وظيفة برمجة متوازية

البرمجة الموزعة باستخدام واجهة تمرير الرسائل MPI

January 22, 2024

تقديم: أحمد عبدالله

السؤال الأول:

الهدف من هذا السؤال كان حساب مجموع البادئة Prefix sum لسلسلة من الاعداد الصحيحة المخزنة ضمن مصفوفة و ذلك بشكل متوازي.

ولتنفيذ ذلك تم تنجيز التابع prefix_mpi الذي يقوم بحساب مجموع البادئة باستخدام MPI، حيث يقوم بأخذ كتلة من الأعداد الصحيحة (block_array) بطول (block_size) ويحسب مجموع البادئة لتلك الكتلة الخاصة باجرائية ما. يتم تخزين النتيجة في المصفوفة block_prefix.

خطوات العمل ضمن التابع prefix_mpi:

- تم بداية حساب مجموع البادئة لكل كتلة على حدا و نخزنها ضمن المصفوفة local_prefix خاصة بكل كتلة (حيث تم توزيع قيم المصفوفة الكلية على الاجرائيات من خلال العملية MPI_Scatter المستدعاة ضمن ال Main و بذلك تحصل كل اجرائية على كتلة القيم الخاصة بها).

- يهمنا من كل مصفوفة مجموع بادئة تم الحصول عليها في المرحلة الاولى العنصر الاخير منها لانه يعبر عن المجموع التراكمي لعناصر كل كتلة، لذلك نقوم في هذه الخطوة بعملية تجميع للعنصر الأخير من كل local_prefix array ضمن الاجرائية الmaster وذلك داخل المصفوفة prefix_sums وذلك من خلال العملية MPI_Gather

MPI_Gather(&local_prefix[block_size - 1], 1, MPI_INT,prefix_sums, 1,MPI_INT, 0, communicator)

- الان ضمن الاجرائية ال Master نقوم باجراء مجموع البادئة لعناصر المصفوفة prefix_sums، وبعدها نقوم بعملية بث MPI_Bcast لهذه المصفوفة (مجموع البادئة ل prefix_sums) لجميع الاجرائيات الاخرى، بحيث يصبح لكل اجرائية مصفوفة مجموع البادئة المحلي الخاص بها local_prefix و ايضا مجموع البادئة ل prefix_sums.

- الان في الخطوة الاخيرة تقوم كل اجرائية بايجاد مجموع البادئة النهائي للعناصر الموجودة في الكتلة الخاصة بالاستفادة من كل مما سبق و ذلك من خلال المعادلة التالية:

```
// Adjust local prefix sums with the global prefix sum
for (int i = 0; i < block_size; i++) {
    block_prefix[i] = local_prefix[i] + prefix_sums[my_rank] - local_prefix[block_size - 1];
}
```

تم تنفيذ البرنامج ولكن باخذ مصفوفة طولها 64 مؤلفة من قيم عشوائية من 0 الى 10، وتم حساب مجموع البادئة لهذه القيم و كان الخرج على الشكل التالي:

```
2. 172.25.1.81 (ahmad_abdallah)
ahmad_abdallah@master:/nfs/Ahmad_Abdallah/Homework/Q1$ make run
mpirun -np 4 -f hosts ./output
Total Array: 6 10 6 2 1 4 0 6 3 1 8 7 5 3 7 4 9 10 2 0 10 8 5 0 4 6 0 10 3 10 10 7 10 3 7 9 7 8 2 10 7 10 4 1 0 10 4 9 7 6 8 6 2 2 7 6 6 5 5
 7 4 4 4 3

total Prefixes:
6 16 22 24 25 29 29 35 38 39 47 54 59 62 69 73 82 92 94 94 104 112 117 117 121 127 127 137 140 150 160 167 177 180 187 196 203 211 213 223 2
30 240 244 245 245 255 259 268 275 281 289 295 297 299 306 312 318 323 328 335 339 343 347 350

 for each process :
Process 0: 6 16 22 24 25 29 29 35 38 39 47 54 59 62 69 73
Process 1: 82 92 94 94 104 112 117 117 121 127 127 137 140 150 160 167
Process 2: 177 180 187 196 203 211 213 223 230 240 244 245 245 255 259 268
Process 3: 275 281 289 295 297 299 306 312 318 323 328 335 339 343 347 350
Test completed!
ahmad_abdallah@master:/nfs/Ahmad_Abdallah/Homework/Q1$
```

