Performance-Analyse und -Optimierung paralleler Eigenwertlöser auf Blue Gene/P

T. Berg, I.Gutheil

Juelich Supercomputing Centre
Forschungszentrum Juelich GmbH
52425 Juelich
t.berg@fz-juelich.de
i.gutheil@fz-juelich.de

Abstract: Viele Anwendungen aus den Natur- und Ingenieurwissenschaften erfordern die Berechnung der Eigenwerte und -vektoren von symmetrischen, dichtbesetzten Matrizen. Aufgrund der Größe der auftretenden Matrizen ist häufig eine parallele Berechnung der Eigenwerte und -vektoren unumgänglich. In diesem Artikel werden verschiedene Routinen zur parallelen Lösung symmetrischer Eigenwertprobleme aus den Bibliotheken ScaLAPACK und Elemental mit ihren jeweils genutzten Algorithmen vorgestellt. Für die Verteilung der Matrizen auf die Prozessoren verwenden beide Bibliotheken unterschiedliche Ansätze. Während ScaLAPACK eine zweidimensionale blockzyklische Verteilung der Matrizen nutzt, verfolgt die noch in der Entwicklung befindliche Bibliothek Elemental einen elementweisen zweidimensionalen zyklischen Ansatz zur Verteilung der Matrizen. Die Routinen zur Eigenwertberechnung wurden auf einem Blue Gene/P System getestet und bezüglich ihres Laufzeit- und Skalierungsverhaltens verglichen.

1 Mathematische Beschreibung

Es sei eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$, das heißt $A = A^T$, gegeben. Als Eigenwertproblem bezeichnet man das Auffinden einer Lösung des Gleichungssystems

$$A\nu = \lambda\nu, \qquad \|\nu\| = 1, \tag{1}$$

wobei $\lambda \in \mathbb{R}$ als Eigenwert und $\nu \in \mathbb{R}^n$ als zugehöriger Eigenvektor bezeichnet wird. Zusammen nennt man Eigenwert λ und Eigenvektor ν ein Eigenpaar (λ, ν). Eine symmetrische Matrix besitzt nur reelle Eigenwerte und darüber hinaus bilden die zugehörigen Eigenvektoren immer ein Orthogonalsystem. Daher ist das Eigenwertproblem (1) äquivalent zu

$$A = Q_A \Lambda Q_A^T, \tag{2}$$

wobei die Matrix $Q_A \in \mathbb{R}^{n\times n}$ die normierten Eigenvektoren von A als Spalten besitzt und die Diagonalmatrix $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$ die Eigenwerte von A als Diagonaleinträge hat.

Die übliche Vorgehensweise zur numerischen Berechnung der Eigenwerte bzw. Eigenvektoren einer dicht besetzten symmetrischen Matrix A besteht aus drei Schritten:

1. Zuerst wird die dicht besetzte Matrix A auf eine symmetrische Tridiagonalmatrix $Tr \in \mathbb{R}^{n \times n}$ reduziert. Dies geschieht mit Hilfe von n orthogonalen Ähnlichkeitstransformationen, in der Regel Householder-Transformationen, $H_i^T A H_i, i = 1, \ldots, n$. Es gilt dann $H^T A H = Tr$, wobei die Orthogonalmatrix $H = H_1 \cdot \ldots \cdot H_n \in \mathbb{R}^{n \times n}$ das Produkt der einzelnen Transformationsmatrizen ist.

2. Der zweite Schritt besteht aus der Lösung des symmetrischen tridiagonalen Eigenwertproblems $Trz = \lambda z$. Hierfür gibt es einige Algorithmen, auf welche in Abschnitt 2 eingegangen wird.

3. Falls die Eigenvektoren ν der Matrix A auch von Interesse sind, muss für die Eigenvektoren z der Matrix Tr eine Rücktransformation angewendet werden. Dafür gilt $Q_A = HQ_{Tr}$, wobei $Q_{Tr} \in \mathbb{R}^{n \times n}$ die normierten Eigenvektoren der Matrix Tr als Spalten besitzt.

