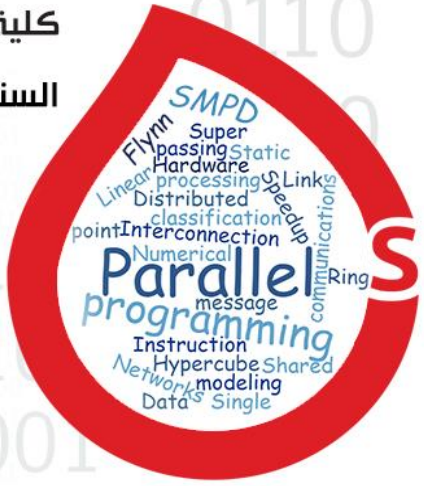
عنوان المحاضرة



م. عمار المصري

09/03/2024

RB Informatics;

البرمجة التفرعية

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 إنشاء Group حيث سيتم إضافة مجموعة من المهام إليه كل مهمة تملك Tid والتي تمثل ال id الخاص بها وعند دخولها إلى المجموعة ستعطى رقما مميزا Gid داخل هذه المجموعة هذا الرقم يعبر عن موقع هذه المهمة ضمن المجموعة وهو يبدأ من الصفر ويزداد ن الصفر ويزداد بشكل تصاعدي وعند مغادرة المهمة لهذه المجموعة سيصبح هذا ال gid فارغا وسيتم إسناده لأول مهمة جديدة تقوم بالدخول إلى المجموعة
- لكل مجموعة يوجد Root Task تكون مسؤولة عن كافة عمليات المجموعة (وليس بالضرورة أن تكون المهمة الأب)

لكل مجموعة يوجد Root task تكون مسؤولة عن كافة عمليات المجموعة (وليس بالضرورة أن تكون المهمة الأب) إذ أنه من الممكن عدم تواجد الأب في المجموعة من الأساس
لدينا شرطين أساسين لإنشاء أي مجموعة:

- أي task من الممكن أن تنتمي لأي مجموعة وليس شرطا أن يكون الأب متواجد ضمن المجموعة
- أي مجموعة لديها root وال rootليس شرطا أن يكون الأب

1. لنفترض في الحالة هذه أن الأب = root
2. ولدينا بينات معينة موجودة broad cast(one to all) ضمن root أريد إرسالها لجميع الأبناء الموجودة ضمن لمجموعة
3. هي رسالة من root لعنصر أو عنصرين Multi casrt (one to many)أو 3 عناصر من عناصر المجموعة هي عملية ليست تجميعية (العملية التجميعية هي عملية مرسلة لجميع أعضاء المجموعة)

Gather function (all to one)

رسالة من جميع أعضاء المجموعة إلى root لنفترض أن root هو $s1$ حيث لدينا المصفوفة التالية لأعضاء المجموعة

$f, s1, s2, s3, s4 \leftarrow$

$S4$الابن الرابع

$S1$ الابن الثالث

$S2$ الابن الثاني

$S3$الابن الأول

f الأب

ومن الممكن لكل عملية تحديد root مختلف

نقوم باستخدام تابع ال gather لإرسال مجموعة من القيم من جميع أعضاء المجموعة إلى root

كيف يتم استقبال البيانات من قبل ال root؟

- يقوم بتجهيز مصفوفة بشرط حجم المصفوفة مساويا لعدد الأعضاء ضمن المجموعة
- عندما يقوم root باستلام رسالة من S3 مثلا (بفرض أن tasks تعمل على التوازي وبالتالي من الممكن أن تكون task أسرع من أخرى يقوم بوضع هذه القيمة ضمن موقع المصفوفة المساوي لموقع العنصر ضمن المجموعة (موقع العنصر = موقع المهمة)

أي في مثالنا

$f, s1, s2, s3, s4$

بفرض تم استلام الرسالة من s3 يقوم ال root=s1 بوضع القيمة في العنصر الرابع والمساوي لموقع العنصر S3 ضمن المجموعة

العملية المعاكسة لل gather

Scatter (one to all)

