

AUFGABE 7 DIE PARALLELE RICHARDSON-ITERATION

Wir möchten das Gleichungssystem $Ax = b$ mit der Richardson-Iteration

$$x^{(k+1)} = x^{(k)} + \omega(b - Ax^{(k)})$$

lösen. Die Matrix A sei die Steifigkeitsmatrix einer P^1 -Diskretisierung der Poissongleichung auf dem Einheitsquadrat. Dabei verwenden wir ein strukturiertes Dreiecksgitter mit $N = n^2$ Freiheitsgraden (Dabei bezeichnet n die Anzahl der Freiheitsgrade auf einer Horizontalen oder Vertikalen des Gitters).

Um die Berechnung zu beschleunigen, möchten wir die Iteration parallel durchführen. Dafür unterteilen wir das Einheitsquadrat in p kleinere Quadrate und verteilen die Freiheitsgrade in diesen Untergebieten auf p Prozessoren. (Dabei nehmen wir an, dass p eine Quadratzahl ist; bei vier Prozessoren erhält beispielsweise jeder Prozessor $(n/2)^2$ Freiheitsgrade.) Die Indexmenge aller Freiheitsgrade bezeichnen wir mit I , die des Prozesses k mit I_k . Jeder Prozessor speichert die zu seinen Freiheitsgraden gehörigen Einträge von $x^{(k)}$ und die relevanten Zeilen von A .

Eine Iteration des parallelen Verfahrens besteht nun aus folgenden Schritten:

- Kommunikation der von den Nachbarprozessoren benötigten Einträge von $x^{(k)}$
- Berechnung von $x^{(k+1)}$

- a) Beschreiben Sie die Indexmengen I_k und geben Sie an, mit welchen Prozessoren der Prozessor k welche Einträge von $x^{(k)}$ kommunizieren muss.
- b) Die Rechenzeit für eine beliebige arithmetische Operation (Addition, Subtraktion oder Multiplikation) betrage t_{op} , die Zeit zur Übertragung eines Bytes an einen anderen Prozessor t_{byte} , und die Zeit, um eine Nachricht aufzusetzen, sei t_{msg} . Geben Sie eine Formel für die Gesamt-Rechenzeit für eine Iteration auf p Knoten an. Die Einträge von $x^{(k)}$ seien mit doppelter Genauigkeit gespeichert, so dass jeder Eintrag 8 Byte belegt. (Die Formel soll nur asymptotisch korrekt sein; beispielsweise muss nicht gesondert berücksichtigt werden, dass die Matrixzeilen zu Randknoten weniger Einträge haben.)
- c) Geben Sie tabellarisch den Speedup des parallelen Verfahrens bei folgenden Parametern an:

$$\begin{aligned}t_{\text{op}} &= 2 \text{ ns} \\t_{\text{byte}} &= 20 \text{ ns} \\t_{\text{msg}} &= 5000 \text{ ns} \\n &\in \{1024, 4096\} \\p &\in \{1, 4, 16, 256, 4096\}\end{aligned}$$

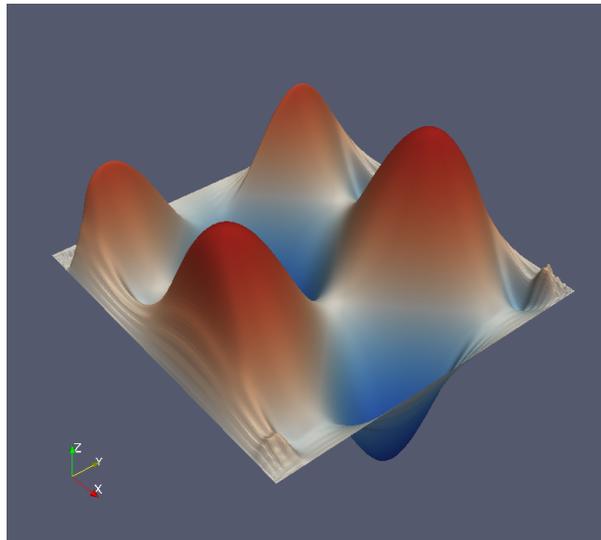
AUFGABE 8 LÖSERKONVERGENZ IN ABHÄNGIGKEIT VON DEN ANFANGSWERTEN

In dieser Aufgabe betrachten wir die Laplace-Gleichung mit homogenen Dirichlet-Randbedingungen

$$\begin{aligned} -\Delta u &= 0 && \text{in } \Omega = (0, 1)^2 \subset \mathbb{R}^2, \\ u &= 0 && \text{auf } \partial\Omega. \end{aligned}$$

Diese Gleichung hat offensichtlich die Lösung $u = 0$. Wir nutzen diese einfache Gleichung, um das Konvergenzverhalten verschiedener Löser zu visualisieren, indem wir verschiedene Startvektoren ungleich 0 vorgeben und beobachten, wie welche Startvektoren durch die iterativen Löser gedämpft werden.

Dazu gibt es im Modul `dune-parsolve` das neue Beispielprogramm `sequentialconvergentest.cc`, im wesentlichen eine vereinfachte Version von `sequentialmodelproblems.cc`. In der vorliegenden Version diskretisiert das Programm das Problem mit Q^1 -Elementen auf einem 256×256 -Gitter und wählt einen niederfrequenten Sinus als Startfunktion. Darauf werden 10 Iteration CG-Löser mit SSOR-Vorkonditionierung angewendet. Die dann entstehende Näherungslösung ist gleich dem Restfehler des linearen Lösers, da die exakte Finite-Elemente-Lösung ebenfalls identisch verschwindet. Diesen Fehler können wir nun mittels ParaView visualisieren:



Schreiben Sie dieses Programm so um, dass es VTK-Ausgaben für alle Kombinationen folgender Komponenten erstellt:

- Löser: Alle Löser, von denen am Beginn des Hauptprogramms Objekte angelegt werden
- Startwerte: Die Startwert-Funktionen

$$\begin{aligned} u_{01}(x) &= 1, \\ u_{02}(x) &= \cos(10x) + \sin(10y), \\ u_{03}(x) &= \cos(100x) + \sin(100y) \end{aligned}$$

- Iterationszahlen: 1, 10 oder 100 Iterationen

Es ist zweckmäßig, das Program dafür ein wenig umzustrukturieren. Beispielsweise dürfte es vorteilhaft sein, die Initialisierung des Lösungsvektors `x` aus der Funktion `single_solve()` in die Funktion `main()` zu verlegen.

Und natürlich sollen Sie die Ausgabedateien auch mit ParaView anschauen... *8 Punkte*