Singulärwertzerlegung nach dem Divide-and-Conquer-Prinzip auf Massiv-Parallelrechnern

von

Udo Schendel   Alke Ziegler   Stefan Walser

Forschungsprojekt *Parallele Numerik*
Fachbereich Mathematik und Informatik
Freie Universität Berlin

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der Computertechnologie wird es in absehbarer Zeit weiter stark wachsende Rechenleistungen geben. Das ergibt sich aus der Entwicklung von immer leistungsfähigeren Übertragungswegen, höheren Speicherkapazitäten und schnelleren Reaktionszeiten bei (mathematischen und logischen) Operationen. In dieser Entwicklung ist die Forderung nach effizienten Algorithmen, die die Rechenleistung heutiger und künftiger Computersysteme ausschöpfen, eingeschlossen. Beispiele dafür sind die mathematische Modellierung von Problemen in der Meteorologie, Strömungslehre, Reaktion von chemischen Substanzen usw.

Massiv-parallele Rechner, d. h. räumlich zusammengefasste Systeme mit mehreren Prozessoren (bzw. Prozessorelementen), versprechen die höchste Leistung. Strukturell unterscheiden sich die realisierten Modelle aufgrund ihrer physikalischen Eigenschaften

- in der Anzahl ihrer Prozessoren (bzw. Prozessorelemente)
- in der Organisation des Speichers
- in der Organisation der Datenübertragung

und weiteren signifikanten Details. Daraus ergibt sich das Bedürfnis, ein Problem und seine mögliche Lösung derart aufzuteilen, daß möglichst viele unabhängige Teilprobleme mit unabhängigen Lösungen entstehen. Das *Divide-and-Conquer-Prinzip* ist in diesem Zusammenhang ein zentrales Gebiet der Forschung in der Numerischen Mathematik.

2 Grundlagen

In den Anwendungen sind im Zusammenhang mit der Singulärwertzerlegung folgende Begriffe von Bedeutung.

2.1 Sparse Matrizen

Die Charakteristik sehr großer, schwach besetzter Matrizen [32] [33] ist, daß für weniger als ca. 10% der Komponenten einer Matrix $a_{ij} \neq 0$ gilt. In der Numerischen Mathematik bestehen bei der Behandlung von Sparse Matrizen folgende Ziele:

- Minimierung des *Fill-in*
- Stabilitätsbetrachtungen

Unter dem *lokalen Fill-in* versteht man die Anzahl der Übergänge von Null-Komponenten aus $A^{(k)}$ in Nichtnull-Komponenten aus $A^{(k+1)}$ beim $(k+1)$-ten Schritt. Sparse-Matrizen eignen sich aufgrund ihrer Struktur selten für die konventionelle Speicherung in zweidimensionalen Arrays. In [32] und [33] werden Techniken erläutert, mit denen wesentlich effizientere Speicherbelegungen erreicht werden können. Bei der Programmierung müssen Algorithmen eingesetzt werden, die der Datenstruktur entsprechen.

2.2 Spezielle Matrizen und Eigenschaften

Eine tridiagonale Matrix $B \in I\!\!R^{n \times n}$ der Form

$$B = \begin{pmatrix} \alpha_{1,1} & \beta_{1,2} & & \\ \beta_{2,1} & \alpha_{2,2} & \ddots & \\ & \ddots & \ddots & \beta_{n-1,n} \\ & & \beta_{n,n-1} & \alpha_{n,n} \end{pmatrix} \tag{1}$$

ist symmetrisch, wenn für alle $i \in \{1, \ldots, n-1\}$, $j \in \{2 \ldots n\}$ gilt $\beta_{i,j} = \beta_{j,i}$. Tridiagonalmatrizen können bei der Berechnung von Splines [23] auftreten.
Eine Matrix $B \in I\!\!R^{n \times n}$ heißt *obere* (bzw. *untere*) *Bidiagonalmatrix*, wenn die Komponenten der unterern (bzw. oberen) Nebendiagonalen Null sind. Tridiagonal- und Bidiagonal-Matrizen gehören schon für $n > 10$ zu den *Sparse Matrizen.*
Matrizen, bei denen unterhalb (bzw. oberhalb) der Hauptdiagonalen alle Matrix-Komponenten gleich Null sind, nennt man *obere* (bzw. *untere*) *Dreicksmatrizen.*

2.3 Rang-1-Modifikation

Seien $u, v \in I\!\!R^n \backslash \{0\}$ und $A \in I\!\!R^{n \times n}$ eine reguläre Matrix. Eine reguläre Matrix $\tilde{A} \in I\!\!R^{n \times n}$, die durch die Gleichung

$$\tilde{A} = A + uv^T \tag{2}$$

entsteht, nennt man auch eine durch *Rang-1-Modifikation* entstandene Matrix.

2.4 Elementare Rotationsmatrizen und das Jacobi-Verfahren

Das Jacobi-Verfahren wird wegen seiner numerischen Gutartigkeit und Einfachheit häufig zur Lösung des vollständigen Eigenwertproblems verwendet. Da es auch zu den ältesten numerischen Verfahren gehört, gibt es mehrere Varianten.

Eine Matrix des Typs

$$G_{ij} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c_{ii} & & -s_{ij} & & \\ & & & \ddots & & & \\ & & s_{ji} & & c_{jj} & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \tag{3}$$

$$\text{mit } c_{ii} = c_{jj} = c \text{ und } s_{ij} = s_{ji} = s \text{ und } c^2 + s^2 = 1 \tag{4}$$

heißt *Jacobi-Rotations-Matrix* oder auch *Givens-Rotations-Matrix.* Die Matrix unterscheidet sich von einer Einheitsmatrix I in nur vier Elementen. Aufgrund der Bedingungen in (4) gilt $GG^T = I$, d. h. Jacobi-Rotationsmatrizen gehören zur Klasse der *orthogonalen Matrizen.*

Die Anwendung einer Matrix G_{ij} als Abbildungsmatrix

$$\begin{aligned} G_{ij} : I\!R^n &\longrightarrow I\!R^n \\ x &\longmapsto y := G_{ij}x \end{aligned} \tag{5}$$

mit $c = \cos\varphi$, $s = \sin\varphi$ und $|\varphi| \leq \pi$ läßt sich als Drehung um den Winkel $-\varphi$ in der durch e_i und e_j aufgespannten Ebene deuten. Es wird nur die i-te und j-te Komponente verändert.

2.4.1 Einseitiges Jacobi-Verfahren

Beim einseitigen Jacobi-Verfahren werden orthogonale Rotationsmatrizen G_{ij} so bestimmt, daß gilt:

$$AG_1 \cdots G_n = \tilde{A} = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)\ . \tag{6}$$

Für die Spalten von $\tilde{A}$ gilt $\|\tilde{a}_i\| \geq \|\tilde{a}_j\|$ für $i < j$ und es gilt $\tilde{a}_i^T \tilde{a}_j = \delta_{ij} \sum\limits_{k=1}^{n} \tilde{a}_{ki}^2$. Dabei ist δ_{ij} das Kroneckersymbol: $\delta_{ij} = 1$ für $i = j$, $\delta_{ij} = 0$ sonst.

Allgemein können mit dem einseitigen Jacobi-Verfahren Matrizen auf Dreiecksgestalt transformiert werden. Ein Spezialfall ist die Diagonalisierung einer Dreicksmatrix.

2.4.2 Zweiseitiges Jacobi-Verfahren

Das *zweiseitige Jacobi-Verfahren* wird auch als *Jacobi-Verfahren* bezeichnet. Das Jacobi-Verfahren ist ein (im allgemeinen unendlicher) iterativer Prozess zur Umwandlung einer symmetrischen Matrix auf Diagonalgestalt. Bei jedem Iterationsschritt werden Jacobi-Rotationsmatrizen dergestalt bestimmt, daß ein Nicht-Diagonal-Element eliminiert wird. In der Literatur wird gewöhnlich unterschieden in

- das klassische Jacobi-Verfahren: Dabei wird in jedem Iterationsschritt das betragsgrößte Nicht-Diagonal-Element der Matrix als Pivotelement gewählt.
- das zyklische Jacobi-Verfahren: Der Algorithmus geht bei der Elimination der Nicht-Diagonal-Elemente zeilenweise oder spaltenweise vor. Dabei wird die Suche nach dem betragsgrößten Nicht-Diagonal-Element vermieden.
- die Schwellenmethode (*Threshold-Method*): Die Schwellenmethode ist eine spezielle Variante des zyklischen Jacobi-Verfahrens. Man bestimmt eine absteigende Folge von Schwellen und eliminiert in beliebiger Reihenfolge alle Nicht-Diagonal-Elemente, die größer als die aktuelle Schwelle sind. Danach geht man zur nächsten Schwelle und setzt den Algorithmus fort.

2.5 Singulärwertzerlegung

Der *Satz über die Singulärwertzerlegung* (SVD[1]) zeigt, daß auch nicht-quadratische Matrizen mittels orthogonaler Transformationsmatrizen auf Diagonalgestalt umgeformt werden können.

Satz 2.1 (Satz über die Singulärwertzerlegung) *Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ gibt es orthogonale Matrizen $U \in \mathbb{R}^{m\times m}$ und $V \in \mathbb{R}^{n\times n}$, so daß*

$$U^T AV = \Sigma = diag\,(\sigma_1, \ldots, \sigma_l) \in \mathbb{R}^{m\times n} \tag{7}$$

mit $l := \min\{m,n\}$ und

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \text{ und } \sigma_{r+1} = \cdots = \sigma_l = 0 \tag{8}$$

gilt. Die Zahlen $\sigma_1, \ldots, \sigma_l$ sind durch A eindeutig festgelegt und heißen Singulärwerte von A. Die Anzahl r der positiven Singulärwerte ist gleich dem Rang der Matrix A.

Beweis: Die Matrix A^TA ist symmetrisch und wegen $x^TA^TAx = \|Ax\|_2^2 \geq 0$ positiv semidefinit. Aufgrund der Eigenwertzerlegung einer symmetrischen Matrix existiert dann eine orthogonale Matrix $V = (v_1, \ldots, v_n) \in \mathbb{R}^{n\times n}$ mit $V^T\left(A^TA\right)V = \Lambda = diag\,(\lambda_i)$ und $\lambda_i \geq 0$.
O.B.d.A. kann für die Eigenwerte von A^TA angenommen werden, daß $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ und $\lambda_{r+1} = \cdots = \lambda_n = 0$ mit $r \in \{0, \ldots, n\}$ gilt. Wegen $r = rang\left(A^TA\right)$ und $rang\left(A^TA\right) = rang\,(A)$ folgt $rang\,(A) = r$.
Aus $A^TAv_j = \lambda_j v_j$ folgt $\lambda_j = v_j^TA^TAv_j = \|Av_j\|_2^2$, also $Av_j \neq 0$ für $j \in \{1, \ldots, r\}$ und $Av_j = 0$

[1] Es handelt sich dabei um die Abkürzung der englischen Bezeichnung: *singular value decomposition*.

für $j \in \{r+1,\ldots,n\}$.
Sei $\sigma_j := \sqrt{\lambda_j}$ und $u_j := Av_j \frac{1}{\sigma_j}$ für $j \in \{1,\ldots,r\}$, dann ergibt sich für $i,j \in \{1,\ldots,r\}$

$$u_i^T u_j = \frac{1}{\sigma_i \sigma_j} v_i^T A^T A v_j = \frac{\lambda_j}{\sigma_i \sigma_j} v_i^T v_j = \begin{cases} 1 & \text{für} \quad i = j \\ 0 & \text{für} \quad i \neq j \end{cases} \tag{9}$$

Die Vektoren $\{u_1,\ldots,u_r\}$ bilden daher ein Orthonormalsystem und können durch Hinzunahme weiterer Vektoren $u_{r+1},\ldots,u_m$ zu einer Orthonormalbasis von $I\!R^m$ ergänzt werden. Dann ist $U = (u_1,\ldots,u_m)$ orthogonal. Mit der Festlegung $\sigma_{r+1} = \cdots = \sigma_l = 0$ folgt wegen

$$\left(U^T A V\right)_{ij} = u_i^T A v_j = \begin{cases} \sigma_j u_j^T u_j & \text{für} \quad j = 1,\ldots,r \\ 0 & \text{für} \quad j = r+1,\ldots,n \end{cases} \tag{10}$$

die gesuchte Beziehung $U^T A V = \Sigma$. Ebenso folgt aus der Konstruktion, daß es genau r positive Singulärwerte gibt, diese also mit dem Rang der Matrix A übereinstimmt.□

2.6 Das symmetrische Eigenwertproblem

In den Arbeiten [26] und [20] werden Ergebnisse der nachfolgenden Sätze und Beweise in leicht umformulierter Form zur Lösung der Singulärwertzerlegung verwendet. Die nachfolgenden Sätze und Beweise sind [16] entnommen.

