الاسم: محمد سلامه
إشراف: ما. محمد بشار دسوقي

أولاً:

السؤال الأول:

يمكن تنجيز الحل بأي طريقة من الطرق السابقة، ولكن إنشاء نياسب محلية (OpenMP) هي الأسرع وفقاً للتجربة عندما يكون عدد مجالات التقسيم (عدد المستطيلات) نوعاً ما صغير والمجال ليس بكبير، أما عند تضخيم المجال وأعداد المستطيلات يصبح لكل نيسب قسم كبير من الحسابات المتسلسلة وقد لا تنصف خوارزمية الجدولة في النظام هذا الأمر في جدولة النياسب والإجرائيات ويأخذ وقتاً أطول.

MPI لوحدها تسبب التسريع فقط عندما يكون world size كبيراً حيث تصبح تكلفة الحساب المحلي أكبر من تكلفة الاتصال وإرسال البيانات والتي تستهلك وقتاً كبيراً جداً مقارنةً بسرعة الحساب المحلي.

عند دمج الاثنان: نحصل على حل مثالي من أجل أعداد مستطيلات كبيرة (زيادة الدقة) حيث نقسم المسألة إلى عدة مسائل جزئية (عددها بحجم world size) وكل مسألة جزئية نقوم بحسابها بأسرع طريقة ممكنة عبر OpenMP على كل آلة من MPI World.

الحل الأخير يتيح تكبير المجال قدر الإمكان وزيادة الدقة (عدد المستطيلات التي يتقسم لها المجال) بأكبر ما يمكن، مع مراعاة توجيه كمية حسابات كبيرة لكل آلة بحيث يفوق زمن حسابها زمن إرسالها بكثير.

تم التحقق من النتائج السابقة عبر تجريب البرامج الملحقة بعدة حالات ولأجل مجالات عدة وعدد مستطيلات مختلف.

الاسم: محمد سلامه
إشراف: ما. محمد بشار دسوقي

ثانياً:

السؤال الأول:

في البرنامج add_Vec2.cu وضمن تابع vector_add نقوم بحساب المجموع عبر المرور على عناصر المصفوفة وحداً واحداً ضمن حلقة for وهذا يتسبب في حسابه عبر نيسب وحيد بسبب عدم إتاحية القدرة على التوازي، أي يتم تخصيص 512 نيسب وكل نيسب يكرر كامل الحلقة، أي يتم تنفيذها (threads number * array length)، فيتحول لبرنامج تسلسلي عادي ولكن يتم حسابه عبر GPU بدلاً من CPU ويأخذ وقتاً طويلاً، يمكن التحقق من ذلك عبر طباعة index و threadIdx.x ضمن الحلقة فنجد أنه لكل index تتكرر كل النياسب المخصصة.

وبما أن هناك كل نيسب مسؤول عن كل المصفوفة ينفذ فإن السجلات الخاصة به فقط سوف تعمل وبالتالي عندما يكون هناك حجوم كبيرة للمصفوفة سوف تأخذ وقتاً طويلاً بسبب إدارة الذاكرة لأنه لن نستطيع وضع كل المصفوفة بذاكرة النيسب الخاصة به وهذا يستهلك وقتاً أطول وموارد أكثر.

يتم حل مشكلة عدم تقسيم العمل في add_Vec3.cu حيث تم تقسيم المصفوفة إلى شرائط يتساوى طولها مع عرض الكتلة (عدد النياسب بها) وبالتالي نكون قسمنا المصفوفة إلى عدد مجالات يساوي عرض الكتلة (عدد النياسب) ونحقق التوازي، ولكن تبقى مشكلة صغيرة في هذا الحل أنه لا يستفيد من وجود أكثر من كتلة، وهو الأمر الذي تم حله في add_Vec4.cu ويقدم البرنامج المثالي.

الاسم: محمد سلامه إشراف: ما. محمد بشار دسوقي

السؤال الثاني:

نقوم بتشغيل أكثر من كتلة معاً وهو ما نعرفه بـ grid_size حيث تعرف عدد الكتل المطلوبة ولكل كتلة عدد من النياسب المتطابق.

أفضل شيء نستطيع القيام به هو ضمان عدد كتل (حجم شبكة) بحيث يكون لكل نيسب عنصر من المصفوفة يحسبه فقط ونعرف رقم مميز للنيسب ضمن الشبكة كلها : tid = blockIdx.x * blockDim.x + threadIdx.x ولكن عند حساب حجم الشبكة قد نعرف عدة كتل فائضة يتغير عددها بعدد النياسب ضمن كل كتلة وعدد الكتل وغيره، وبالتالي قد نحصل على أرقام مميزة (tid) أكبر من عدد عناصر المصفوفة ولذلك يجب أن نراعي هذا الشرط ولا نحسب عنصر المصفوفة إذا ما كان (tid > n).

وبهذه الطريقة الوحيدة نضمن أنه عند وجود عدة كتل وعدة نياسب ضمن الكتلة فإن كل نيسب يقوم بمهمة وحيدة فقط ومتمايزة على مستوى الشبكة.

عند تنفيذ البرنامج المعدل عدة مرات نلاحظ أنه عند تغيير حجم الكتلة، ووضع كتلة بحجم أقل من N وتخصيص عدد نياسب عام أقل من N فإن هناك أخطاء بالمصفوفة لأنه فقط بعض العناصر سوف تحسب (تساوي عدد النياسب المتاميز في الشبكة) أما باقي العناصر لن تحسب، أما عند تخصيص عدد نياسب أكبر فإنن سوف نحاول الوصول لفهارس خارج مجال المصفوفة لذلك يجب وضع الشرط(tid < n).

وبالتالي ليكون البرنامج صحيحاً يجب تخصيص عدد نياسب كلي في الشبكة كلها على الأقل N.

(البرنامج المرفق Second Question - Part 2 - Finding Error.cu يحتوي طريقة للتأكد من حصول أخطاء عبر حساب المصفوفة محلياً ومقارنتها مع الحساب الخارجي).

ملاحظة: لكل طريقة تحسينات في الحساب، حيث مثلاً الطريقة الأخيرة هي المثلى في الحساب وبالأخص عند وجود عمليات حسابية كثيرة ومكلفة جداً ومعقدة (أكثر من الجمع)، ولكن يجب مراعاة المسألة عند تقسيمها فعندما نزيد التقسيم نزيد عدد مرات التواصل والاتصال وإرسال المعطيات مما قد يجعل عملية الاتصال مكلفة أكثر من الحسابات، مثلاً في add_vec4.cu عندما نضع N=100,000,000 يكون الحساب بالنانو ثانية أما زمن إرسال البيانات من مرتبة 100 ms أي أكثر من مليون ضعف !!.

ملاحظة 2: النتائج السابقة والملاحظات تم التأكد منها عبر تجريب البرامج على google colab مع Tesla 4 ومقارنة الأزمنة والعمل لمختلف البارامترات، مثال: عندما تم التأكد في add_Vec3.cu أن زيادة عدد الكتل لا يسرع الحساب بسبب طريقة تقسيم المسألة.

ملاحظة 3: add_Vec3 إذا وضعنا أكثر من block فإن كل نيسب ومقابلة من الكتلة الأخرى (لهما نفس id ضمن الكتلة الواحدة) سوف يقومان بتنفيذ نفس العملية وبالتالي سوف يتم تكرار حساب كل stride بعدد الكتل.

يمكن التأكد مما سبق عبر طباعة index ورقم block الذي يتم العمل به.

الاسم: محمد سلامه

إشراف: ما. محمد بشار دسوقي