1 Batch Queuing (30 Punkte)

1. **Frage:** Was bedeutet der Begriff Batch-Queuing?
2. **Frage:** Nennen Sie Beispiele für Batch-Queuing-Systeme!
3. Machen Sie sich mit der Manpage von `sbatch` vertraut.
   - **Frage:** Welches Batch-Queuing-System wird auf dem Cluster verwendet?
   - **Frage:** Gibt es eine Möglichkeit, einen bereits abgeschickten Job zu löschen (bevor oder während er läuft)?

2 Paralleles Starten eines Shell-Scripts (40 Punkte)

1. Erstellen Sie ein Shell-Script `timescript` welches Folgendes nach `stdout` ausgibt:

   `HOSTNAME: TIMESTAMP`

   `HOSTNAME:` Einfacher Hostname des Rechners, auf dem das Script ausgeführt wird.

   `TIMESTAMP:` Zeitstempel zur Zeit der Ausführung des Scripts in einem mindestens auf die Mikrosekunde genauen Format.

   (Tipp: Sehen Sie sich die Manpages von `hostname` und `date` an.)
2. Erstellen Sie ein Job-Script `job_script`, das `timescript` gleichzeitig auf 4 Knoten mit je 4 Prozessen startet. Dabei soll als Ausgabe eine Datei `timescript.out` entstehen, die die Ausgabe von *jedem* Aufruf von `timescript` beinhaltet. Das Script soll mit `sbatch job_script` aus der Shell aufgerufen werden können.
   (Tipp: `mpiexec`, `#SBATCH -N`, `#SBATCH -n`)
3. Nachdem `job_script` die Datei `timescript.out` geschrieben hat, soll es *fertig* ausgeben. Modifizieren Sie das Script so, dass dieses *fertig* in einer Datei mit dem Namen `job_script.out` steht.
4. Führen Sie das Script mehrmals aus.
   - **Frage:** Was fällt Ihnen auf? Versuchen Sie Ihre Beobachtung zu erklären!
   - **Frage:** Könnte man die Datei `timescript.out` auch innerhalb des Scriptes `timescript` erzeugen? Falls ja: Wie? Falls nein: Warum nicht?

3 Leistungsoptimierung (240 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 $N = 9 + 8 \cdot \texttt{interlines} - 1$ 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.)

Im sequentiellen Code sind zwei Algorithmen zur Lösung des angeführten Poissonproblems implementiert (4.4): Das Jacobi-Verfahren und das Gauß-Seidel-Verfahren. Ein Überblick über die mathematischen Grundlagen ist im Anhang gegeben.

In diesem Code sind einige Leistungsprobleme vorhanden, die Sie finden und beheben sollen. Gehen Sie dabei wie folgt vor:

- Kompilieren Sie das Programm mit verschiedenen Compiler-Optionen und -Optimierungen.
- Profilen Sie das Programm mit Hilfe von `gprof`, um zeitintensive Funktionen zu identifizieren und zu optimieren.
- Überprüfen und optimieren Sie die Speicherzugriffsmuster.
- Überlegen Sie sich, ob die mathematischen Berechnungen optimal durchgeführt werden.
- *Hinweis:* Ihre Änderungen sollten sich zum Großteil auf die `calculate()`-Funktion beschränken.

Um `gprof` benutzen zu können, passen Sie das `Makefile` an, um `partdiff-seq` mit der `-pg`-Option zu kompilieren. Lassen Sie danach das Programm laufen, wird eine Datei `gmon.out` geschrieben. Rufen Sie jetzt `gprof ./partdiff-seq` auf, um sich die aufgezeichneten Daten anzusehen.

Dokumentieren Sie Ihre Änderungen und messen Sie für jede Änderung die Verbesserung in der Programmlaufzeit. Die Programmlaufzeit können Sie mit Hilfe von `time` bestimmen, indem Sie einfach `time ./partdiff-seq ...` ausführen. Wählen Sie aussagekräftige Programmlaufzeiten, d. h. mindestens mehrere Sekunden.

Abgabe

Als Abgabe erwarten wir ein gemäß den Vorgaben benanntes komprimiertes Archiv (`.tar.gz`), das ein gemäß den Vorgaben benanntes Verzeichnis mit folgendem Inhalt enthält:

- Eine Datei `antworten.txt` mit Ihren Antworten zu den Aufgaben 1 und 2.
- Die auf dem Cluster ausführbaren Shell- bzw. Job-Scripte `timescript` und `job_script`.
- Eine Datei `timescript.out` mit der Ausgabe eines Durchlaufs Ihres Scriptes (mit mehreren Prozessen).

- Eine Datei `leistungsoptimierung.txt` mit Ihren Ergebnissen zu Aufgabe 3.
  - Beschreiben Sie die von Ihnen vorgenommenen Optimierungen und die dadurch erreichten Leistungsgewinne.
- Der überarbeitete Code des `partdiff-seq`-Programms im Unterverzeichnis `pde`.
- **Keine** Binärdateien.

Senden Sie das Archiv per E-Mail an `hr-abgabe@wr.informatik.uni-hamburg.de`.

4 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.

4.1 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 $0 < x,y < 1$.

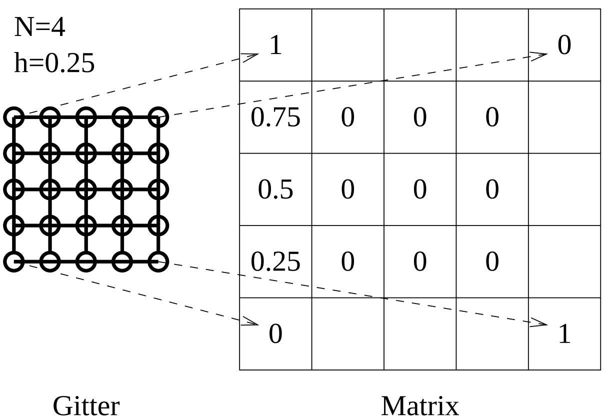
4.2 Diskretisierung

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:



4.3 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 $\| r \|_\infty < \textit{Schranke}$, Abbruch, sonst zu 2.

4.4 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.

4.5 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