Institut für Informatik — TU München
Scientific **C**omputing in **C**omputer **S**cience
Dipl.Inf. T. Auckenthaler, Dipl.Inf. W. Eckhardt

WS 2010/11

Masterpraktikum Scientific Computing (High Performance Computing) Übungsblatt 2: Cache-Blocking und Shared-Memory-Parallelisierung

Zur Übung am 09.11.2010

Aufgabe 5 „Matrix-Matrix-Multiplikation I“(2 Punkte)

Der C-Quellcode `dgemm.c` enthält eine Variante zur Berechnung der Matrix-Matrix-Multiplikation

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \ , \qquad i,j = 1 \ldots n.$$

Untersuchen Sie den Algorithmus mit Blick auf die Speicherzugriffe.

- Messen Sie die MFLOPS- und die Cache-Miss-Rate auf dem Rechner `lx64ia2.lrz-muenchen.de` für unterschiedliche Problemgrößen. Benutzen Sie `Valgrind` für die Messung der Cache-Miss-Rate.
- Welchen Einfluss haben die verschiedenen Compiler-Flags auf die Floatingpoint-Leistung?
- Bei welcher Problemgröße erhalten Sie die maximale Leistung?
- Ab welcher Problemgröße bricht die Leistung ein?
- Versuchen Sie die Performance-Kurve anhand der Systemeigenschaften (Cachegröße, Speicherbandbreite) zu erklären.

Versuchen Sie nun, die MFLOPS-Rate durch *Cache-Blocking* zu erhöhen. Durch das Hinzufügen von zwei weiteren Schleifen soll der Code so abgeändert werden, dass die Register und der Cache (L1-Cache) optimal genutzt werden. Wie ändert sich nun die Leistung mit der Problemgröße? Wie ändert sich die Cache-Miss-Rate?

Hinweis zu Valgrind auf dem LRZ Linux-Cluster

- Laden Sie das Modul valgrind mit `module load valgrind`.
- Compilieren Sie das Programm mit der zusätzlichen Option `-g`.
- Starten Sie Valgrind mit `valgrind --tool=cachegrind myprog [myprog-options]`.
- Mit der Option `--branch-sim=yes` kann zusätzlich noch die Sprungvorhersage simuliert werden.

Aufgabe 6 „Shared-Memory π-Berechnung“(2 Punkte)

Mit $\phi(x) = \dfrac{1}{1+x^2}$ gilt $\displaystyle\int \phi(x)dx = arctan(x).$

Somit kann π durch Integration der Funktion $\phi(x)$ berechnet werden.

- Entwickeln Sie ein serielles Programm, das die Funktion $\phi(x)$ über das Einheitsinterval $[0,1]$ integriert.
- Portieren Sie das serielle Programm auf den Rechner `lx64ia2.lrz-muenchen.de`. Parallelisieren Sie das Programm mit `OpenMP`. Implementieren Sie zwei unterschiedliche Varianten (*critical directive*, *reduction clause*), die jeweils die korrekte parallele Ausführung der Summation sicherstellen.
- Lassen Sie das serielle Programm durch den Intel Compiler autoparallelisieren.
- Mit der Umgebungsvariable `OMP_NUM_THREADS` geben Sie die Anzahl der parallelen Threads an. Starten Sie das Programm mit 1, 2 und 4 Threads. Berechnen Sie jeweils den Speedup. Interpretieren Sie die bei steigender Zahl an Threads erhaltenen Ergebnisse!

Hinweis: Verwenden Sie bei der Integration die Mittelpunktsregel. Unterteilen Sie dazu das Einheitsinterval in n äquidistante Abschnitte der Länge $h = \frac{1}{n}$. Berechnen Sie im Mittelpunkt $\check{x}_i$ jedes Teilintervals den Wert der Funktion $\phi(\check{x}_i)$. Summieren Sie die Funktionswerte auf. Führen Sie die Stützstellenberechnung und die Summation in einer Schleife über alle Teilintervalle aus. Parallelisieren Sie diese Schleife! Am Ende multiplizieren Sie die Summe mit $4 \cdot h$. Warum?

Aufgabe 7 „Matrix-Matrix-Multiplikation II“(2 Punkte)

Parallelisieren Sie Ihr Programm aus Aufgabe 4 mit OpenMP! Stellen Sie sicher, dass jeweils ein Thread einen Cache-Block bearbeitet. Untersuchen Sie das Skalierungsverhalten (weak scaling und strong scaling) Ihres Programms auf der “kleinen Altix”

des LRZ-Linux-Clusters. Machen Sie sich dazu mit dem vorhandenen Batch-System vertraut (`http://www.lrz.de/services/compute/linux-cluster/batch/`).

Viel Erfolg beim Bearbeiten!

Die Abgabe ist bis 30.11.2010, 9.00 Uhr möglich.

Alle Programme finden Sie zum Herunterladen auf der Praktikumsseite unter:
`http://www5.in.tum.de/wiki/index.php/Masterpraktikum_Scientific_Computing_-_High_Performance_Computing_-_Winter_10`