Lemma 2.1 *Sei $D = diag\,(d_1,\ldots,d_n) \in I\!R^{n\times n}$ mit $d_1 > \cdots > d_n$. Sei $\rho \neq 0$ und $z \in I\!R^n$ hat keine Nullkomponenten. Wenn*

$$\left(D + \rho z z^T\right) v = \lambda v \text{ mit } v \neq 0 \tag{11}$$

dann ist $z^T v \neq 0$ und $D - \lambda I$ ist nicht singulär.

Beweis: Wenn λ Eigenwert von D ist, dann ist $\lambda = d_i$ für einige i und somit

$$0 = e_i^T \left[(D - \lambda I)\, v + \rho \left(z^T v\right) z\right] = \rho \left(z^T v\right) z_i. \tag{12}$$

Da ρ und z_i nicht Null sind, muß $0 = z^T v$ und $Dv = \lambda v$ gelten. D hat jedoch unterschiedliche Eigenwerte und daher gilt $v \in span\,\{e_i\}$. Aber dann gilt $0 = z^T v = z_i$, was ein Widerspruch ist. Somit haben D und $D + \rho z z^T$ keine gemeinsamen Eigenwerte und $z^T v \neq 0$. □

Satz 2.2 *Sei $D = diag\,(d_1,\ldots,d_n) \in I\!R^{n\times n}$ mit $d_1 > \cdots > d_n$. Sei $\rho \neq 0$ und $z \in I\!R^n$ hat keine Nullkomponenten. Wenn $V = (v_1,\ldots,v_n) \in I\!R^{n\times n}$ eine Orthogonalmatrix ist, so daß*

$$V^T \left(D + \rho z z^T\right) V = diag\,(\lambda_1,\ldots,\lambda_n) \tag{13}$$

mit $\lambda_1 \geq \cdots \geq \lambda_n$ dann gilt:

a) Die λ_i sind die n Nullstellen von $f(\lambda) = 1 + \rho z^T (D - \lambda I)^{-1} z$.

b) Wenn $\rho > 0$ dann gilt $\lambda_1 > d_1 > \lambda_2 > \cdots > \lambda_n > d_n$. Wenn $\rho < 0$ dann gilt $d_1 > \lambda_1 > d_2 > \cdots > d_n > \lambda_n$.

c) Der Eigenvektor v_i ist eine Mannigfaltigkeit von $(D - \lambda I)^{-1} z$.

Beweis: Wenn $\left(D + \rho z z^T\right) v = \lambda v$, dann gilt

$$(D - \lambda I) v + \rho \left(z^T v\right) z = 0. \tag{14}$$

Von Lemma 2.1 wissen wir, daß $D - \lambda I$ nicht singulär ist. Somit ist $v \in span \left\{(D - \lambda I)^{-1} z\right\}$ und dadurch *c)* bestätigt. Wenn man außerdem auf beiden Seiten der Gleichung (14) $z^T (D - \lambda I)^{-1}$ anwendet, erhält man

$$z^T v \left(1 + \rho z^T \left('D - \lambda I\right)^{-1} z\right) = 0. \tag{15}$$

Nach Lemma 2.1 gilt $z^T v \neq 0$ und das zeigt, daß für $\lambda \in \lambda\left(D + \rho z z^T\right)$ $f(\lambda) = 0$. Es muß noch gezeigt werden, daß alle Nullstellen von f Eigenwerte von $D + \rho z z^T$ sind und *b)* erfüllt ist. Um das zu zeigen, schaut man sich die Gleichungen

$$f(\lambda) = 1 + \rho \left(\frac{z_1^2}{d_1 - \lambda} + \cdots + \frac{z_n^2}{d_n - \lambda}\right) \tag{16}$$

$$f'(\lambda) = -\rho \left(\frac{z_1^2}{(d_1 - \lambda)^2} + \cdots + \frac{z_n^2}{(d_n - \lambda)^2}\right) \tag{17}$$

an. Zwischen den Polstellen ist f monoton. Das erlaubt den Schluß, daß wenn $\rho > 0$ ist, f ganz genau n Nullstellen besitzt, jeweils eine davon in den Intervallen

$$(d_n, d_{n-1}), \ldots, (d_2, d_1), (d_1, \infty). \tag{18}$$

Wenn $\rho < 0$ ist, hat f genau n Nullstellen, jeweils eine davon in den Intervallen

$$(-\infty, d_n), (d_n, d_{n-1}), \ldots, (d_2, d_1). \tag{19}$$

Aus beiden Fällen folgt, daß die Nullstellen von f die Eigenwerte von $D - \rho z z^T$ sind. □

3 Massiv parallele Algorithmen und Rechner-Modelle

Um parallele Algorithmen zu behandeln, werden zunächst einige Rechnermodelle skizziert.

3.1 Klassifizierung der Rechnermodelle

Parallelität in Computern läßt sich auf mehreren Ebenen betrachten. Für die unterschiedlichen Rechnermodelle in Kombination mit der Numerischen Linearen Algebra eignet sich die Klassifizierung nach Flynn [14]. Dabei werden die Instruktionen und Daten in je zwei Klassen geteilt.

Tabelle 1: Klassifikation unterschiedlicher Rechnermodelle nach Flynn

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

MISD-Rechner sind unbedeutend; die Rechner nach dem *von-Neumann-Modell* gehören zu der Klasse der SISD-Computer. Für massiv parallele Algorithmen sind die SIMD- und MIMD-Modelle interessant.

Um Daten auszutauschen, können die Prozessorelemente bzw. Prozessoren auf unterschiedlichste Weise miteinander vernetzt sein. Ausführliche Darstellungen über mögliche *Verbindungsnetzwerke* finden sich in [5], [7], [15], [29], u. a.

3.1.1 Das SIMD-Rechnermodell

Unter SIMD-Rechnermodellen versteht man *Arrayrechner* und *Vektorrechner* . Arrayrechner und Vektorrechner unterscheiden sich auf der Hardware-Ebene durch

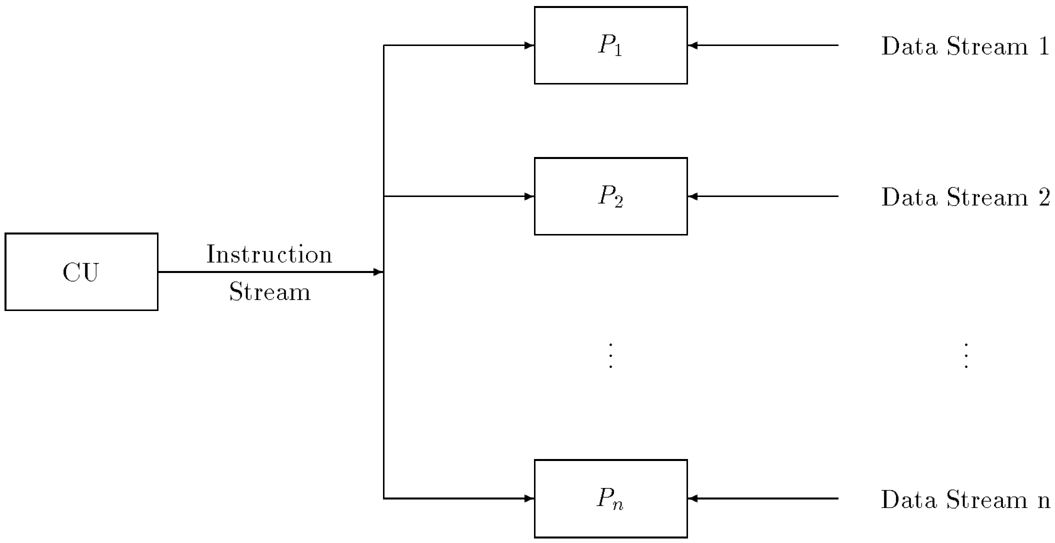
- die Art der Kommunikationsmöglichkeit ihrer Prozessorelemente: Der Arrayrechner besitzt komplexere Verbindungsstrukturen (Netzwerk).
- Organisation des Datenzugriffs der Prozessorelemente: Beim Vektorrechner sind außer LOAD und STORE alle Befehle Registeroperationen.

SIMD-Rechner besitzen einen *Steuerrechner,* der über den Daten- und den Steuerbus die *Prozessorelemente* (PE) kontrolliert. Der Steuerrechner bestimmt, ob ein Prozessorelement aktiv oder inaktiv ist, er ist als einziger mit weiteren Peripheriegeräten verbunden.

Aufgrund des einfachen Aufbaus kann die Zahl der Prozessoren bzw. Prozessorelemente sehr groß sein. Die beiden bekanntesten Realisierungen sind Array-Rechner und *gepipelinte* Floating-Point (überlappende Fließkomma) Vektorfunktionseinheiten. Von Natur aus eignen sich daher SIMD-Rechner für Anwendungen, die auf Vektoren bzw. Matrizen operieren.

Die *Prozessorelemente* bestehen aus arithmetisch-logischen Einheiten (ALU) und lokalem Speicher. SIMD-Rechner können auch mit gemeinsamem Speicher (shared memory) ausgestattet sein.

Abbildung 1: SIMD-Rechner-Modell



SIMD-Rechner arbeiten *synchron*: Die aktiven Prozessorelemente können zu einem Zeittakt nur den gleichen Befehl ausführen. Der Datenaustausch zwischen einer Auswahl von bzw. allen Prozessorelementen findet ebenfalls synchron statt. Bei dieser Art der Parallelverarbeitung entfallen aufwendige und unter Umständen auch fehlerhafte Software-Synchronisationsmechanismen (Semaphore, Monitore, Rendevous u. ä.).

Tabelle 2: Beispiele für SIMD-Rechner

Hersteller	Modell	Prozessoren	Speicher je PE	Peak Performance
Thinking Machines Corporation Massachusetts USA	CM-2	65.535	128 KB	2.500 MIPS (32-Bit-Operationen) 10.000 MFLOPS (Skalarprodukt, 32-Bit) 5.000 MFLOPS (Skalarprodukt, 64-Bit)
MasPar Computer Corporation California USA	MP-1216	16.384	64 KB	30.000 MIPS (32-Bit-Operationen) 1.500 MFLOPS (Add./Mul., 32-Bit) 600 MFLOPS (Add./Mul., 64-Bit)

3.1.2 Das MIMD-Rechnermodell

MIMD-Rechner besitzen im allgemeinen Universalprozessoren, die unabhängig voneinander parallel arbeiten können. Man spricht deshalb auch von Multiprozessor-Systemen. Rechnermodelle, die asynchrone Parallelität bieten, sind bezüglich der Gesamtauslastung der Prozessoren sehr flexibel, da unterschiedliche Verarbeitungsschritte zum selben Zeitpunkt durchgeführt werden können. Die Anzahl der Prozessoren ist wegen des komplexeren Aufbaus geringer als bei SIMD-Rechnern.

