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

Übungsblatt 11 zur Vorlesung
Hochleistungsrechnen
im WiSe 2025/2026
Abgabe: 24.01.2026, 23:59

1 Leistungsevaluation des parallelen PDE-Lösers (200 Punkte)

Evaluieren Sie die Qualität Ihrer Parallelisierung vom letzten Übungsblatt. Hierfür sollen nachfolgend drei verschiedene Untersuchungen (Weak Scaling, Strong Scaling, Communication) durchgeführt werden.

- Visualisieren Sie für die drei nachstehenden Fälle (1.1, 1.2 und 1.3) die Ergebnisse und interpretieren Sie diese.
- Welche Effekte können beobachtet werden? Interpretieren Sie diese.
- Alle Grafiken, Tabellen etc. müssen beschriftet und verständlich sein.

Durchführung

Um sinnvolle Messungen zu ermöglichen, wird wieder für alle Messungen das Batch-Queueing-System auf dem Cluster genutzt. Jede Messung soll **drei Mal** durchgeführt werden. (**Hinweis:** Die mitgelieferten Job-Scripte wiederholen die Messung bereits drei Mal.)

Die Job-Scripte für die Aufgaben werden auf der Materialienseite der Vorlesung zur Verfügung gestellt. Diese können bei Bedarf auch mit `SLURM_generator.sh` neu generiert werden. Des Weiteren kann das Skript `gather_results.sh` benutzt werden, um aus der Ausgabe der Job-Scripte Tabellen zu erstellen. Diese können zur Visualisierung der gewonnenen Daten genutzt werden.

Definition

Konfiguration (K, P, N) beschreibt einen Programmlauf auf K Knoten mit insgesamt P gleichmäßig auf die Knoten verteilten Prozessen und N Interlines.

1.1 Weak Scaling

Bei Weak Scaling wird die Laufzeit betrachtet, wenn die Problemgröße proportional zur Anzahl der verwendeten Prozesse wächst. Ermitteln Sie zuerst den Speedup Ihres Programms indem Sie es in den folgenden Konfigurationen laufen lassen:

(1, 1, 400), (1, 2, 564), (2, 4, 800), (4, 8, 1128), (4, 16, 1600), (4, 24, 1960), (8, 64, 3200)

- Jacobi, 1.000 Iterationen, Störfunktion $f(x,y) = 2 \cdot \pi^2 \cdot \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$
- Gauß-Seidel, 1.000 Iterationen, Störfunktion $f(x,y) = 2 \cdot \pi^2 \cdot \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$

Frage: Wie erklären Sie sich die Wahl der Interlines in Bezug auf die Prozesszahl?
Frage: Wie sähe die Speedupkurve bei idealem Weak Scaling aus?

1.2 Strong Scaling

Bei Strong Scaling wird die Laufzeit betrachtet, wenn die Problemgröße konstant bleibt, während die Anzahl der Ausführungseinheiten (z.B. Prozesse) erhöht wird.
Lassen Sie Ihr Programm in folgenden Konfigurationen laufen:

(1, 12, N) (2, 24, N), (4, 48, N), (8, 96, N), (10, 120, N), (10, 240, N)

mit $N = 1920$. Nutzen Sie folgende Programmparameter:

- Jacobi, 500 Iterationen, Störfunktion $f(x,y) = 2 \cdot \pi^2 \cdot \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$
- Gauß-Seidel, 500 Iterationen, Störfunktion $f(x,y) = 2 \cdot \pi^2 \cdot \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$

Zeichnen Sie neben dem Speedup auch die Effizienz ein (im Idealfall in die gleiche Grafik, ansonsten in ein eigenes Diagramm).
Frage: Wie sähe die Speedupkurve bzw. Effizienzkurve bei idealem Strong Scaling aus?

1.3 Kommunikation und Teilnutzung der Knoten

Lassen Sie Ihr Programm in folgenden Konfigurationen laufen:

(1, 10, N) (2, 10, N) (3, 10, N) (4, 10, N) (6, 10, N) (8, 10, N) (10, 10, N)

mit $N = 200$. Nutzen Sie folgende Programmparameter:

- Jacobi, Genauigkeit $\leq 3.3504^{-5}$, $f(x,y) = 0$
- Gauss-Seidel, Genauigkeit $\leq 3.3504 \times 10^{-5}$, $f(x,y) = 0$

Frage: Gibt es Auffälligkeiten? Wie erklären Sie sich diese?

2 Manuelles Profiling (180 Punkte)

In dieser Aufgabe sollen Sie mittels manuellem Profiling einen Text-Trace generieren, indem Sie das MPI Profiling Interface benutzen.

2.1 Implementation

Erweitern Sie ihre Implementation von *partdiff* um eine Re-Definition der von Ihnen verwendeten MPI-Kommunikations-Operationen, um ein eigenes Profiling zu implementieren.

