرج mpi

```
C:\Users\Lenovo\source\repos\thelastonetaf\thelastonetaf\bin\Debug>mpiexec -n 2 thelastonetaf.exe
Enter number of pharmacies: 6
Enter number of clinics: 3
Enter number of distributors (workers): 1
Enter average time per point: 5

Worker 1 will handle 9 points, total time = 45
Received result from worker 1: 45

? Total parallel distribution time: 45 units
```

_كما نلاحظ أعلاه فإن الخرجفي mpi كان ب45 وحدة وقت بينما ال pvm كان ب 90 وحدة !

و هذا غالبا لسبب ان الكود pvm حسب زمن الارسال و الاستقبال فأصبح 90=2*45 و لا هو يفترض يكون 45 ...

أي ان نتيجة المقارنةفي حال وجود موزع واحد هي التعادل و ذلك لان الاختلاففي الأداء لن يظهر فرق الافي حال وجود عدة موزعين يعملون على التوازي .

دراسة حالة (ان يكون عدد الموزعين اكبر من عدد المهمات) في mpi :

المدخلات :

عدد النقاط (صيدليات + عيادات) = 9

عدد الموزّعين = 8

متوسط الزمن لكل نقطة = 5 وحدات زمن

الخرج كالتالي :

```
Worker 1 will handle 0 points, total time = 0
Worker 2 will handle 0 points, total time = 0
Worker 3 will handle 0 points, total time = 0
Worker 4 will handle 0 points, total time = 0
Worker 6 will handle 0 points, total time = 0
Worker 5 will handle 0 points, total time = 0
Worker 7 will handle 0 points, total time = 0
Worker 8 will handle 0 points, total time = 0
Received result from worker 1: 0
Received result from worker 2: 0
Received result from worker 3: 0
Received result from worker 4: 0
Received result from worker 5: 0
Received result from worker 6: 0
Received result from worker 7: 0
Received result from worker 8: 0

? Total parallel distribution time: 0 units
```

و هذا خطأ .. لذلك سنقوم بتعديل الشرط في الكود ليقوم بزيادة نقطة في حال كان عدد النقاط لا يقبل القسمة على عدد الموزعين

```
int basePoints = totalPoints / numDistributors;
int extraPoints = totalPoints % numDistributors;

for (int i = 1; i < size; i++)
{
    int assignedPoints = basePoints + (i <= extraPoints ? 1 : 0);
    comm.Send(assignedPoints, i, 1);
    comm.Send(avgTime, i, 1);
}
```

بعد تعديل الشرط سيصبح الخرج 5 وحدات .. كل موزع فعلي سيحسب5 = 5 × 1 :

والماستر سيأخذ أقصى زمن بين كل العمال = 5

```
C:\Users\Lenovo\source\repos\thelastonetaf\thelastonetaf\bin\Debug>mpiexec -n 9 thelastonetaf.exe
Enter number of pharmacies: 6
Enter number of clinics: 3
Enter number of distributors (workers): 8
Enter average time per point: 5

Worker 2 will handle 1 points, total time = 5
Worker 4 will handle 1 points, total time = 5
Worker 1 will handle 1 points, total time = 5
Worker 3 will handle 1 points, total time = 5
Worker 5 will handle 1 points, total time = 5
Worker 6 will handle 1 points, total time = 5
Worker 7 will handle 1 points, total time = 5
Worker 8 will handle 1 points, total time = 5
Received result from worker 1: 5
Received result from worker 2: 5
Received result from worker 3: 5
Received result from worker 4: 5
Received result from worker 5: 5
Received result from worker 6: 5
Received result from worker 7: 5
Received result from worker 8: 5

? Total parallel distribution time: 5 units
```

دراسة حالة (ان يكون عدد الموزعين اكبر من عدد المهمات) في pvm :

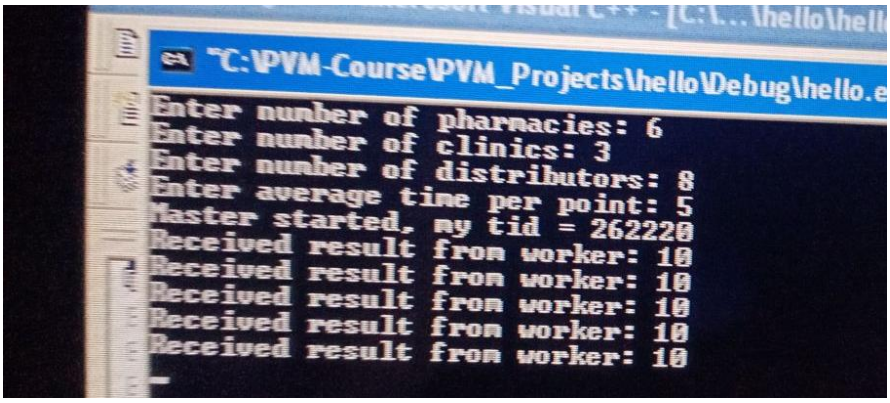
في حال عدم تعديل الشرط

```
points_per_distributor = total_points / num_distributors
```

سيكون الخرج على الحالة السابقة 0=8/6

أي ان ال total =0 و هذا خطأ لذلك سيتم أيضا تعديل الشرط كمافي ال mpi

و الخرج مساوي لخرج ال mpi لكن تم احتساب الزمن للارسال و الاستقبال ايضا



اما حالة ان يكون عدد النقاط او المهمات اكبر من عدد الموزعين فان الخرج سيكونفي

mpi :

هو مدة أطول موزع

```
Enter number of pharmacies: 6
Enter number of clinics: 3
Enter number of distributors (workers): 8
Enter average time per point: 64

Worker 1 will handle 1 points, total time = 64
Worker 2 will handle 1 points, total time = 64
Worker 4 will handle 1 points, total time = 64
Worker 3 will handle 1 points, total time = 64
Worker 5 will handle 1 points, total time = 64
Worker 6 will handle 1 points, total time = 64
Worker 7 will handle 1 points, total time = 64
Received result from worker 1: 64
Received result from worker 2: 64
Worker 8 will handle 1 points, total time = 64
Received result from worker 3: 64
Received result from worker 4: 64
Received result from worker 5: 64
Received result from worker 6: 64
Received result from worker 7: 64
Received result from worker 8: 64

? Total parallel distribution time: 64 units
```

Pvm :

64 وحدة و قد يعطي أخطاء منطقية في حال كان التوزيع غير عادل ..

تم بحمد الله ,مشكورين لحسن قرائتكم ..