Universität Hamburg, Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Prof. Dr. T. Ludwig, Dr. Jannek Squar

Übungsblatt 03 zur Vorlesung
Hochleistungsrechnen
im WiSe 2026/2027
Abgabe: 31.10.2026, 23:59

1 Batch Queuing (80 Punkte)

Beantworten Sie die folgenden Fragen und notieren sie gegebenenfalls die Ausgaben der Befehle. Beachten Sie dabei, dass sich alle Fragen auf die Umgebung des WR-Clusters beziehen. Auf anderen HPC-Systemen sind u.U. andere Konfigurationen im Einsatz.

1. Was bedeutet der Begriff Batch Queuing im HPC-Bereich, welches Problem wird damit gelöst?
2. Nennen Sie drei Beispiele für HPC-Batch-Queuing-Systeme.
3. Welches Batch-Queuing-System wird auf dem WR-Cluster verwendet?
4. Machen Sie sich mit der Manpage von `sbatch` vertraut. Beschreiben Sie die Funktionsweise des Kommandos.
5. Wie lassen sich die aktuellen Jobs und deren Status auf der Kommandozeile anzeigen?
6. Machen Sie sich mit dem Befehl `sview` vertraut. Beschreiben Sie die wichtigsten Vorteile gegenüber dem vorigen Punkt.
7. Gibt es eine Möglichkeit, einen bereits abgeschickten Job zu löschen (bevor oder während er läuft)? Wenn ja, wie?
8. Finden Sie heraus wie Sie sich den detaillierten Status eines Jobs (seine Konfiguration) ausgeben lassen können.
9. Können auf dem Cluster mehrere Nutzer oder Jobs gleichzeitig denselben west-Knoten benutzen?
10. Welche Scheduling-Verfahren sind bei dem auf dem Cluster benutzten System möglich? Welches wird benutzt? Erläutern Sie jedes der Verfahren kurz.
11. Finden Sie heraus, wie Sie einen einzelnen Knoten allokieren können. Allokieren Sie gezielt einen bestimmten Knoten (z. B. `west4`), bauen Sie mittels `ssh` eine Verbindung zu diesem Knoten auf und führen sie `hostname` darauf aus.
12. Wie hoch ist das Zeitlimit auf dem Cluster, bis ein Knoten spätestens wieder freigegeben wird?
13. Welche unterschiedlichen Partitionen sind auf dem Cluster eingerichtet? Wie kann die zu benutzende Partition geändert werden?
14. Wie können Sie die Priorität Ihrer Jobs nachschauen? Können Sie diese verändern oder beeinflussen?

2 Paralleles Starten eines Shell-Scripts (40 Punkte)

1. Erstellen Sie ein Shell-Script `print_output.sh` welches folgende Ausgabe auf `stdout` erzeugt:

   ```
   HOSTNAME // TIMESTAMP
   ```

   HOSTNAME: Kurzer Hostname (`hostname --short`) des Rechners, auf dem das Script ausgeführt wird.

   TIMESTAMP: Zeitstempel zur Zeit der Ausführung des Scripts in einem mindestens auf die Mikrosekunde genauen Format (`date --iso-8601=ns`).

   (**Tipp:** Sehen Sie sich die Manpages von `hostname` und `date` an.)

2. Erstellen Sie ein Job-Script `jobscript.slurm`, das `print_output.sh` gleichzeitig auf 5 Knoten mit jeweils 5 Prozessen startet. Dabei soll als Ausgabe eine einzige Datei `print_output.out` entstehen, die die Ausgabe von **jedem** Aufruf von `print_output.sh` beinhaltet. Das Script soll mit `sbatch jobscript.slurm` aus der Shell aufgerufen werden können.
   (**Tipp:** `srun`, `#SBATCH -N`, `#SBATCH -n`)

3. Nachdem `jobscript.slurm` die Datei `print_output.out` geschrieben hat, soll es „Fertig“ ausgeben. Modifizieren Sie das Script so, dass dieses „Fertig“ in einer Datei mit dem Namen `jobscript.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 `print_output.out` auch innerhalb des Scriptes `print_output.sh` erzeugen? Falls ja: Wie? Falls nein: Warum nicht?

