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

Übungsblatt 07 zur Vorlesung
Hochleistungsrechnen
im WiSe 2026/2027
Abgabe: 28.11.2026, 23:59

1 Circle (120 Punkte)

Vervollständigen Sie ein Programm `circle.c`.

Beachten Sie dabei folgende Anforderungen:

1.1 Programmverhalten

- Ein eindimensionales Array der Länge `N` wird auf `nprocs` Prozesse möglichst **gleichmäßig** aufgeteilt.
- Das Teilarray wird mit zufälligen Werten im Bereich 0–9 (nutzen Sie `rand()%10`) initialisiert. Achten Sie darauf, dass die Prozesse nicht die gleichen Zufallszahlen generieren. Jeder Prozess allokiert nur so viel Speicher, wie er für seinen Anteil benötigt. Etwaige notwendige Buffer (Zwischenspeicher) sind hiervon ausgenommen, müssen aber angemessen klein gehalten werden.
- Die Werte sollen ihrer Reihenfolge im Array nach ausgegeben werden (in der Vorlage im Bereich *VORHER*). Rang 0 soll alleine die Ausgabe auf das Terminal übernehmen.
- Die Prozesse sollen einen Kommunikationsring bilden, wobei jeder Prozess mit seinem Vorgänger und dem Nachfolger kommuniziert (also Rang n mit den Rängen $n-1$ und $n+1$); der letzte Prozess kommuniziert dabei mit Rang 0 und umgekehrt.
- In der Funktion `circle` kommunizieren die Prozesse ihre Daten entsprechend weiter.
- In jedem Schleifendurchlauf versendet jeder Prozess seine Daten an seinen Nachfolger und empfängt die Daten seines Vorgängers.
- Dies passiert solange bis der letzte Prozess am Anfang seines Arrays den Wert hat, den der erste Prozess vor der allerersten Iteration ($i == 0$) hatte.

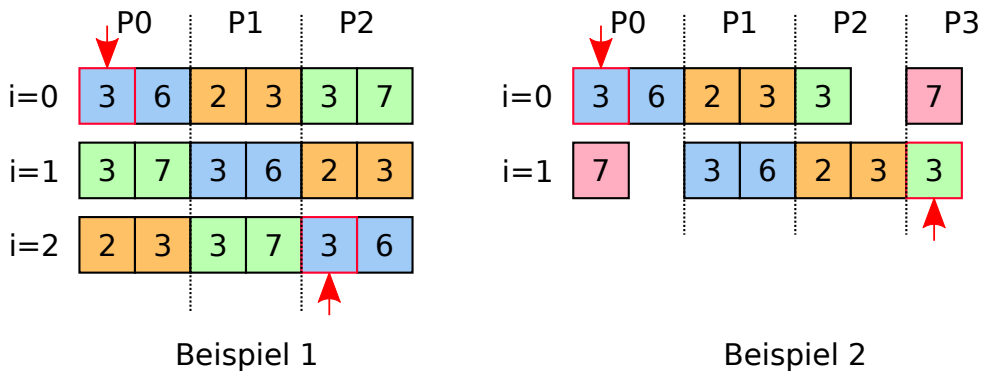


Abbildung 1: Abstrahierte Darstellung des Kommunikationsmusters

- Danach sollen die Werte wieder ihrer Reihenfolge im aktuellen Array nach ausgegeben werden (in der Vorlage im Bereich *NACHHER*). Rang 0 soll alleine die Ausgabe auf das Terminal übernehmen. Zusätzlich soll der erste Prozess die Anzahl der Iterationen und den Abbruchwert ausgeben (im Beispiel 3). Anschließend wird das Programm beendet.
- Bei der Terminal-Ausgabe soll kenntlich gemacht werden, welcher Rang die Werte besessen bzw zwecks Ausgabe an Rang 0 gesendet hat.

```
VORHER
rank 0: 3
rank 0: 6
rank 1: 2
rank 1: 3
rank 2: 3
rank 2: 7

NACHHER
rank 0: 2
rank 0: 3
rank 1: 3
rank 1: 7
rank 2: 3
rank 2: 6

Iterationen: 2
Abbruchwert: 3
```

