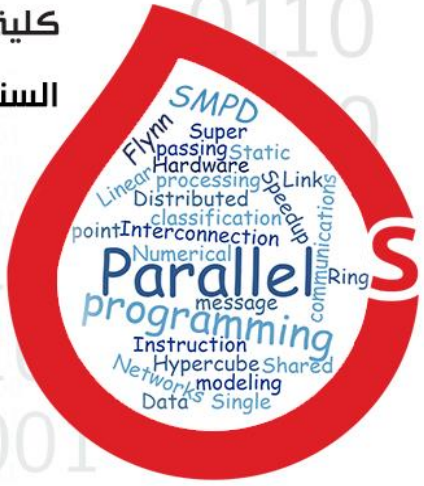

MPI (Message PASSING INTERFACE)

الفرق بين PVM وMPI:

- الـ PVM هي عبارة عن Framework ثابتة، بينما الـ MPI هي مكتبة برمجية تحتفظ بذات الترويسة لمعظم التوابع الخاصة بها، لكن يختلف الـ Implementation باختلاف اللغات البرمجية المستخدمة.
- في الـ MPI لا يوجد Parent يقوم بعمل Spawnللأبناء، اذ نلاحظ اختفاء مفهوم (أب-ابن)، الجميع في MPI هي مهام (الكود يتم تنفيذه nمرة حسب عدد الـ Processes التي نحددها عند تشغيل البرنامج).
- في الـ MPI هناك مجموعة تدعى (MPI Communicator World) موجودة بشكل افتراضي، كافة المهام تنضم تلقائياً لهذا الـ Communicator عند انشائها، والمهام داخله تأخذ Rank (ID) يبدأ من الصفر.

1. مكتبة مبنية على TCP/IP، مخصصة لتبادل البيانات من خلال الـ Message Passing، يتم استخدامها على نطاق واسع لبناء البرامج التفرعية عالية الأداء.
2. إن المهام ضمن الـ MPI معزولة عن بعضها البعض بشكل كلي، ولا يمكن لأي مهمة ملاحظة مهمة أخرى أو التعديل عليها إلا عبر الاستجابة للرسائل المتبادلة فيما بينها.
3. إن معظم تطبيقات الـ MPI تتم كتابتها عبر النموذج التفرعي SIMD (Single Program Multiple Data Stream).
4. إن الـ MPI Communicators تسمح بالتخاطب والاتصال بين مختلف المهام ضمن الـ MPI، حيث كل تطبيق MPI يبدأ بـ 2 Communicators فقط:

a. World Communicator: يحوي كافة مهام الـ MPI التي يبدأ برنامج الـ MPI بها.
b. Self-Communicator: يحوي فقط المهمة الخاصة به، حيث كل مهمة تنشأ ضمن البرنامج يكون لها Self-Communicator خاص بها، ولا يحوي سوى هذه المهمة.

MPI Point-to-Point Communication:

1. عند الإرسال، يتم تحديد ثلاث معاملات بالترتيب التالي:
   a. البيانات التي سيتم ارسالها.
   b. الـ Destination Rank Process.
   c. Message Tag.
2. عند الاستقبال، يتم تحديد ثلاث معاملات بالترتيب التالي:
   a. الـ Source Rank Process.
   b. Message Tag.
   c. البيانات التي سيتم استقبالها.

MPI Point-to-Point Communication:

- Blocking Communication:
  - Operations will wait until a communication has Completed in its local Process Before Continuing.
- Non-Blocking Communication:
  - It will Initiate a Communication Without waiting for that Communication to be Completed.

في الـ MPI ضمن الـ C#، تختلف آلية الاستقبال بناءً على نمط البيانات التي تم إرسالها:

1. فإذا تم إرسال Primitive Data Type (int, char, String, etc...)، يتم استلام هذه البيانات وتخزينها بشكل مباشر ضمن متحول معرف بشكل مسبق.
2. أما اذا كانت البيانات المرسلة Public Structures (Arrays, Objects, etc...)، فيتم استلامها ضمن المستقبل كـ Reference كما سنرى ضمن الأمثلة.

في الـ MPI ضمن الـ C#، تختلف آلية الاستقبال بناءً على نمط البيانات التي تم إرسالها:

1. فإذا تم إرسال Primitive Data Type (int, char, String, etc...)، يتم استلام هذه البيانات وتخزينها بشكل مباشر ضمن متحول معرف بشكل مسبق.
2. أما اذا كانت البيانات المرسلة Public Structures (Arrays, Objects, etc...)، فيتم استلامها ضمن المستقبل كـ Reference كما سنرى ضمن الأمثلة.

MPI FILES

- First :
  - Make sure that you have Visual c++ on your Laptop or you Should Install it first.
- Second :
  - Install The HCPC Pack on your Device (mpi_x64 or x86) - **The Execution Environment**
  - you can Check the Process by Running the "**mpiexec**" Command on CMD ("C:\Program Files\Microsoft HPC Pack 2012")
- (You Might need to install .net Framework 2.0 before the next Step)j
- Finally:
  - Install the MPI.NET Runtime then MPI.NET SDK ("C:\Program Files (x86)\MPI.NET") – The Development Environment.

