البرمجة التفرعية
Parallel Programming

References

- Peter S. Pacheco, An Introduction to Parallel Programming, Morgan Kaufmann Publishers is an imprint of Elsevier 2011

البرمجة المتوازية البرمجة عن طريق تمرير الرسائل

- مقدمة:
  - كيف نقوم ببرمجة المسائل المجزأة على الحاسب المتوازي؟
    - الجواب: باستخدام نوع من البرمجة يطلق عليه "البرمجة المتوازية".
    - إن التعبير عن التوازي في برامج المستخدم يتطلب تعبيرًا في اللغات البرمجية وشكلها، فهي تتطلب مثلا:
      - بعض التعليمات الأولية للتعبير عن التوازي بين مهمتين،
      - وأخرى من أجل التخاطب والتزامن وما شابه ذلك
  - بالتعريف:
    - هي البرمجة بلغة تتضمن البنى أو المميزات المتوازية
    - يمكن أن يتم بناء المميزات المتوازية من خلال:
      - بعض اللغات البرمجية التي تعتمد مبدأ التوازي في تصميمها
      - توسيع لغات البرمجة التسلسلية من أجل احتواء تعليمات التوازي
      - إلحاق المزايا المتوازية إلى لغة تسلسلية تقليدية وذلك باستخدام إجرائيات المكتبات مثل مكتبة MPI
- واجهة تمرير الرسائل Message Passing Interface MPI:
  - تعتبر مكتبة قياسية يعتمد عليها منتجو الحاسبات المتوازية
  - عرّفت هذه المكتبة معايير قياسية لتمرير الرسائل
  - يمكن أن تستخدم لتطوير برامج تمرير الرسائل من خلال لغتي البرمجة FORTRAN أو C/C++
  - تحتوي على أكثر من 125 إجرائية
    - لكن يمكن كتابة برامج متوازية كاملة الوظائف باستخدام 6 إجرائيات فقط تستخدم لبدء وإنهاء المكتبة، ولإحضار معلومات عن بيئة التشغيل المتوازية التي يعمل عليها البرنامج بالإضافة إلى إرسال واستقبال الرسائل

مجموعة من الروتينات الأساسية الخاصة بالمكتبة MPI	
MPI_Init	بداية MPI
MPI_Finalize	إنهاء MPI
MPI_Comm_size	تحديد عدد المعالجات
MPI_Comm_rank	تحديد عنوان أو رتبة المعالج المستدعي
MPI_Send	إرسال رسالة
MPI_Recv	استقبال رسالة

- الهيكل العام لبرامج MPI:
  - قبل أي استدعاء لأي إجرائية ضمن المكتبة MPI يتم استدعاء الإجرائية MPI_Init
    - وظيفته هي تشغيل بيئة MPI
    - استدعاء هذا الإجراء أكثر من مرة خلال فترة عمل البرنامج سيتسبب في حدوث خطأ
  - تستدعى الإجرائية MPI_Finalize لإنهاء الحساب
    - يجب ألا يستدعى أي إجرائية ضمن الـ MPI بعد استدعاء هذه الإجرائية
  - طريقة الاستدعاء:
    - int MPI_Init(int *argc, char ***argv)
    - int MPI_Finalize()
  - إن جميع الإجرائيات و أنواع البيانات و الثوابت ضمن الـ MPI تسبق بالبادئة MPI_
    - MPI_Init
    - MPI_SUCCESS وهو ثابت يتم إرجاعه عندما تتم العملية الحسابية بنجاح
  - جميع هذه الإجرائيات وأنواع البيانات والثوابت تكون معرفة ضمن المكتبة mpi.h والتي يجب تضمينها في جميع برامج الـ MPI
  - مثال:

```
#include <iostream.h>
#include <mpi.h> // لتضمين المكتبة في برنامجنا
int main(int argc, char ** argv){
MPI_Init( &argc, &argv);
cout << "Welcome!" << endl;
```

MPI include file – ملف التضمين

Initialize MPI environment – بدأ البيئة

Do work and make message passing calls - أداء العمل

Terminate MPI Environment – إنهاء البيئة

```
MPI_Finalize();
}
```

- إذا كان البرنامج متوازي تماما، كما في المثال السابق، فإن العمليات التي تحدث بين عبارتي التهيئة والإنهاء لا تَستخدِم أي اتصالات