- لدينا مصفوفة في ال root سيقوم بتوزيع عناصر هذه المصفوفة على tasks في المجموعة (عناصر المجموعة)
- كل task سوف تستلم رسالة (قيمة ضمن المصفوفة بما يناسب بموقعها مثلا القيمة الثالثة ستذهب ل S2 ضمن مثالنا القيمة الرابعة ستذهب ل s3 وهكذا

Reduce (all to one)

مثل gather ولكن بفرق واحد

- Reduce نقوم بتعريف عملية حسابية واحدة

لنفترض أن root=s2 لأعرّف ؟؟ عملية حسابية ولكن الضرب جميع المهام (العناصر) الموجودة ضمن المجموعة سترسل قيمة معينة للroot ف مثلا $f, s1, s2, s3, s4$ قاموا بإرسال قيمة معينة (ال S2 موجود لديه قمة بالأساس فال root يشارك بالمهام إذ يقوم بإعادة توجيهها لنفسه) سيقوم ال S2 بضرب هذه العناصر ببعضها وسيقوم بطباعة الخرج عن طريق تخزينه بمتحول معين

لندخل بتفاصيل إنشاء المجموعة والتوابع التجميعية وكيفية استخدامها في PVM وكيفية توزيع المهام بين أعضاء المجموعة

سنتحدث عن 6 عمليات collective operation في ال PVM

خمسة منهم عمليات تجميعية

Group Function

pvm_joingroup(char*groupName)

- **تابع الانضمام لمجموعة يحدد اسم المجموعة ضمن متحول الدخل الوحيد إنشاء المجموعة يتم من خلال التابع ذاته عند استدعائه من أجل مجموعة غير موجودة سابقا عندها تنشأ المجموعة الجديدة بالاسم وتضاف المهمة التي قامت باستدعاء التابع كعنصر أول ضمن المجموعة وتحصل على gid=0 يمكن للمهمة أن تضم إلى أكثر من المجموعة وتحصل على gid خاص لها ضمن كل مجموعة**

Pvm_ivgroup(char*groupNAme,int taskID) ↙

تابع للحصول على ال gid لمهمة ما ضمن مجموعة حيث يتم تحديد اسم المجموعة وال Task ID للمهمة كمتحولات للدلخ

Pvm_gettid(char*groupName,int GID) ↙

تابع للحصول على ال tid لمهمة ما ضمن مجموعة حيث يتم تحديد اسم المجموعة وال gid للمهمة كمتحولات للدخل

Collective Operation:

MultiCAst:

pvm_mcast(childid, intCount, int msgTag)

- **لا يصنف من التوابع التجميعية يقوم بإرسال رسالة من مهمة ما لعدة مهام (ont to many) يقوم بتحديد مصفوفة من ال Task Id's تعبر عن المهام التي سيقوم بالإرسال إليها ثم يقوم بتحديد عدد المهام التي ستستلم الرسالة المراد إرسالها 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()

Pvm-joingroup(char*groupNAme)

خلاصة:

- **التابع يأخذ بارامتر واحد وهو اسم المجموعة**
- **إذا تم استدعاء التابع ولم يكن اسم المجموعة موجود سابقا وسيتم إنشاء المجموعة وإعطائها الاسم الجديد**

Pvm_gsize()

- **تابع لمعرفة حجم المجموعة (يرجع عدد المهام الموجودة ضمن المجمعة "قيمة طبيعية")**

PVM_LVgroup(char*groupName)

PVM_getinst(char*groupName, int TaskID)

(group instance)

خلاصة :

يقوم بإرجاع ال gid(groupid) الخاص بمجموعة معينة بناء على اسم group و taskID لمهمة معينة

يستفاد منه بمعرفة ال root الخاص بالعمليات أو المسؤول عن العمليات وإدارتها والتي سيتم تطبيقها ضمن البرنامج الحالي

ملاحظة:

الفرق بين

Pvm_mytid : يعطي task id خاص ب task معينة

Pvm_gettid: يعطي task id خاص ب task معينة ولكن بناء على اسم group و Gid(Group id)

