Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2009-12

11/May/2009

On Parallelizing the MRRR Algorithm for Data-Parallel Coprocessors

C. Lessig and P. Bientinesi

RWTH AACHEN UNIVERSITY

AI ces

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.

©C. Lessig and P. Bientinesi 2009. All rights reserved

List of AICES technical reports: http://www.aices.rwth-aachen.de/preprints

On Parallelizing the MRRR Algorithm for Data-Parallel Coprocessors

Christian Lessig[1] and Paolo Bientinesi[2]

[1] DGP, University of Toronto, Canada; lessig@dgp.toronto.edu
[2] AICES, RWTH Aachen, Germany; pauldj@aices.rwth-aachen.de.

Abstract. The eigenvalues and eigenvectors of a symmetric matrix are needed in a myriad of applications in computational engineering and computational science. One of the fastest and most accurate eigensolvers is the Algorithm of Multiple Relatively Robust Representations (MRRR). This is the first stable algorithm that computes k eigenvalues and eigenvectors of a tridiagonal symmetric matrix in $O(nk)$ time. We present a parallelization of the MRRR algorithm for data parallel coprocessors using the CUDA programming environment. The results clearly demonstrate the potential of data-parallel coprocessors for scientific computations: when comparing against routine `sstemr`, LAPACK's implementation of MRRR, our parallel algorithm provides 10-fold speedups.

1 Introduction

The eigendecomposition of a real symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined as $\mathbf{AZ} = \mathbf{Z\Lambda}$, where the columns of $\mathbf{Z}$ and the diagonal matrix $\mathbf{\Lambda}$ contain the eigenvectors and the eigenvalues of $\mathbf{A}$, respectively.[1] The decomposition, also known as the symmetric eigenproblem, is is of significant importance in both science and engineering. This motivated the development and thorough study of a variety of numerical solvers [6, 10]. The efficiency of such eigensolvers depends on the available computational resources and on the required accuracy of the results. With the advent of mainstream parallel architectures such as multi-core CPUs and data-parallel coprocessors also the amenability to parallelization is becoming an important property.

Most eigensolvers do not lend themselves to straightforward parallelizations. In many cases space requirements and/or lack of accuracy become limiting factors. The Algorithm of Multiple Relatively Robust Representations (MRRR) [3] does not suffer from the same constraints. An efficient

[1] In this report we will employ Householder's notation. Matrices will be denoted by bold capital letters such as $\mathbf{A}$ and $\mathbf{T}$; vectors by bold small letters such as $\mathbf{a}$ and $\mathbf{b}$; and scalars by non-bold letters such as a, b, and α.

Algorithm 1: Sketch of the MRRR algorithm for the tridiagonal symmetric eigenproblem.

Input: A tridiagonal symmetric matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$.
Output: The eigenvalues $\mathbf{\Lambda}$ and the eigenvectors $\mathbf{Z}$ of $\mathbf{T}$.

1 Initialize the index set K to $[1..n]$.
2 Find a RRR $\mathbf{LDL^T}$ for a shift of $\mathbf{T}$ with respect to K.
3 Compute or refine the eigenvalues of $\mathbf{LDL^T}$ in K.
4 **for** each $i \in K$ **do**
5   Classify λ_i as singleton or cluster.
6 **for** each cluster $\lambda_{k_1:k_m}$ **do**
7   Set $\hat{K}$ to $[k_1..k_m]$.
8   Compute a new RRR $\mathbf{\hat{L}\hat{D}\hat{L}^T}$ for $\mathbf{LDL^T} - \mu\mathbf{I}$ with respect to $\lambda_{k_1}, \ldots, \lambda_{k_m}$, where μ is a shift close to the cluster $\lambda_{k_1:k_m}$.
9   Let $\mathbf{LDL^T} := \mathbf{\hat{L}\hat{D}\hat{L}^T}$, $K := \hat{K}$, and go to Line 3.
10 **for** each singleton λ_i **do**
11   Compute the eigenvalue and eigenvector to high relative accuracy.

variant of MRRR for distributed systems was recently developed [2], but no data-parallel version exists yet.

In this article we present a parallelization of the MRRR algorithm for data-parallel coprocessors. By using the Compute Unified Device Architecture (CUDA) programming environment [9], we efficiently map MRRR onto data-parallel architectures. When compared to `sstemr`, LAPACK's [1] implementation of the MRRR algorithm, we obtain up to 10-fold speedups.

