البرمجة التفرعية

المهندس عمار المصري

- الجلسة الخامسة -

PVM Groups

- يستفاد من المجموعات في PVM في تبسيط عمليات تبادل الرسائل بين عدة مهام، حيث عندما نقوم بإرسال رسالة ما لمجموعة من الأبناء سأقوم بإرسالها لكل ابن على حدى، وهذا ما سينجم عنه تأخير في زمن التنفيذ واستهلاك أكبر لموارد الجهاز.

- يتمثل الحل في انشاء Group حيث سيتم إضافة مجموعة من المهام إليه، كل مهمة تملك Tid والتي تمثل الـ id الخاص بها، وعند دخولها إلى المجموعة ستعطى رقماً مميزاً Gid داخل هذه المجموعة، هذا الرقم يعبر عن موقع هذه المهمة ضمن المجموعة وهو يبدأ من الصفر ويزداد بشكل تصاعدي، وعند مغادرة المهمة لهذه المجموعة سيصبح هذا الـ gid فارغاً، وسيتم اسناده لأول مهمة جديدة تقوم بالدخول إلى المجموعة.

- لكل مجموعة يوجد Root Task تكون مسؤولة عن كافة عمليات المجموعة (وليس بالضرورة أن تكون المهمة الأب).

Group Functions

- **pvm_joingroup(char *groupName)**
  - تابع الانضمام لمجموعة، يحدد اسم المجموعة ضمن متحول الدخل الوحيد، انشاء المجموعة يتم من خلال التابع ذاته عند استدعائه من أجل مجموعة غير موجودة سابقاً، عندها تنشأ المجموعة الجديدة بالاسم، وتضاف المهمة التي قامت باستدعاء التابع كعنصر أول ضمن المجموعة و تحصل على gid=0، يمكن للمهمة أن تنضم إلى أكثر من مجموعة وتحصل على gid خاص لها ضمن كل مجموعة.
- **pvm_lvgroup(char *groupName)**
  - تابع لمغادرة المجموعة.
- **pvm_getinst(char *groupName, int taskID)**
  - تابع للحصول على الـ gid لمهمة ما ضمن مجموعة، حيث يتم تحديد اسم المجموعة والـ Task ID للمهمة كمتحولات للدخل.
- **pvm_gettid(char *groupName, int GID)**
  - تابع للحصول على الـ tid لمهمة ما ضمن مجموعة، حيث يتم تحديد اسم المجموعة والـ gid للمهمة كمتحولات للدخل.

Collective Operations

- **MultiCast:** **pvm_mcast(childId, int childCount, int msgTag)**

- لا يصنف من التوابع التجميعية، يقوم بإرسال رسالة من مهمة ما لعدة مهام (One-to-Many)، يقوم بتحديد مصفوفة من الـ Task Id's تعبر عن المهام التي سيقوم بالإرسال إليها، ثم يقوم بتحديد عدد المهام التي ستستلم الرسالة المراد ارسالها بالإضافة إلى tag الرسالة.
- يتم استلام الرسالة باستخدام التابع **pvm_recv()**.

- **BroadCast:** **pvm_bcast(char *groupName, int msgTag)**

- يقوم بإرسال الرسالة لكافة المهام ضمن مجموعة ما (One-to-All)، حيث يتم تحديد اسم المجموعة و tag الرسالة ضمن متحولات الدخل، حيث يقوم بإرسال الرسالة بذات اللحظة لكافة العناصر، وقد يكون المرسل من خارج أعضاء المجموعة.
- يتم استلام الرسالة باستخدام التابع **pvm_recv()**.

Collective Operations

- **Gather:**

- لدينا مجموعة من البيانات موزعة بين المهام، نقوم بتجميع هذه البيانات حيث كل مهمة ضمن المجموعة تجهز Block of Data وتقوم بإرساله، لتقوم المهمة الجذر بعملية استلام لهذه البيانات وتجميع الكتل الواردة ضمن مصفوفة بالتتالي حسب الـ gid للمهمة المرسلة.
- الإرسال لدى المهام الـ non-Root يتم عبر التابع :

pvm_gather(Null, void *send, int count, int datatype, int tag, char *group, int root)

حيث:

Send: تمثل عنوان كتلة البيانات التي سترسلها المهمة.

