ضرب المصفوفات

م.عمار المصري

محتوى مجاني غير مخصص للبيع التجاري

مقدمة

تعلمنا في المحاضرة السابقة طريقة تشغيل الـ PVM وكيفية تحقيق عمليات الإرسال والاستقبال بين المهمة الأب والمهمة الابن .

- في هذه المحاضرة سنتعلم بناء برنامج تفرعي لجداء مصفوفتين وبالتالي نكون بحاجة إلى مهمة أب ومجموعة من الأبناء حيث كل task سواء كان أب او ابن سيقوم بعملية إرسال واستقبال لتحقيق جداء المصفوفات .

تذكرة :

من أجل ايجاد ناتج ضرب مصفوفتين يجب أن يتحقق الشرط :

عدد الأعمدة في المصفوفة الأولى = عدد الأسطر في المصفوفة الثانية .

✓ لتكن :

$$A=\begin{matrix}\overbrace{\begin{matrix}a_{11} & a_{12} & a_{13}\end{matrix}}\\ \begin{matrix}a_{21} & a_{22} & a_{23}\end{matrix}\end{matrix}\quad , \quad B=\begin{Bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{Bmatrix}$$

فإن :

$$A*B=\begin{Bmatrix} (a_{11}*b_{11}+a_{12}*b_{21}+a_{13}*b_{31}) & (a_{11}*b_{12}+a_{12}*b_{22}+a_{13}*b_{32}) \\ (a_{21}*b_{11}+a_{22}*b_{21}+a_{23}*b_{31}) & (a_{21}*b_{12}+a_{22}*b_{22}+a_{23}*b_{32}) \end{Bmatrix}$$

- كيف سنمثل ذلك تفرعياً ؟

- سنقوم بإعطاء كل ابن سطر وعمود، حيث يقوم كل ابن بإيجاد ناتج جداء السطر والعمود المرسلين له ويرد الناتج للأب، حيث لدينا 4 أبناء:

- المهمة الأولى (السطر الأول والعمود الأول).
- المهمة الثانية (السطر الأول والعمود الثاني).
- المهمة الثالثة (السطر الثاني و العمود الأول).
- المهمة الرابعة (السطر الثاني والعمود الثاني).

➢ عندما يتم إرسال سطر وعمود للابن ، لا يعلم أي سطر وعمود قد استلم ، إنما يستلم مصفوفتين أحاديتين عليه ضربهما ، حيث الأبناء الأربعة قد يعملون على 4 أجهزة مختلفة ويمكن أن ينتهي الابن الرابع أولاً قبل الابن الأول (حيث كل ابن من الأبناء لا يعلم شيئاً عن الآخر) ، فيتوجب علينا أن نعرف كل قيمة أين ستوضع في مكانها الصحيح ضمن مصفوفة الناتج ، لذا نلجأ للاستعمال الـ tag حيث بناءً على قيمته سيتم وضع الناتج المُرسل من قبل المهمة الابن ضمن الموقع الصحيح .

- في مثالنا سنقرأ أبعاد المصفوفات من ملف موجود على القرص C ، و سوف تكون به الأرقام التي ستتم ضربها حيث أبعاد أول مصفوفة 2x3 والثانية 3x2 فتكون مصفوفة الناتج ذات أبعاد 2x2

1. التابع read_matrix :

- وهو تابع قراءة المصفوفة ، حيث نقوم بتمرير مؤشر المصفوفة وأبعادها ، وستتم قراءة عناصر الدخل من الملف النصي c:\m.txt

```
void read_matrix(int* T, int d1, int d2)
{
    int i;
    for (i=0; i<d1*d2; i++)
        scanf("%d", &T[i]);
}
```

2. التابع print_matrix :

يقوم بطباعة المصفوفة ، مررنا له المصفوفة وأبعادها إضافةً لاسمها، والمعادلة [i*d2+j] تعبر عن معادلة الانتقال بين عناصر المصفوفة، حيث يتم استخدامها للمرور على المصفوفة الثنائية كأنها أحادية.

```
void print_matrix(int* T, int d1, int d2, const char *name)
{
    int i, j;
    printf("Matrix %s %ix%i\n", name, d1, d2);
    for (i=0; i<d1; i++, printf("\n"))
        for (j=0; j<d2; j++)
            printf("%3d", T[i*d2+j]);
    printf("\n");
}
```

