Institut für Numerische Methoden und Informatik im Bauwesen
Prof. Dr.-Ing. habil. Lutz Lämmer (apl.)

Hochleistungssimulationen im Ingenieurwesen (HSI)
Wintersemester 2018/19

TECHNISCHE UNIVERSITÄT DARMSTADT

Hausübung 2 – MPI

10.12.2018

Übungsabgabe: Fr. 18.01.2019 (23:55 Uhr)

Einführung

Zu realisieren ist die parallelisierte Matrixmultiplikation für quadratische Matrizen A, B und C der Dimension n: C = AB + C. Die parallelisierte Berechnung erfolgt auf einem 2D-Gitter mit p×p Prozessoren. n sei ein ganzzahliges Vielfaches von p. Es soll mit einer identischen, blockweisen Verteilung der Matrizen A, B und C gearbeitet werden. Die Teilblöcke haben die Dimension d=n/p. Für die Matrix A sind die Verteilung auf die Prozessoren im quadratischen Gitter und die Abbildung zwischen den Indizes der globalen Matrix A und der lokalen Blockmatrix A_{kl} unten angegeben.

Blockweise Verteilung

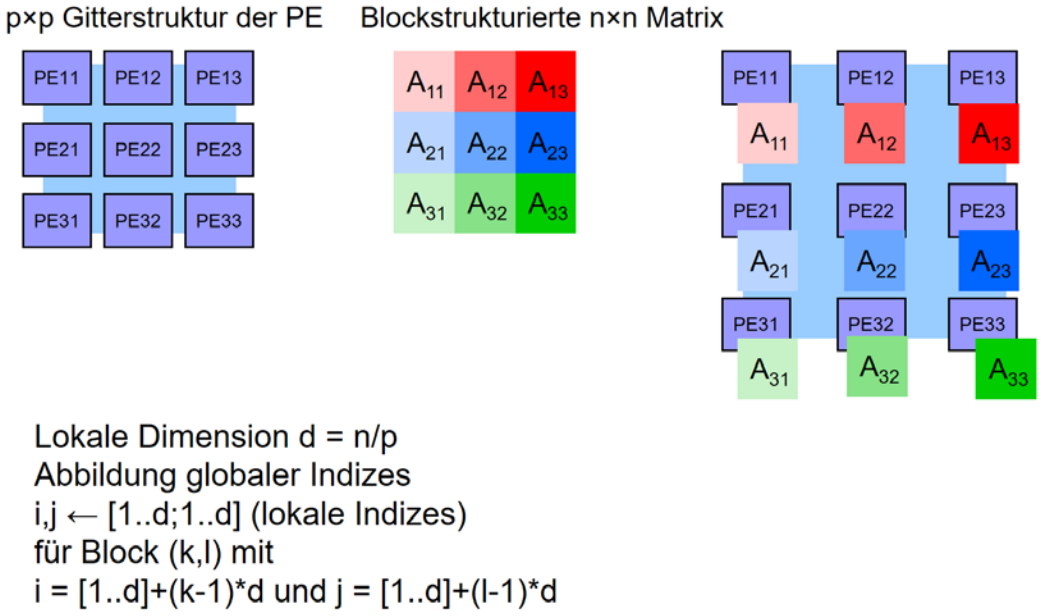

Abbildung 1

Idealerweise ist für die Prozessortopologie ebenfalls ein 2-D-Gitter zu wählen. Die Kommunikation erfolgt dann durch Austausch auf den Achsen des 2-D-Gitters.

Cannon-Algorithmus

Im Folgenden wird der Cannon-Algorithmus für blockweise verteilte Matrizen beschrieben, der im Rahmen der Hausübung umzusetzen ist:

- **Initialisierung von A, B und C mit einer identischen blockweisen Verteilung auf die p^2 Prozessoren.** Wählen Sie für die Verprobung die lokale Blockgröße 2 und eine deterministische und nachvollziehbare Matrixbelegung. Idealerweise ist die Blockgröße aber ein Programmargument und die Belegung erfolgt durch einen einfachen Generator.

- **Initialisierung des Cannon-Algorithmus für A und B.** Diese Operation ist im quadratischen Prozessorgitter so zu implementierten, dass die Blöcke der Matrix A zeilenweise und die Blöcke der Matrix B spaltenweise zyklisch vertauscht werden. In Abbildung 2 ist für eine 3×3 Blockverteilung durch die Pfeile je Spalte bzw. Zeile angegeben, wie oft getauscht werden muss. Es ist das Resultat **nach** der zyklischen Vertauschung angegeben, d.h. für Zeile bzw. Spalte 0 keine, für Zeile bzw. Spalte 1 eine und für Zeile bzw. Spalte 2 zwei Vertauschung(en).