Count: عدد العناصر ضمن كتلة البيانات التي سيتم ارسالها.

Root: يمرر له الـ gid للمهمة الجذر (عندما يكون gid المهمة المستدعية لتابع الـ gather مختلف عن قيمة متحول الـ rootعندها يعمل gather للإرسال فقط).

Collective Operations

- **Gather:**
  - الاستقبال لدى الجذر يتم عبر ذات التابع مع اختلاف في قيمة المدخل الأول :

pvm_gather(void *gatherData, void *send, int count, int datatype, int tag, char *group, int root)

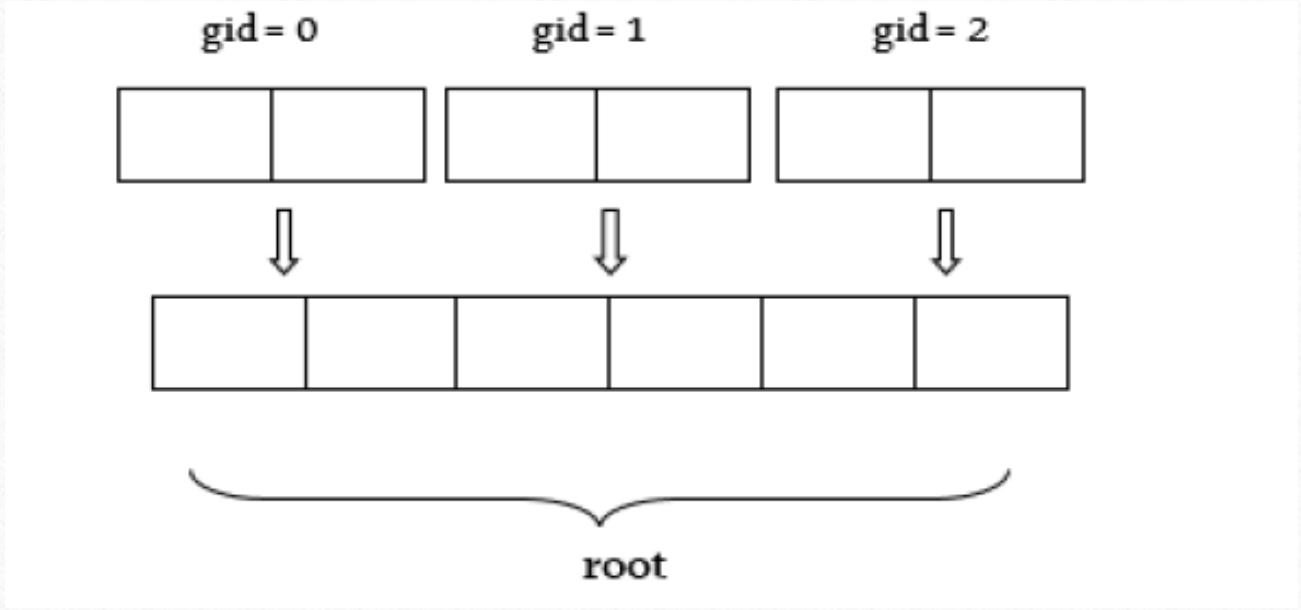
حيث:

GatherData: عنوان مصفوفة الاستلام (التجميع)، وتمرر فقط لدى المهمة الجذر حيث تعطى Null في بقية المهام.

Send: تمثل عنوان كتلة البيانات التي سترسلها المهمة.

Count: عدد العناصر ضمن كتلة البيانات التي سيتم ارسالها.

