Übungsblatt 7 zur Vorlesung Parallele Programmierung

Abgabe: 06.01.2024, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik • Otto-von-Guericke-Universität Magdeburg
https://parcio.ovgu.de

---

1. Parallelisierung mit MPI (300 Punkte)

Parallelisieren Sie das Jacobi-Verfahren im `partdiff`-Programm mithilfe von MPI. Dabei soll nach gleicher Iterationszahl das Ergebnis identisch bleiben. Außerdem soll bei Abbruch nach Genauigkeit im parallelen Programm nach derselben Iterationszahl wie im seriellen abgebrochen werden (und das gleiche Ergebnis geliefert werden).

Es gibt hier zwei Fälle, die auf Korrektheit der Parallelisierung zu prüfen sind:

1. Jacobi-Verfahren mit Abbruch nach fester Iterationszahl
2. Jacobi-Verfahren mit Abbruch nach Genauigkeit

Überprüfen Sie, dass die Ergebnisse mit mehreren Prozessen identisch zum seriellen Fall sind. Wichtig: Wenn dies nicht der Fall ist, ist Ihr Programm falsch!

Vorgaben und Hinweise

- Das Programm darf nicht langsamer als die serielle Variante sein.
- Zu keinem Zeitpunkt darf ein Prozess die gesamte Matrix im Speicher halten.
- Das Programm muss weiterhin mit einem Prozess funktionieren.
- Das Programm muss mit beliebigen Prozesszahlen funktionieren.
- Erstellen Sie eine eigene Funktion für die MPI-Parallelisierung des Jacobi-Verfahrens.

  Hinweis: Sie können die in den Materialien bereitgestellte `displayMatrixMpi`-Funktion als Grundlage für die parallele Ausgabe der Matrix benutzen.

Hinweis: Die Bearbeitungszeit ist schwer zu schätzen, da sie sehr stark von Ihren Vorkenntnissen und dem Glück, mit dem Sie auf Anhieb eine einigermaßen fehlerfreie MPI-Implementierung hinbekommen, abhängt. Bei komplexen Fehlern kann sich der Aufwand aber leicht stark erhöhen, fangen Sie deshalb frühzeitig an.

2. Leistungsanalyse (120 Punkte)

Ermitteln Sie die Leistungsdaten Ihres Programms und visualisieren Sie die Laufzeiten für folgende Konfigurationen in einem Diagramm. Wenn im Folgenden von einer Konfiguration (K, P, N) die Rede ist, so ist damit immer gemeint, dass das Programm auf K Knoten mit insgesamt P gleichmäßig auf den Knoten verteilten Prozessen und N Interlines laufen soll.

(1, 1, 836), (1, 2, 1.182), (1, 3, 1.448), (1, 6, 2.048), (1, 12, 2.896), (1, 24, 4.096),
(2, 48, 5.793), (4, 96, 8.192), (8, 192, 11.585)

Vergleichen Sie außerdem ihr Programm mit der ursprünglichen seriellen Variante. Der kürzeste Lauf sollte mindestens 30 Sekunden rechnen; wählen Sie geeignete Parameter!

Wiederholen Sie dabei jede Messung mindestens drei Mal, um aussagekräftige Mittelwerte bilden zu können. Dafür können Sie beispielsweise das Werkzeug `hyperfine` benutzen, das Sie mit `module load hyperfine` laden können. Es ist außerdem empfehlenswert die Störfunktion $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$ zu verwenden, da der erhöhte Rechenaufwand das Skalierungsverhalten verbessert.

Für `K` Knoten, `P` Prozesse, `N` Iterationen und `I` Interlines können Sie folgendes Script benutzen:

```
#!/bin/bash

#SBATCH --nodes=K
#SBATCH --ntasks=P
#SBATCH --exclusive
#SBATCH --partition=vl-parcio

srun --mpi=pmi2 ./partdiff 1 2 I 2 2 N
```

Die Messungen sollen dabei mit Hilfe von SLURM auf den Rechenknoten durchgeführt werden. Geben Sie die für die Messungen verwendete Hardwarekonfiguration (Prozessor, Anzahl der Kerne, Größe des Arbeitsspeichers etc.) an.

In den Materialien sind bereits entsprechend vorbereitete Job-Skripte enthalten.

Visualisieren Sie alle Ergebnisse in hinreichend beschrifteten Diagrammen. Schreiben Sie ca. eine viertel Seite Interpretation zu diesen Ergebnissen.

3. Hybride Parallelisierung (120 Bonuspunkte)

Parallelisieren Sie Ihre MPI-Version des Jacobi-Verfahrens zusätzlich mittels OpenMP.

Ermitteln Sie die Leistungsdaten Ihres Hybrid-Programms und vergleichen Sie die Laufzeiten für folgende Konfigurationen in einem Diagramm:

- 1 Prozess × 24 Threads
- 2 Prozesse × 12 Threads
- 3 Prozesse × 8 Threads

- 6 Prozesse × 4 Threads
- 12 Prozesse × 2 Threads
- 24 Prozesse × 1 Thread

Verwenden Sie hierzu 4.096 Interlines. Der kürzeste Lauf sollte mindestens 30 Sekunden rechnen; wählen Sie geeignete Parameter! Wiederholen Sie dabei jede Messung mindestens drei Mal, um aussagekräftige Mittelwerte bilden zu können. Es ist wieder empfehlenswert die Störfunktion $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$ zu verwenden. Die Messungen sollen dabei mit Hilfe von SLURM auf einem der Rechenknoten durchgeführt werden. Geben Sie die für die Messungen verwendete Hardwarekonfiguration (Prozessor, Anzahl der Kerne, Größe des Arbeitsspeichers etc.) an und nutzen Sie den gleichen Rechenknoten für eine Messreihe.

Visualisieren Sie alle Ergebnisse in hinreichend beschrifteten Diagrammen. Schreiben Sie ein halbe Seite Interpretation zu diesen Ergebnissen.

4. Optimierungswettbewerb (120 Bonuspunkte) (Abgabe bis 13.01.24)

Wir veranstalten darüber hinaus einen kleinen Wettbewerb, um festzustellen welche Abgabegruppe die am besten optimierte Version des MPI-Jacobi-Verfahrens erstellen kann. Hierfür haben Sie nach Abgabe dieses Übungsblattes eine weitere Woche Zeit, um mögliche Änderungen in Ihrem Git-Repository vorzunehmen. Die Voraussetzung ist, dass das Programm weiterhin die gleichen Ergebnisse produziert wie vor der Parallelisierung. Wir werden nach der Abgabefrist Benchmark-Messungen an Ihrem Programm vornehmen, um zu bestimmen welche Gruppe die optimierteste Lösung implementiert hat.

Als kleinen Anreiz für die Teilnahme werden wir einen Preis für die Gewinner vergeben.

Abgabe

Als Abgabe werten wir den letzten Commit vor der Abgabefrist in Ihrem Git-Repository. Im Hauptverzeichnis des Repositories wird ein Verzeichnis `PP-2023-Uebung-07-Materialien` mit folgendem Inhalt erwartet:

- Eine Datei `gruppe.txt` mit den Gruppenmitgliedern (eines je Zeile) im folgenden Format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- Der überarbeitete Code des `partdiff`-Programms im Verzeichnis `pde` (Aufgaben 1 und 3)
  - Optional: Ein Target `partdiff-hybrid` für die Binärdatei `partdiff-hybrid`, welche die Hybrid-Parallelisierung umsetzt
- Eine Ausarbeitung `leistungsanalyse.pdf` mit den ermittelten Laufzeiten und der Leistungsanalyse (Aufgaben 2 und 3)