Dr. Andreas Borchert
Prof. Dr. Stefan Funken
Prof. Dr. Karsten Urban
Markus Bantle, Kristina Steih

Institut für Numerische Mathematik
Universität Ulm
Sommersemester 2013

High Performance Computing – Blatt 3

(Präsenzübung 6. Mai 2013)

Diskussion

- *MPI Grundlagen:*
  - Nennen Sie einen wesentlichen Unterschied zwischen openMP und MPI.
  - Wie kompliliert bzw. startet man ein MPI programm?
  - Wir betrachten das folgende Programm:

    ```
    int main(int argc, char** argv) {
      MPI_Init(&argc, &argv);
      std::cout << "MPI Rocks!\n" << std::endl;
      MPI_Finalize();
      return 0;
    }
    ```

    Wenn das Programm für MPI übersetzt wird und mit vier Prozessen gestartet wird, was wird auf der Kommandozeile ausgegeben?

- *Vorbereitung die Aufgaben:*
  - Wie findet man im Programm heraus, wie viele Prozesse gestartet wurden bzw. was die ID eines Prozesses ist?
  - Was machen die Funktionen `MPI_Send` und `MPI_Recv`? Was sind die Eingabeparameter für die Funktionen?
  - Was machen die Funktionen `MPI_Bcast` und `MPI_Reduce`?
    Was sind die Eingabeparameter für die Funktionen?
    Welche Prozesse müssen die Funktion aufrufen?

Aufgabe 1: Stille Post

Schreiben Sie ein MPI-C++ Programm, in dem eine Nachricht zwischen verschiedenen Prozessen im Kreis herum geschickt wird. Das Programm soll wie folgt aufgebaut sein

- Prozess 0 liest die Kommandozeilen-Argumente aus. Falls es ein Argument gibt, ist das die Nachricht, die herum geschickt werden soll. Falls keine Nachricht angegeben ist, soll eine im Code definierte Nachricht geschickt werden. Die Nachricht wird dann von Prozess 0 an Prozess 1 geschickt.
- Alle anderen Prozesse warten auf eine Nachricht vom Vorgänger-Prozess und schicken die erhaltene Nachricht dann weiter an den Nachfolger-Prozess (z.B. empfängt Prozess 3 von Prozess 2 und schickt dann weiter an Prozess 4). Der letzte Prozess (also der Prozess mit der höchsten ID) schickt die Nachricht dann wieder zu Prozess 0.
- Prozess 0 gibt dann am Enden die Nachricht aus.

Aufgabe 2: Skalarprodukt

Schreiben Sie ein Programm, das das Skalarprodukt von zwei Vektoren parallelisiert mit berechnet. Die Länge der Vektoren soll dabei entweder als Kommandozeilenargument angegeben werden, oder, falls das nicht der Fall ist, im im Code gesetzt werden. Die beiden Vektoren sollen zunächst von Prozess 0 initialisiert und dann auf die anderen Prozessoren aufgeteilt werden. Am Ende des Programms soll Prozess 0 das Ergebnis ausgeben.

Zusatzaufgabe: Ändern Sie ihr Programm so ab, dass die Arbeit zum Berechnen des Skalarproduktes mit dem Master-Slave Prinzip aufgeteilt wird (siehe auch das Beispiel aus der Vorlesung auf den Folien 95-104). Hierbei ist Prozess 0 der Master, der nur dafür zuständig ist, die Arbeit zu verteilen und Ergebnisse einzusammeln. Alle anderen Prozesse (Slaves) empfangen vom Master-Prozess Teile der beiden Vektoren, berechnen das Teil-Skalarprodukt und schicken das Ergebnis zurück.

Aufgabe 3: Vektor-Normen

Für einen großen Vektor $v \in \mathbb{R}^n$ sollen die Normen

$$\|v\|_1 = \sum_{i=1}^{n} |v_i|, \qquad \|v\|_2 = \left(\sum_{i=1}^{n} v_i^2 \right)^{1/2} \quad \text{und} \quad \|v\|_\infty = \max_{i=1,\ldots,n} |v_i|$$

berechnet werden. Dies soll mit einem Programm mit vier Prozessen geschehen, die wie folgt arbeiten:

- Prozess 0 initialisiert den Vektor und schickt ihn an alle anderen Prozesse.
- Prozess 1 berechnet $\|v\|_1$ und schickt das Ergebnis zuück zu Prozess 0.
- Prozess 2 berechnet $\|v\|_2$ und schickt das Ergebnis zurück zu Prozess 0.
- Prozess 3 berechnet $\|v\|_\infty$ und schickt das Ergebnis zurück zu Prozess 0.
- Am Ende gibt Prozess 0 die Ergebnisse aus.

Die Größe des Vektors soll hierbei als Kommandozeilenargument übergeben werden. Falls das Programm mit mehr oder weniger als vier Prozessen gestartet wird, soll eine Fehlermeldung ausgegeben werden.