Collective operation

التوابع التجميعية والهدف منها أن يتشارك جميع أعضاء المجموعة ومن ضمنهم ال root في مهمة معينة

PVM_bcast(char *groupName, int msgtag)

- **يستخدم لتنفذ BroadCastoperation**
- **يتميز التابع بأنا لسنا بحاجة لمعرفة root (بالتأكيد المجموعة لها root لكن ليس من الضروري إرسال رسالة Broadcast من قبل root)**
- **هي المهمة الوحيدة التي من الممكن أن يكون مرسل الرسالة خارج المجموعة وليس بالضرورة أن يكون لدينا معرفة بال root**

عندما كنا نقوم بإرسال الرسالة one to one

- **تهيئة الرسالة**
- **تحزيم الرسالة**
- **إرسال الرسالة عن طريق** Pvm_send()

الان سيقوم مرسل الرسالة ont to all **ب**

- **تهيئة الرسالة**
- **تحزيم الرسالة**
- **إرسال الرسالة عن طريق** Pvm_bcast()

حيث يتم استلام الرسالة باستخدام التابع PVM_recv()

وبنفس آلية الاستقبال المعتادة تتم قراءة المعلومات من ال buffer ومن ثم عملية فك التحزيم والاستفادة من القيمة التي تم استقبالها

فقط سيتم استبدال Pvm_bcast() ↔ PVM_recv()

حيث سيتم العمل على نفس توابع الإرسال والاستقبال التي تم العمل عليها الجلسات الماضية

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)

Scatter

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

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

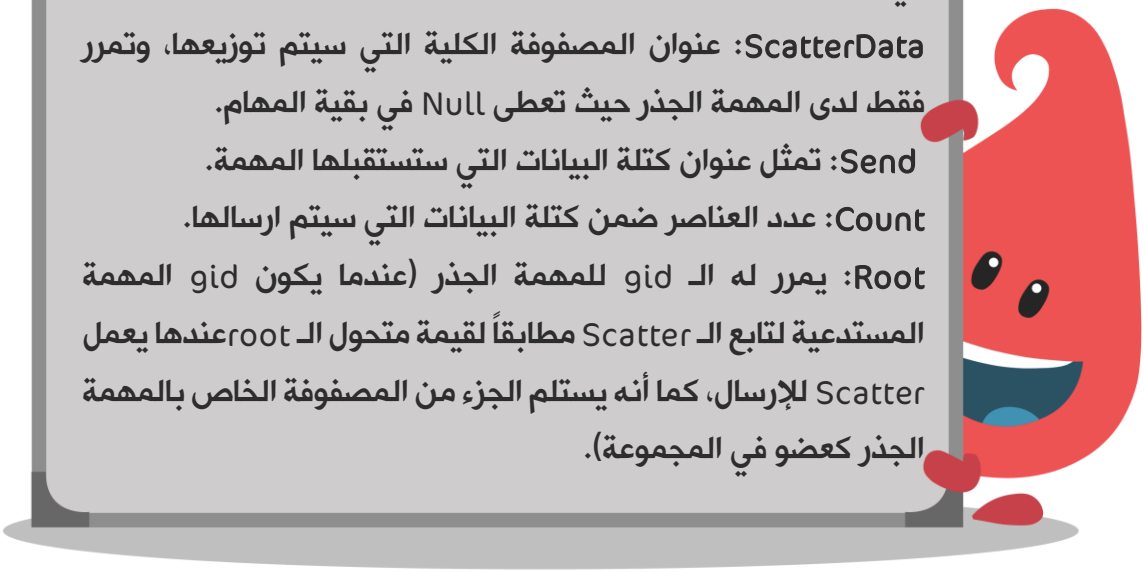
حيث:

