Effizienzsteigerung durch Blockung

Ernst Haunschmid, Christoph Überhuber
Institut für Angewandte und Numerische Mathematik, Technische Universität Wien

Kann eine Aufgabenstellung der Numerischen Datenverarbeitung auf einem gegebenen Computersystem nicht innerhalb eines bestimmten Zeitrahmens gelöst werden und ist man bestrebt, ohne Anschaffung neuer leistungsfähigerer Hardware auszukommen, so muß eine Optimierung des Problemlösungsverfahrens durchgeführt werden. Diese kann auf mehreren Ebenen geschehen:

1. durch Modifikation des zugrundeliegenden mathematischen Modells,
2. durch Änderung der Diskretisierung des mathematischen Modells,
3. durch die Wahl besserer Algorithmen zur Lösung des diskreten Problems,
4. durch eine effizientere Implementierung der gewählten Algorithmen.

In diesem Beitrag wird nur auf die als letzte angesprochene Optimierungsmöglichkeit, die effizientere Programmierung (Implementierung) von Algorithmen der Linearen Algebra, eingegangen. Dies soll aber in keiner Weise eine Wertung (in Bezug auf die leistungssteigernde Wirkung) der verschiedenen Optimierungsebenen zum Ausdruck bringen: Die effizientere Programmierung eines bestehenden Algorithmus gestattet oft nur eine weit geringere Beschleunigung als sie z. B. durch die Entwicklung eines neuen Algorithmus mit stark verringertem Arbeitsaufwand erzielt werden kann.

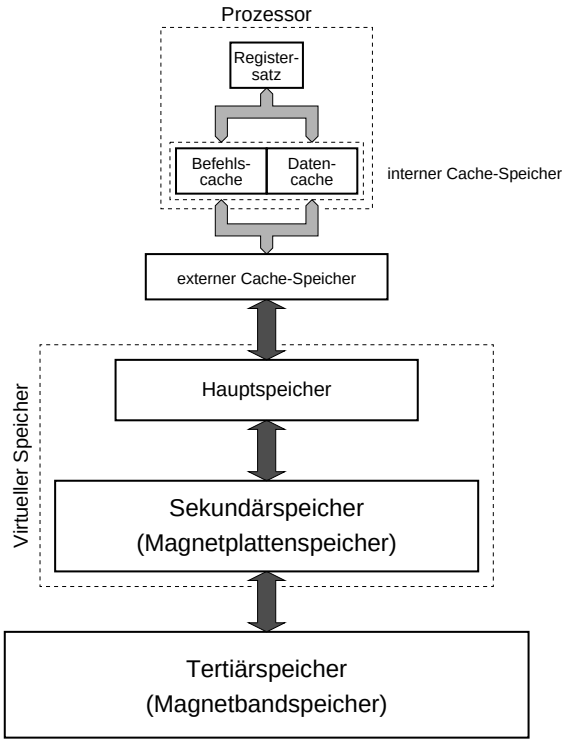

Abb. 1: Speicherhierarchie mit sechs Ebenen

Blocken von Speicherzugriffen

Ein entscheidender Schlüssel zur leistungsorientierten Programmierung liegt darin, einmal in eine höhere (schnellere) Ebene der Speicherhierarchie (siehe Abb. 1) transferierte Daten sooft wie möglich als Operanden zu verwenden, bevor sie – durch neue Daten ersetzt – in die nächsttiefere (langsamere) Ebene ausgelagert werden.

Als wichtiges Hilfsmittel zur Entwicklung von Programmen, die dieses Ziel oft weitgehend erreichen, hat sich das Konzept der *Blockalgorithmen* erwiesen. Unter der *Blokkung* einer $n \times m$ -Matrix A versteht man deren Zerlegung in Untermatrizen A_{ij}, $i = 1,2,...,r$, $j = 1,2,...,s$. Jedes Element der ursprünglichen Matrix fällt in genau einen Block (eine Untermatrix). Die Zerlegung von A nimmt dann die Gestalt

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1s} \\ A_{21} & A_{22} & \cdots & A_{2s} \\ \vdots & \vdots & & \vdots \\ A_{r1} & A_{r2} & \cdots & A_{rs} \end{pmatrix}$$

an. Unter einem *Blockalgorithmus* versteht man einen Algorithmus, der nicht direkt die Elemente von A, sondern die Untermatrizen A_{ij} als Operanden verwendet (Ueberhuber [3]).