- **HELLO Code:**

```
int main(int argc, char* argv[])
{
    int n_child, pr;
    int i, j, k;
    int info, bufid, bytes, tag, tid;
    int n, m, m1, m2, p;
    int s;
    int *childID;
    int *A, *B, *C;

    freopen("c:\\m.txt", "r", stdin);
    scanf("%d %d", &n, &m1);
    A = (int*) malloc(n * m1 * sizeof(int));
    //creat_matrix(A, n, m1);
    read_matrix(A, n, m1);
    scanf("%d %d", &m2, &p);
    if (m1 != m2) {
        printf("error in matrix dimension\n");
        return 1;
    }
    B = (int*) malloc(m2 * p * sizeof(int));
    //creat_matrix(&B, m2, p);
    read_matrix(B, m2, p);

    C = (int*) malloc(n * p * sizeof(int));
    //creat_matrix(&C, n, p);
    m = m1 & m2;
    pr = n * p;
```

3. في التابع main :

سيتم تعريف العديد من المتحولات التي سيتم استخدامها ضمن سياق البرنامج، أبرزها:

- **عدد الأولاد** n_child .
- **عدد المهمات** pr .
- **مصفوفة الأبناء** childID .

- سيتم استخدام التعليمة freopen لفتح الملف الملف النصي m.txt المتواجد ضمن المسار المذكور ضمن البارامتر الأول والقراءة منه.
- التعليمة scanf ستقوم بقراءة أول قيمتين ضمن الملف وتخزينهما كأبعاد للمصفوفة ضمن المتحولين n , m1 .
- سيتم استخدام التعليمة malloc لأجل حجز مصفوفة من النمط int بحجم dimension*sizeof(int) حيث سيتم حجز المصفوفتين A & B ، إضافة للمصفوفة C والتي ستمثل المصفوفة الناتجة عن جداء المصفوفتين السابقتين.
- سيتم التحقق من شرط ضرب المصفوفتين A & B ، فإذا كان الشرط صحيح نتابع العمل، و إلا سيتم طباعة رسالة خطأ ومغادرة البرنامج.
- عدد الأبناء التي سيتم إنشاؤها pr، وهو ناتج جداء أسطر المصفوفة الأولى بأعمدة المصفوفة الثانية.

تابع الـ freopen له ثلاث بارامترات :

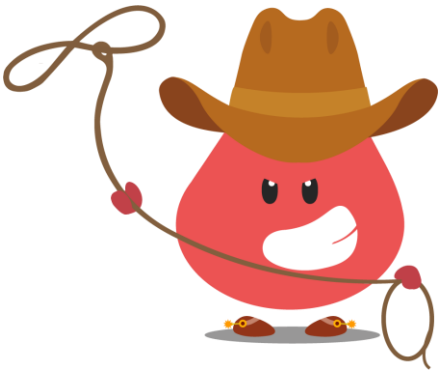
مسار الملف / تحديد اذا كان للقراءة او الكتابة او كلاهما / تحديد الملف الأساسي للقراءة stdin .

```
childID = (int*) malloc(pr * sizeof(int));

print_matrix(A, n, m, "A");
print_matrix(B, m, p, "B");

pvm_mytid();
n_child = pvm_spawn("hello_other", (char**)0, 0, "", pr, childID);

if (n_child == pr) {
    for (i=0; i<n; i++) {
        for (j=0; j<p; j++){
            pvm_initsend(PvmDataDefault);
            pvm_pkint(&m, 1, 1);
            pvm_pkint(A+i*m, m, 1);
            pvm_pkint(B+j, m, p);
            pvm_send(childID[i*p+j], i*p+j);
        }
    }
    for (k=0; k<pr; k++) {
        bufid = pvm_recv(-1, -1);
        info = pvm_bufinfo(bufid, &bytes, &tag, &tid);
        pvm_upkint(&s, 1, 1);
        C[tag] = s;
    }
```



- سيتم استخدام التعليمة malloc لأجل حجز المصفوفة childID ، والتي سيتم بها تخزين الـ tid's للمهام المنشاة بواسطة الأب.