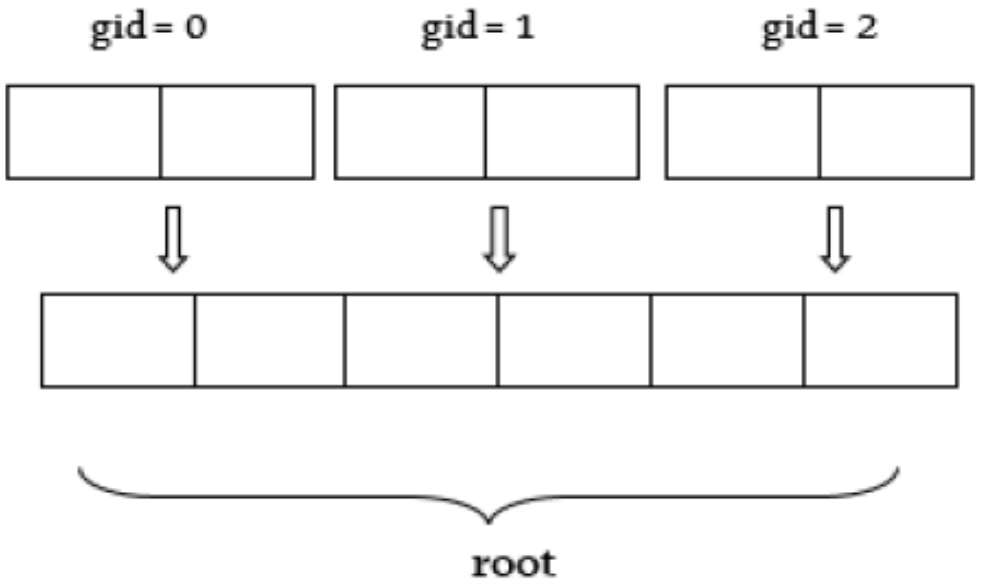
ScatterData: عنوان المصفوفة الكلية التي سيتم توزيعها، وتمرر فقط لدى المهمة الجذر حيث تعطى Null في بقية المهام.

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 rootعندها يعمل Scatter للإرسال، كما أنه يستلم الجزء من المصفوفة الخاص بالمهمة الجذر كعضو في المجموعة).





خلاصة:

ضمن المجموعة ستستلم قيمة من الموقع ضمن المصفوفة المساوي لموقعها ضمن المجموعة

بفرض لدينا $f, s1, s2, s3$

فالمهمة $s3$ ستستلم القيمة 4

يقوم ال root بتوزيع مهام المجموعة ويقوم بإرسال رسالة ont to All إلا أنه مشارك في هذه المهمة فهو يقوم بجزء من المهمة (يقوم باستلام قيمة من نفسه مثل باقي الأعضاء)

الآلية تختلف قليلا عن BroadCast

سنستخدم بعملية الإرسال التابع Pvm_scatter وبعملية الاستقبال التابع Pvm_scatter "بفرق بسيط بالبارامترات

- عند root كمسؤول عن المصفوفة أول بارامتر هو مصفوفة القيم التي سيتم على أساسها استلام القيمة التي سيتم إرسالها
- في مثالنا سيتم استقبال قيمة واحدة as(non_root, root) سنخزنها في متحلو int (القيمة التي سيتم استلامها من ال root
- البارامتر root هو Grid لل root

non_root مماثل البارامتر عند root بفرق بسيط هو ال root من يقوم بتوزيع المصفوفة بالتالي ضمن تابع root سيتم كتابة اسم المصفوفة

ضمن non_root سيتم ووضع null (لأنه لا يوجد مصفوفة ليتم توزيعها)

ملاحظة:

عندما يقوم root باستدعاء Pvm_scatter يكون gid الخاص بال root نفسه gid الخاص بالتابع

أما عندما يقوم أي عضو آخر باستدعاء التابع gid لهذا العضو لا يكون مساويا ل gid الخاص بالتابع

والهدف كالتالي: تتم مقارنة gid التابع مع gid العضو الذ استدعى التابع بهدف معرف ال root

- إذا كانا متساويين فهو root
- غير متساويين ليس root

Broadcast	Scatter
يتم إرسال قيمة واحدة لكل أعضاء المجموعة	يتم إرسال مجموعة من القيم وكل عضو يأخذ القيمة التي تناسبه (بآلية ذكرناها سابقا)

