

البرمجة التفرعية باستخدام الـ MPI ضمن بيئة الـ VS2010

1 مفردات الجلسة:

- ✓ الأدوات البرمجية المطلوبة
- ✓ مقدمة عن استعمال الـ MPI والتوابع الرئيسية
- ✓ تدريب عملي

2 الأدوات البرمجية المطلوبة:

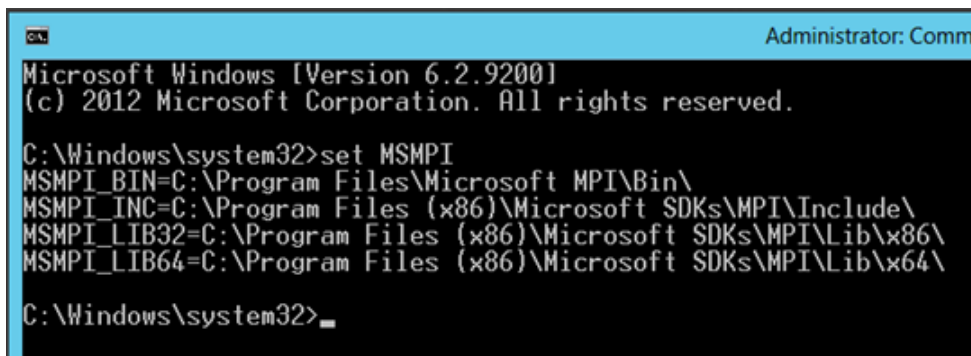
- ✓ تثبيت الـ MPI ضمن بيئة الـ VS2010:

من أجل استخدام البرمجة التفرعية الـ MPI ضمن بيئة الـ VS2010 يجب إتباع الخطوات التالية:

1. تنزيل وتثبيت الحزمة MS-MPI SDK and Redist installers، والتي يمكن الحصول عليها من الرابط

التالي: <https://msdn.microsoft.com/en-us/library/bb524831.aspx>

2. إعداد بارامترات الـ MPI ضمن متغيرات نظام التشغيل كما هو موضح بالشكل:



```
Administrator: Comm...
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Windows\system32>set MSMPI
MSMPI_BIN=C:\Program Files\Microsoft MPI\Bin\
MSMPI_INC=C:\Program Files (x86)\Microsoft SDKs\MPI\Include\
MSMPI_LIB32=C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x86\
MSMPI_LIB64=C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x64\

C:\Windows\system32>
```

- ✓ لغة البرمجة وقد تم استخدام Visual Studio C++

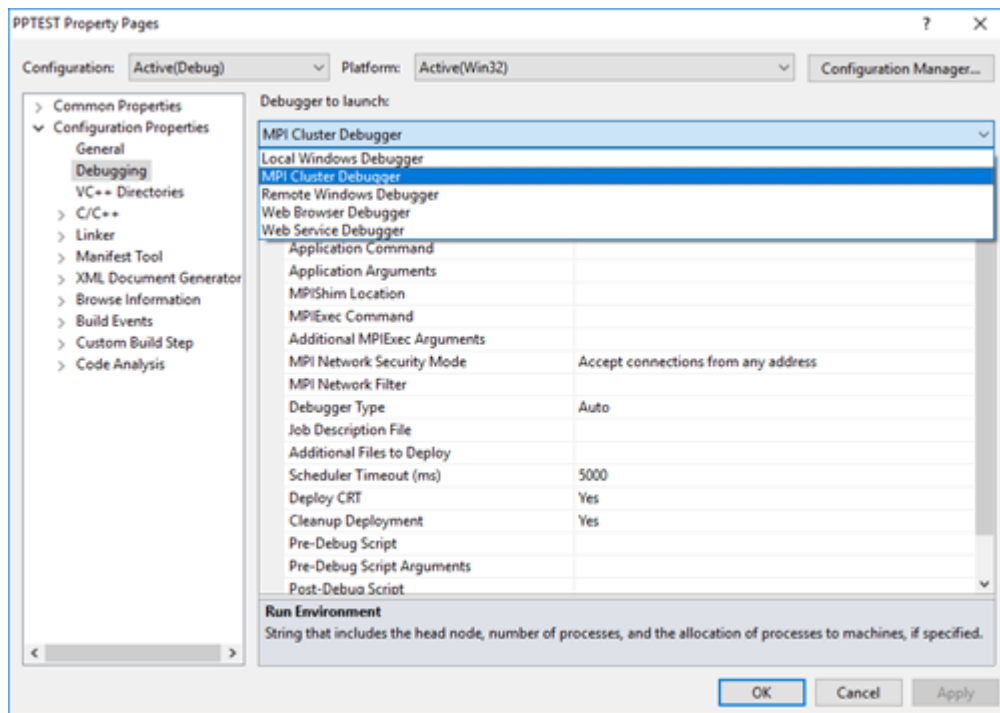
1. البدء بالعمل ضمن بيئة الـ VS2010 عن طريق فتح البرنامج