MIMD-Architekturen lassen sich aufgrund ihrer physikalischen Struktur in zwei Hauptkategorien einteilen:

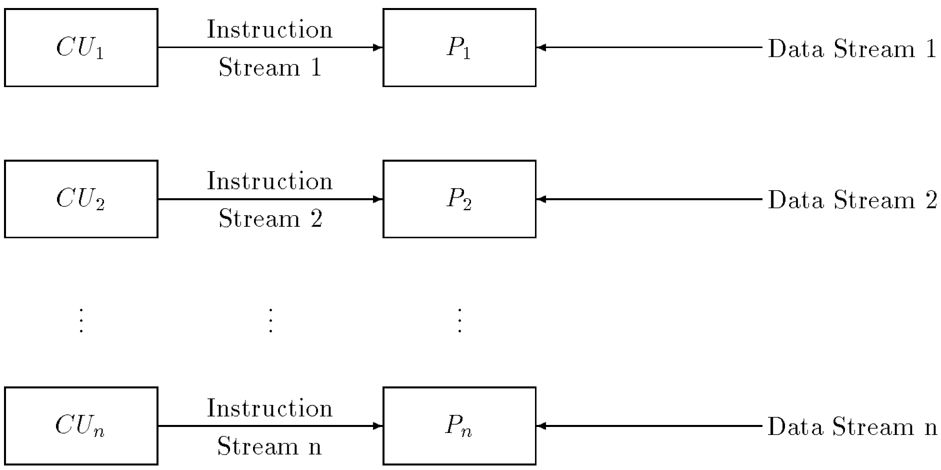
- Systeme mit gemeinsamem Speicher (shared memory architecture)
- Systeme mit verteiltem (lokalem) Speicher (distributed memory architecture)

Man spricht auch dann von einem *System mit gemeinsamem Speicher*, wenn die Prozessoren zusätzlich zu dem allen gemeinsamen Speicher noch jeweils einen lokalen Speicher haben [15].

Systeme mit gemeinsamem Speicher bieten die Möglichkeit einen globalen Adressraum anzusprechen. Die Verwendung von herkömmlichen Programmiersprachen (C, Fortran) mit gering modifiziertem Programmierstil (Thread-Programmierung) ist möglich. Dagegen erfordern Systeme mit verteiltem Speicher vom Benutzer und vom Compiler die Partitionierung der Anwendungsprogramme in eine Vielzahl kooperierender Prozesse. Die Programmierung muss die Kommunikation zwischen den Prozessen und damit deren Synchronisation einschließen. Entsprechende Erweiterungen der Programmiersprachen sind in Entwicklung (beipielsweise *High-Performance Fortran* HPF). Damit ist es möglich, Anwendungsprogramme unabhängig von der aktuellen Konfiguration zu schreiben.

Vertreter der MIMD-Klasse sind beispielsweise die von der Firma *CRAY* realisierten Rechner *X-MP, Y-MP, T3D* und *T3E*. Ebenso gehören die Rechner EGPA (Uni Erlangen), Suprenum und MANNA (beide GMD) zu den MIMD-Rechnern.

Abbildung 2: MIMD-Rechner-Modell



3.2 Algorithmen für Parallelrechner

Der zentrale Begriff des parallelen Rechnens ist die in den auszuführenden Programmen zu findende Parallelität. *Parallelität* ist hierbei eine Kurzform für die Möglichkeit der parallelen (gleichzeitigen) Ausführungen mehrerer Operationen. Der *Grad an Parallelität* gibt an, wieviel Operationen parallel ausführbar sind. Dieser Parallelitätsgrad ist im allgemeinen keine konstante Größe, sondern er hängt vom jeweiligen Stand der Ausführung (d. h. vom Zustand des zugehörigen Programmlaufs) ab. [15]

Von oben nach unten kann man vier Ebenen unterscheiden, auf denen Parallelität gegeben sein kann:

- die Ebene der Benutzerprogramme (spielt für die Betrachtung in den Parallelrechner-Architekturen keine große Rolle)
- die Ebene der parallelen Prozesse oder Tasks
- die Ebene der Anweisungen
- die Ebene der Elementar-Operationen

Parallelarbeit auf der Prozeßebene muß beim gegenwärtigen Stand der Technik vom Programmierer festgelegt werden, wofür ihm u. U. Werkzeuge zur Verfügung stehen. Die Nutzung der Parallelität auf der Anweisungsebene erfordert eine Analyse der zwischen den Anweisungen bestehenden Datenabhängigkeiten. Dieses Gebiet wird inzwischen weitgehend beherrscht. Parallelität auf der Ebene der Elementaroperationen ist vom Compiler leicht zu erkennen. [15]

Das Hauptziel der Parallelen Numerik ist die Konstruktion paralleler Algorithmen zur Ausnutzung mehrerer paralleler Prozessoren (SIMD/MIMD-Systeme). Zur Umsetzung eines parallelen Algorithmus benötigt man eine parallele Programmiersprache ebenso, wie einen guten Compiler. [29]

Ab einer gewissen Größe spricht man bei Rechnern von einem *massiv-parallelen System*, wobei diese Grenze aber nicht definiert, sondern eine Funktion der zur Verfügung stehenden Technik höchstintegrierter Schaltungen ist [15]. Algorithmen, die für massiv-parallele Systeme entwickelt werden, nennt man gewöhnlich ebenfalls massiv-parallel.

3.3 Anwendung

Die Numerische Mathematik und insbesondere die Numerische Lineare Algebra liefert effiziente und stabile Algorithmen für die massiv parallele Programmierung. SIMD- und MIMD-Rechner sind durch ihr oben beschriebenes Modell besonders dafür geeignet. Vektor- und Matrixmultiplikation erreichen einen Parallelisierungsgrad und damit eine Effizienz, die vom klassischen *von-Neumann-Rechner* bei gleichzeitiger Weiterentwicklung bauartbedingt nicht erreicht wird. Die entsprechenden Rechnermodelle können jedoch nicht auf allen Ebenen direkt verglichen werden.

Bei datenparalleler Programmierung tritt häufig eine Analogie zum Konzept des virtuellen Speichers auf. Wenn beispielsweise für einen parallelen Datenverarbeitungsschritt mehr Prozessoren benötigt werden als physikalisch vorhanden sind, dann muß dieser Schritt auf virtuelle Prozesse abgebildet werden, die sich dann bei der Verarbeitung die vorhandenen Prozessoren teilen müssen. Dadurch erhöht sich in erheblichem Maße die Kommunikationszeit und damit die Gesamtlaufzeit. Je größer die Anzahl der Prozessoren, desto seltener tritt dieses Problem auf. Umgekehrt ist es jedoch ebenso notwendig zu prüfen, ob parallele Programme auf Rechnern mit vielen Prozessoren laufen sollen, wenn nachweislich eine ausreichende Prozessorauslastung nicht gewährleistet wird. Diese Prüfung erfolgt einerseits schon bei der Programmierung, bei der eine maximale Problemgröße vorgegeben werden kann, andererseits auch bei der Anwendung, wenn einem Programm eine Aufgabe mit einer definierten Größe übergeben wird.

In der Praxis werden massiv parallele Algorithmen häufig auch auf sogenannten *Hybridsystemen* eingesetzt. Hybride Parallelrechner [5] sind Kombinationen aus SIMD-, MIMD- und Pipelinerechnern. Ergänzend wird an zwei Beispielen ein Einblick gegeben:

- **Multiple-SIMD:** Bei einem Multiple-SIMD-Rechner (MSIMD) sind mehrere Steuerrechner vorhanden, die jeweils für eine Teilmenge aller PEs verantwortlich sind. Dies entspricht einer (MIMD-artigen) Zusammenschaltung von mehreren unabhängigen SIMD-Rechnern. Es können mehrere unabhängige Programme parallel laufen.
- **SPMD-Rechner:** Die Abkürzung steht für *same program multiple date*. Ein Parallelrechner wird dabei von einem einzigen logischen Kontrollfluß (Programm) gesteuert, wobei die Einfachheit der SIMD-Programmierung mit der Flexibilität der MIMD-Rechner kombiniert wird. Beispielsweise müssen bei einem SIMD-Rechner die beiden Zweige einer `IF-THEN-ELSE`-Anweisung nacheinander ausgeführt werden, während ein SPMD-Rechner unter bestimmten Voraussetzungen beide Zweige parallel ausführen kann. Die *CM-5* der *Thinking Machines Corporation*, Cambridge Massachusetts, ist ein SPMD-Modell.

4 Das allgemeine Divide-and-Conquer-Prinzip

Das *Divide-and-Conquer-Prinzip* ist eine algorithmische Technik, die bei der Formulierung von Numerischen Algorithmen und massiv parallelen Programmen untersucht wird.

Ein gegebenes Problem P wird in n kleinere, unabhängige Teilprobleme zerlegt. Diese Teilprobleme müssen nun einzeln gelöst und die so erhaltenen Teilergebnisse zur Lösung L des Gesamtproblems zusammengefaßt werden. Falls die Teilprobleme weiterhin eine bestimmte Größe überschreiten, so kann die D&C-Strategie auf diese erneut angewendet werden.

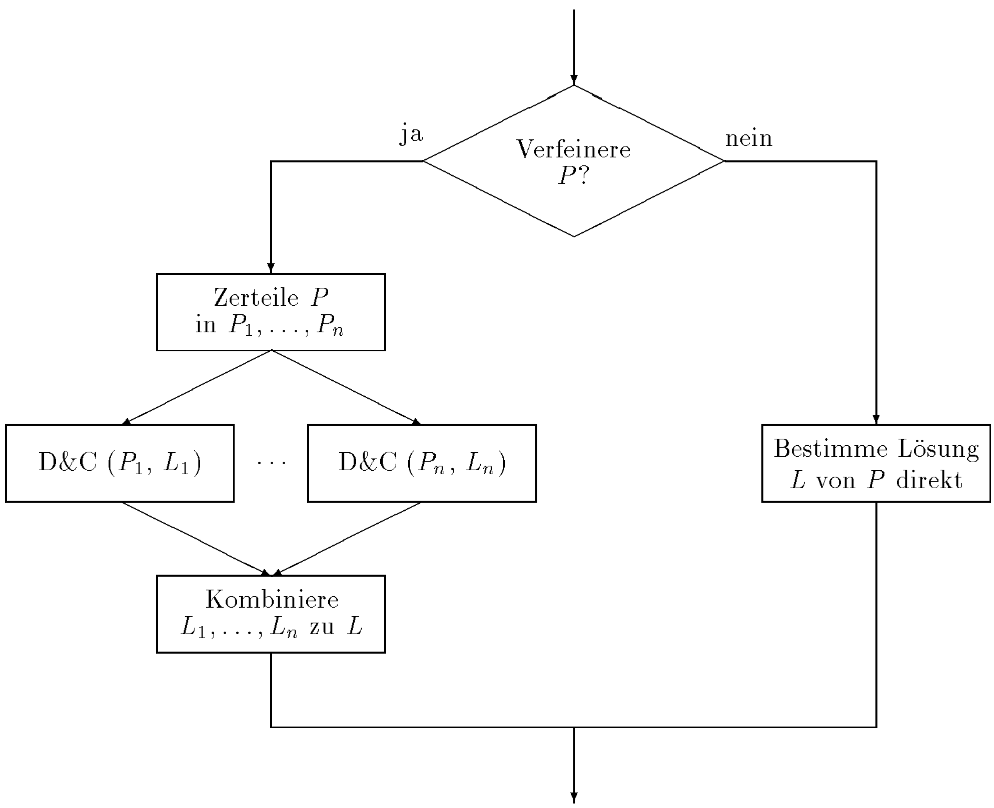
Durch den rekursiven Lösungsansatz eines Problems der Größe n wird das Schema verdeutlicht:

`1. Schritt:`
`Wenn` (n `> 1) dann` Teile das Problem in mindestens zwei Teilprobleme
`sonst` löse das Problem (auf direktem Weg).

