Universität Hamburg
Department Informatik
Wissenschaftliches Rechnen
Prof. T. Ludwig, M. Kuhn, J. Kunkel

Übungsblatt 4 zur Vorlesung
Hochleistungsrechnen
im SoSe 2010
Abgabe: 17.05. bzw. 31.05.

---

1 Partielle Differentialgleichungen (1.050 Punkte)

Mittels iterativer Verfahren (Jacobi, Gauß-Seidel) soll die Poisson-Gleichung für folgende Fälle gelöst werden:

I Störfunktion: $f(x, y) = 0$.
Randbelegung: $v(0, 0) = v(1, 1) = 1$ und $v(1, 0) = v(0, 1) = 0$.
Alle Randwerte zwischen den Ecken sollen linear hochgerechnet werden. Die inneren Werte der Lösungsmatrix sollen auf Null gesetzt werden.

II Störfunktion: $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$.
Randbelegung: $v(\_, 0) = v(\_, 1) = v(0, \_) = v(1, \_) = 0$.
Die inneren Werte der Lösungsmatrix sollen auf Null gesetzt werden.

Wir versetzen Sie jetzt in die typische Situation vieler Diplomanden und Doktoranden, die das Gebiet der parallelen Programmierung betreten: Gegeben ist der Code eines sequentiellen Programms. Er hat eine siebenstellige Zahl von Code-Zeilen, ist in einer Ihnen nicht bekannten Programmiersprache erstellt, außerdem ist er unkommentiert und der ursprüngliche Programmierer ist bereits verstorben. Immerhin aber haben Sie die Quellen (siehe Materialienseite).

Die Quellen beinhalten den kompletten Quelltext, einige weitere Informationen und ein unter Linux ausführbares Programm namens `partdiff-seq`. Verwenden Sie dieses zum Testen des Verfahrens. Spielen Sie ein bisschen mit den Parametern und sehen Sie sich die beigelegten Referenzlösungen an.

Die Problemgröße N wird mittels der Variable `interlines` nach der Formel $N = 9 + 8 \cdot$ `interlines` $- 1$ berechnet. Dadurch wird die Ausgabe mit der Funktion `DisplayMatrix()` überschaubar, die immer genau 9 Zeilen und Spalten ausgibt. (Verwenden Sie diese Ausgabe, um die Richtigkeit Ihrer Änderungen des Programms zu überprüfen.)

Aufgabenstellung

Parallelisieren Sie das Programm mittels Nachrichtenaustausch und MPI. Zunächst soll das Parallelisierungsschema erstellt werden (siehe unten bei der Abgabe). Gehen Sie so vor, dass Sie die gesamte Matrixdaten auf die Prozesse aufteilen, d. h. jeder Prozess bearbeitet einen Teil der Daten. Beachten Sie dabei folgendes: Zur Berechnung der Werte einer Zeile benötigt man immer die Werte der darüber- und der darunterliegenden Zeile. Wenn also die zu berechnende Zeile die erste oder letzte des Blockes eines Prozesses ist, so wird die benötigte Nachbarzeile von einem Nachbarprozess verwaltet; dieser muss sie dann zusenden. Gleichermaßen muss man, wenn man eine Randzeile neu berechnet hat, sie an den entsprechenden Nachbarprozess weitersenden. Die Problematik liegt darin, dies zum richtigen Zeitpunkt mit den richtigen Werten zu tun. Beachten Sie auch, dass der Berechnungsablauf bei den beiden Verfahren unterschiedlich ist und somit auch das Kommunikationsschema unterschiedlich ausfallen wird. Beachten Sie weiterhin, dass der erste und der letzte Prozess keinen vorhergehenden bzw. nachfolgenden Nachbarprozess mehr haben. Um das Speicherverhalten zu optimieren wäre es wünschenswert wenn ein Prozess nur die benötigte Teilmatrix im Speicher halten müsste. Somit können auch Probleme berechnet werden, welche nicht in den Hauptspeicher eines einzelnen Knotens passen.