- المراسِلات Communicators:
  - أحد الأشياء الرئيسية التي تستعمل في جميع برامج الـMPI الحقيقة هو ما يطلق عليه مجال الاتصال (communication domain)
    - مجال الاتصال هو مجموعة من الإجرائيات تسمح بحدوث اتصال فيما بينها.
    - بعض المعلومات حول مجال الاتصال تكون مخزنة في متغيرات من نوع MPI_Comm تدعى بالمراسلات
    - تُستَخدم كبارامترات لجميع إجرائيات نقل الرسائل في الـMPI
    - تستخدم المراسلات لتعريف مجموعة من الإجرائيات يمكن أن تتصل فيما بينها.
      - هذه المجموعة من الإجرائيات تشكل مجال تراسل.
    - بشكل عام قد تحتاج جميع الإجرائيات للاتصال مع بعضها البعض
      - لهذا السبب فإن الـMPI تُعرِّف مراسلات افتراضية تدعى MPI_COMM_WORLD
        - تتضمن جميع الإجرائيات المستخدمة للتنفيذ المتوازي
    - باستعمال مراسلات مختلفة لكل مجموعة يمكن ضمان أن الرسائل لا تتداخل أبدا مع رسائل مجموعة أخرى
  - الحصول على معلومات عن بيئة التشغيل:
    - تستخدم الإجرائيتان MPI_Comm_rank و MPI_Comm_size للحصول على معلومات عن البيئة التي يعمل فيها البرنامج
      - الأولى تستخدم لتحديد عدد الإجرائيات
      - والثانية لتحديد عنوان أورتبة الإجرائية المستدعية
    - تأخذ الصيغة التالية:
      - int MPI_Comm_size(MPI_Comm comm, int *size)
        - تُرجِعُ في المتغير size عدد الإجرائيات التي تنتسب لمجال الاتصال comm
      - int MPI_Comm_rank(MPI_Comm comm, int *rank)
        - كل إجرائية تتبع لمجال الاتصال تُعرّف بواسطة رتبتها rank وهي عدد صحيح يتراوح من صفر إلى حجم مجال الاتصال ناقص واحد

- يجب على الإجرائية التي تستدعي أي من هذه الإجرائيات أن تكون تابعًة لمجال الاتصال، وإلا سوف يحدث خطأ
  - مثال:

```
#include <iostream.h>
#include <mpi.h>
main(int argc, char *argv[]){
int npes, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
cout <<"Welcome! from process "<<myrank;
cout <<"of "<<npes <<endl;
MPI_Finalize();
}
```

- تراسل البيانات ضمن الـMPI :
  - يمكن أن يتم إرسال الرسائل واستقبالها باستخدام الدالتين التاليتين:
    - MPI_Send للإرسال

```
int MPI_Send(
void* message              /*in*/,
int count                  /*in*/,
MPI_Datatype datatype      /*in*/,
int dest                   /*in*/,
int tag                    /*in*/,
MPI_Comm comm              /*in*/
)
```

    - MPI_Recv للاستقبال

```
int MPI_Recv(
void* message              /*out*/,
int count                  /*in*/,
MPI_Datatype datatype      /*in*/,
int source                 /*in*/,
```

```
int tag                 /*in*/,
MPI_Comm comm           /*in*/,
MPI _Status* status     /*out*/
)
```

- البارامتر tag من نوع عدد صحيح يستخدم للتمييز بين أنواع الرسائل المختلفة.
- يمكن أن يأخذ قيم تتراوح من صفر وحتى الحد الأعلى المعرف بواسطة MPI_TAG_UP
- أنواع البيانات ضمن الـ MPI :
  - تتطابق أنواع البيانات ضمن الـ MPI بمثيلاتها الموجودة في لغة البرمجة C++ بالإضافة إلى أنواع أخرى خاصة بها مثل: MPI_PACKED

أنواع البيانات في MPI	أنواع البيانات في C++
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	long double
MPI_BYTE	------
MPI_PACKED	------

- إذا كان هناك العديد من الرسائل لها نفس الـ tag من نفس الإجرائية، فإنه يتم استقبال أي واحدة من هذه الرسائل
- يوجد رمز عام للبارامترات سواء بارامتر المصدر source أو الـ tag:
  - MPI_ANY_SOURCE: أي إجرائية في مجال الاتصال يمكن أن تكون المصدر للرسالة
  - MPI_ANY_TAG: فإن الرسائل يتم قبولها جميعًا بأي tag
- يجب أن تكون الرسالة المستقبلة بطول العازل المجهز ضمن إجرائيتي الارسال والاستقبال
  - إذا كانت الرسالة المستقبلة أكبر من العازل المجهز، فسينتج الخطأ بتجاوز الحد المسموح، وسيعيد الإجراء بالخطأ MPI_ERR_TRUNCATE
- بعد أن تستقبل الرسالة، فإنه يمكن استخدام المتغير status للحصول على معلومات حول عملية الإرسال
  - تركيب البارامتر MPI_Status:

- typedef struct MPI_Status {
  int MPI_SOURCE;
  int MPI_TAG;
  int MPI_ERROR;
  };
- لا يمكن الحصول على المعلومات الموجودة ضمن المتغير status مباشرة، بل يمكن أن نحصل عليها باستدعاء الدالة MPI_Get_count
  - int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)
  - مثال:

- int mynode, totalnodes;
  int datasize; // عددوحدات المعطيات التي سترسل أو تستقبل
  int sender; // رقم الإجرائية المرسلة
  int receiver // رقم الإجرائية المستقبلة
  int tag // عدد صحيح يستخدم آلقب أو علامة للرسالة
  MPI_Status status; // متغير يحتوي معلومات عن الحالة
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
  MPI_Comm_rank(MPI_COMM_WORLD, &mynode); // تحديد datasize عددوحدات المعطيات
  double * databuffer =new double[datasize];
  //Fill in sender,receiver,tag on sender/receiver processes, and fill in databuffer on the sender process.
  if(mynode==sender)
  MPI_Send(databuffer,datasize, MPI_DOUBLE,receiver,
  tag,MPI_COMM_WORLD);
  if(mynode==receiver)
  MPI_Recv(databuffer,datasize, MPI_DOUBLE,sender,tag,
  MPI_COMM_WORLD,&status);// انتهت عملية الإرسال والاستقبال