4. التابع pvm_spawn :

سيقوم بإنشاء الأبناء، حيث سيتم إنشاء عدد من الأبناء pr وتمرير المصفوفة childID والتي سيتم بها تخزين النتائج ، ثم سيتم تخزين عدد الأبناء المنشأة بنجاح ضمن المتحول n_child .

- اذا ما تمت عملية إنشاء 4 أبناء بنجاح، سيتم إرسال سطر وعمود إلى كل مهمة حتى تقوم بتنفيذ عملية الجداء، ثم سيتم استقبال النتيجة من كل مهمة.
- ستتم عملية تهيئة الإرسال عبر التابع ()pvm_ initsend.

5. التابع ()pvm_pkint :

لتحزيم القيم التي سيتم إرسالها إلى الابن، حيث بارامترات هذه التابع هي:

1. البارامتر الأول يمثل العنصر / العناصر التي سوف يتم إرسالها .
2. البارامتر الثاني يعبر عن عدد القيم التي سيتم إرسالها .
3. البارامتر الثالث يعبر عن مقدار القفزة .

- **ضمن عملية التحزيم الأولى يقوم الأب بإرسال القيمة m والتي تمثل بُعد المصفوفات التي نريد ضربها (يتم إرسال هذه القيمة لإعلام الابن بأبعاد المصفوفات التي سيقوم بجدائها حتى يقوم بحجزها) ، عدد القيم يساوي الواحد =1 ، ومقدار القفزة =1 .**

- ضمن عملية التحزيم الثانية سيتم تحزيم الأسطر من المصفوفة الأولى A ، حيث سيتم التحزيم وفق المعادلة [A+i*m] ، حيث m هي عدد القيم المُرسلة ، ومقدار القفزة يساوي الواحد حيث سيتم إرسال القيم ضمن كل سطر والمرور عليها بمقدار قفزة يساوي الواحد .

- ضمن عملية التحزيم الثالثة سيتم تحزيم الأعمدة من المصفوفة B ، حيث سيتم التحزيم وفق المعادلة (B+j) حيث m هي عدد القيم المرسلة ، ومقدار القفزة يساوي عدد الأعمدة حيث سيتم إرسال القيم ضمن كل عمود والمرور عليها بمقدار قفزة يساوي الـ p (عدد الأعمدة ضمن المصفوفة B).

6. التابع pvm_send :

وهو تابع الإرسال ، سيتم فيه تحديد البارامتر الأول وهو id المستقبل الذي سيتم الإرسال إليه ، حيث يتم إرسال السطر والعمود إلى الابن وفقاً للمعادلة [i*p+j] ، البارامتر الثاني يتم به تحديد الـ tag الخاص بالرسالة ، ويتم إرسال الـ tag مع الرسالة بغرض ألا وهو : أنه عندما يتم إرسال نتيجة جداء من الابن للأب سيتم معرفة أين يتم وضع القيمة ضمن أي موقع في مصفوفة النتائج .

- بعد ذلك يتم تشغيل 4 ملفات hello_other.exe ، حيث سنرسل لكل مهمة سطر وعمود مع الـ tag المناسب وبعد إتمام عملية الجداء من قبل كل مهمة سيرسل الناتج الى الأب مع الـ tag ذاته .

7. التابع pvm_recv :

ضمن التابع pvm_recv يتم وضع القيم (-1 , -1) ، حيث سيتم الاستلام من أي ابن أنهى العمل أولاً لغرض توفير الوقت .

8. التابع pvm_bufinfo :

لقراءة معلومات اللصاقة " Buffer " .

9. التابع pvm_upkint :

تابع فك التحزيم ، حيث هذا التابع سيأخذ ثلاث بارامترات :

1. العنصر / العناصر التي سوف يتم استقبالها (حيث القيمة s تمثل ناتج جداء السطر والعمود) .
2. عدد القيم التي سيتم استقبالها .
3. مقدار القفزة .

- أخيراً يتم وضع الناتج في الخانة المناسبة ضمن مصفوفة الناتج C حسب الـ tag وطباعة المصفوفة الناتجة C .

