Übungsblatt 5 zur Vorlesung Parallele Programmierung

Abgabe: 20.11.2021, 23:59

Jun.-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 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 OpenMP erstellt werden.

Mit der Option `-fopenmp` übersetzt GCC OpenMP-Code. Ein Tutorial zur Programmierung mit OpenMP finden Sie unter folgender URL:

https://hpc.llnl.gov/tuts/openMP/

1. Parallelisierung mit OpenMP (180 Punkte)

Parallelisieren Sie das Jacobi-Verfahren aus dem seriellen Programm mittels OpenMP. Die parallele Variante muss dasselbe Ergebnis liefern wie die serielle; sowohl der Abbruch nach Iterationszahl als auch der nach Genauigkeit müssen korrekt funktionieren und dieselben Ausgaben wie die seriellen Gegenstücke liefern! Benutzen Sie die OpenMP-Klausel `default(none)`, um Fehler zu vermeiden.

Der erreichbare Speedup des parallelisierten Programms ist abhängig von der konkreten Hardware, mit 1.024 Interlines sollte sich aber bereits ein erkennbarer Speedup ergeben.

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

Wenn Sie Ihr Programm ausführen, können Sie z. B. mit dem Tool `top` („1“ drücken, um die Cores aufgeschlüsselt zu erhalten) die Auslastung betrachten.

Bitte protokollieren Sie, wieviel Zeit Sie benötigt haben. Wieviel davon für die Fehlersuche?

2. Umsetzung der Datenaufteilungen (60 Punkte)

Zusätzlich zur Standardaufteilung sollen Sie die Daten auf drei verschiedene Arten verteilen:

- Zeilenweise Aufteilung (d. h. Thread 1 bekommt die erste(n) Zeile(n), …)
- Spaltenweise Aufteilung (d. h. Thread 1 bekommt die erste(n) Spalte(n), …)
- Elementweise Aufteilung (d.h. jedes Matrixelement kann von einem anderen Thread berechnet werden)

Implementieren Sie die verschiedenen Datenaufteilungen in Ihrem Programm, wobei für jede Datenaufteilung eine separate `calculate`-Funktion angelegt werden soll. Wenn verschiedene Datenaufteilungen parallelisiert sind, geben Sie den Code bitte so ab, dass die Binärdateien für die entsprechenden Datenaufteilungen jeweils ein Target sind. Hierzu kann beim Kompilieren `-D` verwendet werden, um Flags während der Kompilierung zu setzen. Die mit `-D` definierten Makros können dann wie folgt im Code benutzt werden:

```
#ifdef ELEMENT
...
#endif
```

Hinweis: Für diese Aufgabe muss eventuell der Code der Schleifen umgeschrieben werden.

3. Vergleich Scheduling-Algorithmen (60 Bonuspunkte)

OpenMP kennt verschiedene Strategien zur Verteilung von Arbeit (z. B. Schleifeniterationen) an die Threads. Wenn Sie die Aufgabe verschiedener Datenaufteilungen gelöst haben, dann können Sie auch noch verschiedene OpenMP-Scheduling-Algorithmen evaluieren. Eine Liste mit sinnvollen Scheduling-Einstellungen lautet wie folgt:

- Static (Blockgrößen 1 ,2, 4 und 16)
- Dynamic (Blockgrößen 1 und 4)
- Guided

Hinweis: Diese Aufgabe sollte eine automatische Datenaufteilung verwenden. Dafür ist unter Umständen wieder eine Anpassung der Datenaufteilung und Schleifendurchläufe erforderlich. Vergleichen Sie die Leistungsfähigkeit der Scheduling-Algorithmen für die Datenaufteilung nach Elementen und einer anderen Aufteilung.

4. Leistungsanalyse (180 Punkte)

In dieser Aufgabe soll die Leistungsfähigkeit der parallelisierten Variante analysiert werden. Das parallelisierte Programm sollte bei Ausführung mit 24 Threads und 4.096 Interlines einen Speedup von mindestens 10 erreichen.

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
```

Messung 1

Ermitteln Sie die Leistungsdaten Ihres OpenMP-Programms und vergleichen Sie die Laufzeiten für jeweils 1–24 Threads in einem Diagramm. Vergleichen Sie Ihre Variante auch unbedingt mit dem ursprünglichen seriellen Programm! Verwenden Sie hierzu 4.096 Interlines. Der kürzeste Lauf sollte mindestens 30 Sekunden rechnen; wählen Sie geeignete Parameter aus!

Führen Sie diese Messung sowohl für die Standardaufteilung als auch für die zusätzlichen Datenaufteilungen aus Aufgabe 2 und, falls bearbeitet, die Scheduling-Algorithmen aus Aufgabe 3 durch. Welchen Grund kann es für die gemessenen Ergebnisse geben? Schreiben Sie dazu ca. eine Seite Erklärungen.

Messung 2

Ermitteln Sie weiterhin, wie Ihr OpenMP-Programm in Abhängigkeit der Matrixgröße (Interlines) skaliert. Verwenden Sie hierzu 24 Threads und 13 Messungen zwischen 1 und 4.096 Interlines (wobei Interlines = 2^i für $0 \leq i \leq 12$). Der längste Lauf sollte maximal 30 Minuten rechnen. Starten Sie mit 4.096 Interlines und wählen Sie geeignete Parameter aus; für die folgenden Läufe können Sie die Interlines dann entsprechend der gegebenen Formel verringern. Visualisieren Sie alle Ergebnisse in hinreichend beschrifteten Diagrammen.

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.

Führen Sie diese Messung nur für die Standardaufteilung durch. Schreiben Sie ca. eine halbe Seite Interpretation zu diesen Ergebnissen.

Abgabe

Als Abgabe erwarten wir ein gemäß den Vorgaben benanntes komprimiertes Archiv (`MustermannMusterfrau.tar.gz`), das ein gemäß den Vorgaben benanntes Verzeichnis (`Mustermann-Musterfrau`) mit folgendem Inhalt enthält:

- 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 1–3)
  - Ein `Makefile` mit Targets `partdiff-{element,spalte,zeile}` für Binärdateien `partdiff-{element,spalte,zeile}`, welche jeweils die entsprechende Datenaufteilung umsetzen (Aufgabe 2)
  - Optional: Kommentare im Code, die die Umsetzung der unterschiedlichen Scheduling-Algorithmen erläutern (Aufgabe 3)

- Eine Ausarbeitung `leistungsanalyse.pdf` mit den ermittelten Laufzeiten und der Leistungsanalyse (Aufgabe 4)

Laden Sie das Archiv unter einer der folgenden URLs hoch:

1. Dienstag 7–9 Uhr (Julian Benda): `https://cloud.ovgu.de/s/ZYSSZDZnwc7pi5D`
2. Mittwoch 15–17 Uhr (Michael Blesel): `https://cloud.ovgu.de/s/MK5gKij7MBoDMxX`
3. Mittwoch 15–17 Uhr (Johannes Wünsche): `https://cloud.ovgu.de/s/XYafSH83enW8XNZ`