Listing 1: Beispieloutput

1.2 Code

- `N` wird dem Programm als erstes und einziges Kommandozeilenargument übergeben.
- Das Programm soll mit einer beliebigen Anzahl an Prozessen und einer beliebigen Arraylänge umgehen können. Überlegen Sie sich angemessenes Verhalten für 1 Prozess und `nprocs > N`.
- Ab zwei Prozessen gilt: Zu keinem Zeitpunkt darf ein Prozess das gesamte Array im Speicher halten. Notwendige Buffer minimaler Größe werden dabei nicht beachtet.
- Das Programm muss mindestens eine Iteration laufen.

1.3 Abbruch

Der Abbruch soll **nicht** nach Anzahl der Iterationen erfolgen, sondern nach Eintreten der oben beschriebenen Situation. Beachten Sie, dass dieser Fall auch nach weniger als `nprocs - 1` Iterationen auftreten kann, da derselbe Wert mehrfach vorkommen kann.

1.4 Kommunikation

- Verwenden Sie den synchronen anstelle des standard Sende-Modus. Das bedeutet insbesondere, dass Sie die Funktionen `MPI_Send` und `MPI_Isend` **nicht** verwenden dürfen. Nutzen Sie stattdessen ggf. die Funktionen `MPI_Ssend` und `MPI_Issend`.
- Generell gilt ab sofort: Jeder nichtblockierende Kommunikationsaufruf (meist beginnend mit `MPI_I`...) **muss** mit einem passenden `MPI_Wait` oder einem erfolgreichen `MPI_Test` abgeschlossen werden. Anderenfalls ist der Aufruf falsch.

2 Visualisierung (60 Punkte)

Visualisieren Sie die Kommunikation der Prozesse aus der vorigen Aufgabe mittels Vampir[1]. Gehen Sie dabei in mehreren Schritten vor:

1. Kompilierung des Programms mit Score-P
2. Ausführen des Programms; hierbei entstehen so genannte Spurdaten (Traces). Verwenden Sie 4 Prozesse und $N = 10$
3. Visualisierung der Spurdaten in Vampir

Laden Sie zuerst mit `spack load scorep` das Score-P-Paket. Danach müssen Sie bei der Kompilierung das Kommando `scorep` voran stellen (im Makefile vor `mpicc`), wodurch Ihr Programm automatisch instrumentiert wird. Nach dem Setzen der Umgebungsvariable mit `export SCOREP_ENABLE_TRACING=true` und einem Aufruf des Programms werden Spurdaten generiert; standardmäßig werden alle MPI- und Funktions-Aufrufe erfasst. Die Visualisierung erfolgt dann mittels Vampir:

```
vampir <subfolder>/traces.otf2.
```

Beantworten Sie folgende Fragen und verwenden Sie dabei Screenshots:

1. Wie können Sie in der grafischen Darstellung die Richtung der Kommunikation erkennen?
2. Wie unterscheidet sich die Darstellung von Punkt-zu-Punkt Kommunikation von der Darstellung kollektiver Kommunikation?
3. Lassen Sie sich die Communication Matrix View ausgeben.
4. Markieren Sie die unterschiedlichen Programmphasen (Initialisierung, Iterationen und Beenden) in den Screenshots
5. Wie lange hat die `MPI_Init`-Phase gedauert?
6. Was kann man in der Call Tree Anzeige ablesen?
7. Lassen Sie sich den System Tree anzeigen. Per Rechtsklick kann die angezeigte Metrik im Kontextmenü geändert werden. Lassen Sie sich anzeigen lassen, wie viel Zeit die einzelnen Prozesse in `MPI_Wait` verbracht haben.