Root: يمرر له الـ gid للمهمة الجذر (عندما يكون gid المهمة المستدعية لتابع الـ gather مطابقاً لقيمة متحول الـ root عندها يعمل gather لاستلام وتجميع الرسائل، كما ترسل الكتلة الخاصة بها ليتم تجميعها في المصفوفة الكلية).


gid = 0
gid = 1
gid = 2
root

Collective Operations

- **Scatter:**

- لدينا مصفوفة من البيانات موجودة ضمن الجذر، نقوم بتوزيع هذه البيانات على المهام ضمن المجموعة حيث كل مهمة تستلم Block of Data، ارسال الكتل لمهام المجموعة يتم بالتتالي حسب gid، أي أن المهمة ذات gid=0 تستلم الكتلة الأولى وهكذا ..
- يشترط على حجم المصفوفة المراد ارسالها أن يساوي عدد مهام المجموعة * حجم الكتلة المرسلة لكل مهمة.
- الاستقبال لدى المهام الـ non-Root يتم عبر التابع :

pvm_scatter(void *rec, Null, int count, int datatype, int tag, char *group, int root)

حيث:

rec: تمثل عنوان الكتلة التي ستستقبلها المهمة.

Count: عدد العناصر ضمن كتلة البيانات التي سيتم استقبالها.

Root: يمرر له الـ gid للمهمة الجذر (عندما يكون gid المهمة المستدعية لتابع الـ Scatter مختلف عن قيمة متحول الـ root عندها يعمل gather للاستلام فقط).

Collective Operations

- **Scatter:**
  - الإرسال لدى الجذر يتم عبر ذات التابع مع اختلاف في قيمة المدخل الثاني:

pvm_scatter(void *rec, void *ScatterData, int count, int datatype, int tag, char *group, int root)

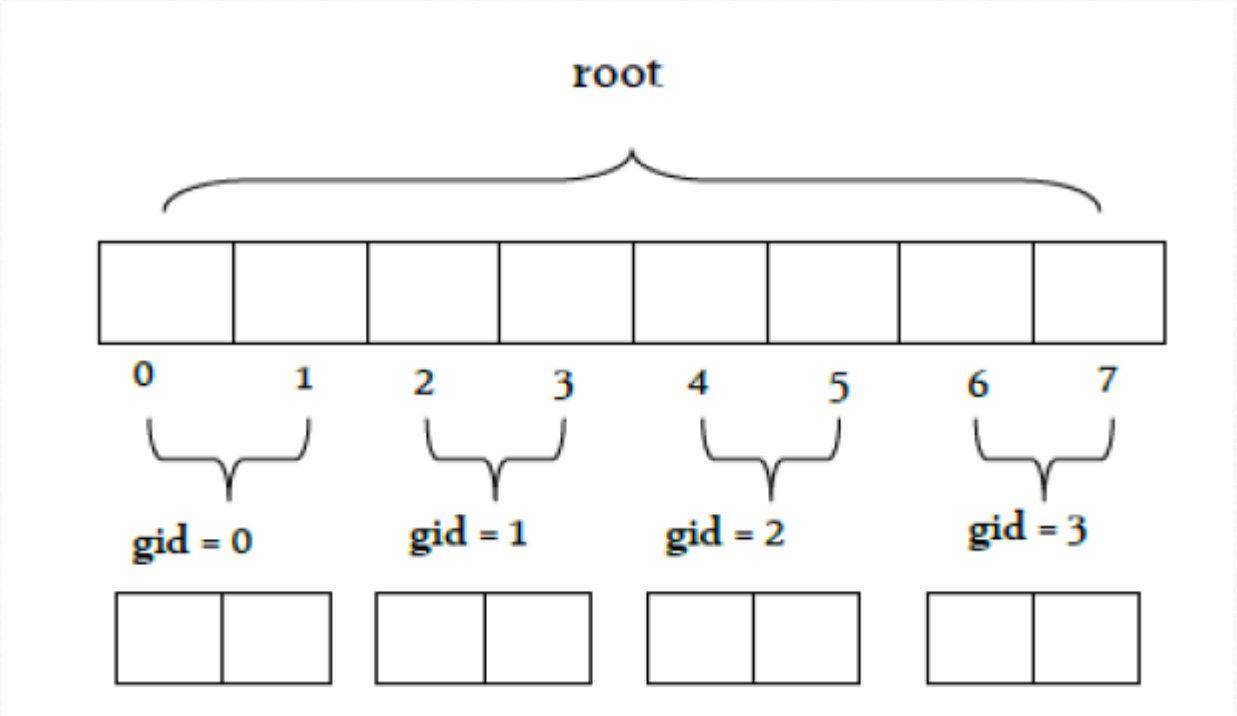
حيث:

ScatterData: عنوان المصفوفة الكلية التي سيتم توزيعها، وتمرر فقط لدى المهمة الجذر حيث تعطى Null في بقية المهام.

