Praktikum

Parallele Programmierung 

Wintersemester 2015/16

Scienti c Computing / Angewandte Informatik

Bergische Universit•at Wuppertal

Prof. Dr. Andreas Frommer,

Dr. Karsten Kahl,

Prof. Dr. Bruno Lang

* Voraussetzungen

Fur• die Bearbeitung der Aufgaben dieses Praktikums sollten Sie (mindestens) eine der Pro-grammiersprachen C/C++ oder Fortran77/Fortran90/Fortran95 beherrschen. Gegebenenfalls sollten Sie sich diese Kenntnisse jetzt aneignen.

Der Besuch der Vorlesung Parallele Algorithmen wird nicht explizit vorausgesetzt, jedoch durf•-te die Kenntnis der Vorlesungsinhalte die Bearbeitung der Aufgaben deutlich erleichtern. Es ist vielleicht hilfreich, die Mitschrift von Kommilitonen zu kopieren. Ein Skript zur Vorlesung nden Sie als "Skript AUD II\ auf der Home-Page des Praktikums

http://www-ai.math.uni-wuppertal.de/SciComp/teaching/ss14/praktikum

Die Aufgaben sollten in Zweiergruppen bearbeitet werden.

Fur• das Erreichen des Scheins sind die Bearbeitung

des "einfuhrenden\• Blocks (Aufgaben 1{3) sowie zweier der drei thematischen Blocke•

{ Direkte L•osung linearer Systeme (Aufgaben 4{5) { Iterative L•osung linearer Systeme (Aufgaben 6{7)

{ Berechnung von Eigenwerten und -vektoren (zwei der Aufgaben 8{10)

hinreichend. Dabei ist

bei den Aufgaben 1, 2 und 4 nur eine Parallelisierung mit MPI,

bei den ubrigen• Aufgaben (3, 5{10) sowohl eine Parallelisierung mit MPI als auch eine Parallelisierung mit OpenMP durchzufuhren•.

Die Abgabetermine fur• die bearbeiteten Aufgaben (Vorfuhrung• und Listings) werden in der Vorbesprechung bekanntgegeben.

2

* Literatur

Die parallelen Programme sollen zum Einen mit Hilfe der MPI-Kommunikationsbibliothek (Message Passing Interface) erstellt werden. Als Einstieg in MPI konnte• etwa die unter

http://moss.csc.ncsu.edu/emueller/cluster/mpi.guide.pdf

erhaltliche• Kurzeinfuhrung• von Peter Pacheco dienen, evtl. erganzt• durch Material aus

http://www-unix.mcs.anl.gov/mpi/tutorial/

Daruber• hinaus bietet

http://www-unix.mcs.anl.gov/mpi/

einen Einstieg in die "MPI-Welt\ mit vielen weiteren Dokumenten (insbesondere den De - nitionen der Standards MPI-1.1 und MPI-2.0) und frei verfugbaren• MPI-Implementierungen (u.a. fur• Linux). Dort sind auch Verweise auf die Lehrbucher• von Gropp, Lusk und Skjellum bzw. Pacheco zu nden.

Es ist empfehlenswert, zumindest Pachecos Kurzeinfuhrung• durchzuarbeiten, bevor mit der MPI-Programmierung begonnen wird.

Fur• die Programmierung mit OpenMP sollte die unter

http://www.openmp.org

erhaltliche• Dokumentation (insbesondere der OpenMP-"Standard\ fur• die jeweils benutzte Programmiersprache) verwendet werden.

3

* Allgemeine Hinweise

Die Aufgaben mussen• naturlich• nicht in der angegebenen Reihenfolge gelost• werden. Es ist jedoch empfehlenswert, zumindest die einfuhrenden• Aufgaben 1{3 zuerst zu be-arbeiten.

Die nachfolgenden Beschreibungen (z.B. der Datenverteilungen etc.) beziehen sich auf die MPI-Parallelisierung. Bei OpenMP liegt ein gemeinsamer Speicher zugrunde, so dass die Daten nicht verteilt werden mussen•. Dies fuhrt• zu einer gegenuber• MPI deutlich vereinfachten Parallelisierung.

Fur• die OpenMP-Parallelisierung gehen Sie am besten von einem seriellen Programm aus, in das dann geeignete OpenMP-Direktiven eingefugt• werden.

Fuhren• Sie numerische Rechnungen grundsatzlich• in doppelter Genauigkeit durch und speichern Sie die Daten (reelle Zahlen, Vektoren und Matrizen) in einem entsprechenden Format (double bei C/C++ bzw. double precision bei Fortran).

4

4 Block 0 | Einfuhrende• Aufgaben

Aufgabe 1 (1 Punkt)

Schreiben Sie ein Programm, welches auf jedem Prozess die Anzahl der aktiven Prozesse sowie die Nummer des eigenen Prozesses bestimmt und ausgibt.

Testen Sie Ihr Programm fur• p = 1; 2; 4; 7.

Auf p = 4 Prozessen sollte die Ausgabe etwa wie folgt aussehen:

Program running on 4 processes, I am process no. 0 Program running on 4 processes, I am process no. 3 Program running on 4 processes, I am process no. 1 Program running on 4 processes, I am process no. 2

Aufgabe 2 (2 Punkte)

Schreiben Sie ein Programm, welches

1. in Prozess 0 eine Zahl vom Benutzer einliest und
2. diese dann durch den Prozessring an alle anderen Prozesse weiterreicht und dabei in jedem Prozess verdoppelt und ausgibt.

Testen Sie Ihr Programm fur• p = 1; 2; 4; 7.

Hinweis: Die Prozesse sind standardma• ig als Ring kon guriert.

Auf p = 4 Prozessen sollte die Ausgabe etwa wie folgt aussehen:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Enter a number in [ 0, 1000 ] : 12.5 | | | |  |  |
| Process | 0 | got | 12.5000 | and computed | 25.0000 |
| Process | 1 | got | 25.0000 | and computed | 50.0000 |
| Process | 3 | got | 100.0000 | and computed | 200.0000 |
| Process | 2 | got | 50.0000 | and computed | 100.0000 |

Aufgabe 3 (4 Punkte)

Implementieren Sie die Simpson-Regel mit n Teilintervallen zur Berechnung von

b

Z

s f(x) dx

a

5

und messen Sie die benotigte• Zeit.

Testen Sie Ihr Programm fur• [a; b] = [0; 2],

f(x) = x***16***

und n = 512; 1024; 65536 und p = 1; 4; 7.

Beschreibung: Bei der Simpson-Regel wird der Integrationsbereich [a; b] durch n + 1 aqui•-distante Punkte

xi = a + ih ; i = 0; : : : ; n;

mit h = (b a)=n in n Intervalle [xi; xi***+1***] mit zugehorigen• Mittelpunkten

|  |  |  |  |
| --- | --- | --- | --- |
| xi***+*** ***21*** = | xi + xi***+1*** | ; i = 0; : : : ; n 1; |  |
| 2 |  |

zerlegt und das Integral durch die Summe

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s = | h n ***1*** | | f(xi) + 4f(xi***+*** | ***21*** ) +f(xi***+1***) |  |
| 6 i***=0*** | |  |
|  |  | X |  |  |  |

approximiert.

Hinweis: Um eine aussagekraftige• Laufzeit zu erhalten, sollten Sie vor Beginn der Zeitmes-sung alle Prozesse mit einer Barriere synchronisieren und am Ende das Maximum uber• die lokalen Zeiten bestimmen.

Auf p = 4 Prozessen sollte die Ausgabe etwa wie folgt aussehen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enter the interval bounds (a, b) | | | : 0 2 |  |
| Enter the number of subintervals | | | (n) : 65536 |  |
| Simpsons Rule for [ | | 0.0000, | 2.0000 ] with | 65536 subintervals |
| yielded the approximation 7710.117647058816 | | | |  |
| and took | 0.042057 seconds | |  |  |

6

* Block 1 | Direkte Losung• linearer Systeme

Aufgabe 4 (6 Punkte)

Eine Matrix A 2 Rn n hei t spaltenzyklisch auf die Prozesse P***1***; : : : ; Pp verteilt, falls Pi genau die Spalten j von A mit j i mod p enthalt•.

1. Schreiben Sie eine Routine DistributeColumns, welche eine in Prozess Pi vollstandig• vorliegende Matrix A 2 Rm n spaltenzyklisch auf die Prozesse verteilt.
2. Schreiben Sie eine Routine CollectColumns, welche eine spaltenzyklisch auf die Pro-zesse verteilte Matrix A 2 Rm n in einem Prozess Pi rekonstruiert.
3. Schreiben Sie eine Routine Transpose, welche zu einer spaltenzyklisch auf die Prozesse verteilten Matrix A 2 Rm n die spaltenzyklisch auf die Prozesse verteilte Transponierte

AT bestimmt, ohne A oder AT in einem einzigen Prozess aufzubauen.

Es sollen jeweils keine einzelnen Elemente der Matrix verschickt, sondern so wenige Kom-munikationsoperationen wie moglich• verwendet werden.

Testen Sie Ihre Routinen, indem Sie etwa fur• verschiedene m; n und p = 1; 3; 4 Prozesse die Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A = (aij) 2 Rm n | mit aij = i + |  | j |  |
|  |  |  |
| 1000 | |  |

jeweils

1. lokal in einem Prozess Pi erzeugen,
2. von dort aus mit DistributeColumns spaltenzyklisch verteilen,
3. mit Transpose (verteilt) transponieren,
4. die Matrix AT mit CollectColumns wieder in Pi einsammeln und
5. dort mit AT vergleichen.

Aufgabe 5 (10 Punkte)

Implementieren Sie einen parallelen Loser• fur• lineare Gleichungssysteme der Gestalt

AX = B; A 2 Rn n; B 2 Rn q;

so dass mehrere Systeme

Ax***(***i***)*** = b***(***i***)***; A 2 Rn n; b***(***i***)*** 2 Rn (i = 1; : : : ; q);

mit gleicher Matrix A gelost• werden konnen•. Gehen Sie hierzu folgenderma en vor.

7

a) Realisieren sie den Gau -Algorithmus mit Spaltenpivotsuche. Transformieren Sie dabei zusatzlich• die rechten Seiten B gleich mit, d.h. arbeiten Sie mit der Matrix ( A j B ) statt A. Gehen Sie hierfur• von der folgenden seriellen (kji-)Form des Gau -Algorithmus aus:

fur• k = 1 : n

f Spaltenpivotsuche g

bestimme betragsgro• tes Element apk;k in A(k : n; k) vertausche Zeilen k und pk von A

f Multiplikatoren bestimmen g fur• i = k + 1 : n

aik := aik=akk

fur• j = k + 1 : n + q

f ( akj)-faches der Multiplikatorenspalte zu Spalte j addieren g fur• i = k + 1 : n

aij := aij aikakj

Verwenden Sie eine spaltenzyklische Speicherung der Daten.

b) Im obigen Gau -Algorithmus wird ( A j B ) uberschrieben• durch

( U j Be ) = ( L ***1***P T A j L ***1***P T B )

mit P T A = LU. Dabei ist P eine durch die Vertauschungen bestimmte Permutations-matrix, L eine untere und U eine obere Dreiecksmatrix.

Realisieren Sie den jetzt noch notwendigen Dreiecksloser• U X = Be. Gehen Sie hierfur• von der folgenden seriellen (ij-)Form aus:

fur• i = n : 1 : 1 fur• k = 1 : q s***(***k***)*** := 0

fur• j = i + 1 : n fur• k = 1 : q

s***(***k***)*** := s***(***k***)*** + aijx***(***jk***)***

fur• k = 1 : q

x***(***ik***)*** := (ai;n***+***k s***(***k***)***)=aii

c) Fur• A 2 Rm n ist die Frobenius-Norm kAkF de niert durch

* 1***1***=***2***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | m | n | A |  |  |
| kAkF := | @Xi | X | : |  |
|  | aij***2*** |  |  |

***=1*** j***=1***

Berechnen Sie jeweils kLU P T AkF und kBe U XkF . Verwenden Sie hierzu ggf. die Routine CollectColumns aus Aufgabe 4.

8

Testen Sie Ihr Verfahren anhand der Matrizen

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A = (aij) 2 Rn n | mit | aij = | i + j | sin | ij | |  |
|  |  |  |  |
| n | n + 1 |  |
| B = (bij) 2 Rn q | mit | bij = i + j | |  |  |  |  |

fur• n = 100; 500 und q = 1; 10 auf p = 1; 3; 4 Prozessen.

9

* Block 2 | Iterative Losung• linearer Systeme

Aufgabe 6 (8 Punkte)

Implementieren Sie das Red-Black-SSOR-Verfahren mit Conrad-Wallach-Trick zum Losen• der diskretisierten Laplace-Gleichung

|  |  |  |
| --- | --- | --- |
| ui;j ***1*** ui ***1***;j + 4uij ui***+1***;j ui;j***+1*** = h***2***fij ; i; j = 1 : N; | | (1) |
| mit den Randbedingungen |  |  |
| u***0***j=g***0***j; uN***+1***;j=gN***+1***;j; | j = 1 : N; |  |
| ui***0*** = gi***0***; ui;N***+1*** = gi;N***+1***; | i = 1 : N: |  |
| Dabei sei N 2 N, h = 1=(N + 1) und |  |  |

uij = u(ih; jh); fij = f(ih; jh); gij = g(ih; jh); i; j = 0 : N + 1:

Beschreibung: Ordnen Sie den Gitterpunkten (ih; jh) des aquidistanten• Gitters

h = f(ih; jh) : i; j = 0 : N + 1g

die Farben Rot und Schwarz zu durch

|  |  |  |
| --- | --- | --- |
| (ih; jh) | ist rot, falls | i + j 0 mod 2; |
| (ih; jh) | ist schwarz, falls | i + j 1 mod 2: |

Ein Iterationsschritt des Red-Black-SSOR-Verfahrens besitzt die Gestalt:

fur• alle roten Gitterpunkte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k***+*** ***1*** | | ! | | | | | | | | h***2***fij + ui;jk ***1*** + uik ***1***;j + uik***+1***;j + ui;jk***+1*** | | | | | | | + (1 !)uijk | |  |  |
| uij | ***2*** | := | | | |  |  |  |  | (2) |  |
| 4 | | |  |
| fur• alle schwarzen Gitterpunkte | | | | | | | | | | | | |  |  |  |  |  |  |  |  |
| k***+*** ***1*** | | ! | | | | | | |  |  | ***2*** | k***+*** ***21*** | k***+*** ***21*** | k***+*** ***21*** | k***+*** ***21*** | | k |  |  |  |
| uij | ***2*** | := | | | | 4 | | |  | h | | fij + ui;j ***1*** + ui ***1***;j + ui***+1***;j + ui;j***+1*** | | | | | + (1 !)uij | | (3) |  |
| fur• alle schwarzen Gitterpunkte | | | | | | | | | | | | |  |  |  |  |  |  |  |  |
| k***+1*** |  | ! | |  |  |  |  |  |  | ***2*** |  | k***+*** ***21*** | k***+*** ***21*** | k***+*** ***21*** | k***+*** ***21*** |  | k***+*** ***21*** |  |  |  |
| uij | := |  | 4 |  | h | | | | | | fij | + ui;j ***1*** + ui ***1***;j + ui***+1***;j + ui;j***+1*** + (1 !)uij | | | | | |  | (4) |  |
| fur• alle roten Gitterpunkte | | | | | | | | | | | | |  |  |  |  |  |  |  |  |
|  |  | ! | | | | | | h***2***fij + ui;jk***+11*** + uik***+11***;j + uik***+1+1***;j + ui;jk***+1+1*** | | | | | | | |  | k***+*** | ***1*** |  |  |
| uijk***+1*** := | | | | |  | | | + (1 !)uij | ***2*** | (5) |  |
| 4 | | |  |

Die Aufdatierungen der roten Gitterpunkte sind voneinander unabhangig• und deshalb parallel ausfuhrbar,• da diese nur von Werten in den jeweils benachbarten schwarzen Gitterpunkten und dem alten Wert im zugehorigen• roten Gitterpunkt abhangen•. Ent-sprechendes gilt fur• die Aufdatierung der schwarzen Gitterpunkte.

10

Hinweise: Beachten Sie, dass sowohl Gro• en aus (3) in (4) als auch Gro• en aus (5) in (2) (fur• den nachsten• Iterationsschritt k + 1) verwendet werden konnen• (Conrad-Wallach-Trick).

Verwenden Sie eine zweidimensionale Aufteilung von h, so dass jedes Teilgebiet von h ein Rechteck von ` m Gitterpunkten darstellt, und ordnen Sie jedem der p = P Q Prozesse P`m, ` = 1 : P , m = 1 : Q, ein Teilgebiet zu. Benutzen Sie lokal ein zweidimensionales Feld u(0 : ` + 1; 0 : m + 1) fur• das Teilgebiet und dessen Rand der Breite 1.

Konstruieren Sie einen geeigneten Datentyp zum Austausch der Werte in den roten und schwarzen Randpunkten. Zur Vereinfachung konnen• Sie ` und m als gerade annehmen. Realisieren Sie die Kommunikation mit MPI CART SHIFT.

Brechen sie die Iteration ab, wenn sich zwei aufeinander folgende Iterierte in der Euklid-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Norm (kvk***2*** = P | n |  | ***1***=***2*** | fur• v 2 Rn) um weniger als " unterscheiden. |  |
| i***=1*** vi***2*** |  |  |

Testen Sie Ihr Verfahren fur• N = 24; 48; 120, P = 2, Q = 3, ! = 1, f 0,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| g(x; y) = | 8 | x |  | x | fur• | y = 1 |  |
|  | > | 1 |  | fur• | y = 0 |  |
|  | 1 |  | y | fur• | x = 0 |  |
|  | > |  |  |  |  |  |  |
|  | < | y |  |  | fur• | x = 1 |  |
|  | > |  |  |  |
|  | > |  |  |  |  |  |  |
|  | : |  |  |  |  |  |  |

und den Startvektor u***0*** = 0. Die exakte Losung• ist die Funktion u(x; y) = 2xy x y + 1.

Geben Sie am Ende die Euklidnorm des Fehlervektors e := (uij)i;j***=1:***N (u(ih; jh))i;j***=1:***N 2 RN***2*** aus.

Aufgabe 7 (8 Punkte)

Implementieren Sie das parallele cg-Verfahren zur Losung• des in Aufgabe 6 beschriebenen linearen Gleichungssystems (1), der sog. diskretisierten Laplace-Gleichung.

Vergleichen Sie die Laufzeit des Verfahrens ohne Pr•akonditionierung mit der des Red-Black-SSOR-pr•akonditionierten Verfahrens.

Beschreibung: Das cg-Verfahren kann zur Losung• linearer Gleichungssysteme Ax = b mit symmetrischer, positiv de niter Matrix A 2 Rn n (d.h. yT Ay > 0 fur• alle 0 6= y 2 Rn) eingesetzt werden. Eine der Standard-Formulierungen des Verfahrens ist nachfolgend angegeben:

11

f Startvektor g

wahle• x***0*** 2 Rn beliebig f Anfangsresiduum g r***0*** := b Ax***0***

***0*** := kr***0***k***22***

f erste Suchrichtung\ g s***0*** :=r***0***"

fur• k = 0; 1; : : :, bis Abbruchkriterium erfullt• ist qk := Ask

k := k = ( (sk)T qk ) f neue Iterierte g

xk***+1*** := xk + ksk

* "billiges\ Aufdatieren des Residuums rk***+1*** = b Axk***+1*** g

rk***+1*** := rk kqk

k***+1*** := krk***+1***k***22***

* neue Suchrichtung g

k := k***+1***= k

sk***+1*** := rk***+1*** ksk

Das cg-Verfahren liefert bei rundungsfehlerfreier Rechnung nach spatestens• n Schrit-ten die exakte Losung• x des Gleichungssystems. In der Praxis wird das Verfahren allerdings nicht in dieser Weise verwendet, sondern als iteratives Verfahren, das nach (i.a. wesentlich) weniger als n Schritten abgebrochen wird, sobald eine hinreichend gute Naherung• fur• die Losung• bestimmt ist.

Die Konvergenzgeschwindigkeit dieses iterativen Verfahrens hangt• stark von der Vertei-lung der Eigenwerte von A ab. Man kann zeigen, dass in der Norm kxkA := (xT Ax)***1***=***2*** fur• die Fehler der Iterierten xk 2 Rn gilt:

kxk x kA k 2kx***0*** x kA ;

wobei

= (A) 1 < 1 (A) + 1

ist mit

|  |  |  |  |
| --- | --- | --- | --- |
| (A) = | ***max***(A) | (Kondition von A) |  |
| ***min***(A) |  |
|  |  |  |

( ***max*** gro• ter, ***min*** kleinster Eigenwert von A). Diese Kondition kann bereits fur• kleine Matrizen sehr gro sein; dann liegt sehr nahe bei 1, und das cg-Verfahren konvergiert sehr langsam.

Aus diesem Grunde geht man meist vom System Ax = b auf ein "aquivalentes\• Sy-stem By = c mit besser konditionierter Matrix B uber,• etwa, indem man mit einer nichtsingularen• Matrix S ***1*** durchmultipliziert:

S ***1***A(ST ) ***1*** ST x = S ***1***b :

|{z} |{z}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | |  | {z |  | } |  |  |
|  |  | B | | y | c |  |
|  |  |  | |  |  |  |

12

Gelingt es, S so zu wahlen,• dass T := SST A gilt, so konvergiert das cg-Verfahren fur• das "pr•akonditionierte\ System By = c deutlich schneller als fur• das ursprungliche• System Ax = b. Fur• die praktische Anwendbarkeit ist es wesentlich, dass dieses prakon•-ditionierte cg-Verfahren auch durchgefuhrt• werden kann, ohne die Matrix B explizit zu berechnen (siehe folgenden Algorithmus).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f | Startvektor | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | ***0*** |  | 2 R | | | | | | ng | | beliebig | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| wahle• x | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Anfangsresiduum | | | | | | | | | | | | | | g | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f***0*** | |  |  |  |  |  |  |  |  | ***0*** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| r |  | := b Ax | | | | | | | | |  |  |  |  | ***0*** | = r | | | | ***0*** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| lose• das System T z | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f | erste Suchrichtung\ | | | | | | | | | | | | | | | | | g | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | "***0*** | ) | | T | | z | ***0*** | | |  |  |  |  |  |  | = |  | ***0 2*** | | | ***0*** |  | By | ***0*** | , wobei y | | | ***0*** | = S | | T ***0*** |  |  |
| ***0*** := (r | | | |  |  |  |  |  |  |  |  |  |  |  | f | |  | k | r^ | k***2*** | | mit r^ = c |  |  |  | x | g |  |
| s***0*** := z***0*** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| fur• k = 0; 1; : : :, bis Abbruchkriterium erfullt• ist | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  | qk := Ask | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | k := k = ( (sk)T qk ) | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | neue Iterierte | | | | | | | | | | | | | gk | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | fk***+1*** | | |  |  |  |  |  |  |  | k |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | x | |  |  | := x | | | | | | | + ks | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | billiges\ Aufdatieren des Residuums rk***+1*** = b | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | Axk***+1*** | | | g |  |  |  |
|  |  | fk"***+1*** | | |  |  |  |  |  |  |  | k | kq | | | | k |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | r |  |  | := r | | | | | | |  |  |  | k***+1*** | | | = r | | | k***+1*** | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | lose• das System T z | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | k***+1*** := (rk***+1***)T zk***+1*** f = kr^k***=1***k***22*** mit r^k***+1*** = c Byk***+1***, yk***+1*** = ST xk***+1*** g | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  |  | f neue Suchrichtung g | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | k := | | | | |  | | | k***+1***= k | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | s | k***+1*** | |  |  |  |  | k***+1*** | | ks | | | | | k | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | := z | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Bei dieser Formulierung erhalt• man auch statt einer Naherung• fur• die Losung• y von By = c direkt eine Naherung• fur• die (eigentlich interessierende) Losung• x von Ax = b. Der einzige Mehraufwand im Vergleich zum nicht prakonditionierten• Verfahren besteht darin, dass in jedem cg-Schritt ein linearen Gleichungssystem mit der Matrix T gelost• werden muss.

Bei der Red-Black-SSOR-Prakonditionierung• wahlt• man die Matrix S so, dass das Losen• des Systems T z = r genau der Durchfuhrung• von 1 Schritten des Red-Black-SSOR-Verfahrens fur• das System Az = r mit dem Startvektor z***(0)*** = 0 entspricht.

Hinweise: Das im cg-Verfahren benotigte• Matrix-Vektor-Produkt q = As ergibt sich fur• die diskretisierte Laplace-Gleichung mit der (fur• zweidimensionale aquidistante• Diskreti-sierungen ublichen)• zweidimensionalen Indizierung der Vektoren durch

qij := si;j ***1*** si ***1***;j + 4sij si***+1***;j si;j***+1*** ; i; j = 1 : N :

Verwenden Sie zur Parallelisierung die in Aufgabe 6 vorgeschlagene zweidimensionale Aufteilung des Gitters h in rechteckige Gebiete.

Testen Sie Ihr Verfahren mit dem in Aufgabe 6 angegebenen Problem. Bestimmen Sie, welche Anzahl von Red-Black-SSOR-Schritten als Prakonditionierer• fur• dieses Problem "vernunf•-tig\ ist.

13

* Block 3 | Berechnung von Eigenwerten und -vektoren

Aufgabe 8 (10 Punkte)

Implementieren Sie eine parallele Variante des zyklischen Jacobi-Verfahrens zur Bestimmung der Eigenwerte und Eigenvektoren einer symmetrischen Matrix A 2 Rn n (n gerade) auf p = n=2 Prozessen.

Beschreibung: Beim Jacobi-Verfahren wird auf die Matrix A eine Folge von Jacobi-Rotati-onen

A A***(0)*** ! J***1***T A***(0)***J***1*** =: A***(1)*** ! J***2***T A***(1)***J***2*** =: A***(2)*** !

angewandt. Dabei ist

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 0 | 1 | ... |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
|  |  |  | B |  | 1 | ck | | 0 |  |  | 0 | sk | |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  | |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  | 0 |  | 1 |  | 0 | |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| Jk |  | J(pk; qk; k) := | B |  |  | . |  |  | . | .. |  | . | |  | C |  | n | n |  |
|  | B |  |  | .. |  |  |  |  | .. | |  | C | 2 | R |  |  |
|  |  | B |  |  | 0 |  |  |  |  | 1 | 0 | |  | C |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  | s | | 0 |  |  | 0 | c | k |  | C |  |  |  |  |
|  |  |  | B |  |  |  | k |  |  | |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  | 1 |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  | . | .. | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  |  |  | B |  |  |  |  |  |  |  |  |  |  | 1 | C |  |  |  |  |
|  |  |  | @ |  |  |  |  |  |  |  |  |  |  |  | A |  |  |  |  |

(der Eintrag sk be ndet sich an der (pk; qk)-Position in Jk) eine Rotation um den Winkel k in der (pk; qk)-Ebene mit ck = cos k und sk = sin k.

Bei geeigneter Wahl des Winkels k kann man durch die Anwendung der Rotation

A***(***k ***1)*** ! J(pk; qk; k)T A***(***k ***1)***J(pk; qk; k) = A***(***k***)***

den Eintrag in der (pk; qk)-Position von A zu Null machen. Wahlt• man etwa fur• apk;qk 6= 0 speziell den Winkel k so, dass

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  | ***(***k ***1)(***k ***1)*** |  |  |  |
| t |  | tan |  | = |  |  |  | sign( ) | | | mit | := | aqkqk apkpk | ; | (6) |  |
|  |  |  |  |  |  |  |  |  |  |
| k | k | j |  | j | + | p1 + ***2*** | | ***(***k ***1)*** |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 2apkqk |  |  |  |

so gilt a***(***pkk***)***qk = 0. (Zur Anwendung der Rotation wird der Winkel k wegen

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ck = |  | 1 | und sk = ck |  | tk | (7) |  |
|  |  |  |  |
| p |  |  |  |
| 1 + tk***2*** |  |
|  |  |  |  |  |

nicht explizit benotigt•.) Im Fall a***(***pkkqk***1)*** = 0 wird man naturlich• k = 0 wahlen,• also keine "echte\ Rotation durchfuhren•.

14

Ein Sweep des zyklischen Jacobi-Verfahrens besteht nun darin, in geeigneter Reihenfolge jedes der Au endiagonalelemente von A genau einmal zu Null zu machen. Wegen der (wahrend• des Verfahrens erhaltenen) Symmetrie wird dabei mit apkqk auch gleichzeitig aqkpk zu Null, so dass n(n 1)=2 Rotationen pro Sweep genugen•. (Es gibt mehrere Varianten des zyklischen Jacobi-Verfahrens, die sich hauptsachlich• in der Wahl der Rei-henfolge fur• die Indexpaare (pk; qk) unterscheiden.) Die Erfahrung zeigt (und fur• einige zyklische Varianten kann die Konvergenz auch bewiesen werden), dass die Matrix A nach wenigen solcher Sweeps (typisch: hochstens• 10) "praktisch\ Diagonalgestalt be-sitzt mit sehr guten Approximationen i der Eigenwerte an den Diagonalpositionen

aii.

Parallelitat• wird durch gleichzeitige Bestimmung und Anwendung von Rotationen er-zielt. Rotationen J(pki ; qki ; ki ) sind vertauschbar (und damit in jeder Reihenfolge an-wendbar, insbesondere parallel), wenn ihre Indexpaare fpki ; qki g paarweise disjunkt sind. Das folgende Vorgehen liefert n 1 Mengen von jeweils n=2 Indexpaaren, so dass

die zu den Indexpaaren jeder Menge gehorigen• Rotationen parallel anwendbar sind und

die n 1 Mengen zusammen genau einen Sweep ergeben (d.h. jedes Au endiago-nalelement wird genau einmal zu Null gemacht); insbesondere tritt kein Indexpaar doppelt auf.

Die erste Menge ist durch die Indexpaare (1; 2), (3; 4), . . . , (n 1; n) gegeben, siehe nachfolgende Skizze fur• n = 8.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| top: | 1 | 3 | 5 | 7 |
| bot: | 2 | 4 | 6 | 8 |
|  |  |  |  |  |

Fur• die zweite Menge verschiebt man alle Indizes (au er der 1) um einen Platz im Uhrzeigersinn

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| top: | 1 |  |  |  | 3 | - | | 5 |  | - | | 7 | |  |
|  |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | ? |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | | |  |  | |  |  | |  | |  |
| bot: | 2 |  |  | | 4 | 6 |  |  | | 8 | |  |
|  |  |  |  |  |  |  |
| und erhalt• |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| top: | 1 |  |  |  | 2 |  | 3 | |  | | 5 | | |  |
|  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  | | , | | |  |
| bot: | 4 |  |  |  | 6 |  | 8 | |  | | 7 | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

entsprechend den Indexpaaren (1; 4), (2; 6), etc. Die nachste• "Verschiebe-Runde\ ergibt

15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| top: | 1 | 4 | 2 | 3 |
| bot: | 6 | 8 | 7 | 5 |
|  |  |  |  |  |

mit den Indexpaaren (1; 6), (4; 8), usw. Nach n 1 solchen Runden ist wieder die Ausgangssituation hergestellt.

Mit dieser Reihenfolge fuhrt• man Sweeps (jeweils n 1 Runden) mit der Matrix A durch, bis die Au endiagonal-"Norm\

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1***1***=***2*** |  |
|  | n | A |  |
| kAk***o*** := | @X X |  |
| i***=1*** j6***=***i | aij***2*** |  |
|  |  |  |

eine vorgegebene Schranke " unterschreitet.

Die Eigenvektoren q***(***i***)*** erhalt• man, indem man die Rotationen gleichzeitig in einer orthogonalen Matrix Q akkumuliert:

In Q***(0)*** ! Q***(0)***J***1*** =: Q***(1)*** ! Q***(1)***J***2*** =: Q***(2)*** !

(In: Einheitsmatrix). Wenn A "hinreichend diagonal\ ist, bilden die Spalten q***(***i***)*** von Q gute Naherungen• fur• Eigenvektoren zu den i.

Zusammen ergibt sich der folgende (zunachst• serielle) Algorithmus fur• ein zyklisches

Jacobi-Verfahren:

Q := In

top := (1; 3; : : : ; n 1) bot := (2; 4; : : : ; n) solange kAk***o*** "

f durchlaufe die Paare-Mengen des Sweeps g fur• s = 1 : n 1

f durchlaufe die Indexpaare der Menge g

fur• k = 1 : n***2***

f Rotation bestimmen g (pk; qk) := (top(k); bot(k))

berechne tk aus aqkqk , apkpk und apkqk gema• (6) berechne ck und sk gema• (7)

f Rotation anwenden g

* := A J(pk; qk; k)
* := J(pk; qk; k)T A

Q := Q J(pk; qk; k) verschiebe( top, bot )

Dabei bezeichnet top(1 : n***2*** ) die "obere\ und bot(1 : n***2*** ) die "untere\ Halfte• der Indizes

•

in der "aktuellen\ Runde (siehe Skizzen), und verschiebe fuhrt• die zum Ubergang in die nachste• Runde notwenigen Indexverschiebungen durch.

16

Parallelitat• wird durch die gleichzeitige Bestimmung und Anwendung der Rotationen jeder Menge erzielt. Hierzu wird die Matrix A spaltenweise auf die Prozesse aufgeteilt, wie durch die aktuellen Indexpaare angegeben. In der ersten Runde jedes Sweeps besitzt also Prozess 1 die Spalten 1 und 2, Prozess 2 die Spalten 3 und 4, usw., in der zweiten Runde besitzt Prozess 1 die Spalten 1 und 4, Prozess 2 die Spalten 2 und 6, usw. Dies bedeutet, dass zur Anwendung der n=2 Rotationen einer Menge

jeder Prozess die "seinem\ Indexpaar der Menge entsprechende Rotation bestim-men und von rechts her auf die Matrix A anwenden kann und hierfur• nur lokale Daten benotigt,•

jeder Prozess alle Rotationen der Menge von links her auf seine beiden Spalten von A anwenden muss und

die Spalten von A zwischen den Runden in gleicher Weise wie die Indizes zwischen den Prozessoren verschoben werden mussen•.

Die Akkumulation von Q kann ebenfalls parallel erfolgen, wenn man die Spalten der Matrix Q ebenso wie die Spalten von A verteilt und mit jeder Runde entsprechend verschiebt. Hier wird die Parallelisierung dadurch erleichtert, dass Q nur von rechts her transformiert werden muss.

Hinweis: Anwenden einer Rotation von links, A ! J(p; q; )T A, andert• nur die Zeilen p und q von A. Analog werden bei Anwendung von rechts, A ! A J(p; q; ), nur die Spalten p und q modi ziert. Nutzt man dies aus, dann erfordert die Anwendung der Rotation (von links oder rechts) 4n Multiplikationen und 2n Additionen. Daher wird die Anwendung einer Rotation nie als Matrix-Matrix-Produkt realisiert, was rund 2n***3*** Operationen erfordern wurde•.

•

Zur Uberprufung• der Ergebnisse lassen Sie das Residuum

kAQ Q kF

sowie die Abweichung

kQT Q IkF

der Eigenvektoren von der Orthogonalitat• ausgeben. Dabei ist = diag( ***1***; : : : ; n) 2 Rn n ( i: Naherungen• fur• die Eigenwerte von A) und Q = ( q***(1)*** j : : : j q***(***n***)*** ) 2 Rn n (q***(***i***)***: Nahe•-

rungen zugehoriger• orthonormierter Eigenvektoren). Testen Sie das Verfahren mit der Matrix

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 n 1 | | | n 1 n 2 | | | | 1 1 | | |  |  |  |
|  | B | n |  | n | 1 | n | 2 |  | 1 | C |  |  |  |
| A = | n . | 2 | n . | 2 | n . | 2 | . 1. | | 2 Rn n | (n = 2p) |  |
|  | B | .. |  | .. |  | .. |  | .. .. | | C |  |  |  |
|  | B |  |  |  |  |  |  |  |  | C |  |  |  |
|  | B | 1 |  | 1 |  | 1 |  |  | 1 | C |  |  |  |
|  | B |  |  |  |  |  |  |  |  | C |  |  |  |
|  | @ |  |  |  |  |  |  |  |  | A |  |  |  |

und " = 10 ***10*** auf p = 1; 3; 4; 8 Prozessen.

Fur• p = 4 (also n = 8) besitzt diese Matrix die Eigenwerte

17

0.258735930272134

0.287520089775684

0.345844044326706

0.457762962433225

0.688385684834675

1.258287827212875

3.338165566772758

29.365297894371945

und ein zum Eigenwert 0:258735930272134 gehoriger• Eigenvektor ist durch

0.0891316083075325 -0.2553571073253758 0.3870952140163506 -0.4665539670857883 0.4830020216355103 -0.4342179767567612 0.3267903880321401 -0.1752279465957299

gegeben.

Aufgabe 9 (10 Punkte)

Implementieren Sie ein zweistu ges paralleles Verfahren zur Berechnung aller Eigenwerte einer symmetrischen Matrix A = (aij) 2 Rn n. Verwenden Sie hierfur• eine spaltenzyklische Abspeicherung des unteren Dreiecks von A.

1. Transformieren Sie in der ersten Stufe die Matrix A mit Hilfe des Householder-Verfahrens auf Tridiagonalgestalt.

Beschreibung: Gehen Sie von der folgenden seriellen Form des Householder-Verfahrens aus:

fur• k = 1 : n 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x := A(k + 1 : n; k) 2 Rn k | | | |  |  |  |  |
| wenn x 6= 0 | k | k | fur• | Tx***1*** | = 0 | |  |
| kxk***2*** |  |
| :=sign(x***1***) | | x ***2*** | fur• | x***1*** | 6= 0 | |  |
| y := (x***1*** + ; x***2***; : : : ; xn k) | | | |  |  |  |  |
| := 2=kyk***22*** |  |  |  |  |  | T |  |
| A(k + 1 : n; k) := ( ; 0; : : : ; 0) | | | | | |  |  |
| v := A(k + 1 : n; k + 1 : n)y | | | | |  |  |  |

z := v ***2*** (yT v)y fur• j = k + 1 : n

fur• i = j : n

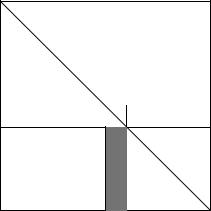
aij := aij yi kzj k zi kyj k

18

Dabei wird fur• festes k gerade die k-te Spalte bzw. Zeile von A "tridiagonalisiert\. Die Schleife uber• j ist eine symmetrische Rang-2-Modi kation, bei der nur das untere Dreieck von A yzT zyT berechnet wird.

Hinweis: Beachten Sie, dass fur• ein paralleles Householder-Verfahren mit spaltenzy-klischer Abspeicherung des unteren Dreiecks von A ein Algorithmus zur Matrix-Vektor-Multiplikation

* 1



* C
* C
  + C v = B  C y
  + C
* C
* A

verwendet werden muss, der nur das untere Dreieck von A benotigt•.

b) Die Eigenwerte einer symmetrischen Tridiagonalmatrix

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | b***2*** | a***2*** | ... | 1 |  |  |
|  | B | a***1*** | b***2*** |  | C |  |  |
| T = |  |  |  | 2 Rn n |  |
|  | B |  |  |  | C |  |  |

* C
* ... ... bn C @ A

bn an

mit bi 6= 0 fur• i = 2 : n konnen• wie folgt durch ein Bisektionsverfahren bestimmt werden (zweite Stufe der Berechnungen).

Beschreibung: De niert man zu 2 R die Gro• en

|  |  |  |
| --- | --- | --- |
| p***0***( ) | := | 1 |
| p***1***( ) | := | a***1*** |

pi( ) := (ai )pi ***1***( ) b***2***ipi ***2***( ) ; i = 2 : n;

so ist die Anzahl s( ) der Vorzeichenwechsel in der Folge p***0***( ); : : : ; pn( ) (Nullen werden ignoriert) gerade die Anzahl der Eigenwerte j von T , welche kleiner als sind.

Das Bisektionsverfahren startet mit einem Intervall [a; b], welches mit Sicherheit alle Eigenwerte enthalt• (also s(a) = 0, s(b) = n). Dieses unterteilt man in der Mitte c = (a + b)=2 und bestimmt s(c). Ist s(a) < s(c), so enthalt• [a; c] Eigenwerte und wird auf die gleiche Weise (rekursiv) behandelt. Ist s(c) < s(b), so wird [c; b] weiter betrachtet. Die Rekursion wird abgebrochen, wenn der Durchmesser des Intervalls [a; b] kleiner als " ist.

19

Bisect( [a; b], sa, sb )

f bestimmt alle Eigenwerte im Intervall [a; b]; es ist sa = s(a), sb = s(b) g wenn b a < "

dann ist (a + b)=2 eine Naherung• fur• sb sa Eigenwerte sonst

c := (a + b)=2 sc := s(c) wenn sc > sa

Bisect( [a; c], sa, sc )

wenn sc < sb

Bisect( [c; b], sc, sb )

Ein geeignetes, alle Eigenwerte enthaltendes, Startintervall ist zum Beispiel durch [ kT k1; +kT k1] gegeben mit

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | n |  |  |  |  |  |  |  |  |
| k |  | k1 | m | Xj |  |  |  |  | 2 | R | m n : |  |
| A | := max |  | A = (a |  | ) |  |
| a | fur• |  |  |
|  | i***=1*** | j ijj |  | ij |  |  |  |
|  |  |  |  | ***=1*** |  |  |  |  |  |  |  |  |

Implementieren Sie eine parallele Variante des Bisektionsverfahrens, bei der ein Prozess (der master)

eine Liste der noch zu bearbeitenden Intervalle verwaltet,

den ubrigen• Prozessen (den slaves) jeweils ein noch zu bearbeitendes Intervall zuschickt und

von jedem Slave die noch weiter zu bearbeitende(n) Halfte(n)• des ihm ubergebenen• Intervalls zuruck• erhalt•

und alle anderen Prozesse (die slaves)

jeweils ein noch zu bearbeitendes Intervall vom master empfangen,

durch Berechnung von s(c) die noch weiter zu bearbeitende(n) Halfte(n)• dieses Intervalls bestimmen und

diese dann wieder an den master senden.

•

Hinweis: Die Berechnung der Folge p***0***( ); : : : ; pn( ) kann leicht zu Uberlauf fuhren•. Dies la• t sich vermeiden, indem man statt dessen die Folge qi( ) := pi( )=pi ***1***( ) verwendet, welche der Rekursionsgleichung

q***1***( ) := a***1***

b***2***

qi( ) := ai i ; i = 2 : n; qi ***1***( )

genugt•. Dabei sind Glieder qi( ) = 0 durch sehr kleine positive Zahlen zu ersetzen, um Division durch Null auszuschlie en. Dann erhalt• man s( ) als Anzahl der negativen Glieder in der Folge q***1***( ); : : : ; qn( ).

20

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Testen Sie Ihre Verfahren anhand der Matrizen | | | | | | |  | 1 1 | | |  |  |
|  | 0 n 1 | | | n 1 n 2 | | | |  |  |
|  | B | n |  | n | 1 | n | 2 |  | 1 | C |  |  |
| A = | n . | 2 | n . | 2 | n . | 2 | . 1. | | 2 Rn n |  |
|  | B | .. |  | .. |  | .. |  | .. .. | | C |  |  |
|  | B |  |  |  |  |  |  |  |  | C |  |  |
|  | B | 1 |  | 1 |  | 1 |  |  | 1 | C |  |  |
|  | B |  |  |  |  |  |  |  |  | C |  |  |
|  | @ |  |  |  |  |  |  |  |  | A |  |  |

fur• n = 50; 100 auf p = 1; 3; 4 Prozessen mit " = 10 ***12***. Die zweite Stufe alleine kann mit den Tridiagonalmatrizen

