

Übungen zur Vorlesung
Paralleles Höchstleistungsrechnen
Dr. S. Lang, J. Pods

Abgabe: 22. Januar 2013, 12 Uhr, per E-mail an jurgis.pods@iwr.uni-heidelberg.de

Übung 21 MPI: Kommunikationszeiten

(5 Punkte)

Wir wollen die Zeiten messen, die bei den MPI-Sende-Routinen `MPI_Send`, `MPI_Ssend`, `MPI_Bsend`, `MPI_Rsend` und `MPI_Isend` (verschiedene Varianten blockenden bzw. nicht-blockenden Sendens) benötigt werden, bis

1. der aufrufende Knoten wieder die Kontrolle über das Programm nach Beenden des jeweiligen Sende-Befehls erlangt hat,
2. der Empfänger die Nachricht ganz erhalten hat.

Eine Übersicht und eine kurze Erklärung der verschiedenen Sende-Modi finden Sie auf <http://www.mcs.anl.gov/research/projects/mpi/sendmode.html>.

Teilaufgabe (a)

Schreiben Sie dazu ein Programm, das per Kommandozeilen-Parameter die gewünschte Routine zum Versenden einer Nachricht an einen anderen Prozess wählt. Die Größe der Nachricht soll ebenso als Programmparameter übergeben werden können. Der Sender misst die Zeit, die es dauert, bis das Send-Kommando abgeschlossen ist, d.h. er wieder die Kontrolle über das Programm erhält, und der Empfänger die Zeit, die er benötigt, die Nachricht vollständig zu empfangen. Um zuverlässige Werte zu erhalten, sollte es möglich sein, diese Prozedur mehrmals hintereinander auszuführen und die Zeiten dann zu mitteln. Die Anzahl Wiederholungen sollte an die Dauer der Nachrichtenübermittlung angepasst sein, d.h. schnelle Übertragungen werden häufiger wiederholt. Das ganze muss natürlich nicht automatisch geschehen.

In http://cpansearch.perl.org/src/JOSH/Parallel-MPI-0.03/contrib/perf/mpi_timing.c ist ein Programm gegeben, das eine "Ping-Pong Kommunikation" zwischen zwei Prozessen testet und die obigen Anforderungen bereits weitgehend erfüllt. Sie müssen dieses noch um die Möglichkeit erweitern, verschiedene Send-/Receive-Methoden auswählen zu können.

Messen Sie für jede Sende-Art die Zeiten mit mehreren Nachrichtengrößen. Die Nachrichtengröße soll dabei von wenigen Byte bis zu einigen MBytes variieren. Messen Sie auch die Hopzeit, d.h. die Übertragungszeit eines einzelnen Bytes.

Achtung: Bei einigen nicht-blockenden Varianten dürfen Sie den Sende- bzw. Empfangspuffer erst dann wieder verwenden, wenn die entsprechende Sende- oder Empfangsoperation (im Hintergrund) ausgeführt wurde. Lesen Sie in der Dokumentation nach, wie Sie dies sicherstellen. Außerdem sollten Sie als Gegenstück zu `MPI_Isend` auf der Empfängerseite auch `MPI_Irecv` statt `MPI_Recv` verwenden.

Teilaufgabe (b)

Die Routine `MPI_Send` kann blockierend oder nicht-blockierend arbeiten. Überlegen Sie sich einen Test, wie herausgefunden werden kann, ab welcher Nachrichtengröße n synchron gesendet wird. Implementieren Sie den Test und finden Sie das passende n im Pool heraus.

Freiwillige Zusatzaufgabe

Wiederholen Sie die Messungen aus Teilaufgabe (a), arbeiten Sie aber nur lokal auf einem Rechner arbeiten (wie das geht, siehe Übung 20). Wie verhalten sich die Zeiten nun?

