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 10

(Präsenzübung 1. Juli 2013)

Teil 1: Theorie verstehen (Hausaufgabe!)

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

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

Um diese PDE mit der Finiten Elemente Methode zu lösen, haben wir die letzten 3 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 und
- Algorithmen zum Lösen des linearen Gleichungssystems implementiert.

Das FEM-Projekt soll nun leicht abgeändert und für das Multigrid-Verfahren vorbereitet werden. Das Multigrid-Verfahren ist ein spezieller, sehr effizienter Löser für Gleichungssysteme, die z.B. in der FEM vorkommen.
Für das Multigrid-Verfahren müssen wir uns bei der Verfeinerung den Zusammenhang der Gitter merken. Hierzu dient die Klasse `Hierarchy`. Ein `Hierarchy`-Objekt beschreibt den Zusammenhang zweier Gitter $\mathcal{T}_\ell$ und $\mathcal{T}_{\ell+1}$, wobei $\mathcal{T}_{\ell+1}$ aus $\mathcal{T}_\ell$ durch Verfeinerung entstanden ist, und enthält die folgenden Daten:

- `int numCoarse;` Anzahl der Knoten von $\mathcal{T}_\ell$
- `int numFine;` Anzahl der Knoten von $\mathcal{T}_{\ell+1}$
- `IndexMatrix fathers;` Jeder Knoten x_k von $\mathcal{T}_{\ell+1}$ hat zwei Vater-Knoten in $\mathcal{T}_\ell$. In der k-ten Zeile von `fathers` stehen die Indizes dieser beiden Vater-Knoten.
- `IndexVector fixedNodesD;` Indizes der Dirichlet Knoten von $\mathcal{T}_\ell$.
- `IndexVector fixedNodesDC;` Indizes der Dirichlet und Coupling Knoten von $\mathcal{T}_\ell$.
- `Coupling coupling;` Eine **Kopie** des Coupling-Objektes von $\mathcal{T}_\ell$.
- `CRSMatrix A;` Eine **Kopie** der Galerkin Matrix von $\mathcal{T}_\ell$.

Ein FEM-Objekt enthält einen Vektor (std::vector) von `Hierarchy`-Objekten, ein Objekt für jedes Gitter bzw. jede Verfeinerung, die durchgeführt wurde.

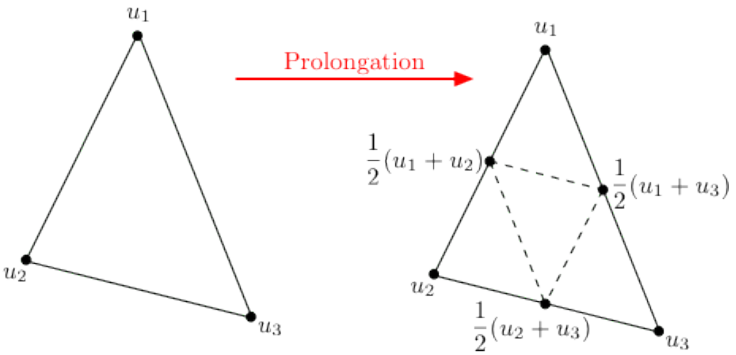
Von der Idee her ist der Ablauf jetzt folgender:

(1) Lese das grobe Gitter ein und erzeuge mit dem groben Gitter ein FEM-Objekt (wie bisher auch)

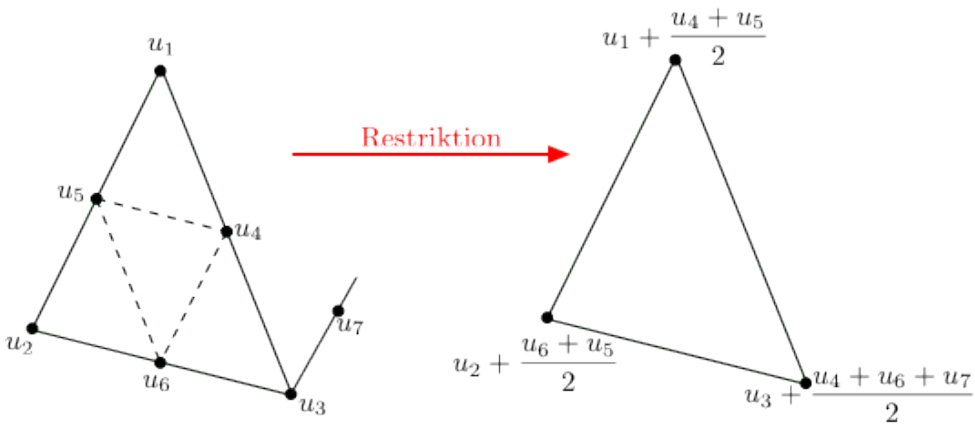
(2) Verfeinere das Gitter des FEM-Objektes einige Male (`fem.refineRed()`). Assembliere für jede Verfeinerungsstufe die Galerkin-Matrix und erstelle ein `Hierarchy`-Objekt (das passiert auch in `fem.refineRed()`)

(3) Starte auf dem feinsten Gitter das Multigrid Verfahren zum Lösen des linearen Gleichungssystems.

Beim Multigrid-Verfahren wird ein Prolongations- und Restriktions-Operator benötigt. Die Prolongation bildet dabei eine Funktion u_ℓ, welche auf einem Gitter $\mathcal{T}_\ell$ definiert ist, auf eine Funktion $u_{\ell+1}$, welche auf dem nächst feineren Gitter $\mathcal{T}_{\ell+1}$ definiert ist, ab. Der Wert der Funktion $u_{\ell+1}$ an den Knoten von $\mathcal{T}_\ell$ wird dabei auf den Wert der gröberen Funktion u_ℓ an diesen Knoten gesetzt. Für die neu hinzugekommenen Knoten wird der Mittelwert der Funktionswerte von u_ℓ an den jeweiligen beiden Vater-Knoten gebildet.



Die Restriktion bildet eine Funktion $u_{\ell+1}$ von einem Gitter $\mathcal{T}_{\ell+1}$ auf eine Funktion u_ℓ, welche auf dem nächst gröberen Gitter $\mathcal{T}_\ell$ definiert ist, ab. Für jeden Funktionswert auf einem Knoten des gröberen Gitters $\mathcal{T}_\ell$ werden der Funktionswert von $u_{\ell+1}$ an diesem Knoten sowie die Hälfte der Funktionswerte an allen Nachbarknoten des feineren Gitters $\mathcal{T}_{\ell+1}$ addiert:



Beide Funktionen sind als Methoden in der Klasse `Hierarchy` implementiert (also in der Datei `hierarchy.cpp`). Es ist zu beachten, dass die parallele Prolongation nur für typeI-Vektoren funktioniert und die parallele Restriktion nur für typeII-Vektoren. Es muss als evtl. eine Typ-Konvertierung durchgeführt werden.

Teil 2: Gegebenen Code verstehen (Hausaufgabe!)

Im Vergleich zum letzten Mal wurden in unserem FEM-Projekt folgende Erweiterungen eingebaut:

- Die Klasse `Hierarchy` wurde eingeführt.
- Die Klasse FEM hat jetzt einen Vektor (std::vector) von Hierarchy-Objekten und die Funktion `refineRed()` in `fem.cpp` muss vervollständigt werden.

Aufgabe

- Schauen Sie den Code von `hierarchy.hpp/cpp` und `FEM.hpp/cpp` durch.
- Schauen Sie die beiden main-Programme im Ordner `examples` durch.

Teil 3: Programmieren (Präsenzaufgabe)

(1) Vervollständigen Sie die `refineRed()` in der Datei `fem.cpp`.

(2) Vervollständigen Sie die Funktion `prolongation` und `restriction` in der Datei `hierarchy.cpp`. Diese Funktionen sind für die **serielle** FEM gedacht.

(3) Vervollständigen Sie die Datei `main-serial.cpp`. Führen Sie das Programm aus und zeichnen Sie die Ergebnisse in Matlab mit Hilfe des Skriptes `plot-serial.m`. Beurteilen und erklären Sie die Ergebnisse.

(4) Vervollständigen Sie die Funktion `prolongation_MPI` und `restriction_MPI` in der Datei `hierarchy.cpp`. Diese Funktionen sind für die **parallele** FEM gedacht.

(5) Vervollständigen Sie die Datei `main-parallel.cpp`. Führen Sie das Programm aus und zeichnen Sie die Ergebnisse in Matlab mit Hilfe des Skriptes `plot-parallel.m`. Beurteilen und erklären Sie die Ergebnisse.