`2. Schritt:`
Löse jedes entstandene Teilproblem wie in: *`1. Schritt`*

`3. Schritt:`
Setze die erhaltenen Teillösungen zusammen und gebe die Gesamtlösung aus.

Abbildung 3: Ablaufschema von $D\&C(P, L)$ aus [18]



Die entstandenen Teilprobleme sind gleich aufgebaut und können rekursiv aufgelöst werden. Die prozedurale Form kann wie folgt beschrieben werden:

Algorithmus 4.1 (Allgemeiner Divide-and-Conquer-Algorithmus)

```
PROCEDURE DaC (Eingabe: Problem; Ausgabe: Loesung)

IF (Problem teilbar)
   THEN Teile das Problem auf in k Teilprobleme;
        DaC (Teilproblem 1, Teilloesung 1);
        ⋮
        DaC (Teilproblem k, Teilloesung k);
        zusammenfuegen (Teiloesung 1, ..., Teilloesung k,
        Loesung);
   ELSE Bestimme die Loesung direkt.
```

Dabei ist `zusammenfuegen` *eine eigene Prozedur, die* `Loesung` *zurückgibt.*

Aus den Algorithmen geht hervor, daß die Rekursion abbricht, sobald sich das Problem nicht weiter aufteilen läßt. Besonders interessant für die parallele Datenverarbeitung ist das Vorliegen von mehreren gleichartigen unabhängigen Teilproblemen nach dem Zerteilen eines größeren Problems. Theoretisch können die Teilprobleme jetzt parallel gelöst werden. In der Praxis geht das nur dann, wenn die entsprechende Anzahl an Prozessoren vorhanden ist. Die Vorteile des Divide-and-Conquer-Prinzips sind bei einfachen Algorithmen offensichtlich, z. B. *Mergesort*. Für numerische Algorithmen, bei denen es nicht nur auf die Effizienz sondern auch auf die Rechengenauigkeit ankommt, sind Untersuchungen mit vorhandenen Rechnerarchitekturen und Datenstrukturen notwendig.

5 Divide-and-Conquer für das bidiagonale Singulärwertproblem

Bei der Modellierung von naturwissenschaftlichen Experimenten tritt häufig ein überbestimmtes System linearer Gleichungen auf:

(20) $$Ax = b,\ A := (a_{ij}),\ A \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ rang\,(A) \leq n,\ m > n.$$

Die Matrix A ist oft dünn besetzt. Ein solches System hat gewöhnlich keine eindeutige Lösung, daher wird das (Fehler-) Minimum gesucht:

(21) $$\min_x \|Ax - b\|_p\,,\ \|\cdot\|_p \text{ geeignete Norm.}$$

Die Euklidische Norm, also der Fall $p = 2$ hat in der Praxis die größte Bedeutung erlangt. Man spricht vom *linearen Quadratmittelproblem*, bzw. von der *Methode der kleinsten (Fehler-) Quadrate* oder im englischen Sprachgebrauch vom *Least Squares Problem*. [16] [27] [31] [34]

Satz 5.1 *Angenommen, für die Matrix $A \in \mathbb{R}^{m\times n}$ mit $rang(A) = r$ sei die Singulärwertzerlegung (vgl. Satz 2.1) $A = U\Sigma V^T$ mit den Spaltenpartitionen $U = (u_1, \dots, u_m)$ und $V = (v_1, \dots, v_n)$ gegeben. Für ein $b \in R^m$ hat der Vektor*

(22) $$x_{LS} = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i \quad \in \mathbb{R}^n$$

die kleinste 2-Norm von $\|Ax - b\|_2$.

Der Satz wurde ohne Beweis [16] entnommen. Für unterschiedliche b ist es sinnvoll, die Singulärwertzerlegung (SVD) von A zu kennen. Für die Berechnung der Pseudoinversen wird ebenfalls die Singulärwertzerlegung herangezogen.

Im nachfolgenden werden zwei D&C-Techniken vorgestellt, die für die Singulärwertzerlegung von Bidiagonal-Matrizen B konzipiert sind.

5.1 SVD auf der Basis der Rang-1-Modifikation

Die erste Methode basiert auf Rang-1-Modifikation (vgl. Abschn. 2.3) und ist eine effiziente Alternative zum D&C-Algorithmus, der zur Eigenwertbestimmung einer $2n \times 2n$-Tridiagonal-Matrix angewendet wird. Er vermeidet durch Umformulierung des Produkts numerische Schwierigkeiten, die sich mit der expliziten Bildung von BB^T oder $B^T B$ ergeben.

5.1.1 Matrixmodifikation

Der Algorithmus baut auf der *Rang-1-Modifikation* (vgl. Abschn. 2.3) auf. Im Detail erlaubt die *Rang-1-Modifikation*

(23) $$B = \begin{pmatrix} B_1 & \beta e_k e_1^T \\ 0 & B_2 \end{pmatrix} = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} + \beta \begin{pmatrix} e_k \\ 0 \end{pmatrix} (0, e_1^T)$$

der Matrix B mit $\beta = \beta_k$ die implizite Bildung von BB^T wie folgt:

$$
\begin{aligned}
BB^T &= \begin{pmatrix} B_1 & \beta e_k e_1^T \\ 0 & B_2 \end{pmatrix} \begin{pmatrix} B_1^T & 0 \\ \beta e_1 e_k^T & B_2^T \end{pmatrix} \\
&= \begin{pmatrix} B_1 B_1^T & 0 \\ 0 & B_2 \left(I - e_1 e_1^T\right) B_2^T \end{pmatrix} + \begin{pmatrix} \beta e_k \\ B_2 e_1 \end{pmatrix} \left(\beta e_k^T, e_1^T B_2^T\right) \\
&= \begin{pmatrix} B_1 B_1^T & 0 \\ 0 & \hat{B}_2 \hat{B}_2^T \end{pmatrix} + \begin{pmatrix} \beta e_k \\ \alpha e_1 \end{pmatrix} \left(\beta e_k^T, \alpha e_q^T\right),
\end{aligned}
\tag{24}
$$

wobei die Matrix

$$
\hat{B}_2 = B_2 \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} \tag{25}
$$

die bidiagonale Matrix B_2 ist, deren erste Spalte durch den Null-Vektor ersetzt wurde, und $\alpha e_1 = B_2 e_1$. Die SVD $B_1 = U_1 \Sigma_1 V_1^T$ und $\hat{B}_2 = \hat{U}_2 \hat{\Sigma}_2 \hat{V}_2^T$ können unabhängig voneinander berechnet werden. Zur Entwicklung von Eliminationsregeln für die SVD ist es notwendig, den grundlegenden Update-Schritt neu zu formulieren.

Um das zu erreichen, sei die $(n-k) \times (n-k-1)$-Matrix $\tilde{B}_2$ definiert durch

$$
\left(0, \tilde{B}_2\right) \equiv \hat{B}_2 \equiv B_2 \left(I - e_1 e_1^T\right). \tag{26}
$$

Bei der SVD von $\tilde{B}_2 = \tilde{U}_2 \tilde{\Sigma}_2 \tilde{V}_2^T$ fällt auf, daß

$$
\hat{B}_2 = \begin{pmatrix} \tilde{U}_2 & \tilde{u} \end{pmatrix} \begin{pmatrix} \tilde{\Sigma}_2 \\ 0^T \end{pmatrix} \tilde{V}_2^T, \tag{27}
$$

wobei $\tilde{u}$ ein Einheitsvektror orthogonal zu den Spalten von $\tilde{U}_2$ ist. Wenn $B_1 = u_1 \Sigma_1 V_1^T$,

$$
B = \begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{U}_2 & \tilde{u} \end{pmatrix} \begin{pmatrix} \Sigma_1 & u_1 & 0 \\ 0 & u_2 & \tilde{\Sigma}_2 \\ 0 & \mu & 0 \end{pmatrix} \begin{pmatrix} V_1^T & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \tilde{V}_2^T \end{pmatrix}. \tag{28}
$$

Für eine einfachere Notation permutieren wir (28) und erhalten

$$
B = \begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{U}_2 & \tilde{u} \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 & u_1 \\ 0 & \tilde{\Sigma}_2 & u_2 \\ 0 & 0 & \mu \end{pmatrix} \begin{pmatrix} V_1^T & 0 & 0 \\ 0 & 0 & \tilde{V}_2^T \\ 0 & 1 & 0 \end{pmatrix}. \tag{29}
$$

Eliminationsregeln werden dann für die innere Matrix

$$
\bar{M} \equiv \begin{pmatrix} \bar{\Sigma} & \bar{u} \\ 0 & \mu \end{pmatrix} \equiv \begin{pmatrix} \Sigma_1 & 0 & u_1 \\ 0 & \tilde{\Sigma}_2 & u_2 \\ 0 & 0 & \mu \end{pmatrix} \tag{30}
$$

benötigt, wobei $\bar{\Sigma} = diag\,(\bar{\sigma}_1, ..., \bar{\sigma}_n - 1)$ und $\bar{u} = (\bar{\mu}_1, ..., \bar{\mu}_n - 1)$. In exakter Arithmetik löst sich das Problem auf, wenn einer der folgenden Fälle eintritt:

1. Ein Element von $\bar{u}$ ist Null: $\bar{\mu}_j = 0$.

2. Diagonalelemente von $\bar{\Sigma}$ sind gleich: $\bar{\sigma}_i \neq \bar{\sigma}_j,\ i = j$.

3. Ein Diagonalelement von $\bar{\Sigma}$ ist Null: $\bar{\sigma}_j = 0$.

Es kann leicht überprüft werden, daß $\bar{\sigma}_j$ ein Singulärwert von $\bar{M}$ ist, wenn $\bar{\mu}_j = 0$ gilt. Die zugehörigen linken und rechten Singulärvektoren stimmen mit dem j-ten kanonischen Vektor e_j überein. Die anderen beiden Fälle können auf den ersten Fall zurückgeführt werden, indem man entsprechende Ebenenrotationen anwendet, um Null-Komponenten in den Vektor $\hat{u}$ einzufügen:

1. Wenn $\bar{\sigma}_i \neq \bar{\sigma}_j$ gilt, dann werden beidseitige Drehungen angewendet. Eine Givensdrehung $G_{i,j}$ in der (i,j)-Ebene wird hergeleitet und auf $\bar{M}$ angewendet, so daß

$$(31)\qquad \bar{M} \longleftarrow \begin{pmatrix} G_{i,j} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \bar{\Sigma} & \bar{u} \\ 0 & \mu \end{pmatrix} \begin{pmatrix} G_{i,j}^T & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} G_{i,j}\bar{\Sigma}G_{i,j}^T & G_{i,j}\bar{u} \\ 0 & \mu \end{pmatrix}$$

mit $\tau^2 = \bar{\mu}_j^2 + \bar{\mu}_i^2$, $c = \frac{\bar{\mu}_j}{\tau}$ und $c = \frac{\bar{\mu}_i}{\tau}$

2. Wenn $\bar{\sigma}_i = 0$ gilt, dann wird eine einseitige Rotation in der (i,n)-Ebene von links angewendet. Dabei wird μ verwendet, um $\bar{\mu}_i$ zu eliminieren:

$$(32)\qquad \bar{M} \longleftarrow G_{i,n}\bar{M}$$

In der praktischen Anwendung müssen diese Eliminationsschritte den Grenzen der endlich genauen Arithmetik angepasst werden. Das Ergebnis ist eine Matrix der Form

$$(33)\qquad \bar{M}' = PH\bar{M}G^TP^T = \begin{pmatrix} \hat{\Sigma}_1 & 0 & 0 \\ 0 & \tilde{\Sigma} & u \\ 0 & 0 & \mu \end{pmatrix},$$

wobei $\tilde{\Sigma}$ unterschiedliche positive und u von Null verschiedene Elemente besitzt. P ist eine angepasste Permutationsmatrix, und G und H sind die zusammengefassten Produkte der Rotationsmatrizen, die sich aus jedem Eliminationsschritt ergeben.

Die Diagonalelemente von $\hat{\Sigma}_1$ sind die Singulärwerte von $\hat{M}$ mit passenden kanonischen Vektoren als Singulärvektoren. Nach der Elimination hat man nur noch die SVD der Matix

$$(34)\qquad M \equiv \begin{pmatrix} \tilde{\Sigma} & u \\ 0 & \mu \end{pmatrix}$$

zu berechnen. Die Quadrate der Singulärwerte von M, und ihre linken Singulärvektoren erhält man durch die Eigenwertzerlegung

$$(35)\qquad Y\Sigma^2Y^T \equiv MM^T \equiv \begin{pmatrix} \tilde{\Sigma}^2 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} u \\ \mu \end{pmatrix} \left(u^T,\ \mu\right).$$

Ein Eigenwert σ^2 von MM^T ist die Wurzel der Gleichung

$$(36)\qquad f\left(\sigma^2\right) = 1 + u^T\left(\tilde{\Sigma}^2 - \sigma^2 I\right)^{-1} u - \left(\frac{\mu}{\sigma}\right)^2$$

und kann unter Verwendung des *root-finder* aus [6] berechnet werden. Wenn $0 = \tilde{\sigma}_1^2 < \tilde{\sigma}_2^2 < \cdots < \tilde{\sigma}_n^2$ die sortierten Diagonalelemente von $diag\left(\tilde{\Sigma}^2, 0\right)$ sind, dann liegt der j-te Eigenwert σ_j^2

von MM^T im Intervall $\left(\tilde{\sigma}_j^2, \tilde{\sigma}_{j+1}^2\right)$ [6], und die Eigenwerte sind alle positiv. Der j-te Singulärwert von B ist σ_j, der zugehörige linke Singulärvektor ist

$$y_j = \begin{pmatrix} \left(\tilde{\Sigma}^2 - \sigma_j^2 I\right)^{-1} u \\ \frac{-\mu}{\sigma_j^2} \end{pmatrix} \theta, \tag{37}$$

wobei θ ein Normalisierungsfaktor ist. Der entsprechende rechte Singulärwert ist

$$x_j = \frac{M^T y_j}{\|M^T y_j\|_2}. \tag{38}$$

Algorithmus 5.1

1. *Löse für σ_j Gleichung (36).*
2. $y = \binom{v_j}{\eta_j} = \begin{pmatrix} \left(\tilde{\Sigma}^2 - \sigma_j^2 I\right)^{-1} u \\ \frac{-\mu}{\sigma_j^2} \end{pmatrix}.$
3. $x = \begin{pmatrix} \tilde{\Sigma}^2 v_j \\ -1 \end{pmatrix}.$
4. $y_j = \frac{y}{\|y\|_2}, x_i = \frac{x}{\|x\|_2}.$

Die Singulärwerte von $B = \hat{Y}\Sigma\hat{X}^T$ sind die von $\bar{M}$ und ihre Singulärvektoren können von $\bar{M}$ hergeleitet werden.

5.1.2 Transformations-Algorithmus und Fehleranalyse

Die Regeln für endlich genaue Floating-Point-Arithmetik sind in Algorithmus 5.2 zusammengefasst. Kleine Elemente von $\bar{u}$ oder $\bar{\Sigma}$ werden als Null angenommen, und eng beieinander liegende Diagonalelemente von $\bar{\Sigma}$ werden als gleich angenähert. Um die Elimination zu verfolgen, werden zwei Listen verwendet:

- *deflate-list* enthält die Indizes aller kleinen Elemente in der letzten Spalte der Matrix.
- *solve-list* enthält die verbleibenden Indizes. *solve-list* wird von 1 bis $n-1$ der Diagonalelemente von $\bar{\Sigma}$ initialisiert.

Nach den Eliminationen, enthält *deflate-list* die Indizes der Diagonalelemente der transformierten Matrix, welche die Singulärwerte von $\bar{M}$ sind, *solve-list* enthält die Indizes der Zeilen von $\bar{M}$ die in der eliminierten Matrix zurückgeblieben sind.

Algorithmus 5.2

1. *Für alle* $k \in$ `solve-list`,
   wenn $|\bar{\mu}_k| < \epsilon$, *verschiebe* k *aus* `solve-list` *nach* `deflate-list`.

2. *Permutiere die Indizes, so daß für die Elemente in* `solve-list` *mit aufsteigendem Index* $\bar{\sigma}_k \leq \bar{\sigma}_{k+1}$ *gilt.*
   ($M \longleftarrow P^T M P$ *so daß* $Pe_n = e_n$ *mit* $k' \in$ deflate-list *für* $1 \leq k' \leq |\text{deflate-list}|$ *und* $k' \in$ solve-list *für* $|\text{deflate-list}| + 1 \leq k' \leq n-1$.*)*

3. *Für alle* $k \in$ `solve-list` *ausser dem letzten,*
   wenn $|\bar{\sigma}_k - \bar{\sigma}_{k+1}|$ *klein ist, wende eine beidseitige Ebenendrehung an, so daß* $\mu_k \leftarrow 0$ *und verschiebe* k *aus* `solve-list` *nach* `deflate-list`.
   (Im PSVD wird eine beidseitige Drehung wie folgt durchgefürt:
   $\tau^2 = \left(\bar{\mu}_k^2 + \bar{\mu}_{k+1}^2\right)$
   wenn $|(\bar{\sigma}_k - \bar{\sigma}_{k+1})\bar{\mu}_k\bar{\mu}_{k+1}| < \epsilon\tau^2$ *dann*
   $c = \frac{\bar{\mu}_{k+1}}{\tau}$ *und* $s = \frac{\bar{\mu}_k}{\tau}$
   $\tilde{\sigma}_k \leftarrow c^2\bar{\sigma}_k + s^2\bar{\sigma}_{k+1}$
   $\tilde{\sigma}_{k+1} \leftarrow s^2\bar{\sigma}_k + c^2\bar{\sigma}_{k+1}$
   $\bar{\mu}_{k+1} \leftarrow \tau$
   $\bar{\mu}_k \leftarrow 0$*)*

4. *Für alle* $k \in$ `solve-list`,
   wenn $|\bar{\sigma}_k|$ *sehr klein ist, wende eine einseitige Ebenenrotation an, so daß* $\mu_k \leftarrow 0$, *und verschiebe* k *aus* `solve-list` *nach* `deflate-list`.
   (Im PSVD wird die einseitge Rotation wie folgt durchgeführt:
   $\tau^2 = \left(\bar{\mu}_k^2 + \bar{\mu}^2\right)$
   wenn $|\bar{\sigma}_k\mu_k| < \epsilon\tau^2$ *dann*
   $c = \frac{\bar{\mu}_k}{\tau}$ *und* $s = \frac{\mu}{\tau}$
   $\bar{\sigma}_k \leftarrow c\bar{\sigma}_k$
   $\mu \leftarrow \tau$
   $\bar{\mu}_k \leftarrow 0$*)*

In [26] wurde gezeigt, daß bei jeder Drehung in Schritt 2 ein Fehler der Form

$$E_k^{(2)} = \epsilon_k \left(e_{k+1}e_k^T + e_k e_{k+1}^T\right) \tag{39}$$

mit $\epsilon_k = \frac{(\bar{\sigma}_k - \bar{\sigma}_{k+1})\bar{\mu}_k\bar{\mu}_{k+1}}{\bar{\mu}_k^2 + \bar{\mu}_{k+1}^2}$ auftritt. In Schritt 3 tritt ein Fehler der Form

$$E_k^{(3)} = \epsilon_k \left(e_n e_k^T\right) \tag{40}$$

auf. Die Größe des Gesamtfehlers hängt von der Eliminationstoleranz ϵ ab. Eine sinnvolle Wahl von ϵ ist $m * \tilde{\sigma}_{max}$, wobei m die Maschinengenauigkeit und $\tilde{\sigma}_{max}$ das größte Diagonalelement von $\tilde{\Sigma}$ in der noch nicht bearbeiteten Matrix $\bar{M}$ ist. Da die sorgfältige Bestimmung kleiner Singulärwerte wichtig ist, kann die Toleranz bei jedem Eliminationsschritt entsprechend variiert werden.

5.2 SVD ohne erweiterte Rechengenauigkeit

In [20] wird nahezu derselbe Dividing-Schritt zur Bestimmung der Singulärwerte wie in [26] verwendet. Der Ansatz zur Berechnung der Singulärvektoren ist jedoch völlig anders: Gu und Eisenstat stellen eine stabile und effiziente Methode vor, die bei der Berechnung der Singulärvektroren auf erweiterte Rechengenauigkeit verzichtet, d. h. für die Berechnung steht die gleiche Anzahl Bits zur Verfügung wie für die Zahldarstellung.

5.2.1 Aufteilung der Matrix

Sei $B \in I\!R^{(n+1)\times n}$ eine untere Bidiagonalmatrix. B wird wie folgt in zwei Teilprobleme gespalten:

$$B = \begin{pmatrix} B_1 & \alpha_k e_k & 0 \\ 0 & \beta_k e_1 & B_2 \end{pmatrix} \tag{41}$$

mit $1 < k < n$, $B_1 \in I\!R^{k\times(k-1)}$ und $B_2 \in I\!R^{(n-k+1)\times(n-k)}$ sind untere Bidiagonalmatrizen und e_j der j-te Einheitsvektor der entsprechenden Dimension. Gewöhnlich gilt $k = \lfloor \frac{n}{2} \rfloor$. Sei

$$B_i = (Q_i \; q_i) \begin{pmatrix} D_i \\ 0 \end{pmatrix} W_i^T \tag{42}$$

die Singulärwertzerlegung von B_i. Sei l_1^T die letzte Zeile von Q_1 und λ_1 die letzte Komponente von q_1, sei f_2^T die erste Zeile von Q_2 und φ_2 die erste Komponente von q_2. Eingesetzt in (41) erhält man

$$B = \begin{pmatrix} q_1 & Q_1 & 0 & 0 \\ 0 & 0 & Q_2 & q_2 \end{pmatrix} \begin{pmatrix} \alpha_k\lambda_1 & 0 & 0 \\ \alpha_k l_1 & D_1 & 0 \\ \beta_k f_2 & 0 & D_2 \\ \beta_k\varphi_2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & W_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & W_2 \end{pmatrix}^T . \tag{43}$$

In der mittleren Matrix gibt es in der ersten und letzten Zeile nur ein Nicht-Null-Element. Um $\beta_k\varphi_2$ zu eliminieren wird eine Givens-Rotation angewendet:

$$B = \left(\begin{pmatrix} c_0 q_1 & Q_1 & 0 \\ s_0 q_2 & 0 & Q_2 \end{pmatrix} \begin{pmatrix} -s_0 q_1 \\ c_0 q_2 \end{pmatrix} \right) \begin{pmatrix} r_0 & 0 & 0 \\ \alpha_k l_1 & D_1 & 0 \\ \beta_k f_2 & 0 & D_2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & W_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & W_2 \end{pmatrix}^T \tag{44}$$

$$\equiv (Q \; q) \begin{pmatrix} M \\ 0 \end{pmatrix} W^T,$$

wobei $r_0 = \sqrt{(\alpha_k\lambda_1)^2 + (\beta_k\varphi_2)^2}$, $c_0 = \frac{\alpha_k\lambda_1}{r_0}$ und $s_0 = \frac{\beta_k\varphi_2}{r_0}$. Somit wurde B durch Orthogonaltransformationen (Q, q) und W zu $\begin{pmatrix} M \\ 0 \end{pmatrix}$. M hat nur in der ersten Spalte und in der Diagonalen Nicht-Null-Elemente.

5.2.2 Singulärwertzerlegung von M

Sei

$$M = \begin{pmatrix} z_1 & & & \\ z_2 & d_2 & & \\ \vdots & & \ddots & \\ z_n & & & d_n \end{pmatrix} \tag{45}$$

wobei $D = diag(d_1, d_2, \ldots, d_n)$ mit $0 \equiv d_1 \leq d_2 \leq \cdots \leq d_n$ und $z = (z_1, \ldots, z_n)^T$. Wir nehmen an, daß

$$d_{j+1} - d_j \geq \tau \|M\|_2 \quad \text{und} \quad |z_j| \geq \|M\|_2, \tag{46}$$

wobei τ ein kleines Vielfaches von ϵ ist, das in Abschnitt 5.2.5 spezifiziert wird. Jede Matrix dieser Form kann so umgeformt werden, so daß diese Bedingungen erfüllt sind. Das nachfolgende Lemma ([20] [26]) charakterisiert die Singulärwerte und die Singulärvektoren der Matix M:

Lemma 5.1 *Sei $U\Omega V^T$ die Singulärwertzerlegung der Matrix M mit $U = (u_1, \ldots, u_n)$, $\Omega = (\omega_1, \ldots, \omega_n)$ und $V = (v_1, \ldots, v_n)$. Dann gilt für die Singulärwerte $\{w_i\}_{i=1}^n$ die Ungleichung $0 = d_1 < \omega_1 < d_2 < \cdots < d_n < \omega_n < d_n + \|z\|_2$ und die Gleichung $f(\omega) = 1 + \sum_{k=1}^{n} \frac{z_k^2}{d_k^2 - \omega^2} = 0$. Die entsprechenden Singulärvektoren erhält man durch*

$$u_i = \frac{1}{\sqrt{\sum_{k=1}^{n} \frac{z_k^2}{d_k^2 - \omega_i^2}}} \left(\frac{z_1}{d_1^2 - \omega_i^2}, \cdots, \frac{z_n}{d_n^2 - \omega_i^2} \right)^T \tag{47}$$

$$v_i = \frac{1}{\sqrt{1 + \sum_{k=2}^{n} \frac{(d_k z_k)^2}{(d_k^2 - \omega_i^2)^2}}} \left(-1, \frac{d_2 z_2}{d_2^2 - \omega_i^2}, \cdots, \frac{d_n z_n}{d_n^2 - \omega_i^2} \right)^T . \tag{48}$$

Lemma 5.2 *Gegeben sei eine Diagonalmatrix $D = diag(d_1, \ldots, d_n)$ und eine Menge $\{\hat{\omega}_i\}_{i=1}^n$ so daß*

$$0 = d_1 < \hat{\omega}_1 < d_2 < \ldots < d_n < \hat{\omega}_n \tag{49}$$

gilt. Dann existiert eine Matrix

$$\hat{M} = \begin{pmatrix} \hat{z}_1 & & & \\ \hat{z}_2 & d_2 & & \\ \vdots & & \ddots & \\ \hat{z}_n & & & d_n \end{pmatrix} \tag{50}$$

deren Singulärwerte $\{\hat{\omega}_i\}_{i=1}^n$ sind. Der Vektor $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_n)^T$ wird durch die Gleichung

$$|\hat{z}_i| = \sqrt{\left(\hat{\omega}_n^2 - d_i^2\right) \left(\prod_{k=1}^{i-1} \frac{\hat{\omega}_k^2 - d_i^2}{d_k^2 - d_i^2} \right) \left(\prod_{k=i}^{n-1} \frac{\hat{\omega}_k^2 - d_i^2}{d_{k+1}^2 - d_i^2} \right)} \tag{51}$$

bestimmt, wobei das Vorzeichen von $\hat{z}_i$ willkürlich gewählt werden kann.

Beweis: Wenn $\hat{M}$ und somit $\hat{z}$ existieren, dann gilt

$$det\left(D^2 + \hat{z}\hat{z}^T - \omega^2 I\right) = det\left(\hat{M}\hat{M}^T \omega^2 I\right) = \prod_{k=1}^{n} \left(\hat{\omega}_k^2 - \omega^2\right). \tag{52}$$

Andererseits gilt auch

$$
\begin{aligned}
(53) \qquad det\left(D^2 + \hat{z}\hat{z}^T - \omega^2 I\right) &= det\left(I + \left(D^2 - \omega^2 I\right)^{-1} \hat{z}\hat{z}^T\right)\ det\left(D^2 - \omega^2 I\right) \\
&= \left(1 - \sum_{k=1}^{n} \frac{\hat{z}_k^2}{d_k^2 - \omega^2}\right) \prod_{k=1}^{n} \left(d_k^2 - \omega^2\right).
\end{aligned}
$$

Aus beiden Gleichungen folgt nun

$$
(54) \qquad \prod_{k=1}^{n} \left(\hat{\omega}_k^2 - \omega^2\right) = \left(1 - \sum_{k=1}^{n} \frac{\hat{z}_k^2}{d_k^2 - \omega^2}\right) \prod_{k=1}^{n} \left(d_k^2 - \omega^2\right).
$$

Setzt man $\omega = d_i$ erhält man

$$
(55) \qquad \hat{z}_i^2 = \frac{\prod_{k=1}^{n} \left(\hat{\omega}_k^2 - d_i^2\right)}{\prod_{\substack{k=1 \\ k \neq i}}^{n} \left(d_k^2 - d_i^2\right)}.
$$

Wegen Eigenschaft (49) ist $\hat{z}_i^2$ ein positiver Wert, zieht man davon die Wurzel, erhält man (51). Wenn $\hat{z}$ wie in (51) gegeben ist, dann erhält man durch die Umkehrung die Singulärwerte $\{\hat{\omega}_i\}_{i=1}^n$ von $\hat{M}$. □

5.2.3 Berechnung der Singulärvektoren

Wenn ω_i gegeben ist, dann kann die Differenz $d_k^2 - \omega_i^2$ aus (47) und (48) mit hoher Genauigkeit als $(d_k - \omega_i)(d_k + \omega_i)$ berechnet werden.

Produkte, Summen und Differenzen einzelner Zahlen lassen sich aufgrund der Computer-Zahl-Darstellung nur bis zu einer bestimmten Genauigkeit berechnen. In der praktischen Anwendung können daher nur *Näherungen* $\hat{\omega}_i$ der Singulärwerte ω_i berechnet werden. Probleme treten auf, wenn wir die Gleichungen (47) und (48) zur Approximation der Singulärvektoren u_i und v_i verwenden. Die Approximationen $\hat{u}_i$ und $\hat{v}_i$ sind in der Regel nicht orthogonal [20].

Lemma 5.2 erlaubt uns, das Problem zu umgehen. Nachdem man alle approximierten Singulärwerte $\{\hat{\omega}_i\}_{i=1}^n$ von M berechnet hat, bildet man eine neue Matrix $\hat{M}$, deren *exakte* Singulärwerte $\{\hat{\omega}_i\}_{i=1}^n$ sind und berechnet danach mit Hilfe von Lemma 5.1 die Singulärvektoren von $\hat{M}$. Jede Differenz

$$
(56) \qquad \hat{\omega}_k^2 - d_i^2 = (\hat{\omega}_k - d_i)(\hat{\omega}_k - d_i) \text{ und } d_k^2 - d_i^2 = (d_k - d_i)(d_k - d_i)
$$

kann mit relativ hoher Genauigkeit bestimmt werden. Die Vorzeichen von $\hat{z}_i$ können als Vorzeichen von z_i verwendent werden. Somit kann $\hat{z}_i$ mit hoher Genauigkeit berechnet werden. Setzt man die *exakten* Singulärwerte $\{\hat{\omega}_i\}_{i=1}^n$ und die berechneten $\hat{z}_i$ in die Gleichungen (47) und (48) ein, dann kann jeder Singulärvektor von $\hat{M}$ komponentenweise mit hoher Genauigkeit berechnet werden. Nachdem man alle Singulärvektoren berechnet hat, folgt, daß die damit erhaltenen Singulärvektor-Matrizen von $\hat{M}$ numerisch orthogonal sind.

Für die Existenz von $\hat{M}$ müssen $\{\hat{\omega}_i\}_{i=1}^n$ die Bedingung (49) aus Lemma 5.2 erfüllen. (Da die exakten Singulärwerte von M dieser Bedingung wegen Lemma 5.1 ebenfalls genügen, ist dies keine Einschränkung.)

Wir können die SVD von $\hat{M}$ als Approximation der SVD von M annehmen. Wegen $\|M - \hat{M}\|_2 = \|z - \hat{z}\|_2$ ist eine derartige Substitution stabil, solange $\hat{z}$ nahe bei z liegt.

5.2.4 Berechnung der Singulärwerte

Damit $\hat{z}$ nahe bei z liegt, müssen die approximierten Werte $\{\hat{\omega}_i\}_{i=1}^n$ für die Singulärwerte genügend genau sein. Das Verfahren lehnt sich an die Eigenwertbestimmung in [22] an und unterscheidet drei Fälle. Mit Gleichung (16) wird das Intervall, indem ein Singulärwert liegt bestimmt. Wenn $f\left(\frac{d_i+d_{i+1}}{2}\right) > 0$ ist, dann ist $\omega_i \in \left(d_i, \frac{d_i+d_{i+1}}{2}\right)$, andernfalls $\omega_i \in \left[\frac{d_i+d_{i+1}}{2}, d_{i+1}\right)$. Für $1 \le i \le n-1$ sei $\omega_i \in \left(d_i, \frac{d_i+d_{i+1}}{2}\right)$, sei $\delta_j = d_j - d_i$ und seien

$$\psi_i(\mu) \equiv \sum_{j=1}^{i} \frac{z_j^2}{(\delta_j - \mu)(d_j + d_i + \mu)} \quad \text{und} \quad \phi_i(\mu) \equiv \sum_{j=i+1}^{n} \frac{z_j^2}{(\delta_j - \mu)(d_j + d_i + \mu)}. \tag{57}$$

Für $\omega = d_i + \mu$ sucht man die Nullstelle $\mu_i = \omega_i - d_i \in \left(0, \frac{\delta_{i+1}}{2}\right)$ der umformulierten Gleichung

$$g_i(\mu) \equiv f(\mu + d_i) = 1 + \psi(\mu) + \phi(\mu) = 0. \tag{58}$$

Jeder Bruch $\frac{z_j^2}{(\delta_j-\mu)(d_j+d_i+\mu)}$ in $g_i(\mu)$ kann für jedes $\mu \in \left(0, \frac{\delta_{i+1}}{2}\right)$ mit relativ hoher Genauigkeit berechnet werden.
Entweder ist $\delta_i - \mu$ ein negativer oder ein positiver Faktor. Da $\psi(\mu)$ und $\phi(\mu)$ Summen von Termen mit dem gleichen Vorzeichen sind, kann ein Fehler bei der Berechnung von $g_i(\mu)$ beschränkt werden durch

$$\eta n\left(1 + |\psi_i(\mu)| + |\phi_i(\mu)|\right), \tag{59}$$

wobei η ein kleines Mehrfaches von ϵ ist, das von n und μ abhängig ist.