Hinweise

- Da wir auch den Overhead durch die Kommunikation messen wollen, messen wir die Echtzeit unter Verwendung von `MPI_Wtime()`. Im Pool hat `MPI_Wtime()` die Auflösung $1\mu s$ (Abfrage über `MPI_Wtick()`), was für unsere Zwecke ausreichen sollte.
- Synchronisieren Sie die Prozesse vor dem Starten der Zeitmessung mittels einer `MPI_Barrier()`.
- Profiling und Zeitmessung für MPI-Programme ist nicht trivial. Hier sind einige Links auf hilfreiche alte und neue Dokumente:
 - Ein Beispiel wie wir es in Teilaufgabe (a) implementieren wollen:
http://cpansearch.perl.org/src/JOSH/Parallel-MPI-0.03/contrib/perf/mpi_timing.c
 - Pitfalls in der Zeitmessung:
<http://www.mcs.anl.gov/research/projects/mpi/mpptest/hownot.html>
 - MPI-Profiling Suite:
<http://www.mcs.anl.gov/research/projects/mpi/mpptest/>
 - Ein recht ausführliches Paper:
<http://perplexity.org/Publications/hoefler-collmea.pdf>

Übung 22 MPI: Matrixmultiplikation

(10 Punkte)

In den Übungen 4 und 9 haben wir zwei (quadratische) Matrizen $A, B \in \mathbb{R}^{n \times n}$ miteinander multipliziert, $C = A \cdot B$, und die Flop-Raten mit Tiling bzw. OpenMP gemessen. Wir wollen das nun unter Verwendung von MPI wiederholen und die Skalierbarkeit des Problems hinsichtlich der Anzahl der Prozesse untersuchen.

Aufgabe

Schreiben Sie ein Programm, das die Matrizen-Multiplikation MPI-parallel ausführt. Auf der Homepage steht eine sequentielle Variante inklusive Tiling bereit (Übersetzen mit `g++ -fopenmp mm_vanilla.c`), die Sie natürlich nicht verwenden müssen.

Man könnte daran denken, den Cannon-Algorithmus zur Lösung dieser Aufgabe zu verwenden, was aber einige Schwierigkeiten mit sich bringt: Zunächst einmal muss bei Cannon $N \bmod \sqrt{P} = 0$ gelten (die Anzahl der Rechenknoten muss dem Quadrat einer natürlichen Zahl entsprechen und N muss ein Vielfaches dieser Zahl sein), was einige der unten geforderten Kombinationen für N, P unmöglich macht. Außerdem ist der Cannon-Algorithmus vor allem dann effizient, wenn man große Matrizen, die nur verteilt gespeichert werden können, multipliziert. Hier sind die Matrizen aber maximal $2048 \cdot 2048 \cdot 8$ Bytes = 32 MBytes groß und passen somit problemlos in den Speicher jedes Knotens.

Es soll hier also eine andere Strategie verwendet werden. Überlegen Sie sich einen geeigneten Algorithmus zur Aufteilung und erklären Sie dessen Konzept. Die Matrix soll aber initial so auf die Prozessoren verteilt sein, dass nicht jeder Prozessor beide Ausgangs-Matrizen voll kennt (sonst fällt ja ein Großteil des Kommunikations-Overheads weg). Initialisieren Sie die Matrizen so, daß sie vollbesetzt sind, etwa $a_{ij} = b_{ij} = i + j$, $i, j = 0 \dots n - 1$. Messen Sie die Rechenzeit in s und FLOP-Rate für $m = 1, 2, 4, 6, 8, 12, 16, 20, 25, 32$ beteiligte Prozesse und für die Matrix-Größen (Grundseite der Matrix) $n = 256, 512, 1024, 2048$. Zur Messung mit $m = 1$ können Sie das bereitgestellte sequentielle Programm verwenden. Nehmen Sie zur Zeitmessung wie gehabt `MPI_Wtime()` und synchronisieren vor Beginn der Zeitmessung mittels einer `MPI_Barrier()`. Plotten Sie Diagramme Rechenzeit über Problemgrößen sowie des Speed-Ups über Prozessoranzahl P für die verschiedenen Problemgrößen. Diskutieren Sie kurz die Ergebnisse. Falls noch vorhanden, können Sie auch mit den alten Ergebnissen der Aufgaben 4 und 9 vergleichen.

Freiwillige Zusatz-Aufgabe

Verwenden Sie auf den einzelnen Rechenknoten zusätzlich ein Tiling, um den Cache besser auszunutzen, und vergleichen Sie die Messungen mit den Ergebnissen der ungekachelten Variante.