البرمجة المتوازية

OpenMP && CUDA

تقديم:

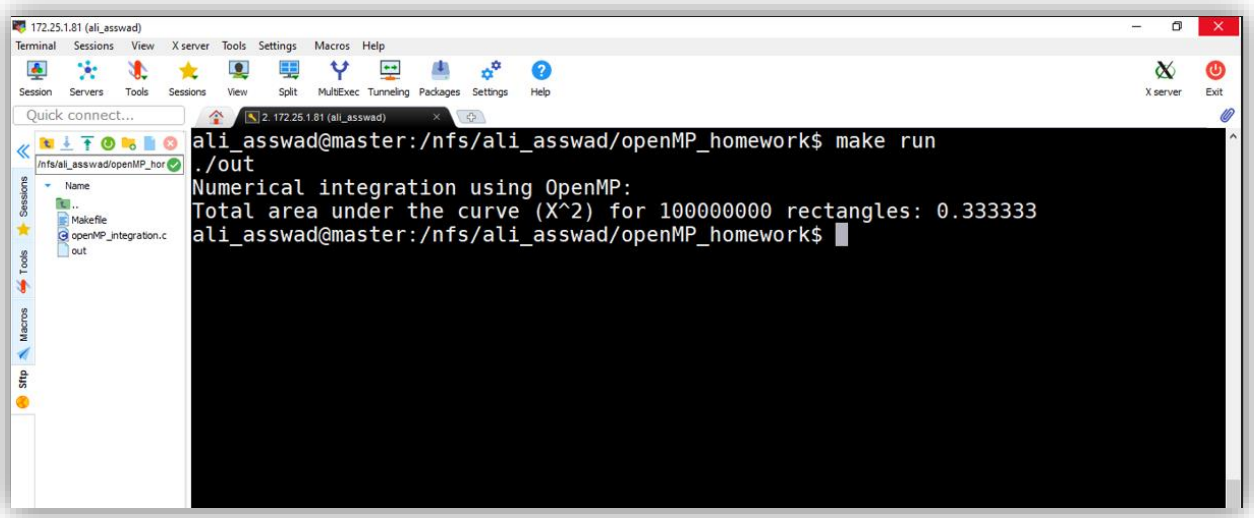
علي أسود

السؤال الأول:

المطلوب في هذا السؤال تطبيق مفاهيم OpenMP لحساب تكامل التابع X^2 بطريقة عددية من خلال حساب المساحة للمنطقة الواقعة تحت منحنى التابع وذلك على المجال [0,1].

ولهذا الغرض نعتمد طريقة التجزئة والتي تقوم بتقسيم المساحة الكلية إلى مستطيلا جزئية لها مساحات منفصلة يمكن حسابها، فتكون المساحة الكلية هي مجموع المساحات الجزئية لهذه المستطيلات.

تم تنجيز المطلوب في هذا السؤال باستخدام مفاهيم ال OpenMP و كان الخرج على الشكل التالي:



بالنسبة للطريقة الافضل لتنجيز هذه المسألة بشكل عام هذا يتعلق بعدة عوامل مثل سهولة التنفيذ وقابلية التوسع وخصائص الأداء والموارد الحاسوبية لنناقش خصائص كل خيار موجود لدينا بناء على ذلك

- استخدام OpenMP لوحدها:
  - يعد OpenMP مناسبًا تمامًا لتوازي الذاكرة المشتركة، حيث يمكن للنياسب المتعددة العمل على البيانات المشتركة داخل عقدة حسابية واحدة.
  - عادةً ما يكون تنفيذه أسهل.
  - يعد OpenMP مناسبًا عندما يكون لديك بنية وحدة المعالجة المركزية متعددة النواة وترغب في الاستفادة من جميع النوى المتاحة بكفاءة.ولكن هذا يشكل عامل سلبي في نفس الوقت حيث في حال كانت الموارد الحاسوبية محدودة والمسألة معقدة نوعا ما فان استخدام OpenMP لوحدها سيكون خيارا غير موفق.

لكن ضمن معطيات مسألتنا استخدام OpenMP لوحدها يمكن ان يكون خيار جيد نوعا ما.