Will man einen Matrix-Algorithmus entwickeln, der eine bestimmte Ebene der Speicherhierarchie gut ausnutzt, so formuliert man den Algorithmus als Blockalgorithmus und wählt die Größen der Teilmatrizen einerseits so, daß bei jeder Teiloperation des Blockalgorithmus sämtliche beteiligten Operandenblöcke in der betreffenden Speicherebene Platz finden, und andererseits so, daß die Anzahl der mit den Elementen dieser Blöcke ausgeführten Operationen maximal wird.

Bei der Wahl der Blockgrößen ist zu berücksichtigen, daß die effektive Größe einer Speicherebene wesentlich kleiner sein kann als die physische. So müssen z. B. in den Operandenregistern eines Prozessors nicht nur Programmdaten, sondern auch Zwischenergebnisse von Berechnungen gespeichert werden. Auch bei Cache-Speichern kann die effektive Kapazität (unter Umständen erheblich) kleiner sein als die physische. Beispiele für eine solche Kapazitätsreduktion auf der SGI Power Challenge XL findet man am Ende dieses Artikels.

Zur Erhöhung der effektiven Kapazität von Cache-Speichern können die in einem Algorithmus-Teilschritt verwendeten Teilmatrizen in einen zusammenhängenden Speicherbereich kopiert werden, wodurch eine Diskrepanz zwischen effektiver und physischer Kapazität vermieden wird. Man spricht dann von einer *Blockung mit Kopieren*.

Hierarchische Blockung

Durch einen Blockalgorithmus wird die auszuführende Operation lediglich auf Operationen mit kleineren Matrizen zurückgeführt. Wie diese Teiloperationen selbst festgelegt werden, bleibt offen. So können die Operationen auf den Untermatrizen selbst wieder durch Blockalgorithmen beschrieben werden – man spricht dann von *hierarchischer Blockung* (Ueberhuber [3]).

Will man mehrere Speicherebenen möglichst gut ausnutzen, so kann man auf hierarchische Blockalgorithmen zurückgreifen und mit Hilfe der verschiedenen Blockungsebenen verschiedene Speicherebenen ausnutzen.

Im Programmpaket LAPACK [1] zur Lösung linearer Gleichungssysteme und Eigenwertprobleme gibt es für viele Algorithmen sowohl eine ungeblockte als auch eine geblockte Programmversion. Bei den geblockten Algorithmen wird allerdings nur *eine* Blockungsebene eingesetzt. Dadurch kann es bei Hauptspeicher-Fehlzugriffen (*page faults*) – trotz optimaler Wahl der Blockgröße – zu starken Leistungseinbrüchen kommen.

Durch eine zweite Blockungsebene kann man auch bei größeren Matrizen, die nicht im Hauptspeicher Platz finden, eine deutlich verbesserte Referenzlokalität und damit signifikante Leistungssteigerungen erzielen (siehe Abb. 2).

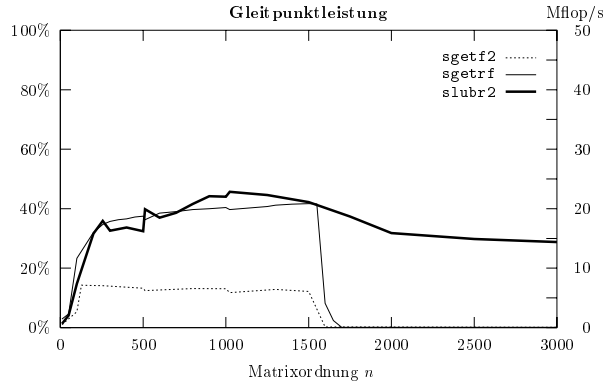

Abb. 2: LU-Faktorisierung auf einer Workstation: Das Programm `LAPACK/sgetf2` ist völlig *ungeblockt*, `LAPACK/sgetrf` ist *einfach geblockt*, `slubr2` ist eine, von den Autoren dieses Beitrages entwickelte, *zweifach hierarchisch geblockte* Modifikation von `LAPACK/sgetrf`.

Cholesky-Faktorisierung

Die Cholesky-Faktorisierung ist die am häufigsten verwendete Methode zur direkten (nicht-iterativen) Lösung linearer Gleichungssysteme mit symmetrischer, positiv definiter Matrix. LAPACK enthält zwei Unterprogramme, die den Cholesky-Algorithmus implementieren:

`dpotrf` liefert auf vielen Computern gute Gleitpunktleistung, läßt aber die Symmetrie der Matrix unberücksichtigt.

`dpptrf` ist für Matrizen in gepackter Speicherung gedacht; es kann daher fast die Hälfte des konventionellen Speicherbedarfs eingespart werden. Der Teil oberhalb (oder unterhalb) der Hauptdiagonale der Matrix wird dabei spaltenweise gepackt in einem eindimensionalen Feld (Vektor) gespeichert. Das Programm `dpptrf` hat den Nachteil sehr schlechter Gleitpunktleistung (siehe Abb. 3). Die Ursache dieser Laufzeit-Ineffizienz liegt darin, daß die gepackte Speicherung die Verwendung von BLAS-3-Programmen unmöglich macht. Diese Unterprogramme für elementare Matrix-Matrix-Operationen sind so programmiert, daß sie die potentielle Leistungsfähigkeit von Computern mit stark gestaffelter Speicherhierarchie in einem hohen Ausmaß nutzen können.

Es war daher das Ziel der Autoren, neue Programme zu entwickeln, die sowohl die Speicher-Effizienz von `dpptrf` als auch die Laufzeit-Effizienz von `dpotrf` besitzen. Entscheidende Hilfsmittel zum Erreichen dieses Ziels war die Einführung einer zweiten Blockungsebene, d. h. hierarchische Blockung (Ueberhuber [3]) zur Sicherstellung hoher Laufzeit-Effizienz und die Verwendung eines modifizierten Speicherschemas, um zufriedenstellende Speicher-Effizienz zu erreichen.

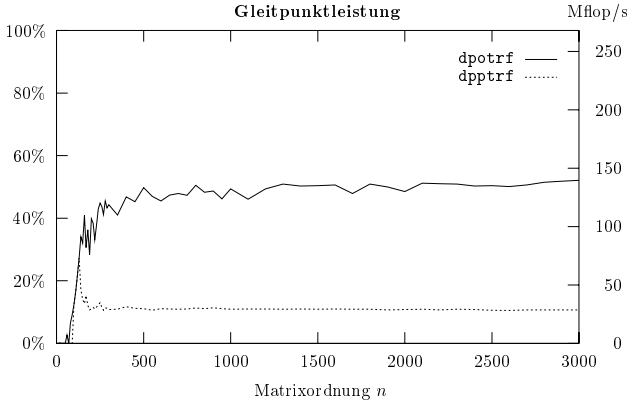

Abb. 3: Gleitpunktleistung der Programme `LAPACK/dpotrf` und `LAPACK/dpptrf` auf einer Workstation der Type IBM RS/6000-390.

Speicherformate

LAPACK unterstützt verschiedene Speicherungsformen für Matrizen. Symmetrische Matrizen können entweder auf konventionelle Art in einem zweidimensionalen Feld oder in gepackter Form in einem eindimensionalen Feld gespeichert werden. Beide Formen haben Nachteile: die konventionelle Speicherung ist ineffizient und die gepackte Speicherung verhindert gute Gleitpunktleistung. Es ist daher zweckmäßig eine Kombination von beiden Formen – die *gepackte Blockspeicherung* – zu verwenden (siehe Abb. 4).