2. إنشاء مشروع جديد ضمن الـ VS2010 (هنا يمكن اختيار مشروع فارغ أو Win32 Console

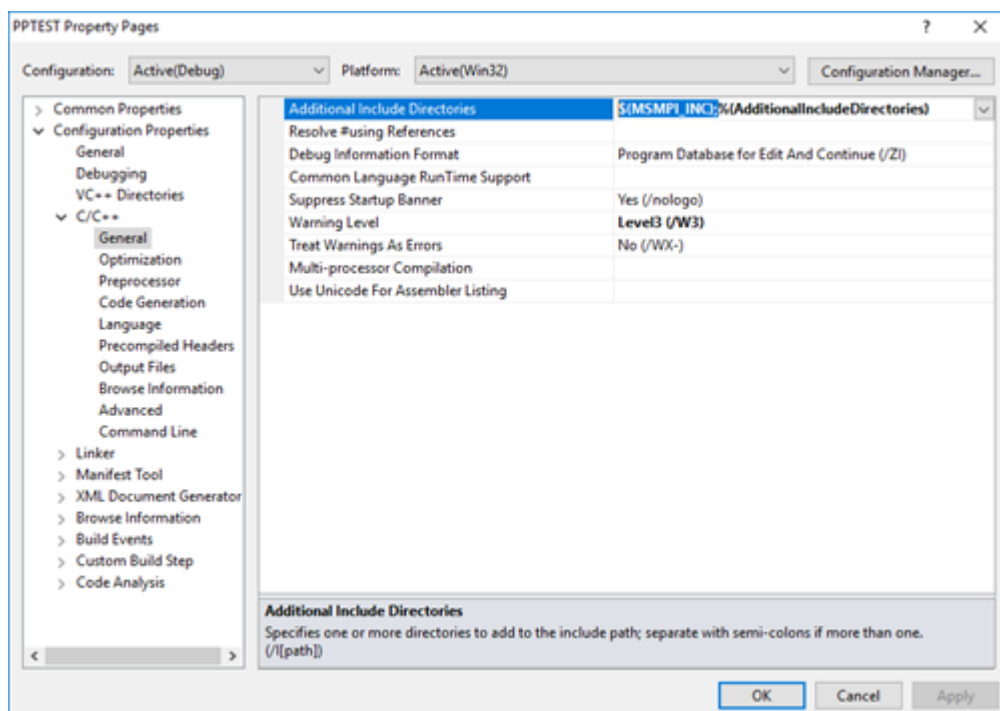
(Application

3. البدء بإعداد المشروع كي يعمل مع الـ MPI على النحو التالي:

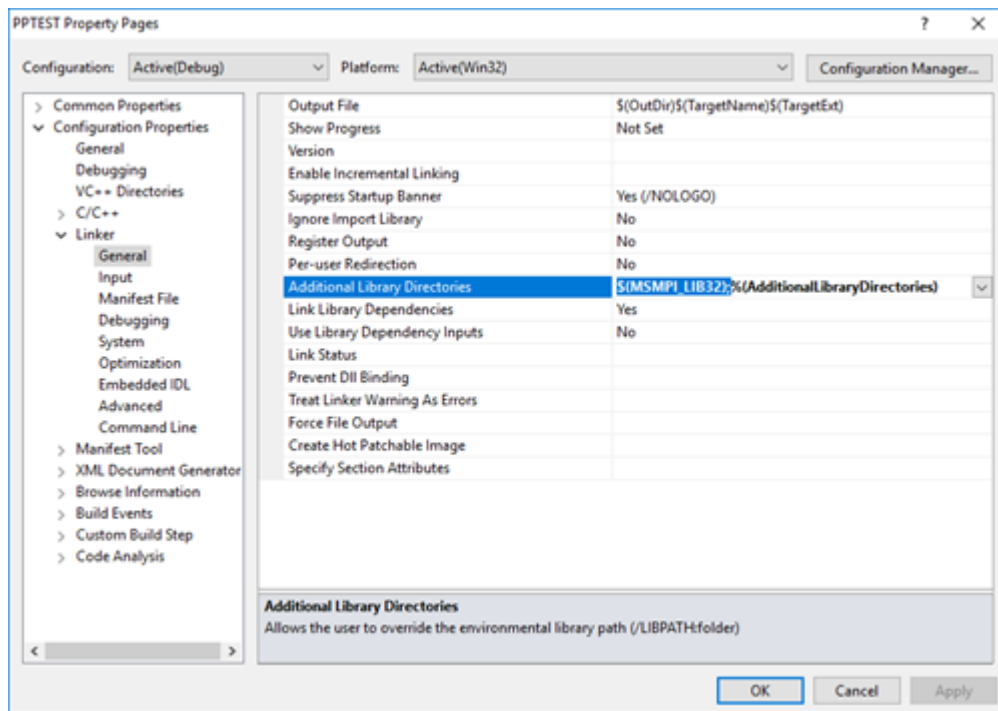
- النقر بالزر الأيمن على المشروع الذي تم إنشاؤه واختيار Properties
- اختيار الخاصية Debugging من النافذة اليسارية ومن ثم تغيير نوع الـ Debugger إلى النمط MPI Cluster Debugger



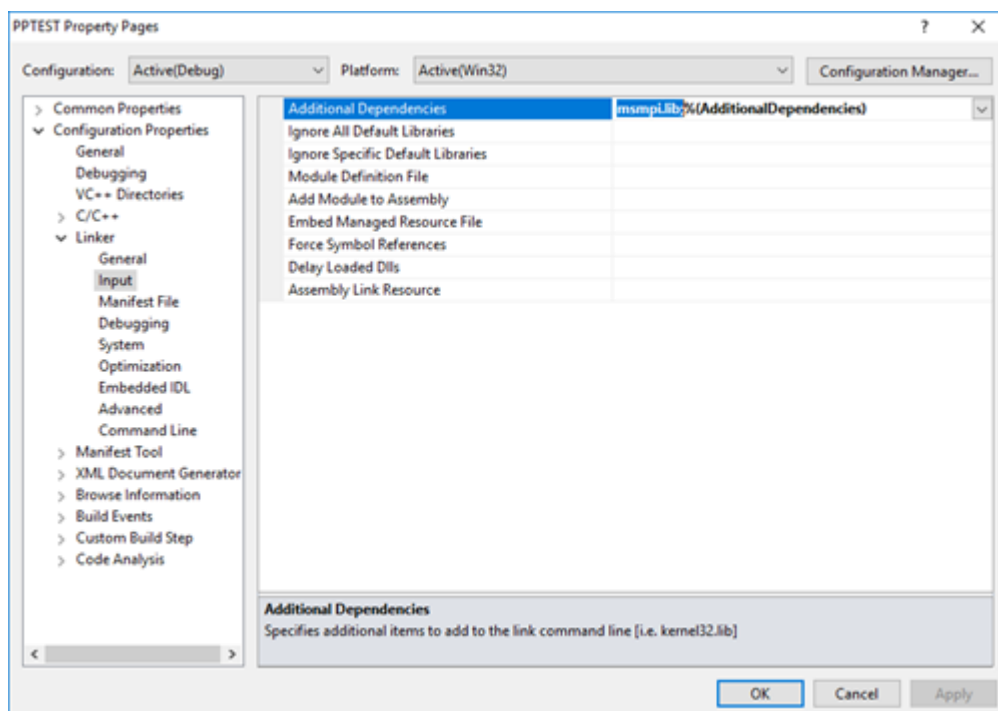
- تغيير قيمة الخاصية Run Environment إلى القيمة localhost/4 والتي ستؤمن لنا العمل مع أربع عمليات Process
- اختيار الخاصية General -> C/C++ من النافذة اليسارية واعداد الخاصية Additional Include Directories لتضمين المكتبة \$(MSMPI_INC)