[1] `http://www.vampir.eu/`

3 Schema der Parallelisierung mit MPI (150 Punkte)

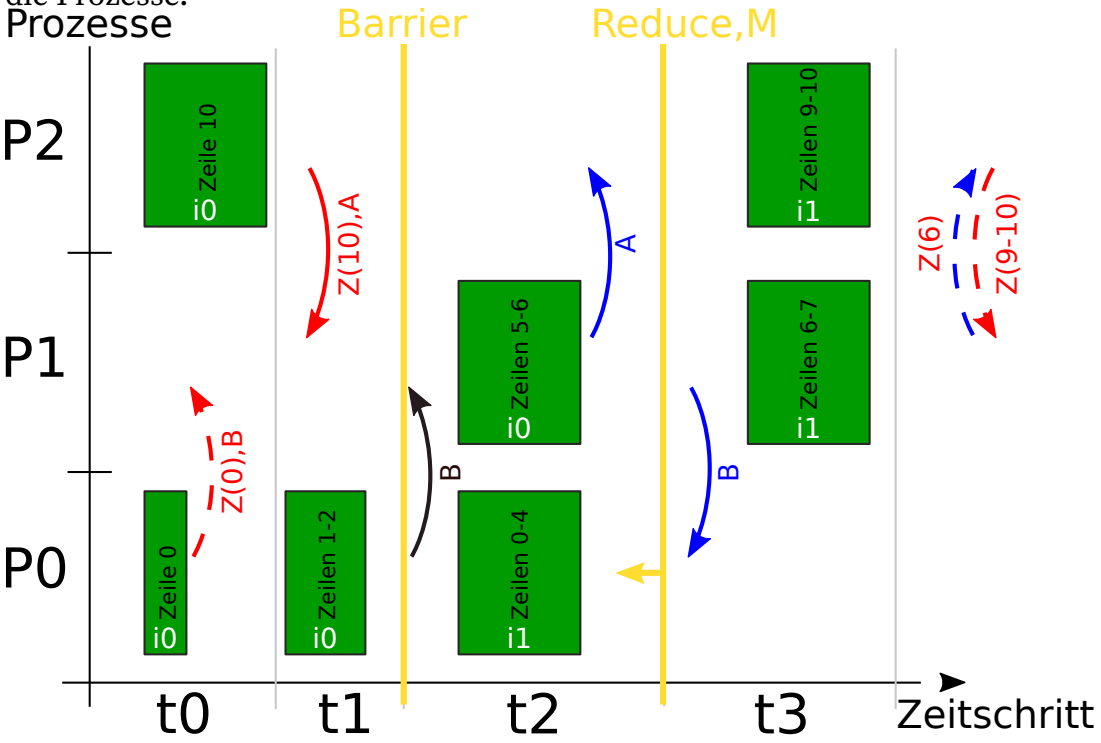
In den kommenden Wochen werden Sie das partdiff-Programm mittels Nachrichtenaustausch und MPI zu parallelisieren. Erstellen Sie hierfür ein Parallelisierungsschema für das **Jacobi**-Verfahren. Beschreiben Sie Probleme und knifflige Stellen, die sich später bei der Implementation ergeben können. Die Visualisierung und erläuternde Texte geben Sie im PDF ab. Das svg-File der Beispielvisualisierung liegt dem Material mit bei, Sie müssen diese Vorlage aber nicht benutzen. Die Formen und Farben müssen aber übereinstimmen.

3.1 Visualisierung Aufteilung

Visualisieren Sie, wie sie die Matrix auf die Prozesse aufteilen für $i = 1$ Interlines und $nprocs = 3$ Prozesse. Kennzeichnen Sie die Daten, die jweils kommuniziert werden müssen und zeigen Sie mit Pfeilen, an welche(n) Prozess(e) diese Daten geschickt werden müssen. Beachten Sie dabei, dass zur Berechnung eines Wertes immer jeweils der Wert oben, unten, links und rechts benötigt wird. Geben Sie für jeden Prozess an, für wie viele Zellen/Zeilen Speicher allokiert werden muss.

3.2 Visualisierung Kommunikation

Visualisieren Sie das Kommunikationsschema für $nprocs = 3$ Prozesse, $i = 1$ Interlines und $T = 3$ Iterationen in einem Diagramm. Legen Sie **je ein** Diagramm für die zwei unterschiedlichen Abbruchbedingungen an. Überlegen Sie sich vorher, mit welchen Operationen (Punkt-zu-Punkt/kollektiv, blockierend/nicht-blockierend) Sie arbeiten möchten. Auf der x-Achse werden abstrakte Zeitschritte aufgetragen, auf der y-Achse die Prozesse.



