Universität Hamburg, Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Prof. Dr. T. Ludwig, Anna Fuchs, Jannek Squar

Übungsblatt 4 zur Vorlesung
Hochleistungsrechnen
im WiSe 2024/2025
Abgabe: 09.11.2023, 23:59

---

1. Parallelisierung mit OpenMP (150 Punkte)

Parallelisieren Sie das **Jacobi-Verfahren** des **sequentiellen** Programms zur Lösung der Poisson-Gleichung mittels OpenMP.

- Korrektheit
  - Die parallele Variante muss dasselbe Ergebnis liefern wie die sequentielle: Gleiche Matrix und gleiche Norm des Fehlers. Sowohl der Abbruch nach Iterationszahl als auch der nach Genauigkeit müssen korrekt funktionieren und dieselben Ausgaben wie die sequentiellen Gegenstücke liefern!
- Code
  - Die Threads **müssen** außerhalb der `while`-Schleife erzeugt werden. Achten Sie dabei auf die nötige Synchronisation und den Geltungsbereich der Variablen.
  - Die Anzahl der Threads **muss** über den ersten Parameter gesetzt werden können.
  - Sie **müssen** die Klausel `default(none)` benutzen
  - Testen Sie Ihr Programm mit mehreren gleichen Läufen hintereinander, um so ggf. zeitbedingtes fehlerhaftes Verhalten aufzuspüren.
  - Gauß-Seidel muss weiterhin sequentiell funktionieren.
- Laufzeit
  - Das parallelisierte Programm sollte bei Ausführung mit 12 Threads und 512 Interlines einen Speedup von ungefähr 10 erreichen. 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.
  - Gilt für alle Messungen in HLR, wenn nicht anders verabredet:
    * Wiederholen Sie dabei jede Messung mindestens **drei** Mal, um aussagekräftige Mittelwerte bilden zu können
    * Geben Sie die Messwerte in einer Tabelle ab (inkl. der Messungen des sequentiellen und des parallelisierten Programms mit einem Thread)
    * Messen Sie nur auf den West-Rechenknoten und immer auf dem selben Knoten
    * Geben Sie für jede Messung die verwendeten Rechenknoten an
- Alternativen
  - Überlegen Sie eine alternative Umsetzung der Parallelisierung mit anderen OpenMP-Konzepten. Beschreiben Sie diese z.B. durch Pseudocode der relevanten Schleife. Diskutieren Sie die Effizienz der beiden Ansätze im Vergleich zueinander. Abzugeben in `leistungsanalyse.pdf`.

Hinweise

Mit der Option `-fopenmp` erzeugt `gcc` OpenMP-Code. Ein Beispielprogramm ist unter `/home/hr/openmp/hello` verfügbar und lässt sich direkt aufrufen. Es arbeitet standardmäßig mit so vielen Threads, wie logische Cores vorhanden sind (24 auf den West-Rechenknoten). Den Quellcode dazu finden Sie unter `/home/hr/openmp/hello.c`.

Tutorials zur Programmierung mit OpenMP finden Sie unter folgender URL:

`https://hpc-tutorials.llnl.gov/openmp/`

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

2a. Umsetzung der Datenaufteilungen (60 Bonuspunkte)

Erweitern Sie ihre Implementierung um verschiedene Datenaufteilungen:

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

Implementieren Sie für jede Datenaufteilung eine separate `calculate`-Funktion. Kompilieren Sie jede Datenaufteilung zu einem separaten Target – SPALTE, ZEILE oder ELEMENT. Im Makefile wird dazu `-D` verwendet, um Flags während der Kompilierung zu setzen. Nutzen Sie dabei umschließende Bedingungen wie folgt:

```
1 #ifdef SPALTE
2   ... // dieser Code wird nur in partdiff-openmp-spalte kompiliert
3 #endif
```

Hinweis: Überlegen Sie sich für den Vergleich geeignete Parameter für den Start ihres Programmes. Welchen Grund kann es für die gemessenen Ergebnisse geben? Schreiben Sie dazu ca. eine halbe Seite Erklärungen.

2b. Vergleich der Scheduling-Algorithmen (45 Bonuspunkte)

Vergleichen Sie folgende Scheduling-Algorithmen für die zeilenweise und elementeweise Aufteilungen. Schreiben Sie eine ausreichende Interpretation Ihrer Messergebnisse.

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

3. Leistungsanalyse (120 Punkte)

Messung 1

Ermitteln Sie die Leistungsdaten Ihres OpenMP-Programms und vergleichen Sie die Laufzeiten für jeweils 1–12 Threads in einem beschrifteten Diagramm.

- Vergleichen Sie Ihre Variante auch unbedingt mit dem sequentiellen ursprünglichen Programm
- Verwenden Sie hierzu 512 Interlines
- Der kürzeste Lauf sollte mindestens 30 Sekunden rechnen; wählen Sie geeignete Parameter aus

Messung 2

Ermitteln Sie weiterhin, wie Ihr OpenMP-Programm in Abhängigkeit von der Matrixgröße (Interlines) skaliert.

- Verwenden Sie hierzu 12 Threads und 11 Messungen zwischen 1 und 1024 Interlines (wobei Interlines = 2^i für $0 \leq i \leq 10$).
- Der längste Lauf sollte 20 bis 40 Minuten rechnen; starten Sie mit 1024 Interlines und wählen Sie geeignete Parameter aus. Für die folgenden Läufe können Sie die Interlines-Zahl dann entsprechend der gegebenen Formel verringern.

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

Hinweis: Es ist empfehlenswert die Störfunktion $f(x,y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$ zu verwenden, da der erhöhte Rechenaufwand das Skalierungsverhalten verbessert.

Abgabe

Abzugeben ist ein gemäß den bekannten Richtlinien erstelltes und benanntes Archiv. Das enthaltene und gewohnt benannte Verzeichnis soll folgenden Inhalt haben:

- Alle Quellen, aus denen Ihr Programm besteht (`askparams.c` und `partdiff.{c,h}`); gut dokumentiert (Kommentare im Code!)
- Ein Makefile welches mittels `make` automatisch alle benötigten Binärdateien erzeugt
- **Optional:** Code `partdiff-openmp-{element,spalte,zeile}` für Binärdateien `partdiff-openmp-{element,spalte,zeile}`, welche jeweils die entsprechende Datenaufteilung umsetzen
- Eine Ausarbeitung `leistungsanalyse.pdf` mit den ermittelten Laufzeiten, der Leistungsanalyse und Erklärungen

Senden Sie Ihre Abgabe an `hr-abgabe@wr.informatik.uni-hamburg.de`.