Gather

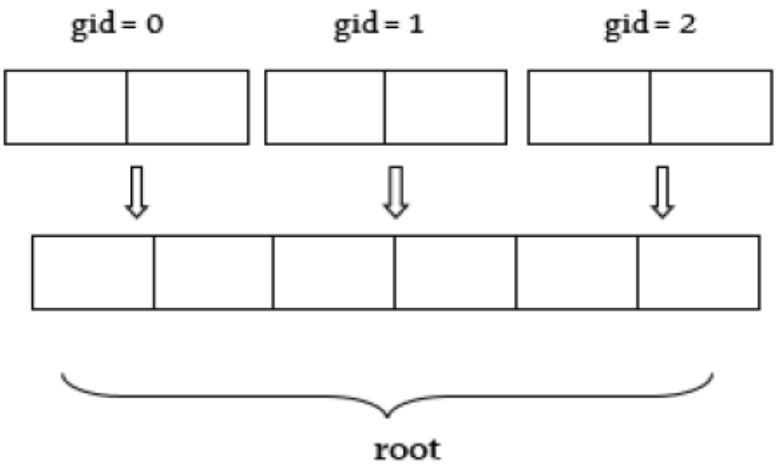
- لدينا مجموعة من البيانات موزعة بين المهام، نقوم بتجميع هذه البيانات حيث كل مهمة ضمن المجموعة تجهز Block of Data وتقوم بإرساله، لتقوم المهمة الجذر بعملية استلام لهذه البيانات وتجميع الكتل الواردة ضمن مصفوفة بالتتالي حسب الـ gid للمهمة المرسلة.
- الإرسال لدى المهام الـ non-Root يتم عبر التابع :

