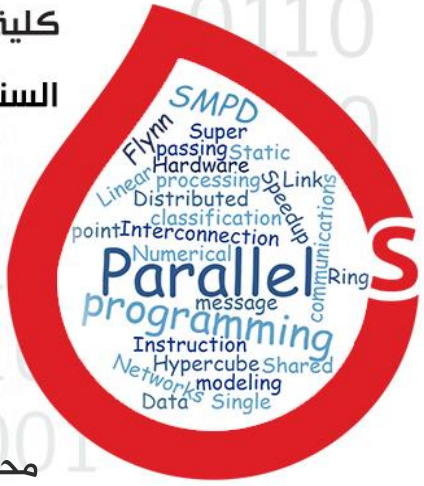
MPI Collective



م.عمار المصري

محتوى مجاني غير مخصص للبيع التجاري

RB Informatics; 23/05/2023

البرمجة التفرعية

MPI COLLECTIVE OPERATIONS

تكلمنا سابقاً عن التوابع التجميعية وكيفية عمل الـ implementation لها بطريقة الـ PVM ،
في هذه المحاضرة سنقوم بتطبيق جميع التوابع السابقة وتحقيقها ولكن بطريقة الـ MPI .

1. BroadCast

comm.Broadcast(ref Data, int Root_Rank)

- يقوم بإرسال الرسالة لكافة المهام ضمن Communicator ما (One to all) ، حيث يتم تحديد البيانات التي سوف يتم ارسالها .
- المهمة الجذر هي التي ستقوم بإرسال الرسالة ، حيث يتم الإرسال بذات اللحظة لكافة العناصر .
- البيانات يتم إرسالها كـ Reference سواءً كانت Primitive أو Public Structure .
- يتم كتابة التابع ضمن الجزء البرمجي الخاص بالمهمة الجذر أو ضمن القسم المشترك بين المهام ، ولا حاجة لاستخدام أي تابع لاستقبال الرسالة المرسلة حيث يتم إنشاء نسخة من الرسالة المرسلة ضمن الـ Address Space الخاص بكل مهمة .
- الـ Rank الخاص فيه متغير بناءً على المجموعة .

2. Gather

comm.Gather(Data, int Root_Rank)

- في هذه الحالة يكون لدينا مجموعة من البيانات موزعة بين المهام ، نقوم بتجميع هذه البيانات حيث كل مهمة ضمن المجموعة تقوم بإرسال البيانات المتواجدة لديها ، لتقوم المهمة الجذر بعملية استلام لهذه البيانات وتجميع الكتل الواردة ضمن مصفوفة بالتتالي حسب الـ Rank للمهمة المرسلة .

- خرج هذا التابع هو مصفوفة التجميع ، وتكون متواجدة ضمن الـ Address Space للمهمة الجذر فقط .

- يتم كتابة هذا التابع ضمن القسم المشترك بين المهام ، ولا حاجة لاستخدام أي تابع لاستقبال البيانات التي سيتم إرسالها من قبل كافة المهام حيث يتم إنشاء مصفوفة التجميع مباشرة ضمن المهمة الجذر.

3. Scatter

- في هذه الحالة يكون لدينا مصفوفة من البيانات موجودة ضمن الجذر ، حيث تقوم بتوزيع هذه البيانات على المهام ضمن الـ Communicator فيتم إرسال البيانات بالتتالي حسب الـ Rank ، أي أن المهمة ذات الـ Rank = 0 تستلم القيمة المتواجدة ضمن الـ index = 0 من المصفوفة وهكذا .

- يشترط على حجم المصفوفة المراد توزيعها ان يساوي عدد المهام المتواجدة ضمن الـ Communicator أي حجم المصفوفة يساوي عدد العناصر .

▪ الإرسال في Scatter :

- الإرسال لدى الجذر يتم عبر التابع :

```
comm.Scatter(Array_Name);
```

  - حيث يتم تحديد مصفوفة التوزيع المتواجدة لدى الجذر ليتم توزيعها على المهام الموجودة ضمن الـ Communicator.

▪ الاستقبال في Scatter :

- الاستقبال لدى المهام الـ non – Root يتم عبر التابع نفسه مع اختلاف المدخلات :

```
comm.Scatter< -- >(int Root_Rank);
```

  - حيث يتم تحديد المهمة الجذر التي قامت بإرسال البيانات .
  - خرج التابع لدى كافة المهام (Root / non – Root) هو متحول يتم تخزين فيه القيمة التي سيتم استقبالها من الجذر .

4. Reduce

comm.Reduce(Data, Operation, int Root_Rank)

- يقوم التابع بتنفيذ عملية محددة بين مجموعة من البيانات المتواجدة ضمن المهام ضمن Communicator ما ، ويقوم بتخزين القيمة الناتجة عن العملية ضمن المهمة الجذر .

- يُستدعى من أجل كافة المهام لمرة واحدة .
- يُستخدم في القسم المشترك للكود .
- يُقصد بـ operation : العملية الحسابية .
- خرج العملية هو متحول من نفس نمط المعطيات ويكون موجود ضمن الروت.