3 Leistungsoptimierung (200 Punkte)

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

I Störfunktion: $f(x,y) = 0$.
Randbelegungen: $v(0,0) = v(1,1) = 3$ und $v(0,1) = 4$ und $v(1,0) = 2$.
Alle Randwerte zwischen den Ecken sollen linear interpoliert 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: $v(0,0) = v(1,1) = v(1,0) = v(0,1) = 0$.
Alle Randwerte zwischen den Ecken sollen ebenfalls auf Null gesetzt werden. 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 signifikante Zahl von Code-Zeilen, ist in einer Ihnen nicht bekannten Programmiersprache erstellt, beinhaltet fehlerhaften Umgang mit dynamischen Speicher, außerdem ist er unkommentiert und der ursprüngliche Programmierer ist bereits verstorben. Immerhin aber haben Sie die Quellen (Material auf der Website).

Die Quellen beinhalten den kompletten Quelltext. Beigelegt sind einzelne Referenzlösungen, sie können aber auch weitere Referenzlösungen mit dem unverändertem Material erstellen.

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. **Tipp:** Verwenden Sie diese Ausgabe, um die Richtigkeit Ihrer Änderungen des Programms zu überprüfen. Dafür können Sie auf selbsterstellte Referenzen nutzen.

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 neben ungünstigem bzw. sogar falschem Memorymanagement auch einige Leistungsprobleme vorhanden, die Sie finden und beheben sollen. Achten Sie darauf, dass der Compiler am Ende keine Warnungen oder Fehler anzeigt. Das Programm sollte auf west-Rechenknoten des WR-Clusters nach Ihren Änderungen ungefähr um einen Faktor von 7 (± 2) schneller laufen. Gehen Sie dabei wie folgt vor, wobei die Auswirkung **jeder** Änderung auf die Laufzeit gemessen werden soll:

1. Decken Sie mithilfe von `valgrind` fehlerhaften Umgang mit dynamischem Speicher auf und korrigieren Sie ihn.

2. Profilen Sie das Programm mit Hilfe von `gprof` und `perf`, um zeitintensive Funktionen zu identifizieren und zu optimieren bzw. um den Effekt ihrer Änderungen nachzuvollziehen. *Hinweis: `perf` steht auf dem Login-Knoten nicht zur Verfügung. Nutzen Sie hierfür ein jobskript, um automatisch `perf` auf einem compute Knoten auszuführen.*

3. Überprüfen und optimieren Sie die Speicherzugriffsmuster.

4. Überlegen Sie sich, ob die mathematischen Berechnungen optimal durchgeführt werden. Passen Sie sie ggfs. an. Das Endergebnis muss natürlich weiterhin korrekt sein. **Wichtig:** Sie dürfen keine Symmetrieeffekte ausnutzen, die im ursprünglichen Code nicht berücksichtigt wurden. Es reicht also nicht aus die Matrix als symmetrisch zu behandeln und nur bis zur Diagonalen zu berechnen.

5. Kompilieren Sie das Programm mit verschiedenen Compiler-Optionen und -Optimierungen. Dafür muss das Makefile angepasst werden. **Frage:** Was sind mögliche Konsequenzen, wenn `-Ofast` benutzt wird? **Wichtig:** Führen Sie die Compileroptimierungen erst ganz am Ende durch, da ansonsten der Effekt ihrer manuellen Änderungen schwerer zu messen ist.

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

Eine weitere Möglichkeit zur Leistungsanalyse bietet das Tool `perf`. Führen Sie Ihr Programm mit `perf stat ./partdiff-seq.x ...` aus um Performance-Statistiken zu erhalten. Erläutern Sie die Bedeutung der einzelnen Statistiken und mögliche Einflüsse auf die Leistung Ihres Programmes. Beachten Sie, dass perf nicht auf dem Login-Knoten verfügbar ist. Um perf zu nutzen, müssen Sie sich einen Rechenknoten aus der *west*

Partition allokieren, darauf einloggen und dort das Programm mit `perf` ausführen. **Erläutern** Sie die Ausgabe.