2 Bibliotheken und Algorithmen

2.1 ScaLAPACK

ScaLAPACK (**Sca**lable **L**inear **A**lgebra **Pack**age) ist eine in Fortran geschriebene Bibliothek mit Routinen zur Lösung von Problemen aus der Linearen Algebra, wie Lineare Gleichungssysteme oder Eigenwertprobleme, auf Parallelrechnern mit verteiltem Speicher. Die Algorithmen der Bibliothek basieren auf LAPACK (**L**inear **A**lgebra **Pack**age) [ABB99] und für die Kommunikation untereinander werden die BLACS (**B**asic **L**inear **A**lgebra **C**ommunication **S**ubroutines) [DW97] verwendet.
Aus ScaLAPACK wurden vier verschiedene Algorithmen untersucht. Diese unterscheiden sich dabei jeweils nur in der Lösung des symmetrischen tridiagonalen Eigenwertproblems, also dem unter Abschnitt 1 beschriebenen zweiten Schritt. Der erste und der dritte Schritt benötigen jeweils $\mathcal{O}(n^3)$ Flops. Für den zweiten Schritt braucht der Algorithmus Multiple Relatively Robust Representations, kurz MRRR oder MR^3, nur $\mathcal{O}(n^2)$ Flops, wohingegen die anderen drei vorgestellten Algorithmen zumindest im schlechtesten Fall auch für diesen Schritt $\mathcal{O}(n^3)$ Flops benötigen.

1. PDSYEV berechnet alle Eigenwerte der symmetrischen Tridiagonalmatrix $Tr \in \mathbb{R}^{n \times n}$ mit Hilfe des QR-Algorithmus. Dabei werden automatisch auch alle Eigenvektoren berechnet und diese sind orthogonal.

2. PDSYEVX kann alle oder nur k bestimmte Eigenwerte berechnen, was eine Reduzierung der Flops zur Folge hat. Dabei wird Bisektion zur Berechnung der Eigenwerte und inverse Iteration zur Berechnung der Eigenvektoren genutzt. Bei stark geclusterten Eigenwerten sind die Eigenvektoren nicht automatisch orthogonal, sie müssen nachorthogonalisiert werden.

3. PDSYEVD verwendet für die Lösung des tridiagonalen Eigenwertproblems den Divide-and-Conquer-Algorithmus nach Cuppen [Cup81]. Es werden alle Eigenwerte berechnet, und die Eigenvektoren sind orthogonal.

4. PDSYEVR berechnet die Eigenwerte und Vektoren mit Hilfe des MR^3-Algorithmus [DPV06]. Dabei wird die Matrix für jeden zu berechnenden Eigenwert λ_j in eine Relativ Robuste Repräsentation (RRR) zerlegt $Tr - \sigma I = LDL^T$. Die Matrix D ist dabei eine Diagonalmatrix und die Matrix L eine untere Bidiagonalmatrix mit Einsen auf der Diagonalen. Dann werden die Eigenwerte mittels des dqds-Algorithmus [FP94] bestimmt. Mit dem MR^3-Algorithmus sind die Eigenvektoren orthogonal. Außerdem besteht die Möglichkeit, mit nur $\mathcal{O}(kn)$ Flops k Eigenwerte und -vektoren zu berechnen.

2.2 ELEMENTAL

ELEMENTAL ist eine in C++ geschriebene Bibliothek, welche Unterprogramme zur Lösung vieler Probleme aus der Linearen Algebra bereitstellt. In Elemental ist bisher nur eine Routine, HermitianEig, zur Lösung von Eigenwertproblemen implementiert. Die Lösung des tridiagonalen Eigenwertproblems wird mit Hilfe des MR^3-Algorithmus bestimmt, welcher für die ScaLAPACK Routine DSYEVR schon beschrieben wurde.

2.3 Verteilung der Matrizen auf den Speicher

ScaLAPACK und ELEMENTAL unterscheiden sich wesentlich in der physikalischen Aufteilung der Matrizen auf den verteilten Speicher. ScaLAPACK verwendet eine zweidimensionale blockzyklische Aufteilung [BCC$^+$97]. Hierbei wird eine globale Matrix $A_{glob} \in \mathbb{R}^{m \times n}$ in Teilmatrizen mit einer fest gewählten Blockgröße bl aufgesplittet. Wählt man die Blockgröße zu groß, kann dies zu einer schlechten Lastverteilung auf die Prozessoren führen, da P_0 häufig mehr Elemente zu bearbeiten hat als der letzte Prozessor. Eine zu kleine Blockgröße (für eine optimale Lastverteilung=1), führt einerseits zu viel Kommunikation und andererseits können auf Blöcken basierende Algorithmen nicht mehr effizient gerechnet werden. Zusätzlich sind unter Umständen die optimalen algorithmischen Blockgrößen für die einzelnen Teilschritte des Eigenwertlösers verschieden. Um jeweils die optimale Blockgröße nutzen zu können, müsste man die Daten jeweils neu verteilen bzw. umverteilen, was allerdings auf Grund des Aufwands unsinnig wäre.