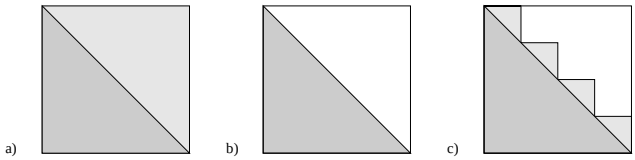

Abb. 4: Speicherungsformen für symmetrische Matrizen: a) konventionelle Speicherung, b) gepackte Speicherung, c) gepackte Blockspeicherung. Die dunkel schattierten Teile der Matrix enthalten unverzichtbare Information. Auf den hell schattierten Teilen ist redundante Information gespeichert.

Der Hauptvorteil der gepackten Blockspeicherung besteht darin, daß sie die Verwendung von BLAS-3-Programmen für Teilblöcke der ursprünglichen Matrix (siehe Abb. 4c) ermöglicht. Auf diese Weise führt dieses Speicherschema zu einer zufriedenstellenden Gleitpunktleistung.

Geblockte Cholesky-Faktorisierung

Die Blockform der LU-Faktorisierung (des Gauß-Algorithmus) und der Cholesky-Faktorisierung können relativ einfach aus den konventionellen, vektorisierbaren Algorithmusformen abgeleitet werden. Es entsteht weder zusätzlicher Speicherbedarf noch werden zusätzliche Gleitpunktoperationen benötigt.

Je nach der Anordnung der drei geschachtelten Schleifen, gibt es sechs verschiedene Algorithmus-Varianten der LU- und der Cholesky-Faktorisierung. Bei einfacher Blockung (wie sie im LAPACK verwendet wird) gibt es daher 36 Varianten und bei zweifacher Blockung 216 Varianten, die sich bezüglich ihrer Gleitpunktleistung erheblich unterscheiden können.

Der Cholesky-Algorithmus zerlegt eine symmetrische und positiv definite $n \times n$-Matrix A in das Produkt einer unteren Dreiecksmatrix L und ihrer Transponierten, d. h. $A = LL^T$ (oder $A = U^T U$, wobei U eine obere Dreiecksmatrix ist). Im folgenden wird nur die Variante mit der unteren Dreiecksmatrix behandelt.

Eine Blockform des Cholesky-Algorithmus kann induktiv gewonnen werden: Beim k-ten Schritt der Berechnung wird angenommen, daß die $n \times n$-Matrizen $A^{(k)}$, $L^{(k)}$, und $L^{(k)T}$ auf folgende Art zerlegt werden können:

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix}.$$

Zunächst wird A_{11} faktorisiert. Sobald L_{11}, die Faktormatrix von A_{11}, vorliegt, kann L_{21} aus der Gleichung

$$L_{21} = A_{21} \left(L_{11}^T\right)^{-1}$$

berechnet werden. Anschließend wird A_{22} aktualisiert:

$$\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T .$$

Die Faktorisierung kann nun rekursiv fortgesetzt werden, indem man die obigen Schritte auf die Teilmatrix $\tilde{A}_{22}$ anwendet.

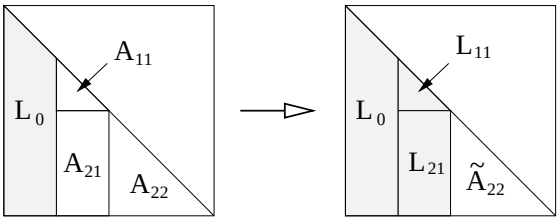

Abb. 5: Zwischenphase des geblockten Cholesky-Algorithmus. Die schattierten Teile wurden bereits berechnet.

Eine Momentaufnahme aus dem Ablauf des geblockten Cholesky-Algorithmus zeigt Abb. 5

Hierarchisch geblockte Cholesky-Faktorisierung

Den hierarchisch geblockten Cholesky-Algorithmus kann man direkt aus der geblockten Cholesky-Faktorisierung ableiten, wenn man die ursprüngliche Blockung weiter verfeinert. Eine Momentaufnahme aus dem Ablauf des hierarchisch geblockten Cholesky-Algorithmus zeigt Abb. 6.