Send: تمثل عنوان كتلة البيانات التي ستستقبلها المهمة.

Count: عدد العناصر ضمن كتلة البيانات التي سيتم ارسالها.

Root: يمرر له الـ gid للمهمة الجذر (عندما يكون gid المهمة المستدعية لتابع الـ Scatter مطابقاً لقيمة متحول الـ root عندها يعمل Scatter للإرسال، كما أنه يستلم الجزء من المصفوفة الخاص بالمهمة الجذر كعضو في المجموعة).


root
0
1
2
3
4
5
6
7
gid = 0
gid = 1
gid = 2
gid = 3

Collective Operations

- **Reduce:**
  - يقوم بتنفيذ عملية محددة بين عناصر مصفوفة لدى كل مهمة في المجموعة، والمصفوفة المقابلة ضمن المهمة الجذر، يعدل ناتج العملية على قيم مصفوفة الجذر.
  - يستدعى من أجل جميع المهام لمرة واحدة:

pvm_reduce(void (*func)(), void *data, int count, int datatype, int msgtag, char *group, int root)

حيث:

func(): العملية المراد تنفيذها على المصفوفات، حيث يتم استخدام توابع معرفة مسبقاً ضمن الـ PVM، ومن التوابع المعرفة ضمن الـPVM : PVMMAX, PVMMIN, PVMSUM.

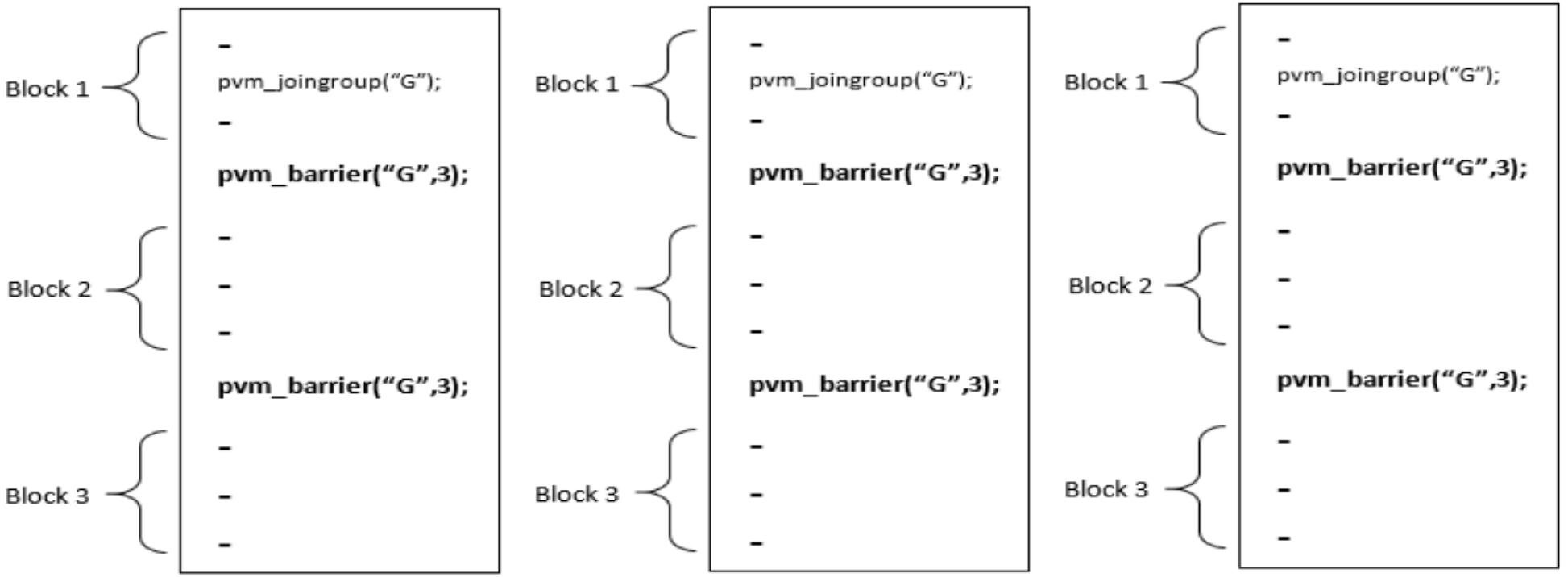
data: المصفوفة التي سيتم تطبيق عملية الـ Reduce عليها.