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

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)
```



gathter(all to one)

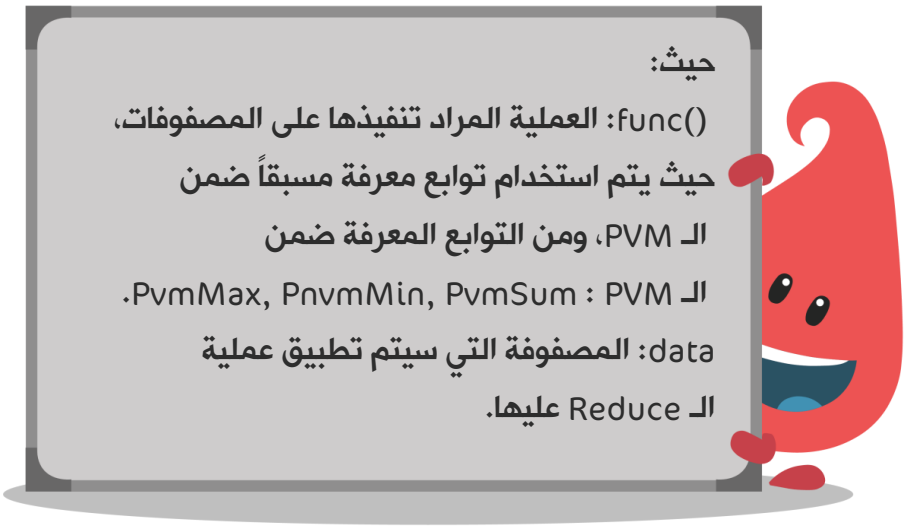
(عملية معاكسة ل Scatter)

- بعملتي الإرسال والاستقبال سيتم استخدام التابع pvm_gather(.....)
- كل task لديها مصفوفة من قيم وسيتم تجميع هذه القيم ضمن root معين
- حيث يتم تهيئة المصفوفة وتخزين القيم التي سيتم استقبالها في هذه المصفوفة
- كل مهمة ستقوم بتخزين القمة الخاصة بها ضمن موقع في المصفوفة مساويgid الخاص بها بالمجموعة
- يتم استخدام التابع pvm_gather من قبل

- root الذي يقوم بتجميع المصفوفة النهائية عنده ولا نفس أنه مشارك ولديه قيمة
- Non_root ذكرنا أن المصفوفة التي يتم تجميع القيم بها هي لدى root هنا ستكون null (بارامتر المصفوفة سيكون null)

- 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:

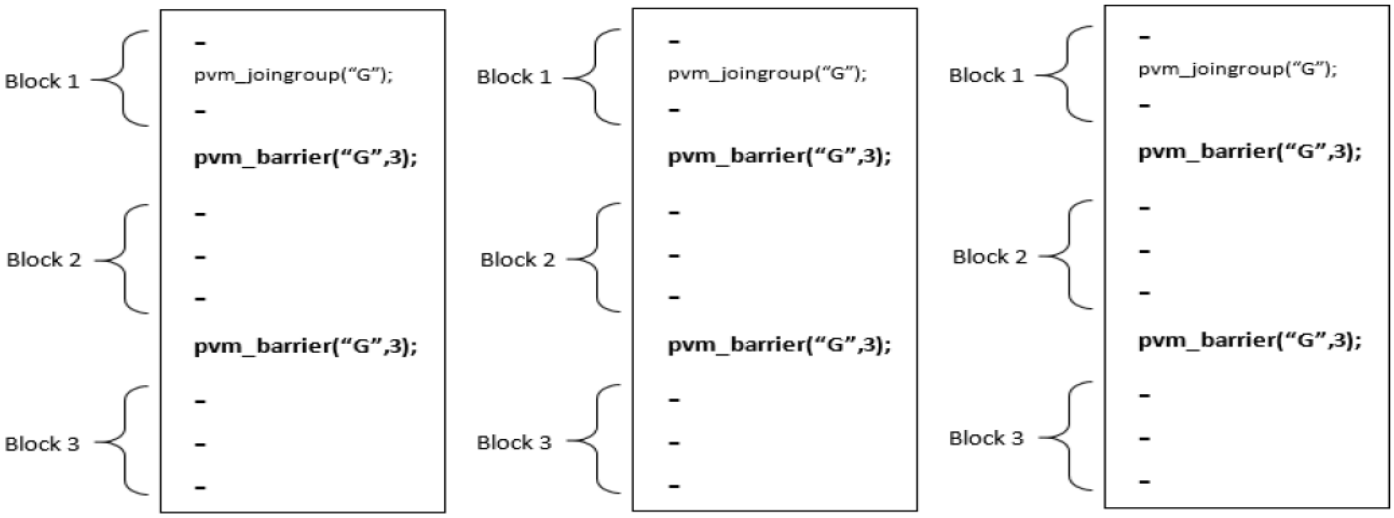
pvm_reduce(void (*func)(), void *data, int count, int datatype, int msgtag, char *group, int root)



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.

Reduce(all to one)

- عملية معينة على البيانات بعد ما يتم استلامها
- تابع ينفذ مرة واحدة بالقسم المشترك من الكود أي بآلية عودية
- لنفترض أن العملية الحسابية التي سيتم تطبيقها هي sum (البارامتر الأول) (البارامتر الثاني هو المتحول الذي سيتم إرساله)

سنقوم بعملية حسابية حيث أن الخرج موجود ضمن ال root

عند طباعة القيم جميع القيم ستبقى نفسها عدا ال root سيحمل قيمة الخرج وهي ناتج مجموع القيم

مثلا: لدينا مصفوفة القيم

$\{3,5,7,6,5 \leftarrow root\}$

$\{3,5,7,6,26 \leftarrow root\}$

ملاحظة:

ويدخل ال root في العملية الحسابية كما في gather,scatter

Pvm_barrier

- عبارة عن نقطة مزامنة
- حيث كل مهمة تعمل على معالج مستقل
- ليس عبارة عن تابع تجميعي إنما هو عبارة عن تابع انتظار

ملاحظة:

شرط جميع المصفوفات أن يكون حجم المصفوفات متساوي بفرض لدينا عنصرين بالابن s

3 عناصر بالابن s1 تفشل عملية الجميع

تمرين

سيتم استخدام fork (إنشاء الابن والأبناء ضمن نفس الكود "آلية عودية")

- انطلاقا من الأب سيتم إنشاء الأبناء عن طريق التابع pvm_spwan()
- إضافة الأبناء إلى المجموعة (عدد الأبناء -3)

- الخطوة الأولى

وهي إنشاء الأبناء وإضافتهم للمجموعة

سنفترض أن الأب root في هذا التمرين سنكون ضمن ال root مصفوفة سيتم عمل scatter لها (مصفوفة منها مصفوفات القيم)

سيتم استلام مصفوفة من ثلاث قيم

- بالطلب الثاني:

سيكون ضمن root مصفوفة من القيم سيتم توزيعها على الأبناء

- بالطلب الثالث:

ستتم مضاعفة كل قيمة من المصفوفة بعد استلامها من ال root (الأبناء هم من سيضاعفون القيمة)

- بالطلب الرابع:

سيتم عمل gather ضمن root

- بالطلب الخامس:

سيتم عمل reduce ضمن root

ملاحظة:

ال root هو المسؤول عن إدارة العملية وبكل عملية يمكن تحديد root مختلف

