Übungsblatt 5 zur Vorlesung Parallele Programmierung

Abgabe: 02.12.2023, 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

Wir gehen jetzt wieder von unserem seriellen Programm zur Lösung der Poisson-Gleichung aus und betrachten dabei aber nur die Variante des Jacobi-Verfahrens. Hierfür sollen jetzt Parallelisierungen mittels POSIX Threads erstellt werden.

Tutorials zur Programmierung mit POSIX Threads finden Sie unter:

https://hpc-tutorials.llnl.gov/posix/

1. Parallelisierung mit POSIX Threads (300 Punkte)

Parallelisieren Sie das Jacobi-Verfahren aus dem seriellen Programm mittels POSIX Threads. Wiederum müssen die parallelen Varianten dasselbe Ergebnis liefern wie die serielle Variante; sowohl der Abbruch nach Iterationszahl als auch der Abbruch nach Genauigkeit müssen korrekt funktionieren und dieselben Ausgaben wie die seriellen Gegenstücke liefern!

Bezüglich der Leistung sollte sich ein akzeptabler Speedup ergeben. Beachten Sie dazu auch die Hinweise zur Leistungsmessung in Aufgabe 2.

Die Threadanzahl soll dabei über den bereits vorhandenen aber bisher ungenutzten ersten Parameter der `partdiff`-Anwendung gesteuert werden können.

Bitte protokollieren Sie, wie viel Zeit Sie benötigt haben. Wie viel davon für die Fehlersuche?

2. Leistungsanalyse (120 Punkte)

Ermitteln Sie die Leistungsdaten Ihres Programms und visualisieren Sie die Laufzeiten für jeweils 1, 2, 3, 6, 12, 18 und 24 Threads in einem Diagramm. Vergleichen Sie außerdem Ihr Programm mit der ursprünglichen seriellen Variante. 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. 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 `t` Threads, `i` Interlines und `n` Iterationen können Sie folgenden Aufruf benutzen:

```
./partdiff t 2 i 2 2 n
```

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.

In den Materialien finden Sie im `slurm`-Verzeichnis vorgefertigte Job-Skripte, welche Sie für die Messreihen verwenden können. Zum Ausführen sollten Sie sich im `slurm`-Verzeichnis befinden und von dort aus die Skripte mit `sbatch` starten:

```
$ cd PP-2023-Uebung-05-Materialien
$ make -C pde
$ cd slurm
$ sbatch messung1.slurm
```

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

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-05-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 Unterverzeichnis `pde` (Aufgabe 1)
- Eine Ausarbeitung `leistungsanalyse.pdf` mit den ermittelten Laufzeiten und der Leistungsanalyse (Aufgabe 2)