Collective Operations

- **Barrier:** **pvm_barrier(char *groupName, int count)**

- وهو تابع انتظار، حيث يقوم بإيقاف المهمة حتى يتم استدعاء التابع ذاته من قبل العدد المحدد Count من مهام المجموعة المحددة باسم المجموعة.

- أي أنه يوقف جميع المهام التي استدعت التابع pvm_barrier() ليحررها معاً عند الوصول للعدد المطلوب من المهام التي قامت باستدعائه من المجموعة، وهو يفيد في ضبط آلية تنفيذ المهام.



لا يبدأ تنفيذ block2 في أي مهمة قبل انتهاء block1 في المهام الثلاثة واستدعاء pvm-barrier ضمن المهام الثلاثة

لا يبدأ تنفيذ block3 في أي مهمة قبل انتهاء block2 في المهام الثلاثة واستدعاء pvm-barrier ضمن المهام الثلاثة.

Code:

```
1) void main(int argc, char* argv[]){
2) clock_t startTime, endTime;
3) char* groupName = "all";
4) int myId = pvm_mytid();
5) int myParent = pvm_parent();
6) bool isParent = (myParent == PvmNoParent) || (myParent == PvmParentNotSet);
7) int childCount, childBlock;
8) int* childId;
```

- السطر الثاني: تابع الـ Clock حيث يتم اسناد المتغيرين (start time, end time) في بداية البرنامج وعند نهايته، يتم من خلال هذين المتغيرين حساب زمن تنفيذ الـ main.

- السطر الثالث: اسم المجموعة التي سنقوم بإنشائها هو "all".

- السطر الرابع: يرد قيمة الـ id الخاص بالمهمة الحالية.

- السطر الخامس: يرد قيمة id الأب للمهمة، ولدينا أربع حالات:
  - يرد id الأب (قيمة موجبة) فالمهمة الحالية هي ابن.
  - Id<0، أي أن المهمة الحالية لا تملك أب (فهي أب).
  - PvmNoParent.
  - PvmParentNotSet.

- السطر السادس: متحول منطقي نسند له قيمة True في حال كانت قيمة myParent تشير إلى المهمة الأب.

- السطر الثامن: المصفوفة التي سيتم بها تخزين الـ tid’s للأبناء المنشأة.

Code:

```
9)  if (isParent) {//Parent Mode
10)     startTime = clock();
11)     cout << "Masetr Mode... ID = " << myId << endl;
12)     if (argc > 1)
13)        childCount = atoi(argv[1]);
14)     else
15)        childCount = 3;
16)     if (argc > 2)
17)        childBlock = atoi(argv[2]);
18)     else
19)        childBlock = 2;
20)     //pvm_catchout(cout);
21)     childId = new int[childCount];
22)     int cc = pvm_spawn("collect", NULL, 0, "", childCount, childId);
23)     if (cc != childCount) {
24)         cout << "\nFaild to spwan required children\n ..press any key to exit ";
25)        pvm_exit();
26)        int c; cin >> c;
27)        exit(-1);  }
28)    pvm_initsend(PvmDataDefault);
29)    pvm_pkint(&childBlock, 1, 1);
30)    pvm_pkint(&childCount, 1, 1);
31)    pvm_mcast(childId, childCount, 1);   }//End Parent Mode
```

- السطر العاشر: يتم تخزين الوقت الحالي ضمن المتحول start time ليعبر عن زمن بداية البرنامج.

