PARALLEL PROGRAMMING

Assignment 4

MPI – Matrix Multiplication

Tammo Johannes Herbert (319391), Rico Jasper (319396), Erik Rudisch (343930)

19.06.2013

Exercise 1 – Going for Speed

TODO

Exercise 2 – Sparse Matrices

(a) Sparse Matrix Vector Multiplication

Wir summieren für jede Zeile der Eingabematrix A bzw. des Ausgabevektor c die Summe von Produkten typisch für eine Matrix-Vektor-Multiplikation. Jedoch werden pro Zeile nur die nicht-null Elemente der Matrix A mit dem Eingabevektor b multipliziert. Daher läuft die innere Schleife nur von `row_ptr[i]` nach `row_ptr[i+1]-1`. Die Spalte wird entsprechend über `col_ind[j]` bestimmt.

```
1  for i in 0 ... n-1                              // for each row
2        c[i] = 0
3        for j in row_ptr[i] ... row_ptr[i+1]-1    // for each column
4              c[i] += val[j] * b[col_ind[j]]
```

(b) Parallelization

Für eine Parallelisierung ist es wünschenswert, dass jede Recheneinheit möglichst dieselbe Menge an Arbeit verrichtet. Die Arbeit pro Zeile ist aber abhängig von der Anzahl der nicht-null Elementen und kann daher variieren.

Die zeilenweise parallelisierte Variante erfordert außerdem das replizieren des Eingabevektors b. Im Falle der CRS-Matrix A kann dieser Vektor allerdings in Relation jedoch groß ausfallen, da die Dichte von nicht-null Elementen recht dünn sein kann.

Hinsichtlich des ersten Problems kann man die Lasten versuchen auszugleichen. Beispielsweise könnten mehrere Zeilen einer Recheneinheit zugeteilt werden, um die Gesamtverteilung auszugleichen. Denkbar ist jedoch auch das Aufbrechen von „großen“ Zeilen in kleinere Stücke.

(c) Embedding into a FEM simulation

Die Matrix kann in den gegebenen Varianten A und B auf Grund der Bandstruktur in einer sparsamen statischen Struktur dargestellt werden. Beispielsweise könnte man in einem Array lediglich das Band speichern, womit die Nullen jenseits des Bandes nicht mehr betrachtet werden müssten. In Variante A könnte eine Eingabematrix beispielsweise so aussehen:

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 5 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 6 & 0 & 0 \\ 0 & 2 & 3 & 4 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 & 0 & 8 \\ 1 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 5 & 6 & 7 & 0 \\ 0 & 0 & 3 & 0 & 0 & 6 & 7 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 7 & 8 \end{pmatrix}$$

In einer kompakteren Darstellung könnte auf Grund der Bandstruktur und das Muster der Nullen innerhalb des Bandes nun die Elemente einer Zeile wie folgt angeordnet werden:

$$A^* = \begin{pmatrix} 0 & 0 & 1 & 2 & 5 \\ 0 & 1 & 2 & 3 & 6 \\ 0 & 2 & 3 & 4 & 7 \\ 0 & 3 & 4 & 5 & 8 \\ 1 & 0 & 5 & 6 & 0 \\ 2 & 5 & 6 & 7 & 0 \\ 3 & 6 & 7 & 8 & 0 \\ 4 & 7 & 8 & 0 & 0 \end{pmatrix}$$

Nach einem ähnlichen Vorgehen für Variante B könnte die kompakte Matrix A^* so aussehen:

$$A^* = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 4 \\ 1 & 3 & 4 & 5 \\ 2 & 3 & 4 & 6 \\ 3 & 5 & 6 & 7 \\ 4 & 5 & 6 & 8 \\ 5 & 7 & 8 & 0 \\ 6 & 7 & 8 & 0 \end{pmatrix}$$

Für eine parallele Matrix-Vektor-Multiplikation ist es auch hier möglich die Matrix A^* zeilenweise aufspalten. Dabei müsste man nun aber nicht mehr zur Berechnung einer Zeile Zugriff auf den vollständigen Vektor besitzen. Beispielsweise benötigt man in Variante B für die Berechnung der fünften Zeile nur die Elemente 3, 5, 6 und 7 des Vektors. Das Ergebnis müsste dementsprechend nur an die Rechenknoten übertragen werden, welche dieses auch benötigen. Auf Grund der symmetrischen Adjazenzmatrix sind das ebenfalls 3, 5, 6 und 7.

Ist die Vernetzung nicht symmetrisch und vorhersehbar, so könnte man A^* jedoch nicht so kompakt gestalten. In diesem Fall müsste man zumindest das Band in der vollen Breite abbilden. Wenn es dann dazu noch dünn ausgefüllt ist, so stößt man auf ähnliche Probleme, wie schon in (b). Je schmaler man in so einem Fall das Band ist, umso besser.