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 9

(Präsenzübung 24. Juni 2013)

Teil 1: Theorie verstehen (Hausaufgabe!)

Gegeben Sei das Gebiet $\Omega \subset \mathbb{R}^2$. Wir betrachten wieder das Poisson-Problem

$$\begin{aligned} -\Delta u &= f && \text{in } \Omega \\ u &= u_D && \text{auf } \Gamma_D \subset \Gamma = \partial\Omega \quad \text{(Dirichlet-Randbedingung)} \\ \frac{\partial u}{\partial n} &= g && \text{auf } \Gamma_N = \Gamma \setminus \Gamma_D \quad \text{(Neumann-Randbedingung).} \end{aligned}$$

Um diese PDE mit der Finiten Elemente Methode zu lösen, haben wir die letzten 2 Wochen schon wichtige Schritte gemacht, nämlich

- die Funktionalität für das unterliegende Gitter implementiert,
- das aus der Galerkin-Projektion resultierende diskrete System

  $$\mathbf{Au} = \mathbf{b}, \qquad \mathbf{A} \in \mathbb{R}^{N \times N}, \mathbf{u}, \mathbf{b} \in \mathbb{R}^{\mathbb{N}},$$

  assembliert.

Nun müssen wir das Gleichungssystem nur noch lösen. Dazu wollen wir in dieser Übung sowohl das *cg*- als auch das *Gauß-Seidel*-Verfahren implementieren.
Beide Verfahren sollen aus Numerik I bzw. der Vorlesung noch bekannt sein. Zur Erinnerung sind in Algorithmus 1 und 2 die Algorithmen noch einmal aufgeschrieben.

Algorithm 1 CG-Verfahren: $x = \mathbf{CG}[A, b]$

Input: x^0

$r^0 = b - Ax^0$
$p^0 = r^0$
for $k = 0, \ldots,$ maxIts **do**
  if $(r^k)^\top r^k <$ tol **then**
    return x^k
  end if
  $z^k = A \cdot p^k$
  $\alpha^k = \frac{(r^k)^\top r^k}{(p^k)^\top z^k}$
  $x^{k+1} = x^k + \alpha^k p^k$
  $r^{k+1} = r^k - \alpha^k z^k$
  $\beta^k = \frac{(r^k)^\top r^k}{(r^{k+1})^\top r^{k+1}}$
  $p^{k+1} = r^{k+1} + \beta^k p^k$
end for

Algorithm 2 Gauß-Seidel-Verfahren: $x = \mathbf{GS}[A, b]$

Input: $x^0 = (x_1^0, \ldots, x_N^0)^\top$

1: **for** $\ell = 0, \ldots,$ maxIts **do**
2: **for** $k = 1, \ldots, N$ **do**
3: $x_k^{\ell+1} = \frac{1}{a_{kk}} \left(b_k - \sum_{i=1}^{k-1} a_{ki} x_i^{\ell+1} - \sum_{i=k+1}^{N} a_{ki} x_i^{\ell} \right)$
4: **end for**
5: **end for**

CG parallel

Wie in der Vorlesung besprochen, lässt sich das *cg*-Verfahren mit wenigen Kommunikationsschritten leicht parallelisieren. Wir bezeichnen mit Oberstrichen immer TypI-Größen (global), während die lokalen TypII-Größen mit Unterstrich gekennzeichnet werden.

Algorithm 3 CG-Verfahren: $x = \mathbf{CG\text{-}Parallel}[A, b]$

Input: $\bar{x}^0$

1: $\underline{r}^0 = \underline{b} - A\bar{x}^0$
2: $\bar{p}^0 = \bar{r}^0 = \sum_{n=1}^{\mathbf{numProcs}} C_n^\top \underline{r}^0$
3: **for** $k = 0, \ldots,$ maxIts **do**
4: **if** $(\bar{r}^k)^\top \underline{r}^k <$ tol **then**
5: **return** $\bar{x}^k$
6: **end if**
7: $\underline{z}^k = A \cdot \bar{p}^k$
8: $\alpha^k = \frac{(\bar{p}^k)^\top \underline{r}^k}{(\bar{p}^k)^\top \underline{z}^k}$
9: $\bar{x}^{k+1} = \bar{x}^k + \alpha^k \bar{p}^k$
10: $\underline{r}^{k+1} = \underline{r}^k - \alpha^k \underline{z}^k$
11: $\bar{r}^{k+1} = \sum_{n=1}^{\mathbf{numProcs}} C_n^\top \underline{r}^k$
12: $\beta^k = \frac{(\bar{r}^k)^\top \underline{r}^k}{(\bar{r}^{k+1})^\top \underline{r}^{k+1}}$
13: $\bar{p}^{k+1} = \bar{r}^{k+1} + \beta^k \bar{p}^k$
14: **end for**

Aufgabe 1

- Schauen Sie sich noch einmal das CRS-Format von Blatt 8 an. Wie kann auf die Datenstrukturen einer CRS-Matrix $A = (a_{ij})_{i,j}$ zugegriffen werden, um

$$\mathrm{tmp}_k := \sum_{i=1}^{k-1} a_{ki} x_i^{\ell+1} + \sum_{i=k+1}^{N} a_{ki} x_i^{\ell}$$

  zu realisieren (siehe Zeile 7 in Algorithmus 2)? Warum möchte man *nicht* über die volle Zeile $i = 1, \ldots, k-1, k+1, \ldots, N$ laufen?

- Welche Funktion brauchen wir für die Umwandlung von $\underline{r}^k$ zu $\bar{r}^k$ (Zeilen 2 bzw. 11 in Algorithmus 3)?

Gauß-Seidel parallel

Das Gauß-Seidel-Verfahren wird oft durch eine sogenannte Red-Black-Nummerierung parallelisiert. Hier wollen wir aber eine Alternative umsetzen, die bei nicht-überlappenden Teilgebieten ausnutzt, dass wir nach einer Umsortierung der Knoten-Indizes die folgende

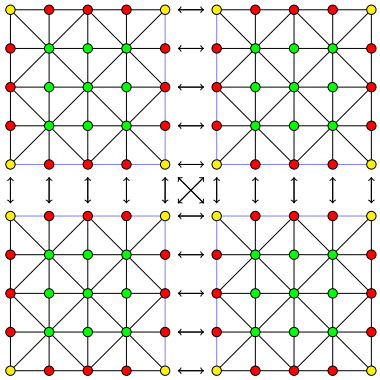

Abbildung 1: Knoten-Aufteilung auf 4 Prozesse

Struktur in der Matrix **A** und damit dem Gleichungssystem haben:

$$\begin{pmatrix} A_{VV} & A_{VE} & A_{VI} \\ A_{EV} & A_{EE} & A_{EI} \\ A_{IV} & A_{IE} & A_{II} \end{pmatrix} \cdot \begin{pmatrix} x_V \\ x_E \\ x_I \end{pmatrix} = \begin{pmatrix} b_V \\ b_E \\ b_I \end{pmatrix},$$

wobei (siehe auch Abbildung 1)

- V: Indexmenge der Kopplungsknoten (gelb),
- E: Indexmenge der Punkte auf Kopplungskanten (rot),
- I: Indexmenge der inneren Knoten (grün).

Wir wissen, dass $A_{VV} =: D_V$ eine Diagonalmatrix und A_{EE} eine Blockdiagonalmatrix mit tridiagonalen Blöcken ist. Beide können also mithilfe weniger Vektoren abgespeichert werden.
Zur Notation: mit $\cdot_{(gl)}$ bezeichnen wir globale (TypI) Matrizen. Damit kann nun der folgende Algorithmus umgesetzt werden:

Algorithm 4 Gauß-Seidel-Verfahren: $x =$ **GS-Parallel**$[A, b]$

Input: $\overline{x}^0$

for $k = 0, \ldots,$ maxIts **do**
  // Schritt 1: $\overline{x}_V^{k+1}$ berechnen
  $\underline{r}_V = \underline{b}_V - A_{VV}\overline{x}_V^k - A_{VE}\overline{x}_E^k - A_{VI}\overline{x}_I^k$
  $\overline{r}_V = \sum_{i=1}^p C_{i,V}^T \underline{r}_{V,i}$
  $\overline{x}_V^{k+1} = \overline{x}_V^k + D_{V,(gl)}^{-1}\overline{r}_V$

  // Schritt 2: $\overline{x}_E^{k+1}$ berechnen unter Verwendung von $\overline{x}_V^{k+1}$
  $\underline{r}_E = \underline{b}_E - A_{EV}\overline{x}_V^{k+1} - A_{EE}\overline{x}_E^k - A_{EI}\overline{x}_I^k$
  $\overline{r}_E = \sum_{i=1}^p C_{i,E}^T \underline{r}_{E,i}$
  $\overline{x}_E^{k+1} = \overline{x}_E^k + A_{EE,(gl)}^{-1}\overline{r}_E$

  // Schritt 3: $\overline{x}_I^{k+1}$ berechnen unter Verwendung von $\overline{x}_V^{k+1}$ und $\overline{x}_E^{k+1}$
  $\underline{r}_I = \underline{b}_I - A_{IV}\overline{x}_V^{k+1} - A_{IE}\overline{x}_E^{k+1} - A_{II}\overline{x}_I^k$
  $\overline{r}_I = \sum_{i=1}^p C_{i,I}^T \underline{r}_{I,i}$
  $\overline{x}_I^{k+1} = \overline{x}_I^k + A_{II}^{-1}\overline{r}_I$
end for

Zur Berechnung von $A^{-1}_{EE,(gl)}\bar{r}_E$ kann die Funktion `solveTridiag(...)` aus *Solver.hpp* verwendet werden, während wir für $A^{-1}_{II}\bar{r}_I$ ein lokales serielles Gauß-Seidel-Verfahren verwenden.

Aufgabe 2

- Was für eine Art von Kommunikation brauchen wir, um $D_{V,(gl)}$ zu bestimmen? Was muss für $A_{EE,(gl)}$ kommuniziert werden und wie kann das umgesetzt werden?
- Welche Funktion kann verwendet werden, um $\bar{r}_V$ (Zeile 4, Algorithmus 4) zu berechnen?
- Welche Funktion kann verwendet werden, um $\bar{r}_E$ (Zeile 9, Algorithmus 4) zu berechnen?
- Muss $\bar{r}_I$ (Zeile 14, Algorithmus 4) explizit berechnet werden? Warum (nicht)?

Teil 2: Gegebenen Code verstehen (Hausaufgabe!)

Im Vergleich zum letzten Mal ist diesmal recht wenig hinzugekommen. Neu sind:

- Die Funktion `FEM::solve(Solver method)`.
- Die Dateien *Solver.hpp* und *Solver.cpp*, in welcher die Lösungs-Verfahren implementiert sind.

Aufgabe 3

- Schauen Sie sich die neuen Code-Teile (und die beiden Hauptprogramme *main-serial.cpp*, *main-parallel.cpp*, um festzustellen, wo was implementiert werden muss.
- Geben Sie einen Überblick darüber, was die Funktion `FEM::solve()` macht. Erklären Sie insbesondere, was es mit den `fixedNodes` auf sich hat.

Teil 3: Programmieren (Präsenzaufgabe)

a) Vervollständigen Sie die seriellen Löser (*cg*- und *Gauß-Seidel*-Verfahren in *Solver.cpp*) und testen Sie Ihre Implementierung mit dem Programm *main-serial.cpp*.

b) Vervollständigen Sie das parallele *cg*-Verfahren und testen Sie Ihre Implementierung mit dem Programm *main-parallel.cpp*.

c) Vervollständigen Sie das parallele *Gauß-Seidel*-Verfahren und testen Sie wiederum Ihre Implementierung.