- استخدام MPI لوحدها
  - تم تصميم MPI لتوازي الذاكرة الموزعة ويستخدم بشكل شائع للحوسبة المتوازية عبر عقد متعددة (عدة حواسب).
  - يتطلب تواصل بين الاجرائيات المختلفة و هذا ما يخلق زمن تواصل قد يؤثر على المسالة والحل.
  - على الرغم من أن MPI يوفر المزيد من المرونة وقابلية التوسع للأنظمة الموزعة الأكبر حجمًا، إلا أنه يتضمن عمومًا المزيد من التعقيد في البرمجة مقارنةً بـ OpenMP.

ولذلك تبعا لمسألتنا و التي تعتبر بسيطة نوعا ما فان استخدام MPI لوحدها قد يكون خيار غير جيد لانه سيعطينا تعقيد نحن في غنى عنه.

- استخدام OpenMP مع MPI

على الرغم من ان هذه الطريقة تكون فعالة و خيار جيد في العديد من المسائل فهي ستجمع بين مزايا الطريقتين سوية، ولكنها ايضا على الطرف الاخر ستجمع بين سلبيات الطريقتين ،ولكن لنفس السبب السابق و هو ان مسألتنا مسألة بسيطة فان ادخال ال MPI ضمن الحل و ادخال عدة حواسب في عملية ايجاد المساحة سيكلفنا زمن تواصل زائد و اتوقع بان كفاءة استخدام OpenMP لوحدها ستكون اكبر.

خلاصة:

بالنسبة لهذه المسألة المتمثلة في حساب التكامل عدديًا باستخدام طريقة التقسيم، حيث يكون التركيز على استغلال التوازي داخل عقدة واحدة، سيكون OpenMP هو الخيار الأفضل. فهو يوفر البساطة في التنفيذ، والاستخدام الفعال للمعالجات متعددة النواة، ويتعامل مع توازي الذاكرة المشتركة بشكل فعال دون الحاجة تواصل كبير.

بينما اذا كنا نتعامل مع مشكلة واسعة النطاق و معقدة للغاية تتطلب توزيع الحسابات عبر عقد متعددة، فقد يكون استخدام MPI مع OpenMP أكثر ملاءمة.

السؤال الثاني:

1- تم المرور عل البرامج المعطاة لاكتشاف الاخطاء و كانت النتائج كالتالي:

- البرنامج الاول صحيح ولا يوجد اي خطأ فيه و يعطي الخرج المطلوب
- البرنامج الثالث ايضا صحيح ولا توجد فيه اية اخطاء و يعطي الخرج المطلوب
- البرنامج الثاني لا توجد فيه اخطاء على مستوى التنفيذ، ولكن يوجد خطأ منطقي في الكود، وذلك على مستوى الاستفادة من التوازي في CUDA حيث يتم تعريف blockواحد يحتوي 256 نيسب كل منها يقوم بحساب مجموع الشعاعين كاملا، أي أن حساب مجمو ع الشعاعين سيتكرر 256 مرة حسب عدد النياسب، اي اننا عرفنا مجموعة كبيرة من النياسب بشكل غير مفيد و بالعكس تماما لان ذلك سينعكس على زمن التنفيذ. وهذا موضح في هذه الصورة

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    for(int i = 0; i < n; i += 1){
        out[i] = a[i] + b[i];
    }
}


   // Executing kernel
   vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

- بينما البرنامج الرابع هو الذي يحتوي على خطأ على مستوى التنفيذ، حيث أن النواة يتم تشغيلها بكتلة واحدة و256 نيسب لكل كتلة ، و نحن لدينا احجام المصفوفات a, b, out هو 10000، وهو أكبر من عدد النياسب 256 ضمن ال gridهذا يعني أنه لن تتم معالجة جميع عناصر المصفوفات بواسطة النياسب التي تم تشغيلها، مما يؤدي إلى نتائج غير صحيحة.

حيث فقط اول 256 قيمة تمت معالجتهم بشكل صحيح واما باقي القيم لم تتم معالجتها و اخذت القيمة صفر

يكمن الخطا في السؤال الرابع في هذا الجزء

```
// Executing kernel
vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

ولحله يمكن زيادة عدد النياسب ضمن الكتلة الواحدة بحيث تحتوي 10000 نيسب.

2- تم اجراء التعديلات على البرنامج cu4.Vec_a حيث تم اضافة حلقة تكرارية للمرور على الاحجام المختلفة (32,128,512,1024). و البرنامج المرفق يوضح ذلك: