MATHEMATISCHES INSTITUT
DER UNIVERSITÄT ZU KÖLN

Dr. Stephanie Friedhoff, Dr. Martin Lanser

Einführung in HPC
Sommersemester 2016

Übung 4

Hinweis: Schreiben Sie bitte jede Aufgabe auf ein neues Blatt und auf **jedes Blatt Ihren Namen**. Auf die erste Seite Ihrer Übung schreiben Sie bitte zusätzlich zu Ihrem Namen Ihre Matrikelnummer.

Aufgabe 1 (6 Punkte).
Es sei $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ eine Bandmatrix mit unterer Bandbreite $b_l < n$ und oberer Bandbreite $b_u < n$ (typischerweise $b_l, b_u \ll n$), d. h. $a_{ij} = 0$ für $j < i - b_l$ oder $j > i + b_u$.

Um Speicherplatz einzusparen, werden Bandmatrizen „diagonalweise“ als Matrix $\widetilde{A} = (\widetilde{a}_{ij}) \in \mathbb{R}^{n\times(b_l+b_u+1)}$ abgespeichert, indem man die Zuordnung

$$\widetilde{a}_{i,j} = \begin{cases} a_{i,i+j} & \text{für } i = 1,\dots,n,\ j = -b_l,\dots,b_u,\ i+j \in \{1,\dots,n\} \\ 0 & \text{sonst} \end{cases}$$

verwendet (vgl. Abschnitt 2.4.2 der Vorlesung). Die Zeilen von $\widetilde{A}$ werden dabei also wie gewohnt von 1 bis n durchnummeriert, die Spaltenindizes jedoch von $-b_l$ bis b_u. Für $n = 6$ sowie $b_l = 1$ und $b_u = 2$ ergibt sich z. B.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & a_{46} \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{pmatrix} \quad \text{und} \quad \widetilde{A} = \begin{pmatrix} 0 & a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} & a_{45} \\ a_{43} & a_{44} & a_{45} & a_{46} \\ a_{54} & a_{55} & a_{56} & 0 \\ a_{65} & a_{66} & 0 & 0 \end{pmatrix}.$$

Nun sei $\widetilde{A}$ auf p Prozessoren spaltenweise verteilt, d. h. jeder Prozessor hat Zugriff auf eine Teilmenge von kompletten Spalten der Matrix. Der Vektor $x \in \mathbb{R}^n$ sei auf jedem Prozessor vorhanden.

Entwerfen Sie einen Pseudocode für die **effiziente** Matrix-Vektor-Multiplikation $b = Ax$. Der Ergebnisvektor $b \in \mathbb{R}^n$ soll nach Abschluss der Berechnung vollständig auf einem (beliebigen) Prozessor vorhanden sein.

Aufgabe 2 (3 + 3 + 2 Punkte).
Gegeben seien zwei Arrays x, y von Integers der Länge n, wobei n ein Vielfaches von 100 sei. Betrachten Sie drei verschiedene Varianten die Arrays komponentenweise zu addieren. Dies soll seriell auf einem Prozessor geschehen.

Variante 1:

for $i = 0$ to $n - 1$
    x[i] += y[i]
end

Variante 2:

for $i = 0$ to 99
    for $j = 0$ to $n/100 - 1$
        $k = i * n/100 + j$
        x[k] += y[k]
    end
end

Variante 3:

for $i = 0$ to 99
    for $j = 0$ to $n/100 - 1$
        $k = i + j * 100$
        x[k] += y[k]
    end
end

(a) Nehmen Sie eine konstante Zeit t_a für eine Addition oder eine Addition plus Multiplikation an. Vergleichen Sie die zu erwartende Rechenzeit der drei Varianten untereinander.

(b) Implementieren Sie alle drei Varianten in C und vergleichen Sie deren Laufzeit für $n = 1.000.000$. Nutzen Sie

(i) eine „einfache“ Übersetzung mit `gcc name_programm.c` und

(ii) eine optimierte Übersetzung mit `gcc -O3 name_programm.c`

(c) Falls die gemessenen Ergebnisse qualitativ nicht mit Ihren Vorhersagen aus (a) übereinstimmen: Bergründen Sie die unterschiedlichen Laufzeiten der drei Varianten.

Aufgabe 3 (2 + 1 + 1 + 1 Punkte).

(a) Geben Sie die beiden top-looking Varianten der Gauß-Elimination *ohne Pivotisierung* an.

(b) Beschreiben Sie für beide Varianten, welche Operation in der innersten Schleife durchgeführt wird (z. B. Addition des Vielfachen einer Zeile zu einer anderen, ...).

(c) Für welche top-looking Variante ist es möglich, die gesamte innerste Schleife durch eine BLAS-Routine zu ersetzen? Welche BLAS-Routine wäre das?

(d) Ist eine Spalten-Pivotsuche für eine der beiden (oder beide) top-looking Varianten sinnvoll? Warum / Warum nicht? Wie sieht es mit einer Zeilen-Pivotsuche aus?

Programmieraufgabe 3 (2 Punkte).
Gegeben sei ein Parallelrechner mit Ringtopologie bestehend aus $p > 1$ Prozessoren $P_0, P_1, \ldots, P_{p-1}$. Implementieren Sie ein paralleles Programm mit p MPI Prozessen (Ränge 0 bis $p-1$), das ein Senden im Ring ohne Deadlock durchführt. Jeder Prozess P_i, $i = 0, \ldots, p$ soll dabei seinen Rang an seinen linken Nachbarn, $P_{(i-1) \bmod p}$, schicken und im Anschluss seinen eigenen Rang sowie den empfangenen Rang ausgeben. Verwenden Sie dabei nicht-blockierende Kommunikationsroutinen.

Hinweise:

- Nicht-blockierende Kommunikation lässt sich in 3 Phasen einteilen:
  1. Kommunikation anstoßen
     - MPI-Funktionsnamen enthalten ein „**I**“ für **immediate**
     - Die Rückkehr aus nicht-blockierenden MPI-Funktionen erfolgt sofort.
     - Nicht-blockierende MPI-Funktionen haben als zusätzliches Argument ein **Request-Objekt**, das benötigt wird um abzufragen, ob die Kommunikation beendet ist.
  2. Andere Arbeit (z. B. weitere Kommunikation oder Berechnungen) fortsetzen
  3. Kommunikation beenden: Kommunikation erfolgt mithilfe von Puffern. Soll auf diese Puffer zugegriffen werden, muss man warten, bis die Kommunikation beendet ist.
- Schauen Sie sich die MPI-Funktionen **MPI_ISend**, **MPI_IRecv** und **MPI_Wait** für die nicht-blockierende Kommunikation an.

Abgabedatum: 30. Mai 2016 bis 12:00 Uhr im entsprechenden Kasten in Raum 3.01 des Mathematischen Instituts oder am Ende der Vorlesung.