- **HELLO_OTHER Code:**

```
int *A, *B;
int i, s, m;
int mytid, master;
int info, bufid, tag;

//mytid = pvm_mytid();
master = pvm_parent();

bufid = pvm_recv(master, -1);
info= pvm_bufinfo(bufid, 0, &tag, 0);
pvm_upkint(&m, 1, 1);

A = (int*) malloc(m * sizeof(int));
B = (int*) malloc(m * sizeof(int));

pvm_upkint(A, m, 1);
pvm_upkint(B, m, 1);

s = 0;
for (i=0; i<m; i++)
    s += A[i] * B[i];

pvm_initsend(PvmDataDefault);
pvm_pkint(&s, 1, 1);
pvm_send(master, tag);

pvm_exit();
return 0;
```

10. التابع pvm_parent() :

يقوم بتسجيل الـ PVM Task ، حيث يرد الـ id الخاص بالأب للمهمة الحالية وتخزينه ضمن المتحول master .

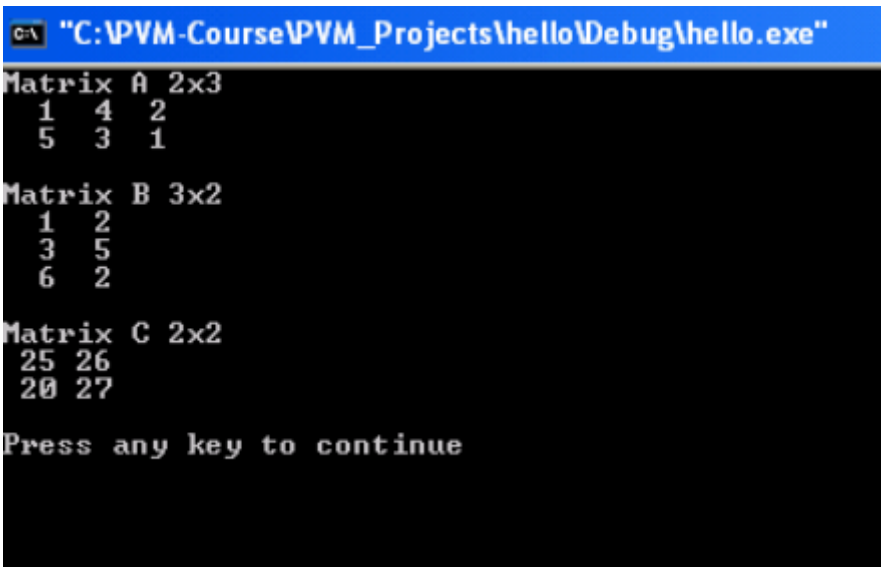
- التابع pvm_recv(master , -1) يقوم باستقبال رسالة من المهمة الأب أياً كانت قيمة الـ tag الخاصة بهذه الرسالة .

- التابع pvm_bufinfo() يقوم بقراءة معلومات اللصاقة .

- تابع فك التحزيم pvm_upkint يقوم بفك تحزيم الرسائل المرسلة من الأب بذات الترتيب ، حيث بعملية فك التحزيم الأولى يتم قراءة قيمة m والتي تمثل عدد العناصر التي سيتم استلامها ضمن عمليات فك التحزيم التالية .

- سيتم استخدام القيمة m لأجل حجز المصفوفتين A & B بحجم m*sizeof(int) والتي سيتم بهما تخزين السطر والعمود المرسلين من قبل المهمة الأب .
- فك تحزيم A ثم B مع ملاحظة أن مقدار القفزة يساوي الواحد ، لأن المصفوفتين اللتين سيتم استلامهما أحاديتين ، ولأن عملية القفز تمت في الأب فتصل للابن جاهزة لعملية الضرب مع الأخرى .

- S هو المتحول الذي سيتم به تخزين ناتج عملية جداء المصفوفتين .
- عملية تهيئة الإرسال عبر التابع pvm_initsend() .
- استخدام التابع pvm_pknit() لتحزيم القيمة التي سيتم إرسالها إلى الأب .
- التابع pvm_send وهو تابع الإرسال ، البارامتر الأول هو الـ id الخاص بالأب حيث سيتم إرسال الرسالة مع الـ tag ذاته الذي تم استلامه من الأب وذلك ليتم معرفة أين سيتم وضع القيمة ضمن أي موقع في مصفوفة الناتج .



يقول نيتشه :

" من يملك سبباً للحياة ، يستطيع تحمّل أي شيء ".

-انتهت المحاضرة -