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 4

(Präsenzübung 13. Mai 2013)

Diskussion

- *C++ Grundlagen: Speicherverwaltung*
  - Was machen die Operatoren `new`, `delete`?
  - Was machen die Operatoren `new[]`, `delete[]`?
  - Was ist der Unterschied zwischen `int a[10]` und `int* a = new int[10]`?
  - Was macht die Funktion `memcpy` (s. Folie 127)? Was sind die Parameter?
- *Vorbereitung Aufgabe 1:*

  In der Datei *mpi_matrixvector.cpp* wird die C++-Schreibweise der MPI-Befehle verwendet.
  - Beschreiben Sie die Unterschiede zur bisherigen C-Schreibweise.
- *Vorbereitung Aufgabe 2:*

  Benötigter Vorlesungsstoff:
  - Was macht die MPI-Funktion `MPI_Sendrecv`? Was sind die Parameter?
  - Wie funktioniert die Kommunikation im Jacobi-Beispiel der Vorlesung (Folien 124-125)? Was ist mit Hin- und Rückwelle gemeint?
  - Was ist der Nullprozess `MPI_PROC_NULL` und wozu kann er eingesetzt werden (Folie 126)?

  Spielregeln und Verständnis:
  - Was sind die Regeln des *Game of Life* (siehe Aufg. 2)?
  - Welche Informationen werden für jede Zelle in jedem Iterationsschritt benötigt?
  - Wie groß sind die lokalen Teilmatrizen, wenn das Spielfeld $m \times n$ Zellen groß ist?
  - Wie kann das Spiel organisiert werden (d.h. wann werden die Nachbarn gezählt, wann wird kommuniziert)?

Aufgabe 1: Parallele Matrix-Vektor-Multiplikation

Aufgabe ist es, für eine quadratische Matrix $A \in \mathbb{R}^{d \times d}$ die Matrix-Vektor-Multiplikation

$$y = A^k x, \qquad k \in \mathbb{N},$$

mit MPI verteilt zu berechnen, indem die Zeilen der Matrix A auf die verschiedenen Prozesse verteilt werden.

Auf der Homepage finden Sie ein Programm *mpi_matrixvector.cpp*, welches Sie ergänzen können.

Hinweise:

- Prozess 0 legt A an und füllt die Matrix mit Zufallszahlen aus $[0, 1]$.
- Alle Prozesse speichern lokal ihre Teilmatrizen in einer Variablen `local_A`.
- Der Vektor $x := (1, \ldots, 1)^T \in \mathbb{R}^d$ wird von jedem Prozess angelegt und initialisiert.
- Das Produkt $y_{local} = A_{local} \cdot x$ sollte mit der BLAS-Funktion `gemv` berechnet werden.
- In jeder Iteration $l = 1, \ldots, k$ muss x auf jedem Knoten mit den Ergebnissen y_{local} aller Prozesse überschrieben werden.

Aufgabe 2: Game of Life

Conways *Game of Life* ist eine Simulation von Zellen auf einem rechteckigen Spielbrett. Die Zellen können jeweils tot oder lebendig sein, dies kann sich in jedem Zeitschritt ändern. Die Regeln zur Änderung finden sich unter `http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens`.

Das Game of Life soll mit MPI parallelisiert werden, indem die Zeilen des Spielbretts auf die verschiedenen Prozesse aufgeteilt werden.

Auf der Homepage finden Sie die Dateien *GameOfLife.h*, *GameOfLife.cpp* und *play_GameOfLife.cpp*, welche Sie ergänzen können.

Hinweise:

- Der Rand ist bei uns abgeschlossen (entspricht also einer äußeren Reihe von toten Zellen).
- Da zur Bestimmung des Zustandes einer Zelle immer alle Nachbarn benötigt werden, muss die Aufteilung der Matrix und die Kommunikation der Prozesse analog zum Vorlesungs-Beispiel *mpi-jacobi.cpp* passieren.
- Die Klasse `GameOfLife` hat 2 Spielfelder: In der `bool`-Matrix `data` werden die Zustände der Zellen (tot/lebendig) gespeichert, in der `int`-Matrix `counting_grid` in jeder Iteration die Nachbarn gezählt.
- Zur Visualisierung werden die in Textdateien gespeicherten Ergebnisse von Prozess 0 eingelesen und graphisch dargestellt. Dazu muss beim Kompilieren noch die `X11`-Bibliothek eingebunden werden:

  ```
  openmpic++ GameOfLife.cpp play_GameOfLife.cpp -Wall -L/usr/X11R6/lib -lX11
  ```