```
void main(int argc, char * argv[])
{
clock_t startTime, endTime;
char * groupName = "all";
int myId = pvm_mytid();
int myParent = pvm_parent();
bool isParent = (myParent == PvmNoParent) || (myParent ==
= PvmParentNotSet);
int childCount, childBlock;
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.

```
if(isParent)
{//Parent Mode
CopyFile("Debug\\collect.exe","C:\\pvm\\bin\\WIN32\\collect.exe",false);
startTime = clock();
cout << "Master ID = " << myId << endl;
//if(argc > 1) childCount = atoi(argv[1]);
//else
childCount = 3;
//if(argc > 2) childBlock = atoi(argv[2]);
//else
childBlock = 2;
//pvm_catchout(stdout);
childId = new int[childCount];
pvm_catchout(stdout);
int cc = pvm_spawn("collect",NULL,0,"",childCount,childId);
if(cc != childCount)
{
cout << "\nFaild to spwan required children\n...Exit...Press any Key to exit\n";
pvm_exit();
int c; cin >> c;
exit(-1);
}
pvm_initsend(PvmDataDefault);
pvm_pkint(&childBlock,1,1);
pvm_pkint(&childCount,1,1);
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.

```
else
{
pvm_recv(myParent, 1);
pvm_upkint(&childBlock, 1,1);
pvm_upkint(&childCount, 1,1);
cout << "Data Received...\n";
}
```

Scatter The Array

```
int myGroupId = pvm_joingroup(groupName);
pvm_barrier(groupName, childCount + 1);
int groupSize = pvm_gsize(groupName);
int allRoot;
if(isParent)
allRoot = pvm_getinst(groupName, myId);
else
allRoot = pvm_getinst(groupName, myParent);
int * rec = new int[childBlock];
if(isParent)
{
//Prepare Data to be sent
int scatterCount = groupSize * childBlock;
int * scatterData = new int[scatterCount];
for(int i = 0; i < scatterCount; i++) scatterData[i] = i + 1;

pvm_scatter(rec, scatterData, childBlock, PVM_INT, 2, groupName, allRoot);
}
else //Not Root for Scatter...Then I Recieve
pvm_scatter(rec, NULL, childBlock, PVM_INT, 2, 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).

Double The Array Values, then Gather it.

$$int * send = new\ int[childBlock];$$

$$for\ (int\ i = 0;\ i < childBlock\ ;\ i++)$$

$$send[i] = rec[i] * 2;$$

$$if(!isParent)$$

$$pvm\_gather(NULL, send, childBlock, PVM\_INT, 3, groupName, allRoot);$$

$$else$$

$$\{$$

$$int\ gatherCount = groupSize * childBlock;$$

```
int * gatherData = new int[gatherCount];
pvm_gather(gatherData, send, childBlock, PVM_INT, 3, groupName, allRoot);
//Print Gathered Data
cout << "\nGathered Data...\n";
for(int i = 0; i < gatherCount; i++)
{
cout << gatherData[i];
if(i == gatherCount - 1)
cout << endl;
else
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).

Reduce The Array Values using SUM Function.

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