ÜBUNG 22 PARALLELES MEHRGITTER

In der Vorlesung wurden die Restriktionen $r_{l,i}$, $R_{l,i}$ und R_l definiert:

Mit $r_{l,i} : \mathbb{R}^{I_l} \to \mathbb{R}^{I_{l,i}}$ bezeichnen wir die Restriktion auf das „Teilgebiet" i, für $x^l \in \mathbb{R}^{I_l}$ ist

$$(r_{l,i}x^l)_j = (x^l)_j \quad \forall j \in I_{l,i}$$

wie bei den Schwarz-Verfahren. Unter $R_l : \mathbb{R}^{I_{l+1}} \to \mathbb{R}^{I_l}$ verstehen wir die Mehrgitterrestriktion, d.h. für $x^{l+1} \in \mathbb{R}^{I_{l+1}}$ ist

$$(R_l x_{l+1})_\alpha = \sum_{\beta \in I_{l+1}} \theta^{l,l+1}_{\alpha,\beta} (x_{l+1})_\beta.$$

Die Beschränkung von R^l auf Teilgebiet i ist $R_{l,i} : \mathbb{R}^{I_{l+1,i}} \to \mathbb{R}^{I_{l,i}}$, und ist gegeben für $x^{l+1,i} \in \mathbb{R}^{I_{l+1,i}}$ durch

$$(R_{l,i}x_{l+1,i})_\alpha = \sum_{\beta \in I_{l+1,i}} \theta^{l,l+1}_{\alpha,\beta} (x_{l+1,i})_\beta.$$

und danach die Bemerkungen 6.4 und 6.5 bewiesen. In dieser Übung betrachten wir andere Eigenscheften von diesen Operatoren.

1. Beweisen Sie, dass diese Behauptung **nicht** allgemeint gilt:

$$R_{l,i}\, r_{l+1,i}\, x_{l+1} = r_{l,i}\, R_l\, x_{l+1} \tag{1}$$

2. Sei $\hat{\Omega}_i$ die überlappende Zerlegung. Sei $\hat{I}_{l,i} \subset I_{l,i}$ mit Eigenschaften

$$\alpha \in \hat{I}_{l,i} \Rightarrow s_\alpha \in \hat{\Omega}_i \vee s_\alpha \in \partial\Omega,$$

dann gilt auch (1) für $\alpha \in \hat{I}_{l,i}$, d.h.

$$(R_{l,i}\, r_{l+1,i}\, x_{l+1})_\alpha = (r_{l,i}\, R_l\, x_{l+1})_\alpha \qquad \alpha \in \hat{I}_{l,i}.$$

3. Beschreiben Sie, was dies für Konsequenzen für die Mehrgitterimplementierung auf **überlappenden** Gittern hat.

10 Punkte

ÜBUNG 23 BPX-VERFAHREN UND ADDITIVES MEHRGITTERVERFAHREN

Nach einem Update Ihres `dune-parsolve` Moduls finden Sie zwei neue Programme im Verzeichnis `uebung/uebung09/`, welche einen Multilevel-Diagonal-Scaling-Verfahren (MDS oder auch BPX) und additives Mehrgitterverfahren implementieren.

Die Programme bekommen 4 Parameter auf der Kommandozeile übergeben:

1. Die Anzahl der Zellen in jeder Richtung (Level 0)
2. Den gewünschten Überlapp auf dem grobsten Gitter
3. Den gewünschten Überlapp auf dem feinsten Gitter
4. Das Verfeinerungslevel L

Auf dem verwendeten `YaspGrid` wird L-mal `globalRefine()` aufgerufen, d.h. 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

Aufgabe 1 Wie könnte man das Ergebnis der vorherigen Aufgabe in der Praxis ausnutzen? Was musste man im Kode ändern?

Aufgabe 2 Führen Sie Testrechnungen mit dem Programm aus. Lassen Sie einmal die globale Grobgittergröße (starke Skalierbarkeit) und einmal die Grobgittergröße pro Prozess (schwache Skalierbarkeit) fest. Nehmen sie als Standard ein feines Gitter mit maximal 1024×1024 Elementen. Messen Sie dabei folgendes für $2, 4, 16, 64$ Prozessoren:

- Die benötigte Anzahl an Iterationsschritten.
- 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.
- Den erreichten minimalen Defekt.

Interpretieren Sie die Ergebnisse. Wie erklären Sie sich das beobachte Verhalten?

Um diese Programme auf mehreren Rechnern im CIP-Pool zu starten, sollten Sie als erstes eine Datei mit den Namen der Rechner erzeugen, die an der Rechnung beteiligt sein sollen. Dazu können Sie im Verzeichnis `dune-parsolve/uebung/uebung09` das Skript

```
./create_mpihosts.sh or bash create_mpihosts.sh
```

aufrufen, dass in die Datei `mpihosts` die Namen derjenigen Rechner schreibt, die im Augenblick nicht ausgelastet sind. Wenn Sie Problem mit `ssh keys` haben(kein Zugang auf andere Rechnern ohne Password), könnten Sie zuerst die Dateien im Verzeichniss `.ssh` löschen:

```
rm ~/.ssh/*
```

und danach Skript `create_mpihosts.sh` aufrufen. Neue `known_hosts` Datei wird erzeugt.

Danach können Sie die parallele Rechnung mittels

```
mpirun -np <p> -machinefile mpihosts ./programm
```

10 Punkte