ÜBUNG 9 ADDITIVE SCHWARZ-ITERATION

In der Vorlesung wurden das additive und das multiplikative Schwarz-Verfahren beschrieben. Die additive Schwarz-Iteration lautet

$$x^{k+1} = x^k + \omega \sum_{i=1}^{p} R_{\hat{I}_i}^T A_{\hat{I}_i\hat{I}_i}^{-1} R_{\hat{I}_i}(b - Ax^k),$$

wobei $\omega \in \mathbb{R}^+$ der Dämpfungsfaktor ist. Beweisen Sie, dass die Matrix $B = \sum_{i=1}^{p} R_{\hat{I}_i}^T A_{\hat{I}_i\hat{I}_i}^{-1} R_{\hat{I}_i}$ positiv definit ist. *4 Punkte*

ÜBUNG 10 GROBGITTERKORREKTUR

In der Vorlesung haben Sie bereits die Grobgitterkorrektur kennengelernt. Diese werden wir jetzt variationell formulieren. Sei $\mathcal{T}^H$ eine Triangulierung des Gebietes Ω und $\mathcal{T}^h$ eine Verfeinerung von $\mathcal{T}^H$. Die zugehörigen Finite-Elemente-Basen bezeichnen wir mit

$$\Phi^H = \{\varphi_i^H \mid i \in \mathcal{I}^H\},$$
$$\Phi^h = \{\varphi_i^h \mid i \in \mathcal{I}^h\}$$

und die Finite-Elemente-Räume mit

$$V^H = \operatorname{span} \Phi^H,$$
$$V^h = \operatorname{span} \Phi^h.$$

Wie definieren nun die Restriktionsabbildung $R^H\colon V^h \to V^H$, die bezüglich der oben angegebenen Basen durch die Matrix

$$R_{ij}^H = \varphi_i^H(x_j), \quad i \in \mathcal{I}^H, \quad j \in \mathcal{I}^h$$

beschrieben ist. Dabei sind die x_j die Knotenpositionen mit $\varphi_i^h(x_j) = \delta_{ij}$.

Sei nun ein $u_h^{(k+\frac{1}{2})}$ gegeben. Die Grobgitterkorrektur $w_H^{(k+\frac{1}{2})} \in V^H$ wird dann durch die Variationsgleichung

$$a(u_h^{(k+\frac{1}{2})} + w_H^{(k+\frac{1}{2})}, v) = l(v) \qquad \forall v \in V^H$$

charakterisiert. Der Startwert für die folgende Iteration ist dann

$$u_h^{(k+1)} = u_h^{(k+\frac{1}{2})} + w_H^{(k+\frac{1}{2})}.$$

1. Zeigen Sie, dass die Gleichung

   $$\varphi_i^H = \sum_{j \in \mathcal{I}^h} R_{ij}^H \varphi_j^h$$

   gilt.

2. Zeigen Sie die Beziehung

   $$R^H A^h (R^H)^T = A^H,$$

   wobei

   $$A^h = a(\varphi_j^h, \varphi_i^h),$$
   $$A^H = a(\varphi_j^H, \varphi_i^H).$$

3. Leiten Sie aus der Variationsformulierung die in der Vorlesung angegebene algebraische Schreibweise der Grobgitterkorrektur her.

6 Punkte

In dieser Aufgabe experimentieren wir erstmals praktisch mit einer Implementierung eines parallelen Lösers. Wir implementieren und vergleichen insgesamt vier Varianten des überlappenden Schwarz-Verfahrens, die sowohl in zwei als auch in drei Dimensionen funktionieren. Für die Bearbeitung dieser Aufgabe haben sie zwei Wochen Zeit, wobei es zweckmäßig ist, zumindest die ersten drei Teilaufgaben in der ersten Woche zu bearbeiten.