3.2.1 Zeitschritte

- Ein abstrakter Zeitschritt ist ein ungefährer Zeitraum, in dem Prozesse gleichzeitig arbeiten. Je nach Bedarf können diese feiner oder gröber ausfallen

- Die Zeitschritte müssen nicht gleich lang sein, sie stellen nur einen abstrakten Zeitraum dar.
- Eine lang dauernde Operation darf über mehrere Zeitschritte verteilt werden
- Wichtige Details dürfen nicht einem zu groben Zeitschritt zum Opfer fallen!
- **Wichtig:** Alle Operationen müssen unverzögert ohne künstliche Lücken ohne Bewandnis stattfinden (d.h. so weit links wie möglich), außer Synchronisation o.ä erzwingt eine Verzögerung

3.2.2 Farbschema

Verwenden Sie das folgende Farb- und Objektschema, es ist nicht erlaubt davon abzuweichen (es muss der generelle Farbton stimmen, minimale farbliche Abweichungen sind erlaubt).

- Berechnungsphasen: Grüner Block
  - Jeder Block muss beschriftet werden, zu welcher Iteration er gehört (weiße Beschriftung *iX*) und was inhaltlich passiert
  - In jedem Block müssen die verarbeiteten Zeilen angegeben werden. Verwenden Sie **global** nummerierte Zeilen.
  - Zeitschritte und Iterationen sind nicht das gleiche. Sie können die Zeitschritte beliebig fein auflösen, wenn die Gleichzeitigkeit bestimmter Operationen betont werden soll
  - Unterteilen Sie die Berechnungsphasen innerhalb einer Iteration, wenn es sich anbietet
- Punkt-zu-Punkt Kommunikation:
  - Blockierende Kommunikation: Durchgezogener Pfeil
  - Nicht-blockierende Kommunikation: Gestrichelter Pfeil
  - Nicht-blockierende Kommunikation **muss** abgeschlossen werden: Schwarzer Pfeil
  - Versenden: Roter Pfeil
  - Empfangen: Blauer Pfeil
  - `MPI_Sendrecv`: Eng verbundener blauer und roter Pfeil
  - Beschriftung: Z(x-y) am Pfeil für die zu kommunizierende Zeile(n). Bei Empfangsoperationen können sie die Zeile weglassen, wenn alles von der dazugehörigen Sende-Operation empfangen werden soll
  - Buchstaben (z.B. A, B, C, ...), um zusammengehörige Operationen (z.B. Senden und Empfangen) zu kennzeichnen. Die Zuordnung muss **eindeutig** sein, d.h. Sie dürfen verwendete Buchstaben nicht mehrfach für unterschiedliche Operations-Paare verwenden.
- Kollektive Operationen: Senkrechte, gelbe Linie
  - Wenn es hervorstechende Ränge gibt (z.B. bei einem Reduce oder Broadcast), markieren sie diesen mit einem kleinen Pfeil (siehe Reduce-Beispiel)
  - Wertname an der Linie für den zu kommunizierenden Wert (im Beispiel: `M`)

- *Hinweis:* Nur `MPI_Barrier` bietet eine garantierte Synchronisation. Um die Visualisierung aber übersichtlich und einfacher zu halten, dürfen alle kollektiven Operationen mit einer Vertikalen dargestellt werden, auch wenn sie keine echte Synchronisation erzwingen.

Abgabe

- Den Quelltext des C-Programmes `circle.c` sowie das zugehörige Makefile
- Das PDF mit allen Antworten und Abbildungen
- **Keine** Binär- oder Punkdateien!
- **Keine** Daten, die von scorep erstellt worden sind
- Warnungen von scorep bzgl. Genauigkeit der Zeitstempel können Sie ignorieren

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

`/home/USERNAME/HR-Abgaben-2627/Blatt-07`

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-07/MustermannMusterfrau.tar.gz`.

Setzen Sie Ihre Gruppenmitglieder auf CC.