Insgesamt gibt es vier Varianten, die nun zunächst auf Korrektheit der Parallelisierung geprüft werden müssen: Jeweils das Verfahren nach Jacobi als auch das nach Gauß-Seidel mit den beiden Abbruchkriterien „Anzahl der Iterationen“ bzw. „erreichte Genauigkeit“. Überprüfen Sie, dass bei einer Parallelisierung mit acht Prozessen auf vier Knoten in allen vier Fällen genau dasselbe Ergebnis herauskommt, wie in den zugeordneten sequentiellen Fällen, wenn nach einer festen Iterationszahl beendet wird. Nicht notwendigerweise muss das sequentielle Progamm bei dem Abbruchkriterium „erreichte Genauigkeit“ bei der selben Iteration abbrechen wie das parallele Programm. Klarerweise muss aber bei einer festen Iterationszahl für jedes Verfahren das Ergebnis

unabhängig von der Anzahl der Prozesse stets **identisch** sein. Wichtig: wenn dies nicht der Fall ist, ist Ihr Programm falsch! Wichtig: wenn dies nicht der Fall ist, ist Ihr Programm falsch! :-) Es steht hier doppelt, weil die Chance, dass zunächst ein abweichendes Ergebnis herauskommt wohl gegen 100% geht. Als Eingabeparameter verwenden Sie zunächst die Daten aus der Datei der Referenzlösungen und geben als Abbruchkriterium jeweils die Iterationenzahl oder die Genauigkeit an. Dies garantiert natürlich nicht das ihr Programm in allen Fällen funktioniert.
Diskutieren Sie Auffälligkeiten bei der Realisierung der Beendigung des Programms. Geben Sie für jeden der vier Fälle eine kurze Beschreibung an, welche Probleme aufgetreten sind und welche Mittel Sie zu deren Lösung eingesetzt haben (falls eine Lösung möglich war). Übrigens, es gibt nur einen unkritischen Fall: Jacobi-Verfahren mit gegebener Iterationenzahl.

Punktevergabe

- Parallelisierungsschema – 150 Punkte
- Jacobi – 300 Punkte
- Gauß-Seidel – 600 Punkte

Abgabe des Parallelisierungsschemas (bis 17.05.)

Abzugeben ist eine Textdatei, mit folgendem Inhalt (ca. 1/2 Seite):

- Prosabeschreibung der Datenaufteilung der Matrix auf die einzelnen Prozesse. Welche Daten der Matrix werden von welchem Prozess verwaltet?
- Parallelisierungsschema für das Jacobi-Verfahren. Beschreiben Sie aus Sicht eines Prozesses wann die Kommunikation mit seinen Nachbarn erfolgt und wann die Berechnung – welche Daten benötigt der Prozess von seinen Nachbarn und wann tauscht er die Daten aus? Wann wird dieser Prozess feststellen, dass das Abbruchkriterium erreicht wurde und er seine Arbeit beenden kann? In welcher Iteration befindet sich der Prozess im Vergleich zu seinen Nachbarn, wenn er das Abbruchkriterium erreicht?
- Parallelisierungsschema für das Gauß-Seidel-Verfahren (vgl. Jacobi).

Senden Sie das Archiv per Mail an `michael.kuhn@informatik.uni-hamburg.de`.

Abgabe des Programs (bis 31.05.)

Abzugeben ist ein gemäß den bekannten Richtlinien erstelltes und benanntes Archiv (`.tar.gz`). Das enthaltene und gewohnt benannte Verzeichnis soll folgenden Inhalt haben:

- Alle Quellen, aus denen Ihr Programm besteht; gut dokumentiert (Kommentare bei geänderten Code-Teilen!)
- Ein `Makefile` derart, dass `make partdiff-par` ihr parallelisiertes Programm mit dem Namen `partdiff-par` generiert, das sich **genauso** aufrufen lässt wie das vorgegebene `partdiff-seq`. Auch für das parallele Programm irrelevante Parameter müssen erhalten bleiben. `make clean` und `make` sollen erwartungsgemäße funktionieren. `make` soll dabei natürlich `partdiff-par` erzeugen.
- Eine Datei `diskussion.txt` mit der geforderten Beschreibung zur Problematik der Programmbeendigung.
- **Keine** Binärdateien!

Senden Sie das Archiv per Mail an `michael.kuhn@informatik.uni-hamburg.de`.

Bewertungskriterien

- Strukturierter, gut dokumentierter Programmcode
- Lauffähiges Programm
- Pünktliche Abgabe der Ergebnisse
- Vollständigkeit der geforderten Materialen
- Korrektheit der Form der Abgabe

Die Bearbeitungszeit ist schwer zu schätzen, da sie sehr stark von Ihren Vorkenntnissen und dem Glück, mit dem Sie auf Anhieb eine einigermaßen fehlerfreie MPI-Implementierung hinbekommen, abhängt. Bei komplexen Fehlern kann sich der Aufwand aber leicht stark erhöhen, fangen Sie deshalb **frühzeitig** an. Das bedeutet: sofort in diesen Tagen!
Bitte protokollieren Sie mit, wieviel Zeit Sie benötigt haben. Wieviel davon für die Fehlersuche?