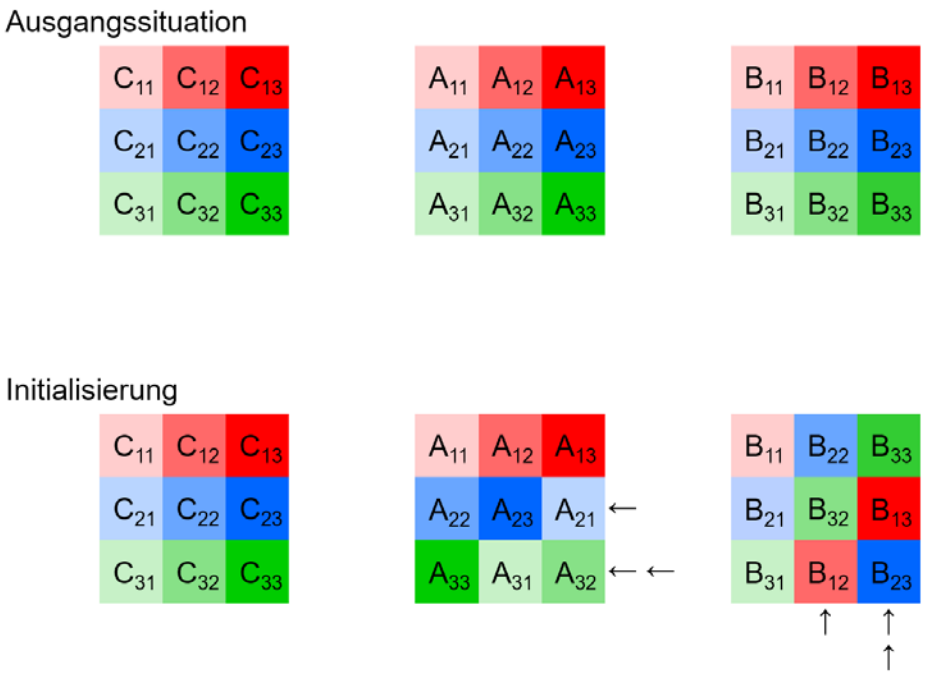

Abbildung 2

- **Cannon-Iteration C = AB+C.** Die Cannon-Iteration erfolgt nun für jede Blockspalte bzw. –zeile genau einmal, bei einer 3×3 Blockung also insgesamt dreimal in den folgenden Schritten:
  1. Lokale Matrixmultiplikation AB und Aktualisierung des lokalen Anteils von C.
  2. Danach erfolgt in dem jeweiligen Iterationsschritt die zyklische Vertauschung aller Blockzeilen von A und aller Blockspalten von B. In Abbildung 3 ist angegeben, wie die Verteilung für die Matrixmultiplikation unter 1. und **vor** der zyklischen Vertauschung aussieht.

  Nach dem letzten Austauschschritt liegen A und B wieder in der Ausgangsverteilung in den Prozessoren vor.

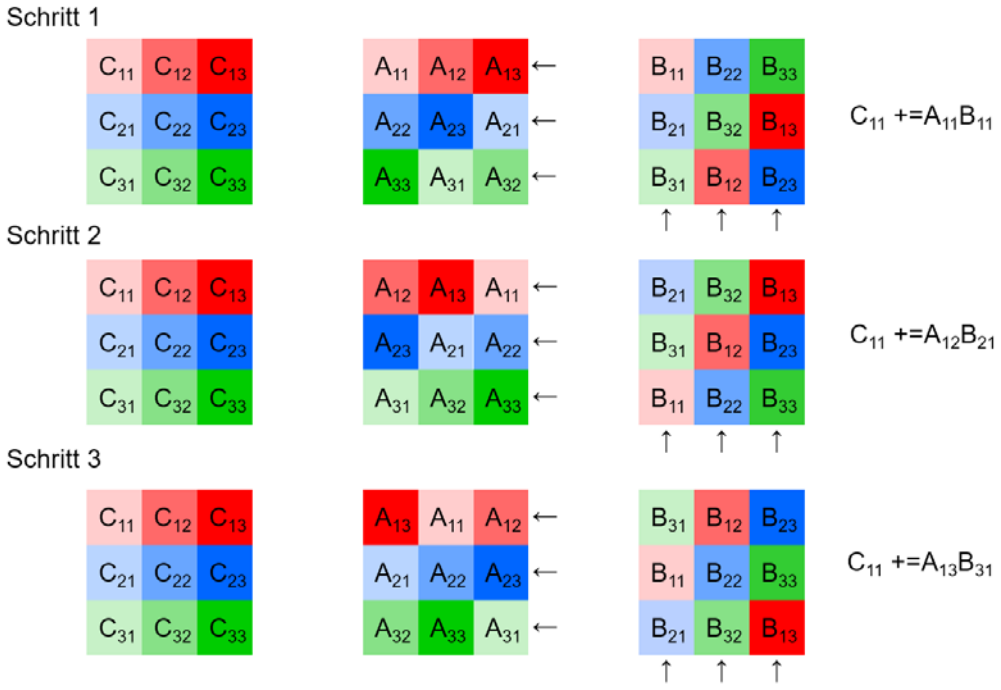

Abbildung 3

Aufgabe

1.1. Sequentieller Algorithmus

Schreiben Sie zunächst ein sequentielles Java-Programm für die Matrixoperation für allgemeine (vollbesetzte) quadratische Matrizen nach folgender Vorgehensweise:

- Initialisierung von A, B und C
- Direkte Berechnung von C = AB + C

1.2. Paralleler Algorithmus

Setzen Sie nun den in der Einführung beschriebenen Algorithmus nach Cannon für eine parallelisierte Lösung der Matrix-Matrix-Multiplikation in Java um. Nutzen Sie hierzu die Java message passing Bibliothek MPJ-Express.

1.3. Zeitmessung

Führen Sie sowohl für die sequentielle als auch die parallele Abarbeitung eine Zeitmessung durch.
Hinweis:
Die CPU-Zeit ist ein guter Anhaltspunkt, die Laufzeit für einen sequentiellen Prozess ohne I/O und andere Einflüsse des Betriebssystems zu ermitteln. Allerdings ist die CPU-Zeit nicht geeignet, die Laufzeit eines parallelen Programms zu ermitteln. Hier ist die tatsächlich verstrichene Zeit (elapsed time – wall-clock time) zu ermitteln, um alle Aspekte des parallelen Programms, insbesondere auch die im Vergleich zur sequentiellen Abarbeitung zusätzlichen Kommunikations- und Synchronisationszeiten zu erfassen. Die Messung sollte auf einer ansonsten nicht weiter belasteten Maschine erfolgen. Üblicherweise sind weitere

Einflussgrößen, wie Start des Message-Passing-Systems, Initialisierung und Beenden des Programms für die Messung nicht von Interesse. Die Zeitmessung sollte daher üblicherweise so erfolgen:

- Alle Prozesse führen unmittelbar vor Beginn des zu messenden Algorithmus eine Barrieroperation zur Synchronisation aus.
- Ein Prozess x misst die lokale Zeit.
- Alle Prozesse führen den zu messenden Algorithmus aus.
- Alle Prozesse führen wieder eine Barrieroperation zur Synchronisation aus.
- Der Prozess x misst die verstrichene Zeit.

Solange die verstrichene Zeit sehr groß im Vergleich zu der Zeit für die Barrieroperation ist, ist das Ergebnis ausreichend genau. Um verlässliche Daten zu ermitteln, ist die Messung mehrfach zu wiederholen und der Durchschnitt der Zeit zu bilden.

1.4. Ausführung auf dem Lichtenberg-Hochleistungsrechner

Schreiben Sie die zur Ausführung des sequentiellen und des parallelen Algorithmus auf dem Lichtenberg-Hochleistungsrechner erforderlichen Batchscripte (Job Script).

1.5. Erwartete Ergebnisse

a) Zeigen Sie, dass die parallele Umsetzung für eine 6×6 Matrix auf 3×3 Prozessoren identische Ergebnisse mit der sequentiellen Implementierung liefert.
b) Untersuchen Sie auf dem Hochleistungsrechner die Laufzeit des sequentiellen und des parallelen Algorithmus für wachsendes n (n = 250 * p * i | i=1…8).
c) Bestimmen und untersuchen Sie überdies für gewählte n den Speed-up in Abhängigkeit von p (bis p = 64). Berücksichtigen Sie hierbei auch verschiedene Verteilungen der Prozesse auf die verfügbaren Knoten, Prozessoren und Kerne.

Abgabe

Die Abgabe muss bis **Freitag, den 18.01.2019, 23:55 Uhr** in 2er-Gruppen über moodle erfolgen. Abzugeben sind (mindestens) der Quellcode der sequentiellen und parallelen Matrix Multiplikation, Job Scripte zur Ausführung auf dem Lichtenberg-HLR, Ergebnisse der Zeitmessung sowie eine Dokumentation der Bearbeitung der Übung. Bitte fassen Sie die Dateien zum Upload in einem gepackten Dateiformat (*.zip, *.7z, *.tar) zusammen.

Weiterführende Links

- Einführung Paralleles Programmieren & Beispiele
  https://computing.llnl.gov/tutorials/parallel_comp/
- MPI
  http://www.mpi-forum.org/docs/docs.html
- MPJ-Express
  http://mpj-express.org/
- A Comprehensive MPI Tutorial Resource
  http://mpitutorial.com/
- Lichtenberg-Hochleistungsrechner
  http://www.hhlr.tu-darmstadt.de
- Slurm Documentation – sbatch
  https://slurm.schedmd.com/sbatch.html