Sei nun $\omega_i \in \left[\frac{d_i+d_{i+1}}{2}, d_{i+1}\right)$ für $1 \le i \le n-1$. Sei $\delta_i = d_j - d_{i+1}$ und seien

$$\psi_i(\mu) \equiv \sum_{j=1}^{i} \frac{z_j^2}{(\delta_j - \mu)(d_j + d_{i+1} + \mu)} \quad \text{und} \quad \phi_i(\mu) \equiv \sum_{j=i+1}^{n} \frac{z_j^2}{(\delta_j - \mu)(d_j + d_{i+1} + \mu)}. \tag{60}$$

Für $\omega = d_{i+1} + \mu$ sucht man die Nullstelle $\mu_i = \omega_i - d_{i+1} \in \left[\frac{\delta_i}{2}, 0\right)$ der Gleichung

$$g_i(\mu) \equiv f(\mu + d_{i+1}) = 1 + \psi_i(\mu) + \phi_i(\mu) = 0. \tag{61}$$

Wie oben haben auch hier alle Faktoren das gleiche Vorzeichen und analog kann ein Fehler bei der Berechnung von $g_i(\mu)$ beschränkt werden.

Zum Schluß sei der Fall $i = n$. Sei $\delta_i = d_j - d_n$ und sei

$$\psi_n(\mu) \equiv \sum_{j=1}^{i} \frac{z_j^2}{(\delta_j - \mu)(d_j + d_n + \mu)} \quad \text{und} \quad \phi_n(\mu) \equiv 0 \tag{62}$$

Für $\omega = d_n + \mu$ suchen wir die Nullstelle $\mu_n = \omega_n - d_n \in (0, \|z\|_2)$ der Gleichung

$$g_n(\mu) \equiv f(\mu + d_n) = 1 + \psi_n(\mu) + \phi_n(\mu) = 0. \tag{63}$$

Für die Beendigung der Nullstellenberechnung wird

$$|g_i(\mu)| \leq \eta n \left(1 + |\psi_i(\mu)| + |\phi_i(\mu)|\right). \tag{64}$$

vorgeschlagen, wobei die rechte Seite eine obere Schranke des Rundungsfehlers bei der Berechnung von $g_i(\mu)$ ist. Für jedes i gibt es mindestens eine Floatingpoint-Zahl, das der numerischen Abbruchbedingung genügt, nämlich $fl(\mu_i)$.

5.2.5 Numerische Stabilität

Das nachfolgende Lemma zeigt, daß $\hat{z}$ nahe bei z liegt.

Lemma 5.3 *Für $\tau = 2\eta n^2$ aus (46) und jedes μ_i erfüllt die Bedingung in (64), dann gilt*

$$|\hat{z}_i - z_i| \leq 4\eta n^2 \|z\|_2 . \tag{65}$$

Der Beweis ist nahezu identisch mit dem für die Eigenwertberechnung aus [22].

6 SVD durch Kombination von parallelen und sequentiellen Algorithmen

Im nachfolgenden wird ein Algorithmus beschrieben, der die SVD durch Kombination von parallel und - bezogen auf das Gesamtproblem - sequentiell ausführbaren Schritten löst. Die Entwicklung wuchs aus der Notwendigkeit, auf einem Echtzeit-System die Lösung der SVD in möglichst kurzer Zeit zu erhalten. Sei $A \in \mathbb{R}^{m \times n}$ mit $m \gg n$.

6.1 Housholder-Transformation

Zuerst wird eine Housholder-Transformation durchgeführt, so daß

$$A = QR. \tag{66}$$

$Q \in \mathbb{R}^{m \times n}$ hat orthogonale Spalten und $R \in \mathbb{R}^{n \times n}$ ist eine obere Dreicksmatrix. Die Matrix R ist singulär, wenn der Rang r der Matrix A kleiner als n ist, $(r < n)$. Dabei wird eine von Bishof und van Loan beschriebene Methode [4] verwendet, die auf der Beobachtung beruht, daß das Produkt $H_k = P_k \cdots P_2 P_1$ der k Householder-Transformationen in der Form

$$H_k = I - V_k U_k^T \tag{67}$$

geschrieben werden kann. Dabei sind $U_k, V_k \in \mathbb{R}^{m \times k}$, mit $U_k = (u_1, u_2, \ldots, u_k)$ und $V_k = (P_k V_{k-1}, u_k)$. In [4] wurde gezeigt, daß dieses Schema dieselben numerischen Eigenschaften wie die klassische Housholder-Faktorisierung besitzt. Auf einer Alliant FX/8 konnte gegenüber dem klassischen Schema für eine Matrix $A \in \mathbb{R}^{1000 \times 100}$ eine Geschwindigkeitssteigerung um den Faktor 9 erreicht werden [1].

6.2 Einseitiges Jacobi-Verfahren

Das einseitige Jacobi-Verfahren wird auf die Matrix $R \in \mathbb{R}^{n \times n}$ aus (66) angewendet. Man bestimmt eine orthogonale Matrix $V = \left[\tilde{V}, \tilde{W}\right]$, wobei $\tilde{V} \in \mathbb{R}^{n \times r}$ ein Produkt von Ebenendrehungen der Art ist, so daß

$$R\tilde{V} = \tilde{Q} = (\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_r) \tag{68}$$

mit $\tilde{q}_i^T \tilde{q}_j = \sigma_i^2 \delta_{ij}$; dabei ist δ_{ij} das Kronecker-Symbol. Die Spalten von $\tilde{Q}$ sind orthogonal. Anschließend kann $\tilde{Q}$ wie folgt geschrieben werden:

$$\tilde{Q} = \tilde{U}\tilde{\Sigma} \tag{69}$$

mit $\tilde{U}^T \tilde{U} = I_r$ und $\tilde{\Sigma} = diag\,(\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_r)$. Daraus ergibt sich die Gleichung

$$R = \tilde{U}\tilde{\Sigma}\tilde{V}^T. \tag{70}$$

Zusammen mit (66) erhält man

$$A = QR = Q\tilde{U}\tilde{\Sigma}\tilde{V}^T = U\tilde{\Sigma}\tilde{V}^T. \tag{71}$$

Die Matrix $\tilde{V}$ wird über die (i,j)-te Ebenendrehung hergeleitet, so daß

$$r_i^T r_j = 0, \ \|r_i\|_2 > \|r_j\|_2 \tag{72}$$

gilt, wobei r_i die i-te Spalte von $R \equiv R\tilde{V}$ ist. Weiter wird vorausgesetzt, daß die Norm der Spalten von Q von links nach rechts ansteigt. Daraus ergibt sich σ_i in monoton absteigender Reihenfolge. Bei der Orthogonalisierung von R ergeben sich mehrere Parallelisierungsmöglichkeiten [1]. Eine weitere ausführliche Orthogonalisierungsmöglichkeit wird in [2] und [3] behandelt. Eine Spieglung besteht aus $\frac{n(n-1)}{2}$ Ebenendrehungen und man erhält für jede einzelne Spiegelung

$$V_{s_i} = V_1 V_2 \cdots V_n. \tag{73}$$

Wenn t die Anzahl der notwendigen Spiegelungen für Konvergenz ist, ergibt sich daraus

$$V = V_{s_1} V_{s_2} \cdots V_{s_t}. \tag{74}$$

Es handelt sich dabei um einen iterativen Prozess, da bei einer darunterliegenden Transformation die Orthogonalität einiger Spalten verloren gehen kann.

6.3 Numerische Behandlung

Man kann das einseitige Jacobi-Verfahren auf der oberen Dreiecksmatrix $R \equiv R_1$ definieren durch

$$R_{k+1} = R_k V_k, \ k = 1, 2, \ldots, \tag{75}$$

wobei

$$R_k = D_k + E_k + E_k^T, \tag{76}$$

und

$$S_k = R_k^T R_k = \tilde{D}_k + \tilde{E}_k + \tilde{E}_k^T \tag{77}$$

wobei D_k, $\tilde{D}_k$ Diagonalmatrizen und E_k, $\tilde{E}_k$ obere Dreiecksmatrizen sind. Aus Gleichung (70) folgt, daß die Wurzeln der Eigenwerte von S_k die Singulärwerte von A sind. S_k konvergiert über eine blockweise Diagonalform, wobei jede Hauptuntermatrix mit Diagonalelementen blockweise die Singulärwerte von A enhält. Um S_k in dieser Form zu erhalten, müssen die Diagonalelemente von S_k dem benachbarten Auftreten der gruppierten Singulärwerte entlang der Diagonalen entsprechen. Das wird durch die Forderung der monoton absteigenden Norm in R_k erreicht (72). Nach einer bestimmten Anzahl c kritischer Spiegelungen erhalten wir

$$S_c = \begin{pmatrix} T_1 & & & \\ & T_2 & & \\ & & \ddots & \\ & & & T_p \end{pmatrix}, \tag{78}$$

so daß die Eigenpaare jedes T_i, $i = (1, 2, \ldots, p)$, wobei p die Anzahl der Blöcke ist, entweder durch das einseitige oder beidseitige Jacobi-Verfahren parallel berechnet werden können.

Ein solches gemischtes Verfahren, bestehend aus einer Initialisierungsphase von mehreren einseitigen Jacobi-Iterationen, gefolgt vom beidseitigen Jacobi-Verfahren auf den Unterproblemen, vereinigt die optimale Parallelisierbarkeit des einseitigen Jacobi-Verfahrens und der schnellen Konvergenz des zweiseitigen Jacobi-Verfahrens für blockweise Spektren. Die Schwierigkeit bei der Implementierung einer solchen Methode liegt in der Bestimmung der kritischen Anzahl Spiegelungen c. Ein derartiges Mischverfahren für die SVD ist sehr praktisch für die Anwendung auf Multiprozessorsystemen wie beispielsweise der CRAY X-MP/416, CRAY 2 und CEDAR.

6.4 Gershgorin Kreise

Um die Anzahl der kritischen Spiegelungen c zu bestimmen, greift man auf die Kreisbestimmung von Gershgorin zurück. Der *Satz von Gershgorin* [28] liefert einfach zu berechnende Schranken für die Eigenwerte und Singulärwerte einer Matrix.

Angenommen der Fall $c = 2$. Der i-te Gershgorinkreis (nach der Zeilennorm) im Zusammenhang mit S_k ist definiert durch

$$g_i^k = \sum_{j=1(\neq i)}^{n} \left| s_{ij}^k \right|, \; i = 1, 2, \ldots, n, \tag{79}$$

und der partielle Gershgorinradius

$$\hat{g}_i^k = \sum_{j=i+1}^{n} \left| s_{ij}^k \right|, \; i = 1, 2, \ldots, n-1. \tag{80}$$

Man wählt ein vorgebbares ϵ und bestimmt die Menge

$$\Im_k = \left\{ i \mid \left| \hat{g}_i^k - \hat{g}_{i-1}^k \right| > \epsilon \right\}. \tag{81}$$

Für jedes $i \in \Im_k$ bestimmt man g_i^k und g_{i-1}^k. Wenn

$$g_i^k + g_{i-1}^k < \left| s_{ii}^k - s_{i-1,\, i-1}^k \right| \tag{82}$$

werden zwei Blöcke im Spektrum von A durch s_{jj}^k, $j = [1, i-1]$, und s_{ll}^k, $l = [i, n]$ approximiert. Die Menge $\Im_k$ aus (81) bildet eine notwendige Bedingung für die Trennung zweier Gershgorinkreise, und Ungleichung (82) garantiert, daß die beiden Blöcke in der Reihenfolge der Singulärwerte getrennt werden können.

7 Ergebnisse und Performance

Algorithmen sind in der Regel so flexibel, daß sie Probleme unterschiedlicher Größe verarbeiten können. Für die Anwendung ist daher die *Verhältnisänderung* der durchzuführenden Instruktionen bezüglich der Problemgröße wichtig. Derartige Analysen werden meistens vor der Implementierung eines Algorithmus gemacht.