البرنامج الذي سنعمل عليه هو Microsoft Visual Studio 2012.

قبل البدء يجب إضافة المكتبة البرمجية (MPI.dll) والتي سيتم استخدامها لكتابة تطبيقات MPI بلغة الـ C# ضمن البرنامج:

من خيار Project ثم Add Reference، عند فتح النافذة نختار Browse ثم نقوم بتحديد المسار الذي تتواجد به المكتبة **MPI.dll**.

C:\Program Files (x86) \MPI.NET\Lib

STEP ONE

a. البرنامج الذي سنعمل عليه هو Microsoft Visual Studio 2012.

b. قبل البدء يجب إضافة المكتبة البرمجية (MPI.dll) والتي سيتم استخدامها لكتابة تطبيقات MPI بلغة الـ C# ضمن البرنامج:

من خيار Project ثم Add Reference، عند فتح النافذة نختار Browse ثم نقوم بتحديد المسار الذي تتواجد به المكتبة **MPI.dll**.

C:\Program Files (x86)\MPI.NET\Lib

First Code

```
static void Main(string[] args)
{
    double Start_time, Time;
    using (new MPI.Environment(ref args))
    {
        // MPI program goes here! At then end MPI.Communicator.Dispose will be called
        Start_time = MPI.Environment.Time;
        Console.WriteLine("The size of the communicator is :" + Communicator.world.Size);
        Console.WriteLine("Hello, World! from rank " + Communicator.world.Rank
                          + " (running on " + MPI.Environment.ProcessorName + ")");

        Time = MPI.Environment.Time - Stime;
        Console.WriteLine("MPI Process Requires :" + Time);
    }
}
```

- التعليمة MPI.Environment.Time

ترد الزمن الحالي نستخدم هذه التعليمة لحساب الزمن التفرعي لتنفيذ البرنامج، عن طريق كتابتها مرتين، الأولى عند بدء تنفيذ البرنامج، والثانية بنهاية التنفيذ، ثم نقوم بحساب زمن التنفيذ الإجمالي ضمن المتحول Time عبر طرح زمن نهاية البرنامج من زمن البداية.

- نستخدم التعليمة Communicator.world.Size

لطباعة الـ Size للـ World Communicator والذي يعبر عن عدد المهام ضمنه.

- نستخدم التعليمة Communicator.world.Rank

لطباعة الـ Rank للـمهمة الحالية.

- نستخدم التعليمة MPI.Environment.ProcessorName

لطباعة اسم الجهاز الذي يقوم بتشغيل المهمة الحالية.

Execute the Code

1. الخطوة الأولى: نقوم بعمل Start للبرنامج (Ctrl+F5).
2. الخطوة الثانية: نقوم بالتوجه إلى مسار المشروع ضمن
C:\Users\Ammar000\Documents\Visual Studio 2012\Projects
ندخل إلى مجلد bin ثم debug، نقوم بنسخ كامل المسار وفتحه ضمن الـ CMD (نستخدم التعليمة cd).
3. الخطوة الثالثة: نقوم بتحديد عدد المهام التي سنقوم بتشغيلها عبر استخدام التعليمة
mpiexec -n "num of Processes" "ProgramName.exe"

```
C:\Users\Ammar000\Documents\Visual Studio 2012\Projects\mpi\mpi\bin\Debug>mpiexec -n 1 mpi.exe
The size of the communicator is :1
Hello, World! from rank 0 (running on DESKTOP-OAHVTGM)
MPI with Rank 0 Requires: 0.00113899999996647
```

Second Code

```
using (new MPI.Environment(ref args))
{
    string msg, stringmsg;
    Intracommunicator comm = Communicator.world;
    if (comm.Rank == 0)
    {
        comm.Send("My name is Ammar", 1, 0);

        // receive the final message
        stringmsg = comm.Receive<string>(Communicator.anySource, 0);
        //stringmsg = comm.Receive<string>(1, 0);
        Console.WriteLine("Process with rank " + comm.Rank  + " received message \"" + stringmsg + "\".");
    }
    else // not rank 0
    {
        msg = comm.Receive<string>(comm.Rank - 1, 0);
        Console.WriteLine("Process with rank " + comm.Rank + " received message \"" + msg + "\".");

        comm.Send(msg + ", " + comm.Rank, 0, 0);
    }
}
```

- يستخدم الـ IntraCommunicator Class

لتحقيق عمليات الارسال والاستقبال بين المهام، والتي ستتم عبر الـ Communicator.

- نستخدم التابع comm.Send

بغرض الإرسال المتزامن، مدخلات التابع هي:

1. البيانات التي سيتم إرسالها إلى المستقبل.
2. الـ Target Process.
3. Message Tag

المُجتَهِد الصّادِق لا بُدَّ أن يَصِل.

- نستخدم التابع comm.Receive

للاستقبال المتزامن، مدخلات التابع هي:

1. الـ Source Process:

يحدد Rank مهمة معينة لاستلام الرسالة منها، (الـ Communicator.anySource تمثل الاستلام من أي مهمة دون الاهتمام بالجهة المرسلة).