- First Code

- First. Scatter The Array :

```
static void Main(string[] args)
{
    using (new MPI.Environment(ref args))
    {
        Intracommunicator comm = Communicator.world;
        int[] arr = new int[comm.Size];
        Random rand = new Random();
        int value;

        //scatter the array
        if (comm.Rank == 0)
        {
            for (int i = 0; i < arr.Length; i++)
                arr[i] = rand.Next(1, 1000);
            Console.WriteLine(" max is " + arr.Max() + " sum is " + arr.Sum());

            value = comm.Scatter(arr);
        }
        else
        {
            value = comm.Scatter<int>(0);

        }
        comm.Barrier();

        Console.WriteLine("The value at process " + comm.Rank + " is  " + value);
        comm.Barrier();
```

- نستخدم الـ IntraCommunicator Class لتحقيق عمليات الإرسال والاستقبال بين المهام ، والتي ستتم عبر الـ World Communicator .

- المصفوفة arr وهي مصفوفة التوزيع المتواجدة لدى المهمة الجذر ذات الـ Rank = 0 حيث سيتم تعبئة المصفوفة بقيم عشوائية عبر استخدام الـ Random Class .

- ضمن المهمة الجذر:
  يتم تعبئة مصفوفة التوزيع ثم طباعة مجموع عناصر المصفوفة والقيمة الأعظمية ضمنها عبر استخدام مجموعة من التوابع الجاهزة ضمن اللغة .

- التابع comm.scatter() والذي سيقوم بتوزيع المصفوفة على المهام ضمن المجموعة حيث مدخلات التابع هي مصفوفة التوزيع .

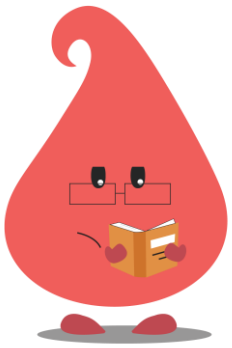
- ضمن باقي المهام التي يتم إنشاؤها الـ (non Root) نستخدم التابع comm.scatter() لاستلام قيمة من المصفوفة التي تم توزيعها ، حيث مدخلات التابع هنا هي الـ Rank للمهمة الجذر ، ويكون خرج هذا التابع هو متحول من النمط int سيتم تخزين فيه القيمة التي سيتم استقبالها من قبل كل مهمة .

- التابع comm.Barrier() هو تابع انتظار ، حيث يقوم بإيقاف المهمة حتى يتم استدعاء التابع ذاته من قبل كافة المهام المتواجدة ضمن الـ Communicator .

- First Code

- Second. Change Array Values, Gather it, then Reduce The Array Values using SUM && MAX Functions :

```
//change the received value
value *= comm.Rank;

Console.WriteLine("The new value at process " + comm.Rank + " is   " + value);
comm.Barrier();

//Gather the new_array
int[] results = comm.Gather(value, 2);

if (comm.Rank == 2)
    for (int i = 0; i < results.Length; i++)
        Console.WriteLine(results[i]);

comm.Barrier();


//reduce the results
int a = comm.Reduce(value, Operation<int>.Max, 0);

int b = comm.Reduce(value, Operation<int>.Add, 1);
comm.Barrier();

if (comm.Rank == 0)
    Console.WriteLine("New max is:    " + a);
if (comm.Rank == 1)
    Console.WriteLine("New sum is:    " + b);
```



- التابع comm.Gather() والذي سيقوم بتجميع البيانات من كل مهمة في الـ Communicator ضمن مصفوفة في الجذر ، ويستخدم مرة واحدة من أجل كافة المهام ، حيث مدخلات التابع :
  - القيمة التي سترسلها كل مهمة إلى الجذر .
  - Root Rank .

- التابع comm.Reduce() يُستخدم من أجل كافة المهام ، حيث مدخلات التابع :
  - البيانات التي سيتم تطبيق العملية عليها .
  - التابع المراد تنفيذه على القيم.
  - Root Rank .

- يتم تخزين القيمة الأعظمية الجديدة ضمن الجذر الجديد ذو الـ Rank = 0 ، والمجموع الجديد للعناصر ضمن الجذر ذو الـ Rank = 1 ومن ثم طباعة القيم .

```
C:\Users\Ammar000\Documents\Visual Studio 2012\Projects\mpi_collective_communication\mpi_collective_communication\bin\Debug>mpiexec -n 7 mpi_collective_communication.exe

 max is 994 sum is 3473
The value at process 0 is  630
The value at process 3 is  133
The value at process 4 is  161
The value at process 5 is  994
The value at process 1 is  242
The value at process 2 is  951
The value at process 6 is  362
The new value at process 3 is  399
The new value at process 0 is  0
The new value at process 1 is  242
The new value at process 2 is  1902
The new value at process 4 is  644
The new value at process 5 is  4970
The new value at process 6 is  2172
0
242
1902
399
644
4970
2172
New max is:   4970
New sum is:   10329
```

- Second Code

```
//using Broadcast

stime = MPI.Environment.Time;


if (comm.Rank == 0)
{
    for (int i = 0; i <= arr.Length - 1; i++)
        arr[i] = rand.Next(1, 10000);
}

comm.Broadcast(ref arr,0);
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 Broadcast on " + comm.Size + " processes is " + extime + " seconds");
```

- تطبيق نفس العملية السابقة ولكن باستخدام عملية الـ BroadCast .

سؤال: من الأسرع من بين العمليتين التاليتين؟ البرودكاست أم الإرسال والاستقبال المتزامن ؟

الجواب: فعلياً يختلف ذلك على حسب الهاردوير المطبق عليه العملية التفرعية ولكن بحالة عامة يكون البرودكاست أسرع ، وللاختبار ذلك نقوم بزيادة عدد الprocess لأنه كلما زاد عدد الـ process الذي نريد التنفيذ عليها ؛ كلما توضح أن عملية البرودكاست أسرع .

– نهاية المقرر –

لا تنسونا من صالح دعاؤكم ..