Tabelle 3: Die häufigsten Funktionen zur Messung der Effizienz in Abhängigkeit einer Problemgröße n sind:

logarithmisches Wachstum:	$\log n$	besser
lineares Wachstum:	n	
$n \log n$-Wachstum:	$n \log n$	
quadratisches, kubisches, ... Wachstum:	$n^2,\ n^3, \ldots$	schlechter
exponentielles Wachstum:	m^n	unbrauchbar für $m \gg 1$

Beim codierten Algorithmus sind die aktuellen Laufzeiten auf den zur Verfügung stehenden Rechnern interessant. Diese werden über die Abfrage der Systemuhr ermittelt.

7.1 Ein Ergebnis in der $\mathcal{O}$-Notation

Für den in Abschnitt 5.2 vorgestellten Algorithmus zur SVD werden für die Aufteilung der Matrix höchstens $\mathcal{O}(\log n)$ Rekursionsaufrufe benötigt. Für die Berechnung der Singulärwerte und -vektoren benötigt der Algorithmus $\mathcal{O}(n^3)$, für die Singulärwerte allein $\mathcal{O}(n^2)$ Schritte. Unter Verwendung der *Fast Multipole Method* [19] [21] kann die Zeit auf $\mathcal{O}(\log n^2)$ bzw. $\mathcal{O}(n \log n)$ verbessert werden. Verglichen mit den entsprechenden Zeiten — $\mathcal{O}(n^3)$ bzw. $\mathcal{O}(n^2)$ — des QR-Algorithmus sind diese besser.

7.2 Rechnerbasierte Ergebnisse

Der von Jessup und Sorensen vorgeschlagene Algorithmus (siehe Abschnitt 5.1) wurde auf einer Alliant FX/8 implementiert und in der Laufzeit mit den Algorithmen B/III [25] und DSVDC [11] [24] verglichen. Die nachfolgenden Tabellen geben die Rechenergebnisse an.

Das unerwartete Ergebnis in der zweiten Spalte von Tabelle 5 ist noch nicht ganz geklärt. Vermutlich liegt es in zwei Ursachen der Implementierung begründet. Zum einen wird jeder Eliminationsschritt innerhalb des SVD-Update-Schrittes seriell bearbeitet und muß vollständig abgeschlossen sein, bevor dynamische Allokation für die Nullstellensuche statt finden kann. Das beschränkt den Speedup. Zum anderen besteht auf der Alliant FX/8 ein potentieller Cache-Konflikt für explizite Parallel-Verarbeitung.

Die hybride SVD wurde auf einer Alliant FX/8 (128 KB Cache-Speicher), einer Alliant FX/80 (512 KB Cache-Speicher), einer CRAY X-MP/416 und einer CRAY 2 implementiert. Von den Testmatrizen wurden alle Singulärwerte in 64-Bit-Arithmetik auf 14 Dezimalstellen genau berechnet. Tabelle 6 zeigt den Vergleich mit dem klassischen SVD-Algorithmus aus [11] und [35]. Dieser wurde auf den beiden CRAY-Rechnern nur auf einem Prozessor implementiert.

Tabelle 4: Residuen-Maximum und Orthogonalität der SVD berechnet durch B/III, PSVD, und DSVDC für fünf Testmatrizen.

Matrix order	Method	Maximum residual $\mathcal{R}$	Maximum orthogonality (left) $\mathcal{O}_Y$	Maximum orthogonality (right) $\mathcal{O}_X$
$n = 32$	PSVD	1.66d-14	7.65d-15	7.54d-15
	DSVDC	1.79d-15	1.13d-14	1.13d-14
	B/III	9.77d-16	9.76d-15	1.02d-14
$n = 100$	PSVD	9.39d-14	2.56d-14	2.37d-14
	DSVDC	4.22d-15	2.68d-14	2.86d-14
	B/III	2.38d-15	1.90d-14	1.87d-14
$n = 200$	PSVD	4.09d-15	1.13d-14	1.64-14
	DSVDC	7.60d-15	8.13d-14	8.14d-14
	B/III	5.99d-15	5.42d-13	5.53d-14

Tabelle 5: Speedup für PSVD

Matrix-größe	Zeitverhältnis DSVDC (1 Prozessor) / PSVD (8 Prozessoren)	Zeitverhältnis PSVD (1 Prozessor) / PSVD (8 Prozessoren)
100	13.4	5.0
150	14.8	5.3
300	15.7	4.7
350	13.9	4.6

Für die hybride SVD wurden sowohl Microtasking [8] (Micro) als auch Macrotasking [8] (Macro) auf den CRAY-Rechnern verwendet. Aus Tabelle 6 geht hervor, daß für $n = 128$ die hybride SVD gegenüber der seriellen SVD um das zwei- (CRAY 2 Macrotasking) bis fast vierfache (CRAY X-MP/416 Microtasking) schneller sein kann.

Die hybride SVD ist jedoch auf den beiden Alliant-Rechnern nicht konkurrenzfähig mit dem seriellen Algorithmus. Die verbesserten Rechenzeiten der Hybrid-SVD auf der Alliant FX/80 gegenüber der Alliant FX/8 sind auf den größeren Cache-Speicher und den höheren Prozessortakt zurückzuführen.

Als Ergänzung seien die in Tabelle 7 aufgeführten rechnerspezifischen Angaben anzusehen, die [13] entnommen wurden. Dabei handelt es sich um theoretisch mögliche Spitzenleistungen, die in der Praxis gewöhnlich kleiner sind [10] [13] [29]. Für unterschiedliche Rechnerstrukturen gibt es bisher noch keine vergleichenden Leistungsaussagen. Für einen effizienten Algorithmus ist neben der Architekturform auch die Qualität des Compilers entscheidend (vgl. Abschnitt 3.2) [15] [29].

Tabelle 6: Performance von Hybrid-SVD und serieller SVD in Gesamtlaufzeit (Belegungszeit der Prozessoren bzw. wall clock time)

Hybrid-SVD						
n	Alliant		CRAY X-MP/416		CRAY 2	
	FX/8	FX/80	Micro	Macro	Micro	Macro
64	0.50	0.33	0.09	0.16	0.26	0.18
128	3.0	2.0	0.60	0.65	0.85	0.75
serielle SVD						
n	Alliant		CRAY X-MP/416		CRAY 2	
	FX/8	FX/80	1 CPU		1 CPU	
64	0.36	0.22	0.11		0.13	
128	1.4	1.0	2.2		1.3	

Tabelle 7: Rechnerspezifische (theoretische) Spitzenleistungen

Rechner-typ	Peak Performance (MFLOPS)	Zeit in ns für die Gleitpunkt-multiplikation	Anzahl der Ergebnisse	Anteilige Zeit für eine Gleitpunkt-multiplikation
Cray X-MP 4	840	9.5	2	4.75
Alliant FX/8	94	170	8	21.3

Literatur

[1] Berry M., Sameh A., *A Hybrid Scheme for the Singular Value Decomposition on a Multiprocessor*, NATO ASI Series. Vol. F70, Numerical Linear Algebra, Springer-Verlag, (1991), pp.419-428

[2] Berry M., Sameh A., *Multiprocessor Jacobi schemes for dense symetric eigenvalue and singular value decompositions*, University of Illinois at Urbana-Champaign, Center for Supercomputing Research and Development, (1986)

[3] Berry M., Sameh A., *Parallel algorithms for the singular value and dense symmetric eigenvalue problems*, University of Illinois at Urbana-Champaign, Center for Supercomputing Research and Development, (1988)

[4] Bishof C., van Loan C., *The WY representation for products of Housholder Matrices*, TR 85-681 , Cornell University, Department of Computer Science, (1985)

[5] Bräunl T., *Parallele Programmierung*, ISBN 3-528-05142-6, vieweg Braunschweig Wiesbaden, (1993)

[6] Bunch J., Nielsen C., Sorensen D., *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48

[7] Burkhardt S., u. a., *Parallele Rechnersysteme*, ISBN 3-341-01076-9, Verlag Technik GmbH Berlin München, (1993)

[8] *CRAY X-MP Multitasking Programmer's Reference Manual*, Cray Research Inc., Mendota Heights, MN, (1987), SR-0222

[9] Cuppen J., *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195

[10] Dongarra J., *The LINPACK Benchmark: An Explanation Speedup*, Vol. 1, No. 1, (1987)

[11] Dongarra J., Bunch J., Moler C., Stewart G., *LINPACK Users Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, (1979)

[12] Dongarra J., Sorensen D., *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., (1987), pp. s139-s154

[13] Erhard W., *Parallelrechnerstrukturen*, ISBN 3-519-02243-5, B. G. Teubner Stuttgart, (1990)

[14] Flynn M. J., *Very high speed computing systems*, Proc. of the IEEE, Vol. 54, No. 12, pp. 1901-1909

[15] Giloi W. K., *Rechnerarchitektur*, ISBN 3-540-56355-5, Springer-Verlag Berlin Heidelberg New York

[16] Golub G. H., van Loan C. F., *Matrix Computations*, ISBN 0-8018-3739-1, The Johns Hopkins University Press

[17] Gottwald S., Schendel U., *Massiv-parallele numerische Algorithmen nach dem Divide&Conquer-Prinzip*, FU Berlin, FB Mathematik, Preprint No. A-10/94

[18] Gottwald S., *Das Divide&Conquer-Prinzip für massiv-parallele numerische Algorithmen*, ISBN 3-8265-1737-7, Shaker Verlag

[19] Gu M., *Studies in Numerical Linear Algebra*, Ph. D. thesis, Department of Computer Science, Yale University, New Haven, CT, (1993)

[20] Gu M., Eisenstat S. C., *A Divide-and-Conquer Algorithm For The Bidiagonal SVD*, SIAM J. Matrix Anal. Appl., (1995), pp. 79-92

[21] Gu M., Eisenstat S. C., *A fast algorithm for updating the singular value decomposition*, manuscript

[22] Gu M., Eisenstat S. C., *A Stable And Efficient Algorithm For The Rank-One-Modification Of The Symmetric Eigenproblem*, SIAM J. Matrix Anal. Appl., (1994), pp. 1266-1276

[23] Hämmerlin G., Hoffmann K.-H., *Numerische Mathematik*, ISBN 3-540-53539-X, Springer-Verlag Berlin Heidelberg New York

[24] Jessup E. R., *Parallel Solution of the Symmetric Tridiagonal Eigenproblem*, Ph. D. thesis, Dept. of Computer Science, Yale University, New Haven, CT, (1989)

[25] Jessup E. R., Ipsen I., *Improving the accuracy of inverse iteration*, SIAM J. Sci. Statis. Comput., 13, (1992), pp. 550-571

[26] Jessup E. R., Sorensen D. C., *A parallel algorithm for computing the singular value decomposition of a matrix*, SIAM J. Matrix Anal. Appl., (1994), pp. 530-548

[27] Kiełbasiński A., Schwetlick H., *Numerische Lineare Algebra*, ISBN 3-87144-999-7, Harri Deutsch Frankfurt/Main

[28] Schendel U., *Einführung in die Numerische Mathematik Teil II*, FU Berlin, FB Mathematik, Skript, (1988)

[29] Schendel U., *Einführung in die Parallele Numerik*, FU Berlin, FB Mathematik, Skript, (1996)

[30] Schendel U., *Introduction to Numerical Methods for Parallel Computers*, Ellis Horwood Ltd.

[31] Schendel U., *Numerische Algorthmen in der Optimierung* FU Berlin, FB Mathematik, Skript, (1994)

[32] Schendel U., *Sparse-Matrizen*, ISBN 3-486-20621-4, Oldenbourg Verlag München Wien

[33] Schendel U., *Sparse-Matrices*, ISBN 0-7458-0635-X, Ellis Horwood Ltd.

[34] Schwarz H. R., *Numerische Mathematik*, ISBN 3-519-22960-9, B. G. Teubner Stuttgart

[35] Smith B., a. o., *Matrix Eigensystem Routines - EISPACK Guide*, second edition, Springer-Verlag, (1976)