ELEMENTAL nutzt ebenfalls eine zweidimensionale zyklische Aufteilung auf die Prozessoren. Allerdings ist hier immer die Blockgröße $bl = 1$ gewählt, das heißt es werden keine Blöcke, sondern einzelne Elemente der Matrix auf die Prozessoren verteilt, was immer zu einer optimalen Lastverteilung führt. Die algorithmische Blockgröße bl_{algo} ist von der Verteilung auf die Prozessoren unabhängig, so dass sie vom Anwender frei gewählt werden kann. Dabei ist es möglich die algorithmische Blockgröße innerhalb eines Programms zu ändern, um so für den jeweiligen Algorithmus die optimale Blockgröße auszuwählen.

3 Laufzeiten und Skalierungsverhalten

In diesem Abschnitt werden die Ergebnisse der durchgeführten Tests vorgestellt. Dazu wurden die Testergebnisse der vier ScaLAPACK-Routinen aus einer Untersuchung im Rahmen des Gaststudentenprogramms 2010 [gas11] verwendet.

3.1 Durchgeführte Tests

Bisher wurden nur Tests mit der reinen MPI-Version von Elemental durchgeführt, da die hybride Version von ELEMENTAL noch in Entwicklung ist. Des Weiteren wurde bisher nur die Berechnung aller Eigenwerte untersucht.
Die Tests wurden mit Matrizen der Dimension $m \in \{1024, 1536, 2048, \ldots, 12800\}$ auf 64 bis zu 1024 Prozessoren durchgeführt. Hierfür wurden zum Einen Matrizen mit isolierten Eigenwerten erzeugt – gleichverteilt im Intervall $[0; 1]$–, zum Anderen solche mit stark geclusterten Eigenwerten – $(m - 1)$ Eigenwerte, die um 0 geclustert liegen, ein Eigenwert gleich 1.

3.2 Laufzeiten

In Abbildung 1 werden die Laufzeiten aller Routinen auf 1024 Prozessoren verglichen. Es ist deutlich zu erkennen, dass PDSYEV aus ScaLAPACK absolut nicht mit den anderen vier Routinen mithalten kann. Diese Routine basiert auf dem QR-Algorithmus, welcher mittlerweile von neueren Algorithmen überholt wurde. Weiter kann man erkennen, dass die Berechnung mit PDSYEVR von ScaLAPACK und einer großen Prozessoranzahl bei isolierten Eigenwerten erst bei großen Matrizen sinnvoll ist. Dies liegt jedoch an einem Fehler in der Implementierung. Bei geclusterten Eigenwerten und auch bei kleineren Prozessorzahlen tritt dieses Problem nicht auf. Diese Routine gehört noch nicht zum offiziellen Release von ScaLAPACK.

Für alle Routinen außer für PDSYEVX und PDSYEVR von ScaLAPACK unterscheiden sich die Laufzeiten bei stark geclusterten Eigenwerten nicht von denen bei isolierten Eigenwerten, daher wurden nur für diese Routinen auch die Zeiten bei geclusterten Eigenwerten dargestellt. Durch die erforderliche Nachorthogonalisierung bei der inversen Iteration ist PDSYEVX bei geclusterten Eigenwerten so viel langsamer, dass sie nicht mehr sinnvoll verwendet werden kann. Da die Nachorthogonalisierung außerdem sequentiell auf einem einzelnen Prozessor stattfindet, ist die Matrixgröße, bis zu der dies möglich ist, auch durch den kleinen Speicherplatz auf der Blue Gene/P begrenzt. Andererseits zeigt der MRRR-Algorithmus jetzt seine Leistungsfähigkeit, da beide Routinen, die ihn verwenden mit geclusterten Eigenwerten die schnellsten sind.

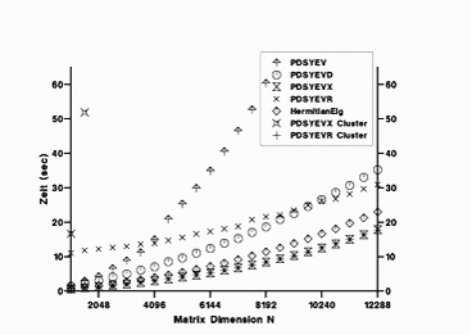

Abbildung 1: Laufzeiten aller getesteten Routinen bei isolierten Eigenwerten und von PDSYEVX und PDSYEVR auch bei geclusterten Eigenwerten auf 1024 Prozessoren

3.3 Skalierungsverhalten

Als Maß für die Skalierung wurde in Abbildung 2 jeweils der Speedup aufgetragen. Der Speedup ist definiert durch $S_p = T_{seq}/T_p$, wobei T_{seq} die sequentielle Laufzeit und T_p die Laufzeit mit p Prozessoren bezeichnet. Hier wurde als Vergleichswert allerdings keine sequentielle Laufzeit, da für parallele Eigenwertlöser nicht sinnvoll, sondern T_{64} genutzt. In Abbildung 2 erkennt man links, dass eine Vergrößerung der Matrixdimension sich positiv auf den Speedup auswirkt, da der Speedup bei einer festen Problemgröße auf Grund von Amdahls Gesetz [Amd67] in eine Sättigung läuft.