- اختيار الخاصية General -> Linker من النافذة اليسارية واعداد الخاصية Additional Library Directories لتضمين المكتبة \$(MSMPI_LIB32)



- اختيار الخاصية Input -> Linker من النافذة اليسارية واعداد الخاصية Additional Dependencies لتضمين المكتبة msmpl.lib عن طريق النقر على Edit



3 مقدمة عن استعمال الـ MPI والتوابع الرئيسية

1.3 كتابة البرامج باستخدام الـ MPI:

تعتمد الـ MPI على نموذج اتصال بسيط بين العمليات التفرعية والذي يعتمد على مبدأ Send/Receive كما هو موضح بالشكل:



عند تهيئة بيئة الـ MPI سيتم تجميع العمليات Processes ضمن مجموعات. يوجد ضمن الـ MPI مجموعات (communicator) معرفة مسبقاً MPI_COMM_WORLD. يتم تحديد كل عملية بمحدد ID فريد ضمن الـ communicator والذي يستخدم من أجل عملية الاتصال ويدعى rank. يمكن لعملية واحدة أن تأخذ محددات مختلفة باختلاف الـ communicator. للعمل ضمن الـ MPI يجب أولاً تضمين المكتبة mpi.h ومن ثم تهيئها عن طريق التابع MPI_Init(&argc, &argv); وفي النهاية يجب إنهاء الـ MPI عن طريق التابع MPI_Finalize(); كما هو موضح بالمثل التالي:

```

#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("I am %d of %d\n", rank, size);

    MPI_Finalize();
    return 0;
}
  
```

Basic requirements for an MPI program

2.3 أهم التوابيع المستخدمة ضمن الـ MPI:

أهم التوابيع المستخدمة ضمن الـ MPI:

MPI_Init	Initialize MPI
MPI_Comm_size	Find out how many processes there are
MPI_Comm_rank	Find out which process I am
MPI_Send	Send a message
MPI_Recv	Receive a message
MPI_Finalize	Terminate MPI

تأخذ التوابيع الثلاثة الأولى الصيغ التالية:

```

int MPI_Init(int *argc, char ***argv)

int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)
  
```

حيث تمثل MPI_Comm comm البارامتر الخاص بالمجموعة communicator والتي يمكن أن تأخذ المجموعة المعرفة مسبقاً ضمن الـ MPI وهي MPI_COMM_WORLD أو تعريف communicator آخر.

3.3 أنواع البيانات المستخدمة في الـ MPI وما يقابلها في لغة الـ C++

تستخدم الـ MPI مجموعة من البيانات الخاصة بها والموضحة بالجدول التالي:

MPI datatype	C/C++ datatype
MPLCHAR	signed char
MPLSHORT	signed short int
MPLINT	signed int
MPLLONG	signed long int
MPLUNSIGNED_CHAR	unsigned char
MPLUNSIGNED_SHORT	unsigned short int
MPLUNSIGNED	unsigned int
MPLUNSIGNED_LONG	unsigned long int
MPLFLOAT	float
MPLDOUBLE	double
MPLLONG_DOUBLE	long double
MPLBYTE	
MPLPACKED	