Zunächst sollten Sie `dune-parsolve` mittels `svn update` auf den neuesten Stand bringen (neu ist `uebung04`). Falls Sie auf Ihrem eigenen Rechner arbeiten, stellen Sie als nächstes sicher, dass Sie über eine Installation der Bibliothek SuperLU zum direkten Lösen linearer Gleichungssysteme verfügen. In den meisten Linux-Distribution gibt es dafür ein Paket, ansonsten werden Sie leicht im WWW fündig. Nach der Installation von SuperLU sollten Sie in Ihrer DUNE-Optionendatei zu den `CONFIGURE_OPTS` den Parameter `--with-superlu=<Installationspfad>` hinzufügen. (Wenn Sie SuperLU als Paket oder unter `/usr` installiert haben, ist dieser Parameter normalerweise nicht erforderlich.) Danach kompilieren Sie am besten DUNE komplett neu.

Führen Sie nun einige Testrechnungen mit dem Programm `additive_schwarz` im Verzeichnis `dune-parsolve/uebungen/uebung04` durch. Dieses Programm implementiert das additive Schwarz-Verfahren (ASM). Betrachten Sie die Funktionalität des Programms, insbesonders die folgende Klassen:

- `Dune::PDELab::OverlappingOperator`
- `Dune::PDELab::ParallelISTLHelper`
- `Dune::PDELab::OverlappingScalarProduct`
- `Dune::PDELab::SuperLUSubdomainSolver`

Die zweite Variante des Verfahrens heißt Restricted Additive Schwarz Method (RASM), implementiert ist sie in der Datei `restricted_additive_schwarz`. Lesen Sie den Code durch und vergleichen Sie diese Variante mit dem `additive_schwarz` Verfahren. Korrigieren Sie den Bug in `dune-istl 2.2` in der Implementation von `GMResSolver` (Löschen Sie die zwei Zeilen mit der Exception).

Aufgabe 1 Warum kann man in diesem Fall nicht `CGSolver` nutzen? Gehen Sie davon aus, dass man insbesonders die Kommunikation untersuchen muss. Obwohl man Methoden mit unsymmetrischen Vorkonditionierern benutzen muss, haben diese auch Nachteile. Können Sie sie finden?

In der Vorlesung wurde auch das multiplikative Schwarz-Verfahren (MSM) beschrieben. Um diesen Algorithmus parallelisieren zu können, muss man zuerst das Gebiet in Teilgebiete zerlegen. Zum Beispiel zeigt Abbildung 1 eine Zusammenfassung von Teilgebieten, deren Korrekturen sich jeweils gleichzeitig berechnen lassen. Diese Methode heißt `coloring`. Man kann diese Idee auch auf drei Dimensionen erweitern. In dieser Übung benutzen wir nur `Yaspgrid`, das lexikographische Verteilung nutzt. Wie sich die Prozessoren auf das Gebiet verteilen, kann man in Paraview anschauen (Datei `s00XX-decomposition.pvtu`), wenn man das Programm parallel laufen lässt.

Aufgabe 2 In der Datei `multiplicative_schwarz_preconditioner.hh` wurde die Klasse *MultiplicativeSchwarzPreconditioner* schon vorbereitet. Implementieren Sie die Methode `apply`.

Aufgabe 3 Man kann diesen Vorkonditionierer modifizieren, so dass er wieder symmetrisch ist. Überlegen Sie sich, wie diese symmetrische Variante (SMSM) aussehen muss, und implementieren Sie sie in der Klasse *SymmetricMultiplicativeSchwarzPreconditioner*.

Die Hauptaufgabe besteht darin, diese verschiedenen Varianten zu vergleichen. Testen das Programm mit verschiedenen Gittergrößen, mit dem Overlap 1 und 2, mit Raumdimension 2 und 3 und mit Prozessorzahl $\{1, 4, 16, 64\}$.

Aufgabe 4 Geben Sie für die verschiedenen Kombinationen jeweils die Rechenzeiten für das Lösen sowie die Anzahl der benötigten Iterationen in einer Tabelle an.