Mathematischer Hintergrund

Viele natürliche und technische Vorgänge lassen sich durch partielle Differentialgleichungen beschreiben. Ein Beispiel hierfür ist die Poisson-Gleichung. Mangels vorhandener analytischer Lösungsformeln muss man sich oft Methoden der numerischen Mathematik bedienen.

Hier gelangt man zunächst durch

(i) Diskretisierung (Festlegung der interessanten Punkte im gewünschten Lösungsgebiet) und

(ii) Ersetzen der Differentialquotienten durch Differenzenquotienten

zu einem System von linearen Gleichungen. Für die Berechnung des Lösungsvektors dieses Systems existieren direkte und indirekte Verfahren. Wegen der Nachteile der direkten Verfahren (Eliminationsverfahren wie z. B. die Gauß-Elimination), nämlich zu hohe algorithmische Komplexität einerseits und numerische Instabilität andererseits, bevorzugt man heute indirekte Verfahren, zum Beispiel Iterationsverfahren, bei denen man sich iterativ bis zu einer gewünschten Genauigkeit der exakten Lösung annähern kann (sofern das Iterationsverfahren konvergiert). Zwei dieser Iterationsverfahren werden hier kurz und pragmatisch vorgestellt.

Problemstellung

Gegeben ist eine partielle Differentialgleichung der Form

$$-u_{xx}(x,y) - u_{yy}(x,y) = f(x,y) \text{ mit } 0 < x, y < 1 \tag{1}$$

Diese Darstellung wird als **Poisson-Problem** bezeichnet. u_{ii} ist die zweite Ableitung der Funktion u nach i. Die Funktion $f(x,y)$ bezeichnet man als **Störfunktion**. Die Randwerte $u(\_,0)$, $u(\_,1)$, $u(0,\_)$ und $u(1,\_)$ sind gegeben. Gesucht wird $u(x,y)$ für $0 < x, y < 1$.

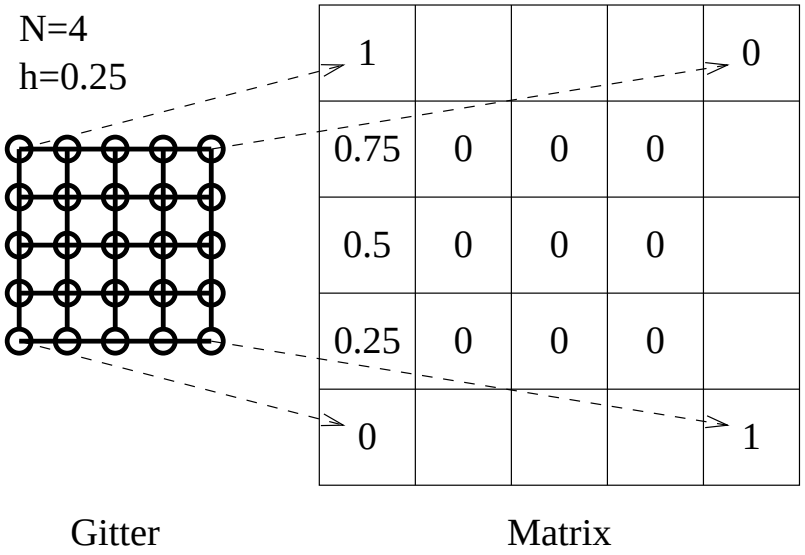
Diskretisierung

Die Lösung der Poisson-Gleichung soll auf dem Gebiet $[0,1]\times[0,1]$ berechnet werden. Einfachste Diskretisierung ist ein **äquidistantes quadratisches Gitter**.

Anzahl Intervalle in jeder Richtung: N
Anzahl Punkte auf Gesamtgebiet (mit Rand): $(N+1)^2$
Anzahl innerer Punkte: $(N-1)^2$
Gitterweite h: $1/N$

Dieses Gitter kann in einer $(N+1)\times(N+1)$-Matrix gespeichert werden. Jeder Eintrag in der Matrix repräsentiert einen Punkt des Gitters. Die Randpunkte des Gitters werden vor Beginn der Berechnung vorbelegt.