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 0 | 1 | 2 | 1 . | |  |  |  |  | 1 | |  |  |  |  |  |  |  |
|  |  |  |  | B | 2 | 1 |  |  |  |  |  |  | C | |  |  |  |  |  |  |  |
|  |  |  | T = |  | 1 2 .. | | | | |  |  | 2 Rn n | |  |  |  |  |  |
|  |  |  |  | B |  |  |  |  |  |  |  |  | C | |  |  |  |  |  |  |  |
|  |  |  |  | B |  |  | ... ... | | | |  |  | C | |  |  |  |  |  |  |  |
|  |  |  |  | B |  |  |  | 1 | C | |  |  |  |  |  |  |  |
|  |  |  |  | B |  |  |  |  |  |  |  |  | C | |  |  |  |  |  |  |  |
|  |  |  |  | @ |  |  |  |  | 1 |  |  | 2 | A | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mit den Eigenwerten |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | k = 4 sin***2*** | | | | k |  |  |  | ; | | k = 1 : n; | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 2(n + 1) | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| und | 1 | 9 | 1 . | | |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B | 10 | 1 | .8.. ..... | | |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
|  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| T = B |  |  |  |  |  | 1 | 0 | 1 |  |  |  |  |  |  |  |  | C |  | R | ***21 21*** |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C | 2 |  |  |
| B |  |  |  |  |  |  | 1 | 1 |  |  | . | 1 | . |  |  |  | C |  |  |  |
| B |  |  |  |  |  |  |  | 1 |  |  | .. | .. | |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  | . |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  | .. |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  | 8 | 1 |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 9 | 1 | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | C |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 10 | C |  |  |  |  |
| @ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | A |  |  |  |  |

mit Eigenwerten

-1.12544152211998 0.25380581709668 0.94753436752929 1.78932135269508 2.13020921936251 2.96105888418573 3.04309929257883 3.99604820138363 4.00435402344086 4.99978247774291 5.00024442500192

21

6.00021752225710

6.00023403158417

7.00395179861638

7.00395220952868

8.03894111581427

8.03894112282902

9.21067864730492

9.21067864736133

10.74619418290332

10.74619418290339

getestet werden.

Aufgabe 10 (10 Punkte)

Implementieren Sie das Lanczos-Verfahren zur Bestimmung der extremalen Eigenwerte und Eigenvektoren einer symmetrischen, dunn• besetzten Matrix A 2 Rn n auf p Prozessen.

Beschreibung: Das Lanczos-Verfahren erweitert, ausgehend von einem normierten Startvek-

tor v***1***, die vorhandene Orthonormalbasis (ONB) fv***1***; : : : ; vkg des Krylov-Unterraums

Kk(A; v***1***) = span(v***1***; Av***1***; : : : ; Ak ***1***v***1***) zu einer ONB fv***1***; : : : ; vk***+1***g von Kk***+1***(A; v***1***). Die algorithmische Formulierung

fur• k = 1; 2; : : :, bis das Abbruchkriterium erfullt• ist

uk = Avk

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v |  |  |  |  | k***+1*** k | | | k |  | ***1*** |  |
| k = uk | | | | ; vk | |  |  |  |  |  |  |
|  | k***+1*** = qk | | | |  | vk | | vk | ***1*** | (mit = 0, v***0*** = 0) |  |
| k***+1*** | | = v | | |  |  | ***2*** |  |  |  |  |
| e | |  | e | | | k | k***+1*** |  |  |  |  |
| v | k***+1*** |  | kk***+1*** | | |  |  |  |  |  |
|  | = v | |  |  | = |  |  |  |  |  |
|  |  |  | e |  |  |  |  |  |  |  |  |

lasst• sich mit Hilfe der Matrizen Vk = v***1***jv***2***j : : : jvk und der symmetrischen Tridiago-nalmatrizen

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | ***2*** | ***2*** | ***3*** | |  |  |  | 1 |  |
|  | B | ***1*** | ***2*** |  |  |  |  |  | C |  |
| Tk = |  | ... | ... | | ... |  |  |  |
|  | B |  |  |  | k ***1*** |  |  | k | C |  |
|  | B |  |  |  | k ***1*** |  | C |  |
|  | B |  |  |  |  |  |  | k | C |  |
|  | B |  |  |  |  | k |  | C |  |
|  | @ |  |  |  |  |  |  |  | A |  |

zusammenfassen zu

AVk = VkTk + k***+1*** vk***+1*** eTk***+1*** mit eTk***+1*** = (0; : : : ; 0; 1):

| {z } k-mal

Die Eigenwerte ***1***; : : : ; k von Tk sind Approximationen an die Eigenwerte von A. Ist Qk = (qij) 2 Rk k die Matrix, deren Spalten die zugehorigen• Eigenvektoren von Tk

22

darstellen, so sind die Spalten wj von Wk = Vk Qk approximative Eigenvektoren von A, und es gilt

|  |  |
| --- | --- |
| kAwj jwjk***2*** k***+1*** jqkjj: | (8) |

Parallelitat• wird im Wesentlichen durch die Parallelisierung der Schritte im Lanczos-Verfahren erzielt. Sie sollen deshalb alle Berechnungen von Vektoren sowie die Matrix{ Vektor-Multiplikation parallel durchfuhren•.

Die Bestimmung der Eigenwerte und -vektoren von Tk soll seriell erfolgen, entweder durch eigenhandige• Programmierung des Jacobi-Verfahrens (vgl. Aufg. 8) oder unter Verwendung einer einzubindenden Standard-Routine, etwa DSTEV aus LAPACK.

Die Bestimmung der approximativen Eigenvektoren von A soll dann wieder parallel erfolgen.

Geben Sie fur• jedes k die jeweils konvergierten Eigenwerte an, d.h. die Eigenwerte, in welchen die rechte Seite in (8) kleiner als eine vorgegeben Schranke ist. Berechnen Sie zur Kontrolle jeweils auch die linke Seite von (8).

Das Lanczos-Verfahren wird abgebrochen, wenn eine zuvor festgelegte Anzahl m von approximativen Eigenpaaren ( j; wj) konvergiert ist, d.h. wenn die durch (8) gegebene Schranke fur• diese Paare kleiner als ist.

Hinweis: Bei dieser Aufgabe kommt es ganz wesentlich darauf an, wie die Matrix{Vektor-Multiplikation mit A realisiert wird. Fur• die vorliegende Aufgabe ist es ausreichend, wenn Sie fur• A die Matrix der diskreten Laplace-Gleichung auf dem Einheitsquadrat verwenden (s. Aufgabe 6).

23