4.3 عملية تبادل البيانات ضمن الـ MPI (Data Communication):

هنا سيتم استخدام التابعين MPI_Send و MPI_Receive والليذان يأخذان الصيغ التالية:

✓ تابع الإرسال:

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

يبين الجدول التالي الوسائط المطلوبة لهذا التابع

```
buf: initial address of send buffer (choice)
count: number of elements in send buffer (nonnegative integer)
datatype: datatype of each send buffer element (handle)
dest: rank of destination process (integer)
tag: message tag (integer)
comm: communicacator (handle)
```

✓ تابع الاستقبال:

```
MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
MPI_Status *status)
```

يبين الجدول التالي الوسائط المطلوبة لهذا التابع

```
buf: initial address of send buffer (choice)
count: number of elements in send buffer (nonnegative integer)
datatype: datatype of each send buffer element (handle)
source: rank of sourc or MPI_ANY_SOURCE (integer)
tag: message tag or MPI_ANY_TAG (integer)
comm: communicacator (handle)
' status: status object (Status)
```

1.4 تدريب 1: المطلوب تنفيذ هذا الكود باستخدام الـ Debugger

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define MASTER 0
int main (int argc, char *argv[]) {
    int numtasks, taskid, len;
    char hostname[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Get_processor_name(hostname, &len);
    printf ("Hello from task %d on %s!\n", taskid, hostname);
    if (taskid == MASTER)
        printf("MASTER: Number of MPI tasks is: %d\n", numtasks);
    MPI_Finalize(); }
```

2.4 تدريب 2: المطلوب تنفيذ هذا الكود باستخدام الـ Debugger

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char ** argv) {
    int rank, data[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
        MPI_Send(data, 100, MPI_INT, 1, 0, MPI_COMM_WORLD);
    else if (rank == 1)
        MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
    MPI_Finalize();
    return 0;
}
```

3.4 تدريب 3: المطلوب تنفيذ هذا الكود باستخدام الـ Debugger

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define MASTER 0
int main (int argc, char *argv[]) {
    int numtasks, taskid, len, partner, message;
    char hostname[MPI_MAX_PROCESSOR_NAME];
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    /* need an even number of tasks */
    if (numtasks % 2 != 0) {
        if (taskid == MASTER)
            printf("Quitting. Need an even number of tasks: numtasks=%d\n", numtasks);
    }
    else {
        if (taskid == MASTER)
            printf("MASTER: Number of MPI tasks is: %d\n", numtasks);
        MPI_Get_processor_name(hostname, &len);
        printf ("Hello from task %d on %s!\n", taskid, hostname);
        /* determine partner and then send/receive with partner */
        if (taskid < numtasks/2) {
            partner = numtasks/2 + taskid;
            MPI_Send(&taskid, 1, MPI_INT, partner, 1, MPI_COMM_WORLD);
            MPI_Recv(&message, 1, MPI_INT, partner, 1, MPI_COMM_WORLD, &status);
        }
        else if (taskid >= numtasks/2) {
            partner = taskid - numtasks/2;
            MPI_Recv(&message, 1, MPI_INT, partner, 1, MPI_COMM_WORLD, &status);
            MPI_Send(&taskid, 1, MPI_INT, partner, 1, MPI_COMM_WORLD);
        }
        /* print partner info and exit*/
        printf("Task %d is partner with %d\n", taskid, message);
    }
}
```

```
MPI_Finalize();  
}
```

4.4 تدريب 4 (وظيفة): المطلوب برمجة المعادلة الرياضية التالية $(a+b)*(c-d)-(e*f)+(h*g)$ باستخدام البرمجة التفرعية (MPI) بحيث يتم استخدام اربع معالجات (عمليات) كل معالج يقوم بتنفيذ عملية حسابية والنتائج يتم ارساله إلى المعالج الأول لإظهار النتيجة النهائية: