Aufgabe 17 Transformation Lagrange Basis zu hierarchischer Basis

Es sei ein 1D Grobgitter gegeben mit N Elementen der Weite H gegeben. Die feineren Gitter der Weite $\frac{H}{2^l}$ werden durch uniforme Verfeinerung gebildet. Auf jedem dieser Gitter kann man sowohl die Standard Knotenbasis als auch die entsprechen Hierarchische Basis verwenden. Siehe auch Abbildung 1 für eine Darstellung der Basen basierend auf einem Grobgitter bestehend aus 2 Elementen.

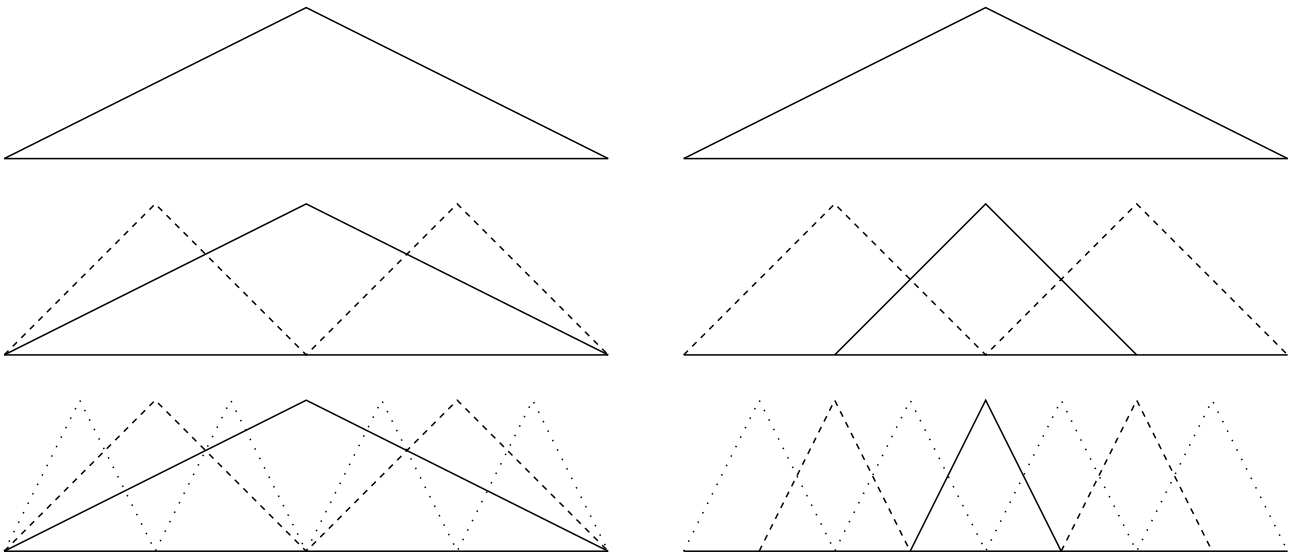

Abbildung 1: Hierarchische (links) gegenüber Standard nodaler Basis (rechts) in 1D

Berechnen Sie die Transformation zwischen den beiden Basen für ein Gitterlevel l. *10 Punkte*

Aufgabe 18 Paralleles Zweigitterverfahren mit Grobgitter

Nach einem Update Ihrer DUNE Module finden Sie ein neues Programm `src/parallel_schwarz_with_coarse_grid.cc`, welches einen parallelen überlappenden Schwarzlöser mit Grobgitterkorrektur bereit stellt. Als Standard werden hier exakte Teilgebietslöser verwendet. Das Programm ist ein 2D Problem und kann nur mit einer quadratischen Anzahl von Prozessoren gestartet werden. Es erwartet zwei Parameter:

a) Das Verfeinerungslevel L. Auf dem verwendeten `YaspGrid` wird L-mal`globalRefine()` aufgerufen. Das heißt, es wird L-mal global uniform verfeinert. Die Aufteilung des Gitters auf Level L auf die Prozessoren stellt die lokalen Teilgebiete dar. Als Grobgitter wird das ursprüngliche Gitter verwendet, welches genau ein Element (+ ensprechenden Überlap) pro Prozess besitzt. Auf diesem Gitter wird das Grobgitterproblem gelöst. (Jeder Prozessor löst das komplette Grobgitterproblem sequentiell.)

b) Den Zweier-Logarithmus ld$o$ des gewünschten Überlapps auf dem feinsten Gitter.

Führen Sie Testrechnungen mit dem Programm aus. Messen Sie dabei folgendes:

- Die benötigte Anzahl an Iterationsschritten für $1, 4, 9, \ldots, 81$ Prozessoren. Nehmen sie als Standard ein maximales Verfeinerungslevel 8 und einen vernünftigen Wert für ld$o$ (z.B. 3 oder 4). Gerne können Sie auch mit den Werten spielen.
- Den Speedup der benötigten Zeit pro Iterationsschritt. Hier müssen Sie natürlich sicher stellen, dass nur ein Prozess pro Prozessor benutzt wird. Der Pool hat maximal 50 Rechner mit je 2 Prozessoren.

Stellen Sie Ihre Ergebnisse graphisch (z.b. mit gnuplot) dar. *6 Punkte*

Aufgabe 19 Paralleles Zweigitterverfahren angewandt auf ein Diffussionsproblem mit heterogenen Permeabilitätsfeld

Das DUNE module `dune-pdelab` stellt unter anderem ein Diffusionsproblem zur Verfügung. Den lokalen Operator namens `Diffusion` findet man in `dune-pdelab/dune/localoperators/diffusion.hh`. Er beschreibt das folgende Problem

$$\begin{aligned} -\nabla \cdot \{K(x)\nabla u\} + a_0 u &= f \text{ in } \Omega, \\ u &= g \text{ on } \partial\Omega_D \\ -(K(x)\nabla u) \cdot \nu &= j \text{ on } \partial\Omega_N \end{aligned}$$

Natürlich bekommt der Konstruktor als Parameter Instanzen der Templateparameter `K`, `A0`, `F`, `B` (beschreibt wo Dirichlet bzw. Neumann Randbedingungen vorliegen) und `J`. Die für das Beispiel-Diffusionsproblem ($K = 1$, $a_0 = 0$, $f = 0$, $j = 0$, $g = 0$) notwendigen Klassen finden Sie im Module `dune-parsolve` im Header `dune-parsolve/src/problemA.hh`. Dieses Problem wurde auch in der letzten Aufgabe benutzt.

In dieser Aufgabe sollen Sie das Programm nun so abändern, dass die Klassen aus `dune-parsolve/src/problemD.hh` benutzt werden. Hier wird die Element-Permeabilität K mit einer vorgegebenen Varianz, Korellationslänge und einem Mittelwert zufällig generiert. In der Klasse `K_D` werden die auf einem Element konstanten Permeabilitätswerte gespeichert. Selbige lassen sich mit der Funktion `getElementPermeability` abfragen. Sie sollen nun das Program so abändern, dass man zumindest Varianz und Korrelationslänge ändern kann. Hierzu finden Sie nach einem `svn update` folgende neue Dateien in Ihrem dune-parsolve Modul:

- `dune-parsolve/src/permeabilitycomponents.hh` enthält eine Klasse mit Namen `LocalPermeabilityDiffusionOperatorHierarchy`, die eine Kopie der Klasse `LocalDiffusionOperatorHierarchy` ist mit noch unvollständigen Änderungen. Der Konstruktor bekommt eine Instanz der Klasse `K_D`, die im Hauptprogramm mit den gewünschten Werten initialisiert werden kann.

- `dune-parsolve/parallel_permeability_twolevel.cc` enthält das Hauptprogram. Hier können die Parameter gesetzt werden.

Ihre Aufgabe besteht nun darin die Methode `interpolate` der Klasse `LocalDiffusionOperatorHierarchy` fertig zu implementieren. Näheres zu der gewünschten Semantik entnehmen Sie bitte der Methoden-Dokumentation im Header.

Es bietet sich an vorher oder auch während der Implementierung die Dokumentation der GridViews `http://www.dune-project.org/doc/doxygen/dune-grid-html/classDune_1_1GridView.html`, und IndexSets `http://www.dune-project.org/doc/doxygen/dune-grid-html/classDune_1_1IndexSet.html` zu lesen. Sowie sich daran zu erinnern, dass man für eine Codim-0-Entität (Element) die Entität, aus der sie durch Verfeinerung hervorgegangen ist, mit der Methode `father()` findet. (Das Gitter wird global verfeinert, es gibt also tatsächlich für jedes Element einen Vater!)

Führen Sie die Messung der Iterationszahlen für verschiedene Prozessorzahlen erneut durch. Nehmen Sie hierzu z.B. als Mittelwert 1, als Varianz 0.2 und als Korellationslänge $H/64$. Eventuell müssen sie das Problem wegen des höheren Speicherverbrauches etwas kleiner machen.

Messen Sie für eine feste Anzahl Prozessoren (z.B. 4), wie sich die Anzahl an benötigten Iterationsschritten bei Änderungen an Varianz und/oder Korellationslänge ($H, H/2, H/4, \ldots, H/64$) ändert. Stellen Sie Ihre Ergebnisse in einer Tabelle dar. *14 Punkte*