Beispiel:



Übergang von Differential- zu Differenzenquotienten

Wir definieren die inneren Gitterpunkte

$$u_{i,j} := u(i * h, j * h) \text{ mit } i, j = 1, ..., N - 1 \tag{2}$$

Die Ersetzung der partiellen Ableitungen aus (1) durch finite Differenzen zweiter Ordnung:

$$u_{xx;i,j} := 1/h^2(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$$
$$u_{yy;i,j} := 1/h^2(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$$

liefert ein **System von linearen Gleichungen** der Form:

$$1/h^2(4v(i,j) - v(i-1,j) - v(i+1,j) - v(i,j-1) - v(i,j+1)) = f(i,j) \tag{3}$$

Vorgehensweise zur Lösung des Systems linearer Gleichungen

Darstellung von (3) in Matrixform: $Au = h^2 f$ (Herleitung und Aufbau der Matrix A ist unbedeutsam für die Anwendung des Iterationsverfahrens.)

Exakte Lösung: u
Näherung: v (soll iterativ berechnet werden, bis v nur noch minimal von u abweicht)
Fehler: $e = u - v$
Leider: u ist unbekannt, d.h. e ist nicht feststellbar.
Aber: berechenbar sind

- **Residuum** $r := h^2 f - Av$ (r ist der Betrag, um den die Näherung von v vom Originalproblem $Au = h^2 f$ entfernt ist)
- **Norm des Residuums** $\| r \|_\infty := \max | r(i,j) |$
- Zusammenhang: $\| r \|_\infty = 0 \Longleftrightarrow e = 0$

Aus $Au = h^2 f$ und $Av = h^2 f - r$ erhält man durch Subtraktion $Ae = r$, bzw. $e = A^{-1} r$, genannt **Residuumsgleichung**.

Zum Berechnen einer Näherung $\hat{e}$ für e:

Verwende in der Residuumsgleichung statt A die einfach zu invertierende Matrix $D = (d_{i,j})$ mit $d_{i,j} = 4\delta(i,j)$ (δ: Kronecker-Symbol).

D ist die Matrix, die aus A durch Nullsetzen sämtlicher Nicht-Hauptdiagonalelemente hervorgeht.

Vorgehensweise:

1. Initiales v^0 berechnen oder raten (einfachheitshalber gleich 0 setzen). Setze $i = 0$.
2. Setze $i = i + 1$. Berechne r^i mittels $r^i = h^2 f - Av^{i-1}$.
3. Berechne $\hat{e}$ mittels $\hat{e} = D^{-1} r$.
4. Berechne neue Näherung $v^i = v^{i-1} + \hat{e}$.
5. Wenn $\| r \|_\infty < Schranke$, Abbruch, sonst zu 2.

Iterative Lösungsmethoden

Prinzip: Erste Näherung raten, dann iterativ verbessern.

Jacobi-Verfahren:
Verwendet zwei Matrizen für v: Die Aktualisierung der neuen Werte erfolgt durch Betrachtung der alten Werte.

Gauß-Seidel-Verfahren:
Verwendet nur eine Matrix für v, d.h. neue Werte werden verwendet, sobald sie berechnet wurden.

Pragmatische Vorgehensweise

Schema für das Programm zur Lösung der Poisson-Gleichung:

```
initialisiere Matrix (Ränder und innere Punkte)
solange (Maximum_Residuum > Schranke)
    über alle Zeilen DO
      über alle Spalten DO
         // berechne den Abtaststern
         star =                    - v[m_old][x][y+1]
               - v[m_old][x-1][y] + 4*v[m_old][x][y] - v[m_old][x+1][y]
                                   - v[m_old][x][y-1];
         // berechne den Korrekturwert
         korrektur = ( f(h*x,h*y) * h_square - star) / 4;
         // für Abbruchbedingung
         berechne Norm von Residuum
         berechne Maximum_Residuum
         // neue Belegung der Matrix
         v[m_new][x][y] = v[m_old][x][y] + korrektur;
         // Gauß-Seidel: m_new = m_old.
```

Literatur (Begleitend und weiterführend)

- Stoer, Bulirsch: Einführung in die numerische Mathematik II, Heidelberger-Taschenbücher, Band 114, Springer
- William H. Press: Numerical Recipes in Pascal/C/Fortran, Cambridge, USA 1990

Anmerkung

Stellen Sie keine Fragen zur Mathematik. Sie müssen das nicht verstehen – nur parallelisieren. ;-)