تم تنفيذ البرنامج باستخدام اربع اجرائيات على التوازي، ويوضح الخرج بداية مصفوفة مجموع البادئة الكلية و اسفلها مصفوفة مجموع البادئة الخاصة بكل اجرائية.

السؤال الثاني:

الهدف من هذا السؤال هو تنفيذ عملية الاختزال reduce يدويًا في MPI دون استخدام MPI_Reduce وذلك بشكل شجري، ومقارنة التنفيذ التسلسلي والتنفيذ الشجري من حيث زمن التنفيذ و ذلك على أبعاد مختلفة من المصفوفات وعدد مختلف من الإجرائيات.

- تم بداية في الطلب الأول تنجيز عملية الاختزال بشكل شجري من خلال تنجيز التابع reduce_tree. حيث يقوم هذا التابع بتنجبز عملية ال reduce بشكل متوازي بطريقة قائمة على الشجرة، حيث يفترض ان الاجرائيات مرتبة ضمن شجرة افتراضية، بحيث تتوافق كل عملية مع عقدة من الشجرة.
  ويتم تنفيذ عملية ال reduce من خلال قيام كل اجرائية باستلام البيانات من الاجرائيات (العقد) الفرعية لها ودمجها مع البيانات الخاصة بها، وارسال البيانات جميعها الى العقدة الاب، تقوم كل الاجرائيات بهذه الخطوات (استلام البيانات من العقد الابناء، وارسال ال العقدة الاب) ما عدا العقدة الجذر فهي تقوم فقط بالاسلام من العقد الابناء لها و اختزال هذه البيانات مع البيانات الخاصة بها.
  خطوات العمل ضمن التابع reduce_tree على الشكل التالي:
  - كل اجرائية يهمها ان تعرف من هم الاجرائيات الابناء لها و من هي الاجرائية الاب لها لذا في الخطوة الأول تم حساب رتب الاجرائيات الأبناء والاجرائية الأب في الشجرة الافتراضية اعتمادا على رتبة الاجرائية الحالية
    و ذلك من خلال العلاقات :

parent_rank = (my_rank - 1) / 2

left_child_rank = 2 * my_rank + 1

right_child_rank = 2 * my_rank + 2

  - بعدها تم حجز مكان في الذاكرة لمصفوفة ال recv_data و المصفوفات التي سيتم استلامها من الاجرائيات الأبناء.
  - بعد ذلك تقوم كل اجرائية بعملية استلام MPI_Recv من الاجرائيات الابناء لها في حال وجودها و يتم جمعها مع البيانات الخاصة بها.
  - و بعدها تقوم كل اجرائية ما عدا ال Master بعملية ارسال MPI_Send للبيانات المدمجة إلى الاجرائية الاب.

- تم تنفيذ البرنامج من اجل عدد مختلف من الاجرائيات و لكل اجرائية تم تجريب اطوال مختلفة من المصفوفات. وكانت النتائج كالتالي:
  - حالة 12 اجرائية:

```
2. 172.25.1.81 (ahmad_abdallah)
mpirun -np 12 -f hosts ./output

size of array : 100
Processing time for reduce tree is: 0.155991
Processing time for reduce sequential is: 0.211972

-----------------------------------------------------------


size of array : 1000
Processing time for reduce tree is: 0.069788
Processing time for reduce sequential is: 0.060007

-----------------------------------------------------------


size of array : 10000
Processing time for reduce tree is: 0.130028
Processing time for reduce sequential is: 0.207496

-----------------------------------------------------------


size of array : 100000
Processing time for reduce tree is: 2.544570
Processing time for reduce sequential is: 1.278798

-----------------------------------------------------------


size of array : 1000000
Processing time for reduce tree is: 5.211179
Processing time for reduce sequential is: 9.103636

-----------------------------------------------------------


size of array : 10000000
Processing time for reduce tree is: 40.477533
Processing time for reduce sequential is: 85.067337
```