The rest of the paper is organized as follows. In the next section we provide a brief introduction to the MRRR algorithm. In Sec. 3 we discuss our implementation and design choices, and Sec. 4 contains experimental results. Conclusions and directions of future work are given in Sec. 5.

2 The MRRR Algorithm

The *Algorithm of Multiple Relatively Robust Representations* (MRRR) [3] is the first stable $O(nk)$ algorithm for computing k eigenvalues $\mathbf{\Lambda}$ and eigenvectors $\mathbf{Z}$ of a symmetric tridiagonal matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$.

MRRR guarantees small residuals, and orthogonality of the eigenvectors:

$$\|\mathbf{TZ} - \mathbf{Z\Lambda}\| = \mathcal{O}\left(n\epsilon\|\mathbf{T}\|\right), \quad \|\mathbf{Z}^T\mathbf{Z} - I\| = \mathcal{O}\left(n\epsilon\right). \tag{1}$$

In the following we present an overview of the MRRR algorithm as shown in Alg. 1. A thorough discussion can be found in [5].

MRRR is eigenvalue-centric: the computation of the eigenvalues precedes that of the eigenvectors and dictates the unfolding of the algorithm. The eigenvectors are obtained only once the computed eigenvalues meet a separation criterion that ensures that the eigenvectors never need explicit re-orthogonalization.
Once an initial estimate of the eigenvalues of the input matrix $\mathbf{T}$ is obtained (Lines 2 and 3 in Alg. 1), the algorithm proceeds by computing relatively robust representations (RRR) for shifts of $\mathbf{T}$ (Line 8). An RRR for a matrix $\mathbf{M}$ is a representation of $\mathbf{M}$ that determines its eigenvalues and eigenvectors to high relative accuracy, provided that the eigenvalues are well separated. An eigenvalue λ_i is called well separated, or a *singleton*, if its relative distance $reldist(\lambda_i, \lambda_j) \equiv \frac{|\lambda_i - \lambda_j|}{|\lambda_i|}$ is greater than a threshold t_c for all λ_j, $i \neq j$ (Line 5). Under these conditions, small component-wise changes to individual entries of the RRR lead to small relative changes in the eigenvalues and eigenvectors. The eigenpairs corresponding to singletons can therefore be directly computed to high accuracy (Lines 10-11).
A set of k eigenvalues that do not satisfy the condition $reldist(\lambda_i, \lambda_j) > t_c$ form a *cluster*. In this case, the eigenpairs cannot be computed directly without loss of accuracy [3] and a new representation, robust relatively to the eigenvalues within the cluster, is needed (Line 8). This is accomplished by employing matrix shifts of the form $\hat{T} = T - \sigma I$ and by refining the clustered eigenvalues (Line 3). The new RRR for $\hat{T}$ guarantees that some of the eigenvalues in the cluster will be well separated, hence the corresponding eigenpairs can be computed to high accuracy. For the eigenvalues that are still clustered the algorithm proceeds recursively until all eigenvalues are well separated.
A *representation tree* is convenient way to capture the sequence of computations in the algorithm [4]. The root node of the representation tree is initialized to be the set of desired eigenvalues. The other nodes in the tree correspond to clusters of eigenvalues and singletons. The edges correspond to matrix shifts. For the remainder of this paper, is suffices to note that nodes at the same depth capture independent calculations, i.e., they can be performed in a task-parallel fashion. In the next section we outline how to efficiently map a representation tree onto data-parallel architectures.

3 Parallelization & Implementation

In this section we present a parallelization of the MRRR algorithm for data-parallel devices using the CUDA application programming interface.

The computational engine of CUDA-capable devices is a task-parallel array of data-parallel multiprocessors. For many practical applications, multiple devices can be considered as one virtual device with an increased number of multiprocessors. Each multiprocessor has an independent SPMD (Single Process, Multiple Data) execution unit and can run up to 512 or 1024 threads simultaneously. In order to exploit these two levels of parallelism, we employ a two-stage approach: in the first stage the eigenproblem is decomposed into a set of smaller problems that can be solved independently on separate multiprocessors. In the second stage, the computation of the eigenpairs (local to one multiprocessor) is organized to be suitable for data-parallel devices.
Input: A symmetric tridiagonal matrix $\mathbf{T}$ and a threshold t_c to classify the eigenvalues' relative gaps.
Output: The eigenvalues and eigenvectors of $\mathbf{T}$ computed to high relative accuracy.
Assumptions: $\mathbf{T}$ is of size less than or equal to 4096, is unreduced and does not contain clusters of size greater than 512.