- السطر 22: لتوليد الأبناء، حيث سيتم توليد ثلاث أبناء وتخزين قيمة الـ id's الخاصة بهم ضمن المصفوفة childID، حيث تعبر القيمة cc عن عدد الأبناء المنشأة بنجاح.

- السطر 28: تهيئة عملية الارسال.

- السطر 31: تابع الارسال MultiCast، حيث تم تحديد المدخلات التالية للتابع:
  - childID: مصفوفة المهام المراد الارسال لها.
  - childCount: عدد المهام التي سيتم ارسال الرسالة لها.
  - Message Tag.

Code:

```
32) else { //Child Mode
33)     pvm_recv(myParent, 1);
34)     pvm_upkint(&childBlock, 1, 1);
35)     pvm_upkint(&childCount, 1, 1);
36)     cout << "Data Received...\n"; }
```

Code:

First. Scatter The Array

```
1)  int myGroupId = pvm_joingroup(groupName);
2)   pvm_barrier(groupName, childCount+1);
3)   int groupSize = pvm_gsize(groupName);
4)   int allRoot;
5)     if(isParent)
6)         allRoot = pvm_getinst(groupName, myId);
7)     else
8)        allRoot = pvm_getinst(groupName, myParent);
9)   int* rec = new int[childBlock];
10)  if(isParent) { //Prepare Data to be sent
11)     int scatterCount = groupSize * childBlock;
12)     int* scatterData = new int[scatterCount];
13)     for(int i = 0; i<scatterCount; i++)
14)         scatterData[i] = i+1;
15)     pvm_scatter(rec, scatterData, childBlock, PVM_INT,2000, groupName, allRoot); }
16)  else //Not Root for Scatter... Then I Receive
17)    pvm_scatter(rec, NULL, childBlock, PVM_INT,2000, groupName, allRoot);
```

- السطر الأول: عندما تضاف المهمة في البداية يتم انشاء المجموعة، ويسند لها gid=0.

- السطر الثاني: لضمان عدم البدء حتى وجود كافة الأبناء ضمن المجموعة، حيث عدد المهام المنتظر انضمامها للمجموعة هي childCount+1.

- السطر الثالث: التابع **pvm_gsize()** وهو تابع يعيد عدد الأبناء ضمن مجموعة.

- السطر السادس حتى الثامن: سيتم تعيين المهمة الأب كجذر للمجموعة، حيث سيتم اسناد قيمة الـ gid للمهمة الأب للمتحول **allRoot**، وللوصول إلى الـ gid الخاص بالمهمة الأب نستخدم التابع pvm_getinst() حيث:
  - اذا كانت المهمة الحالية هي المهمة الأب يتم استدعاء التابع من أجل المتحول myId (وهو خرج التابع **pvm_mytid()**).
  - اذا كانت المهمة الحالية هي مهمة ابن يتم استدعاء التابع من أجل المتحول myParent (وهو خرج التابع **pvm_parent()**).

- السطر التاسع: تهيئة مصفوفة الاستقبال، حيث على كل مهمة حجز مصفوفة بالحجم المحدد للاستلام لتخزن ضمنها الـ Block الذي ستستلمه.

- السطر 11: عدد عناصر المصفوفة التي سيقوم الجذر بتوزيعها يساوي عدد الأبناء * حجم الكتلة الواحدة.

- السطر 15+17: التابع **pvm_scatter()** والذي سيقوم بتوزيع المصفوفة على المهام ضمن المجموعة، حيث مدخلات التابع هي:
  - Rec: عنوان مصفوفة الاستلام التي سيتم تخزين البيانات المستلمة بها.
  - ScatterData: عنوان المصفوفة الكلية التي سيتم توزيعها على الأبناء، ويمرر فقط ضمن المهمة الجذر المسؤولة عن حجز المصفوفة، أما في باقي المهام (non-Root) يكون Null.
  - childblock: حجم الكتلة المحدد ارسالها لكل مهمة.
  - groupName: اسم المجموعة التي سيتم عمل Scatter ضمنها.
  - allRoot: جذر المجموعة (قيمة الـ gid).