- Die re-definierten Funktionen sollen entweder den aktuellen Zeitstempel oder die Zeit, die seit Programmstart vergangen ist, auf das Terminal ausgeben. Sie können z.B. für die Zeitmessung `MPI_Wtime` verwenden. Um die vergangene Zeit seit Programmstart zu bestimmen, können sie eine globale Variable zur Bestimmung eines Startzeitpunkts (Sie können diese direkt nach `MPI_Init` initialisieren) benutzen.
- Die Ausgabe soll auch den Funktionsnamen sowie den Prozessrang beinhalten, damit die Ausgabe dem jeweiligen Prozess zugeordnet werden kann.
- Sollten in Ihrem Code sowohl synchronous als auch standard Sends vorhanden sein, sollten Sie sich für eine Version entscheiden und entsprechend ersetzen. Dann müssen sie weniger Profiling-Funktionen schreiben.
- Andere MPI-Funktionen, die nicht direkt mit dem Datenaustausch zu tun haben (wie z.B. `MPI_Comm_split` oder `MPI_Comm_rank`) müssen nicht neu definiert werden. Beschränken Sie sich daher auf Send, Recv, Wait, Broadcast und Reduce (sowie deren Varianten).
- Beschränken Sie sich auf die Kommunikationsoperationen, die Sie auch tatsächlich in Ihrem Code verwenden.
- Es ist Ihnen überlassen, ob die Zeitmessung nach oder vor der eigentlichen MPI-Funktion ausgeführt wird. Es muss aber konsistent über alle Funktionen hinweg sein.

Hinweis: Sie dürfen den anderen Programmoutput (Anzeige des Headers, der Statistik und der Matrix) für diese Aufgabe deaktivieren, um die Übersichtlichkeit des Text-Traces zu erhöhen, wenn sie wollen.

Ein Ausschnitt der Programm-Ausgabe mit Text-Tracing könnte z.B. so aussehen:

```
[...]
[Rank 1] MPI_Irecv: Seconds since start: 18.230772
[Rank 0] MPI_Isend: Seconds since start: 18.231551
[Rank 0] MPI_Irecv: Seconds since start: 18.231558
[Rank 1] MPI_Wait: Seconds since start: 25.787068
[Rank 1] MPI_Wait: Seconds since start: 25.787111
[Rank 1] MPI_Isend: Seconds since start: 25.787777
[Rank 1] MPI_Irecv: Seconds since start: 25.787786
[Rank 1] MPI_Wait: Seconds since start: 25.787790
[Rank 0] MPI_Wait: Seconds since start: 26.828341
[...]
```

2.2 Auswertung

Generieren Sie mit Ihrer Implementation und zwei Prozessen auf einem Knoten einen Text-Trace für die folgende Konfiguration: `1 1 1000 2 2 3`.
Leiten Sie aus dem Text-Trace ein Sequenz-Diagramm ab. Die Zeit sowie Rang 0 und Rang 1 bilden die drei Spalten. Es geht hierbei nicht primär um die absoluten Zeitwerte sondern vielmehr um den logischen Ablauf der Kommunikation. Daher muss das Sequenz-Diagramm nicht maßstabsgetreu sein, die Zeitangaben sollten nur ungefähre Werte sein. Aber die Reihenfolge der Ereignisse sowie größere Pausen müssen erkennbar sein.

Ein Ausschnitt basierend auf dem obigen Beispiel könnte so aussehen:

Zeit	Rang 0	Rang 1
...	...	...
18s	`MPI_Isend`, `MPI_Irecv`	`MPI_Irecv`
25.7		`MPI_Wait`, `MPI_Wait`
		`MPI_Isend`, `MPI_Irecv`, `MPI_Wait`
26.8s	`MPI_Wait`	
...	...	...

Hinweis: Achten Sie auf die Reihenfolge der Zeitstempel, da die Ausgabe der Prozesse aufgrund ihrer Nebenläufigkeit und den Eigenheiten von Terminal-Output nicht zwangsläufig sortiert ist (es ist nur eine Sortierung innerhalb des eigenen Prozesses garantiert).
Erstellen Sie zusätzlich zu dieser Konfiguration einen Vampir-Trace. Vergleichen Sie den Vampir-Trace mit Ihrem Sequenz-Diagramm. Welche Gemeinsamkeiten und Unterschiede können Sie feststellen?
Abzugeben sind die Implementation, ein Programmoutput mit Ihrem Text-Trace, das zugehörige Sequenz-Diagramm sowie der Vampir-Trace und der Vergleich zwischen beiden.

Abgabe

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

- Alle Quellen, aus denen Ihr Programm besteht
  - Erwartet werden die Dateien `Makefile`, `askparams.c`, `partdiff.c` und `partdiff.h`.
- Ein einzelnes Dokument `antworten.pdf` mit folgendem Inhalt:
  - Antworten aus Aufgabe 1
  - Der Programmoutput von Aufgabe 2 mit eigenem Profiling
  - Das Sequenz-Diagramm aus Aufgabe 2
  - Screenshot des entsprechenden Vampir-Traces
  - Vergleich zwischen Sequenz-Diagramm und Vampir-Trace
- **Keine** Binärdateien!

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/HR-Abgaben-2526/Blatt-11
```

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/HR-Abgaben-2526/Blatt-11/MustermannMusterfrau.tar.gz.
```

Hinweis: `$HOME` ist eine Umgebungsvariable, die auf Ihr Heimatverzeichnis zeigt. Es sollte also nicht wortwörtlich so in der E-Mail stehen.

Führen Sie die Laufskripte aus Aufgabe 1 so früh wie möglich aus, da die Auslastung des Clusters erfahrungsgemäß hoch sein wird. Begründungen für eine verspätete Abgabe wegen einer Überbelegung des Clusters können leider nicht akzeptiert werden.