Beispiel für 2 Raumdimensionen, Overlap 1, Prozessoranzahl 1:

	ASM		RASM		MSM		SMSM	
Gitterweite h	IT	Time	IT	Time	IT	Time	IT	Time
1/32								
1/64								
1/128								
1/256								
1/512								

Um `additive_schwarz` und die andere 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/uebung04` das Skript

```
./create_mpihosts.sh
```

aufrufen, dass in die Datei `mpihosts` die Namen derjenigen Rechner schreibt, die im Augenblick nicht ausgelastet sind. Danach können Sie die parallele Rechnung mittels

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

starten. Für `<p>` setzen Sie die Anzahl der gewünschten Prozesse ein. Achten Sie dabei bitte auf folgende Punkte:

- Rechnen Sie nach Möglichkeit nicht tagsüber, weil dann im Pool Übungen stattfinden. Am Wochenende oder am Abend stören die Rechnungen niemanden.
- Rufen Sie vor jedem parallelen Prozessstart das Skript `create_mpihosts.sh` auf, damit die aktuell nicht ausgelasteten Rechner neu ermittelt werden.
- Achten Sie auf die Anzahl der Rechner, die in Ihrer `mpihosts`-Datei eingetragen wurden. Jeder Rechner verfügt über zwei Prozessorkerne, so dass Sie mit zwei Prozessen pro Rechner optimale Leistung erreichen sollten. Insbesondere die Rechnung mit 64 Prozessen sollten Sie nicht ausführen, wenn nicht mindestens 32 Rechner zur Verfügung stehen.
- Überschlagen Sie vor der Rechnung den Speicherverbrauch. Auf jedem Rechner sollten Sie höchstens 1.5 GB Speicher verbrauchen, damit die Rechner stabil weiterlaufen. Bei Verwendung von CG beträgt der Speicherverbrauch grob 400 Byte pro Unbekannter, bei der Verwendung des direkten Lösers SuperLU ungefähr 2.200 Byte. Berücksichtigen Sie auch, wieviele Prozesse auf einem Rechner entstehen.

Sollten Sie einmal eine parallele Rechnung abbrechen wollen, drücken Sie einfach Strg-C. Dabei kann es passieren, dass auf den Knoten einzelne Prozesse hängebleiben. Diese können Sie mit der Kommandozeile

```
for i in `seq -w 50`; do ssh cip$i killall -9
additive_schwarz; done
```

entfernen.

20 Punkte

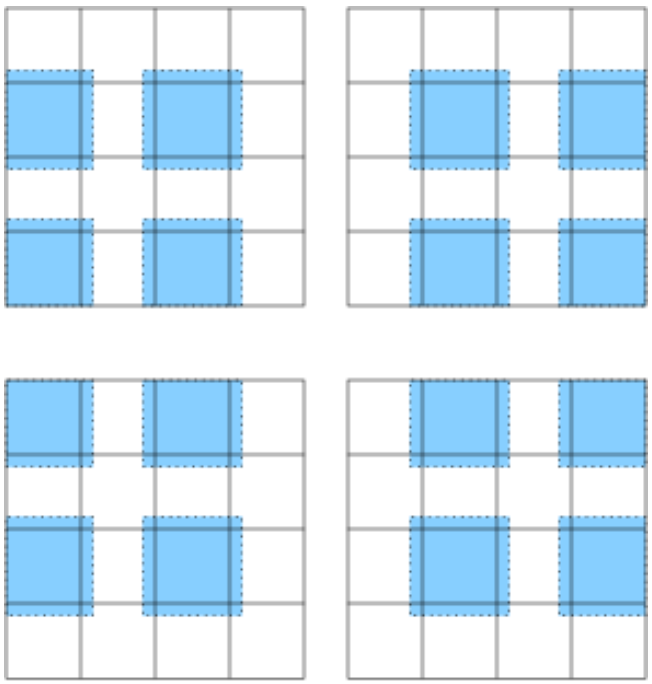

Abbildung 1: Gleichzeitig ausführbare Korrekturen im multiplikativen Schwarz