2. Message Tag.
3. البيانات التي سيتم استقبالها.

عند استخدام تابع الاستقبال نميز حالتين:

- عند استقبال Primitive Data Types يتم استقبال هذه البيانات بشكل مباشر، عندها يتم تحديد معاملين في تابع الاستقبال وهما (Source Process, Msg Tag)، ويكون خرج التابع comm.Receive متحول من ذات نمط المعطيات التي سيتم استلامها، حيث يكتب بالشكل:

String x = comm.Recieve< String>(Source Process, int Msg Tag)

- عند استقبال Public Structures يتم استقبال هذه البيانات كـ Reference حيث يكتب التابع Receive بالشكل: comm.Recieve(Source Process, int Msg Tag, ref Array_Name)

```
C:\Users\Ammar000\Documents\Visual Studio 2012\Projects\mpi\mpi\bin\Debug>mpiexec -n 2 mpi.exe
Process with rank 1 received message "My name is Ammar".
Process with rank 0 received message "My name is Ammar, 1".
```

Third Code

- عند استقبال Public Structures يتم استقبال هذه البيانات كـ Reference، إذ نقوم بتهيئة مصفوفة للاستقبال ضمن المستقبل بحجم أكبر من حجم المصفوفة التي سيتم إرسالها.

تَتَّسِعُ مَيادين الفتى على قَدرِ هِمّته،
وتَعلو جهداً بحَجم قِمَّته

```
static void Main(string[] args)
{
    using (new MPI.Environment(ref args))
    {

        Intracommunicator comm = Communicator.world;


        if (comm.Rank == 0)
        {
            int[] values = new int[5];
            Console.WriteLine("Input of prcoess with Rank " + comm.Rank + " please insert 5 values");
            for (int i = 0; i <= 4; i++) values[i] = Convert.ToInt32(Console.ReadLine());

            comm.Send(values, 1, 0);
        }

        else if (comm.Rank == 1)
        {
            int[] values2 = new int[10];
            comm.Receive(0, 0, ref values2); // okay: array of 10 integers has enough space to receive 5 integers

            values2[5] = values2[0] + values2[1] + values2[2] + values2[3] + values2[4];
            Console.WriteLine("Prcoess with Rank" + comm.Rank + " received the array");

        }


    }
```

```
C:\Users\Ammar000\Documents\Visual Studio 2012\Projects\mpi\mpi\bin\Debug>mpiexec -n 3 mpi.exe
Input of prcoess with Rank 0 please insert 5 values
5
8
4
3
6
Prcoess with Rank1 received the array
```

Forth Code

```
static void Main(string[] args)
{
    using (new MPI.Environment(ref args))
    {
        Intracommunicator comm = Communicator.world;

        int[] arr = new int[1000];
        Random rand = new Random();
        double stime = MPI.Environment.Time;

        //using non-blocking send and receive
        stime = MPI.Environment.Time;
        if (comm.Rank == 0)
        {
            for (int i = 0; i <= arr.Length - 1; i++)
                arr[i] = rand.Next(1, 1000);
            for (int j = 1; j <= comm.Size - 1; j++)
                comm.ImmediateSend(arr, j, 0);
        }
        else
        {
            comm.ImmediateReceive(0, 0, arr);

            Console.WriteLine("process " + comm.Rank + " received the array " + " Length is " + arr.Length);
        }
        comm.Barrier();
        extime = MPI.Environment.Time - stime;
        if (comm.Rank == 0)
            Console.WriteLine("Execution time using non-blocking send/receive on " + comm.Size + " processes is " + extime + " seconds");

    }
```

- نستخدم التابع comm.ImmediateSend

بغرض الإرسال غير المتزامن، مدخلات التابع هي:

1. **البيانات التي سيتم ارسالها إلى المستقبل.**
2. **الـ** Target Process.
3. Message Tag.

- نستخدم التابع comm.ImmediateReceive

للاستقبال غير المتزامن، مدخلات التابع هي :

1. **الـ** Source Process: **يحدد** Rank **مهمة معينة لاستلام الرسالة منها، (الـ** Communicator.anySource **تمثل الاستلام من أي مهمة دون الاهتمام بالجهة المرسلة.)**
2. Message Tag.
3. **البيانات التي سيتم استقبالها: (حيث يتم استقبال هذه البيانات بشكل مباشر سواءً كانت** Primitive **أو** Public Structures).

```
C:\Users\Ammar000>cd C:\Users\Ammar000\Documents\visual studio 2012\Projects\mpi\mpi\bin\Debug

C:\Users\Ammar000\Documents\Visual Studio 2012\Projects\mpi\mpi\bin\Debug>mpiexec -n 2 mpi.exe
process 1 received the array 10000
Execution time using blocking send/receive on 2 processes is 0.00422699999762699 seconds
process 1 received the array  Length is 10000
Execution time using non-blocking send/receive on 2 processes is 0.00131030002376065 seconds
```

النّهاية ❤