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.x ...` ausführen. Zu Testzwecken, um die Effektivität Ihrer Optimierungen auszutesten, können sie auch kürzere Läufe verwenden, aber führen Sie `partdiff-seq.x` für Ihre finalen Vergleichsmessungen wie folgt aus:

```
./partdiff-seq.x 1 2 64 1 2 9000
./partdiff-seq.x 1 2 64 2 2 4000
```

Hinweis: Wenn Sie das Speichermanagement nicht verbessert haben, laufen die Aufrufe mit so vielen Iterationen eventuell nicht durch, weil Ihnen der RAM-Speicher ausgeht. Im Idealfall korrigieren Sie das fehlerhafte Speichermanagement, alternativ setzen Sie die Anzahl der Iterationen entsprechend niedriger.
Wichtig: Die Ergebnisse der Matrix als auch der Fehlerwert müssen vor und nach Ihren Änderungen identisch sein! Sie können die Referenzen zur Unterstützung verwenden. Sie können sich aber auch selbst Referenzen erstellen, wenn sie abweichende Läufe zu Testzwecken ausführen wollen. *Hinweis:* Minimale Abweichungen sind nur für Gauß-Seidel akzeptabel, falls sie z.B. Schleifenreihenfolgen notwendigerweise ändern müssen.

Abgabe

Die Abgabe sollte folgenden Inhalt umfassen:

- Ein einzelnes PDF mit Ihren Antworten zu den Aufgaben 1, 2 und 3.
- Die auf dem Cluster ausführbaren Shell- bzw. Job-Scripte `print_output.sh` und `jobscript.slurm`.
- Eine Datei `print_output.out` mit der Ausgabe eines Durchlaufs Ihres Scriptes (mit mehreren Prozessen).
- Im PDF Ihre Ergebnissen zu Aufgabe 3.
  - Beschreiben Sie die einzelnen von Ihnen vorgenommenen Optimierungen und die dadurch erreichten Leistungsgewinne.
  - Fügen Sie mindestens einen Auszug von `gprof` einer Ihrer Messungen ein.
  - Fügen Sie mindestens einen Auszug von `perf` einer Ihrer Messungen ein.
  - Erläutern Sie kurz die Bedeutung der Metriken von `gprof` und `perf`.
- Der überarbeitete Code des `partdiff-seq`-Programms.

Achten Sie darauf, dass Ihre Programme ohne Warnungen oder Fehler bauen und Valgrind keine Probleme anzeigt. Packen Sie ein komprimiertes Archiv (`.tar.gz`) aus dem sauberen Verzeichnis (**ohne Binärdateien oder versteckte Dateien**). Benennen Sie das Archiv nach den Nachnamen der Gruppenmitglieder (z. B. `MustermannMusterfrau.tar.gz`).
Ein Mitglied Ihrer Gruppe legt das Archiv dann in

ab und schickt nach erfolgter Abgabe eine E-Mail an `jannek.squar@uni-hamburg.de`, in der Sie einfach den absoluten Pfad zu Ihrem Archiv angeben:

```
/home/USERNAME/HR-Abgaben-2627/Blatt-03/MustermannMusterfrau.tar.gz.
```

Setzen Sie Ihre Gruppenmitglieder auf CC.

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 $0 < x,y < 1$.

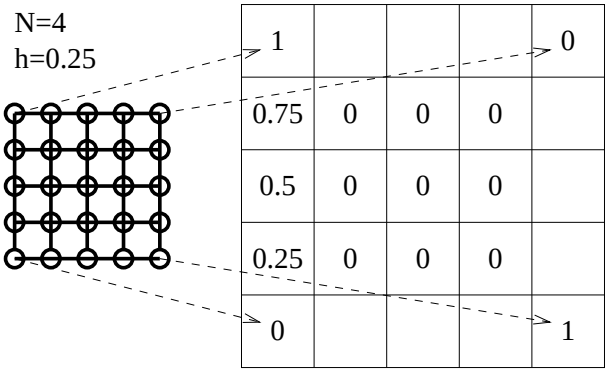
4.2 Diskretisierung

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 $\| r \|_\infty <$ *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