- حالة 8 اجرائيات:

```
2. 172.25.1.81 (ahmad_abdallah)
mpirun -np 8 -f hosts ./output

size of array : 100
Processing time for reduce tree is: 0.151749
Processing time for reduce sequential is: 0.154487

-----------------------------------------------------------


size of array : 1000
Processing time for reduce tree is: 0.019996
Processing time for reduce sequential is: 0.033029

-----------------------------------------------------------


size of array : 10000
Processing time for reduce tree is: 0.079271
Processing time for reduce sequential is: 0.137641

-----------------------------------------------------------


size of array : 100000
Processing time for reduce tree is: 0.630331
Processing time for reduce sequential is: 0.456534

-----------------------------------------------------------


size of array : 1000000
Processing time for reduce tree is: 1.573133
Processing time for reduce sequential is: 4.591607

-----------------------------------------------------------


size of array : 10000000
Processing time for reduce tree is: 9.302163
Processing time for reduce sequential is: 29.544025
```

- حالة 4 اجرائيات:

```
2. 172.25.1.81 (ahmad_abdallah)
mpirun -np 4 -f hosts ./output

size of array : 100
Processing time for reduce tree is: 0.012531
Processing time for reduce sequential is: 0.000299

-----------------------------------------------------------


size of array : 1000
Processing time for reduce tree is: 0.009186
Processing time for reduce sequential is: 0.003543

-----------------------------------------------------------


size of array : 10000
Processing time for reduce tree is: 0.002902
Processing time for reduce sequential is: 0.006104

-----------------------------------------------------------


size of array : 100000
Processing time for reduce tree is: 0.085658
Processing time for reduce sequential is: 0.005064

-----------------------------------------------------------


size of array : 1000000
Processing time for reduce tree is: 0.064479
Processing time for reduce sequential is: 0.034200

-----------------------------------------------------------


size of array : 10000000
Processing time for reduce tree is: 0.490421
Processing time for reduce sequential is: 0.322197
```

و عليه كانت النتائج كالتالي:

الملاحظات	زمن التنفيذ التسلسلي	عدد الاجرائيات	زمن التنفيذ المتوازي	بعد المصفوفة
التسلسلي اقل	0.322197	4	0.490421	10000000
التسلسلي اقل	0.034200	4	0.064479	1000000
التسلسلي اقل	0.005064	4	0.085658	100000
المتوازي اقل	0.006104	4	0.002902	10000
المتوازي اقل	29.544025	8	9.302163	10000000
المتوازي اقل	4.591607	8	1.573133	1000000
التسلسلي اقل	0.456534	8	0.630331	100000
المتوازي اقل	0.137641	8	0.079271	10000
المتوازي اقل	85.067337	12	40.477533	10000000
المتوازي اقل	9.103636	12	5.211179	1000000
التسلسلي اقل	1.278798	12	2.544570	100000
المتوازي اقل	0.207496	12	0.130028	10000

- ملاحظات على النتائج

نلاحظ من الجدول السابق بأنه لا يمكن القول بان اداء طريقة reduce_tree أفضل من أداء طريقة التنفيذ التسلسلي sequential reduce و العكس بالعكس. اي انه في بعض الحالات يتفوق اداء طريقة reduce_tree على اداء طريقة sequential reduce و في حالات أخرى نجد بأن اداء sequential reduce افضل من reduce_tree، وهذه الحالات تتعلق بعاملين اساسيين هما:

- عدد الاجرائيات: بشكل عام كلما زاد عدد الاجرائيات يصبح زمن الحساب اقل لان جميع الاجرائيات ستتشارك بعملية الحساب، من جهة اخرى زيادة عدد الاجرائيات يؤدي الى زيادة عدد مرات التواصل بين الاجرائيات و بالتالي زيادة زمن التواصل.
- بعد المصفوفة : بشكل عام ايضا كلما زاد بعد المصفوفة يصبح زمن التنفيذ المطلوب اكبر، ومن جهة اخرى عمليات ارسال واستقبال المصفوفات ستتطلب وقت اكبر و بالتالي زيادة زمن التواصل.

بناءً على ما سبق يمكن ان نستنتج امكانية تفوق اداء احد الطريقتين على الاخرى.

على سبيل المثال عندما يكون عدد الاجرائيات كبير و بعد المصفوفة كبيرجدا ستتفوق طريقة ال reduce_tree على طريقة ال sequential reduce و ذلك لان زمن التنفيذ التسلسلي سيكون كبير جدا في حالة الsequential reduce ، و اما عند استخدام reduce_tree ستشارك جميع الاجرائيات في عملية الحساب و هذا ما يخفض زمن الحساب بشكل كبير على الرغم من ان عدد مرات التواصل بين الاجرائيات سيكون كبير، وحجم المصفوفة المنقولة في كل عملية تواصل كبير اي ان زمن الاتصال كبير ايضا ، ولكن هنا يطغى زمن الحساب على زمن الاتصال لذلك اللجوء الى عملية تقسيم الحساب على عدد كبير من الاجرائيات سيؤدي الى جعل الاداء افضل بكثير من عملية الحساب التسلسلي و هذا ما تقدمه طريقة reduce_tree.
بينما لو كان عدد الاجرئيات قليل و بعد المصفوفة كبير هنا تقسيم هذا الحساب الكبير على عدد اجرائيات قليل لن يؤدي الى الحصول على اداء افضل بكثير من لو تم استخدام الحساب التسلسلي ، وانما يمكن ان نحصل على اداء متقارب من الحساب التسلسلي لذلك من الافضل استخدام الحساب التسلسلي، حيث انه في طريقة reduce_tree سيكون لدينا عدد مرات قليل من عمليات التواصل ولكن كل عملية ستتطلب وقت كبير لان بعد المصفوفة كبير وبالتالي في هذه الحالة تقريبا يتساوى زمن الحساب وزمن الاتصال، وهذا ما يمكن ملاحظته من نتائج الجدول.

خلاصة:
يمكن ان نستنتج ان استخدام احد الطريقتين السابقتين يتعلق بشكل رئيسي بمعيطات المسالة (عدد الاجرائيات, بعد المصفوفة) فعندما يكون زمن الحساب التسلسلي صغير من الافضل استخدام طريقة sequential reduce و لا داعي لاستخدام reduce_tree لان هذا سيضيف اعباء الاتصال بين الاجرائيات بدون جدوى تذكر على تقليل زمن الحساب اما عندما يكون زمن الحساب التسلسلي المطلوب كبير و يتوفر لدينا عدد اجرائيات لا بأس به فمن الافضل استخدام طريقة reduce_tree.

ملاحظة:
يجب الاخذ بعين الاعتبار انه في طريقة reduce_tree الحساب لا يتم على التوازي بشكل كامل حيث ان كل اجرائية ستنتظر حتى تنتهي الاجرائيات الابناء الخاصين بها من تنفيذ حساباتهم، لتقوم باستلام المعطيات منهن و هذا يؤدي الى انه بالاضافة الى زمن الحساب والتواصل، يوجد زمن انتظار. لهذا السبب يمكن ان لا تكون طريقة reduce_tree ذات اداء افضل من sequential reduce .