In Abbildung 2 werden rechts die Speedups der verschiedenen getesteten Routinen bei einer festen Problemgröße von $n = 12288$ betrachtet. Zu erkennen ist, dass bei dieser noch relativ kleinen Problemgöße alle Routinen schon mit 256 Prozessoren das Ende der Skalierung fast erreicht haben. Eine Erhöhung der Prozessoranzahl auf 1024 und mehr bringt hier fast keinen Gewinn mehr. Da aber alle Routinen mit Ausnahme von PDSYEVR, die fehlerhaft implementiert ist, etwa gleich gut skalieren, ist zu hoffen, dass bei einer größeren Matrixdimension auch bei den ScaLAPACK Routinen die Skalierung besser wird.

4 Fazit und Ausblick

Von den hier vorgestellten Routinen erscheint PDSYEVX (Bisektion) am schnellsten, wenn isolierte Eigenwerte zu berechnen sind. Dies ist allerdings auch die einzige Routine, die bei geclusterten Eigenwerte große Probleme bekommt, so dass in diesem Fall die Routinen, die den MR^3-Algorithmus verwenden, vorzuziehen sind. Die Tests für ELEMENTAL wurden bisher mit einer festen algorithmischen Blockgröße durchgeführt, so dass ein wesentlicher Vorteil, das Variieren der Blockgröße, noch nicht berücksichtigt wurde. Interessant ist die Optimierung dieser Routine durch das Variieren der Blockgröße zwischen den Teilschritten. Außerdem soll in Zukunft noch die Performance bei der Berechnung von nur 10 % der Eigenwerte, sowie bei einer hybriden Version getestet werden.

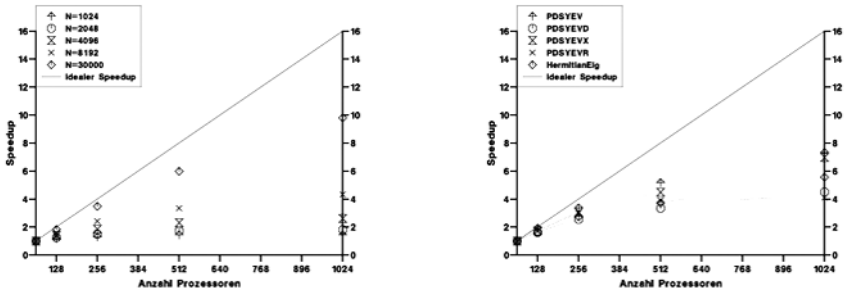

Abbildung 2: Links: Speedup von Elementals HermitianEig bei isolierten Eigenwerten. Rechts: Speedup der Routinen PDSYEV, PDSYEVD, PDSYEVX, PDSYEVR und HermitianEig bei isolierten Eigenwerten und Matrixgröße 12288.

Literatur

[ABB99] E. Anderson, Z. Bai und C. Bischof. *LAPACK Users' guide*. Society for Industrial Mathematics, 1999.

[Amd67] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, Seiten 483–485. ACM, 1967.

[BCC+97] LS Blackford, A. Cleary, J. Choi, E. d'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet et al. *ScaLAPACK users' guide*. Society for Industrial Mathematics, 1997.

[Cup81] JJM Cuppen. A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem. *Numer. Math*, 36:177–195, 1981.

[DP04] I.S. Dhillon und B.N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications*, 387:1–28, 2004.

[DPV06] I.S. Dhillon, B.N. Parlett und C. Vömel. The design and implementation of the MRRR algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 32(4):533–560, 2006.

[DW97] J.J. Dongarra und R.C. Whaley. A User's Guide to the BLACS v1. 1. *LAPACK working notes*, 1997. Available online at `http://www.netlib.org/blacs/lawn94.ps`.

[FP94] K.V. Fernando und B.N. Parlett. Accurate singular values and differential qd algorithms. *Numerische Mathematik*, 67(2):191–229, 1994.

[gas11] Gaststudentenprogramm 2010. Website, Jun 2011. Available online at `http://www2.fz-juelich.de/jsc/gsp_de/archiv/gs2010#`.