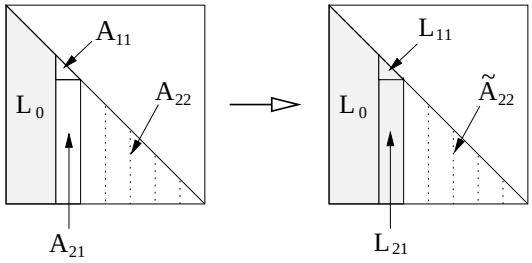

Abb. 6: Zwischenphase des hierarchisch geblockten Cholesky-Algorithmus. Die schattierten Teile wurden bereits berechnet.

Numerische Experimente

Um die Effizienz der neu entwickelten Cholesky-Implementierung zu untersuchen, wurden numerische Experimente auf *einem* Prozessor des SGI Power Challenge XL Hochleistungsservers und auf *einem* Prozessor des neuen NEC SX-4 Applikationsservers Lineare Algebra des EDV-Zentrums der TU Wien durchgeführt.

Resultate auf der SGI Power Challenge XL

Das LAPACK-Programm `dpptrf`, das auf dem gepackten Speicherformat beruht, zeigt hier ein noch schlechteres Leistungsverhalten (schlechtere Wirkungsgrade) als auf anderen Computersystemen (vgl. Abb. 3 und Abb. 7).

Das LAPACK-Programm `dpotrf`, das auf der ineffizienten, vollen Speicherung der symmetrischen Matrix beruht, zeigt ein sehr ungleichmäßiges Leistungsverhalten (siehe Abb. 7). Für manche Problemgrößen sinkt die Gleitpunktleistung auf ein Viertel der sonst beobachtbaren Leistung. Diese Leistungseinbrüche sind auf die ineffiziente Implementierung der BLAS-3 Routine `dgemm` in der Programmbibliothek `CHALLENGEcomplib` Library zurückzuführen. Abb. 8 zeigt dieses Phänomen im Detail. Der Leistungszusammenbruch bei $n = 1536 = 2^{10} + 2^9$ hängt mit der verringerten effektiven Cache-Größe und der damit verbundenen überdurchschnittlich hohen Anzahl von Cache-Fehlzugriffen (*cache misses)* zusammen, die bei dieser Matrix-Ordnung zu beobachten sind.

Das von den Autoren neu entwickelte Programm `dllthbk`, das auf einer hierarchisch geblockten Implementierung des Cholesky-Algorithmus beruht, zeigt keine Leistungseinbrüche. Es ist generell mehr als zehnmal so schnell wie das LAPACK-Programm `dpptrf` und bei manchen Matrixgrößen mehr als doppelt so schnell wie das LAPACK-Programm `dpotrf`.

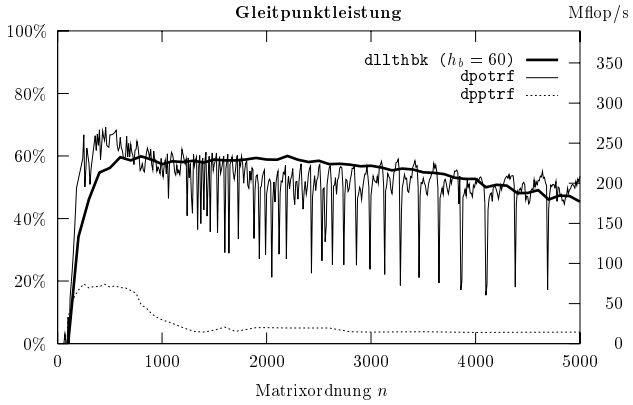

Abb. 7: Leistungsverhalten der Programme `LAPACK/dpotrf` und `LAPACK/dpptrf` im Vergleich mit dem hierarchisch geblockten Cholesky-Algorithmus `dllthbk` (Blockungsfaktor h_b = 60) auf einer SGI Power Challenge XL.

