

Parallelisierung mit OpenMP (240 Punkte)

Wir gehen jetzt wieder von unserem sequentiellen 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` erzeugt `gcc` OpenMP-Code. Das Programm `/home/hr/openmp/hello` lässt sich direkt aufrufen und arbeitet standardmäßig mit so vielen Threads, wie logische Cores vorhanden sind (24 auf den Rechenknoten). Den Quellcode des Programmes finden Sie unter `/home/hr/openmp/hello.c`.

Man kann es mit einer anderen Anzahl laufen lassen, wenn man die Umgebungsvariable `OMP_NUM_THREADS` entsprechend setzt. Tutorials zur Programmierung mit OpenMP finden Sie unter <http://www.llnl.gov/computing/tutorials/openMP/>.

Aufgabenstellung

Parallelisieren Sie das Jacobi-Verfahren aus dem **sequentiellen** Programm mittels OpenMP. Die parallelen Varianten müssen dasselbe Ergebnis liefern wie die sequentiellen. Das parallelisierte Programm sollte bei Ausführung mit 12 Threads und 512 Interlines einen Speedup von ungefähr 10 erreichen. Beachten Sie dazu auch die Hinweise zur Leistungsmessung in der Aufgabe „Leistungsanalyse“.

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

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

Umsetzung der Datenaufteilungen (150 Bonuspunkte)

In Aufgabe 1 haben Sie auf eine bestimmte Weise die Daten auf die Threads verteilt. In dieser Aufgabe sollen Sie die Daten auf drei verschiedenen Arten zu verteilen:

- 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 auf einem anderen Thread berechnet werden)

Implementieren Sie die verschiedenen Datenaufteilungen in Ihrem OpenMP-Programm. 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.

Hinweis: Für diese Aufgabe muss eventuell der Code der Schleifen umgeschrieben werden. Überlegen Sie sich für den Vergleich geeignete Parameter für den Start ihres Programmes. Welchen Grund kann es für die gemessenen Ergebnisse geben (ca. 1/2 Seite)?

Vergleich der OpenMP-Scheduling-Algorithmen (150 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:

- Static (Blockgröße 1, 2, 4, 16)
- Dynamic (Blockgröße 1, 4)
- Guided

Hinweis: Diese Aufgabe sollte sinnvollerweise auch 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.

Leistungsanalyse (120 Punkte)

Messung 1

Ermitteln Sie die Leistungsdaten Ihres OpenMP-Programms und vergleichen Sie die Laufzeiten für jeweils 1–12 Threads in einem Diagramm. Verwenden Sie hierzu 512 Interlines.

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 $\text{Interlines} = 2^i$ für $0 \leq i \leq 10$).

Schreiben Sie ein paar Zeilen (1/2 Seite) Interpretation zu diesen Ergebnissen. Die schnellste Parallelisierung sollte mindestens 50 Sekunden rechnen; wählen Sie geeignete Parameter aus! Außerdem sollten Sie jede Messung mindestens 3 mal wiederholen, um aussagekräftige Mittelwerte bilden zu können.

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 (`.tar.gz`). Das enthaltene und gewohnt benannte Verzeichnis soll folgenden Inhalt haben:

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

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

Bearbeitungszeit			
Schwierigkeit	<input type="checkbox"/> zu leicht	<input type="checkbox"/> genau richtig	<input type="checkbox"/> zu schwer
Lehrreich	<input type="checkbox"/> wenig	<input type="checkbox"/> etwas	<input type="checkbox"/> sehr
Verständlichkeit	<input type="checkbox"/> großteils unklar	<input type="checkbox"/> teilweise unklar	<input type="checkbox"/> verständlich
Kommentar			