Code:

Second. Double The Array Values, then Gather it.

```
 1) int* send = new int[childBlock];
 2) for (int i = 0; i<childBlock ; i++)
 3)    send[i] = rec[i]*2;
 4)  if(!isParent)
 5)     pvm_gather(NULL, send, childBlock, PVM_INT,3, groupName, allRoot);
 6)  else {
 7)  int gatherCount = groupSize * childBlock;
 8)  int* gatherData = new int[gatherCount];
 9)  pvm_gather(gatherData, send, childBlock, PVM_INT,3, groupName,  allRoot);
    //Print Gathered Data
10)    cout<<"\nGathered Data...\n";
11)    for(int i = 0; i<gatherCount; i++) {
12)     cout<< gatherData[i];
13)     if(i==gatherCount -1)  cout<<endl;
14)     else
15)      cout<<" , ";}}
```

- السطر الأول حتى الثالث: تقوم كافة المهام ضمن المجموعة بحجز مصفوفة جديدة بحجم الـ Block المراد ارساله، ويتم تعبئة المصفوفة send بالقيم المستلمة سابقاً من عملية الـ Scatter بعد جدائها بالرقم 2.

- السطر الخامس حتى التاسع: التابع **pvm_gather()** والذي سيقوم بتجميع البيانات من كل مهمة في المجموعة ضمن مصفوفة في الجذر، حيث مدخلات التابع هي:
  - gatherData: عنوان مصفوفة الاستلام التي سيتم تخزين البيانات المستلمة بها، ويمرر فقط ضمن المهمة الجذر المسؤولة عن حجز المصفوفة، أما في باقي المهام (non-Root) يكون Null.
  - send: عنوان المصفوفة التي ستقوم كل مهمة بإرسالها إلى الجذر.
  - childblock: حجم الكتلة المحدد ارسالها لكل مهمة.
  - groupName: اسم المجموعة التي سيتم عمل Gather ضمنها.
  - allRoot: جذر المجموعة (قيمة الـ gid).

Code:

Finally. Reduce The Array Values using SUM Function.

```
//Reduce Operation
    1) pvm_reduce(PvmSum, send, childBlock, PVM_INT,4, groupName, allRoot);
    2) if(isParent) {
    3)      cout<<"\nReduction SUM Operator...Final Data\n";
    4)      for(int i = 0; i<childBlock; i++) {
    5)         cout<<send[i];
    6)         if(i==childBlock -1) cout<<endl;
    7)         else cout<<" , "; }
    8)    endTime = clock();
    9)    double elapsedTime = (double)(endTime - startTime) /    CLOCKS_PER_SEC;
   10)   cout<<"\nElapsed Time = "<<elapsedTime<<endl;   }
   11) pvm_barrier(groupName, groupSize);
   12) pvm_lvgroup(groupName);
   13) pvm_exit();}
```

- السطر الأول: التابع **pvm_reduce()**، يستدعى خارج if, else من أجل جميع المهام، مدخلات التابع هي:
  - PvmSum: التابع المراد تنفيذه على المصفوفات.
  - send: عنوان المصفوفة التي سيتم تطبيق العملية عليها، حيث سيتم اسناد ناتج العملية إلى المصفوفة send ضمن الجذر.

  - groupName: اسم المجموعة التي سيتم عمل Reduce ضمنها.
  - allRoot: جذر المجموعة (قيمة الـ gid).

- السطر الثامن: يتم تخزين الزمن الحالي ضمن المتحول time end ليعبر عن زمن نهاية البرنامج.

- السطر التاسع: حساب زمن تنفيذ البرنامج.

- السطر 11: حجز مهام المجموعة لضمان عدم مغادرة أي مهمة للمجموعة قبل انتهاء جميع المهام من تنفيذ كافة العمليات ثم يتم مغادرة المجموعة.


to be continued