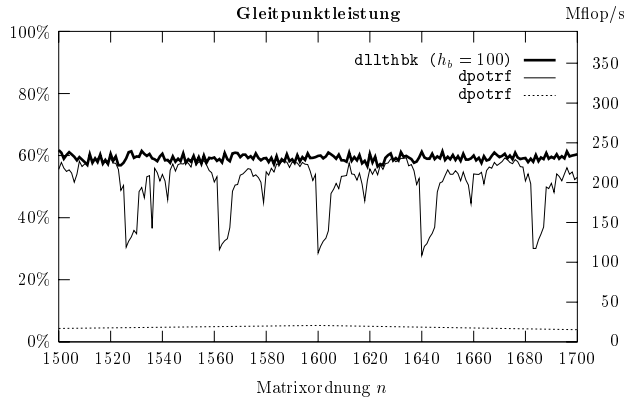

Abb. 8: Leistungsverhalten der Programme `LAPACK/dpotrf` und `LAPACK/dpptrf` im Vergleich mit dem neu entwickelten Programm `dllthbk` (Blockungsfaktor h_b = 100) auf einer SGI Power Challenge XL (Ausschnitt aus Abb. 7).

Resultate auf der NEC SX-4

Das LAPACK-Programm `dpptrf` zeigt auf der NEC SX-4 ein wesentlich besseres Leistungsverhalten als auf anderen Computern (siehe Abb. 9). Die Leistungswerte (Mflops/s) sind bei großen Matrizen mehr als 100 mal so groß wie auf der SGI Power Challenge XL.

Das LAPACK-Program `dpotrf` zeigt auf der NEC SX-4 ein gleichmäßiges und überdurchschnittlich gutes Leistungsverhalten. Die NEC SX-4 liefert mit diesem Programm Leistungswerte, die 8 bis 30 mal so groß sind wie jene der SGI Power Challenge XL.

Das neue Programm `dllthbk` liefert fast durchwegs noch bessere Leistungswerte als `dpotrf`. Dieses erfreuliche Resultat ist besonders bemerkenswert, da auf Grund der Blockung nur Vektorlängen bis 256 auftreten können.

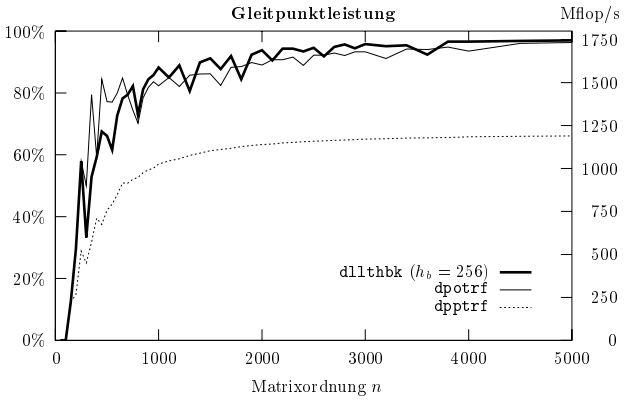

Abb. 9: Leistungsverhalten der Programme `LAPACK/dpotrf` und `LAPACK/dpptrf` im Vergleich mit dem neu entwickelten Programm `dllthbk` (Blockungsfaktor h_b = 256) auf einer NEC SX-4.

Zusammenfassung und Ausblick

In diesem Beitrag wurde gezeigt, daß durch geeignete Implementierung numerischer Algorithmen die empirische Gleitpunktleistung (der empirische Wirkungsgrad) moderner Computersysteme erheblich gesteigert werden kann. Selbst die Leistung von Programmen aus einem ausgezeichneten numerischen Softwarepaket, wie es LAPACK darstellt, kann durch besondere Speicherungstechniken und hierarchische Blockung signifikant gesteigert werden.

Die Autoren werden ihre Arbeit an der Entwicklung effizienter numerischer Algorithmen und deren Implementierung auf modernen Hochleistungsrechnern im Rahmen eines neu gegründeten Spezialforschungsbereiches (SFB „Aurora“) unter Förderung durch den FWF (Fonds zur Förderung der wissenschaftlichen Forschung) fortsetzen.

Literatur

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. McKenney, S. Ostrouchov, D. C. Sorensen: LAPACK *User's Guide*, 2nd ed. SIAM Press, Philadelphia 1995.

[2] K. Dowd: *High Performance Computing*. O'Reilly & Associates, Sebastopol 1993.

[3] C. W. Ueberhuber: *Numerical Computation*. Springer-Verlag, Berlin Heidelberg New York Tokyo 1997.