Host Computations. The MRRR algorithm, as described in Alg. 1, begins with the computation of an $\mathbf{LDL^T}$ representation for a shift of $\mathbf{T}$ (Step 2). Since this is an inherently sequential operation, we assign it to the CPU. The resulting RRR, identified by the vectors $\mathbf{d}$ and $\mathbf{l}$, is then transferred onto the device to calculate a first estimate of the eigenvalues. These initial steps are equivalent to the processing of the root node in the representation tree.
Besides the computation of the initial RRR, the CPU is only responsible for bundling together the singletons and the clusters (resulting from Step 5) into task-batches. These batches are then dispatched and executed on different multiprocessors. In terms of the representation tree, this means that the subtrees rooted at depth one are bundled into "forests" that are assigned to different multiprocessors.

Device Computations. Once the vectors $\mathbf{d}$ and $\mathbf{l}$ have been uploaded onto the device, the bisection algorithm (see below) is used to obtain initial estimates of the eigenvalues (Line 3). The eigenvalues are then subdivided into clusters and singletons (Line 5) according to the threshold t_c. In the representation tree, this stage creates the child nodes of the root node. This list of nodes is passed to the host where batches are created. The computation on the device resumes once the list of clusters and singletons that must be processed is uploaded. Notice that since $\mathbf{T}$ is

unreduced, the RRRs for all the singletons and clusters are of the same size. This makes it possible to take advantage of data-parallelism, as all the clusters can be processed in parallel, and the same is true for the singletons. Specifically, for each cluster a new shift and RRR are computed (Step 8), and a refinement of the eigenvalues is obtained (Step 3). The calculations are carried out by variants of the **qds** transform and by the bisection algorithm, respectively (see below). For each singleton (Step 11), an eigenpair is computed to high accuracy via twisted factorizations, also based on the **qds** transform.

qds *transform.* The **qds** transform is used to determine the RRR of shift matrices, to compute the eigenvectors for isolated eigenvalues, and it also underlies the Sturm count computation, an integral part of bisection. Unfortunately, the transform consists of a loop with dependent iterations and therefore is not task-parallelizable. In data-parallel devices this limitation vanishes, as they operate in SPMD fashion, i.e., multiple **qds** transforms can be computed at once. In fact, the computations for all singletons and clusters at the same tree level and in the same batch are, up to the eigenvalue and cluster location, identical. Hence they can be processed simultaneously in a data-parallel fashion.

Bisection. The bisection algorithm is a remarkably easy and effective method for computing the eigenvalues of a symmetric tridiagonal matrix. Moreover, it can be implemented efficiently on a data-parallel coprocessor [8, 11]. The algorithm starts with a set of intervals containing the desired eigenvalues. At each iteration the intervals are subdivided into two subintervals and the number of eigenvalues contained in each subinterval is determined by performing Sturm counts. All non-empty intervals are stored in a compact list, using a version of the scan algorithm [7], and are again subdivided, starting a new iteration of the algorithm. Convergence is determined by a mixed relative and absolute criterion, as in LAPACK's bisection routine `stebx`. The Sturm count is determined using the **qds** transform and hence not parallelizable. However, Sturm counts for different intervals are independent and can thus be performed in parallel on a data-parallel device.

4 Experimental Evaluation

To assess the quality of our data-parallel MRRR, `mrrr_dp`, we implemented it using CUDA (ver. 2.1) and compared against CLAPACK's `sstemr` routine (ver. 3.1.1). Our test system was a Intel Pentium D CPU (3.0 GHz) with 1GB RAM, and an NVIDIA GeForce 8800 Ultra (driver

version 180.22). The operating system employed for the experiments was Fedora Core 10 and all programs were compiled using `gcc` 4.3.2.
Our testbed consists of random matrices of size ≤ 1024 with entries uniformly distributed on $[0,1]$ and $[-1,1]$. We chose these two types of matrices because of the different eigenspectrum they possess. In a successive report we will test `mrrr_dp` on a much larger variety of test matrices, including tridiagonal matrices arising in scientific applications.

		rand(0,1)				rand(-1,1)			
n	Alg.	Time	Res.	Orth.	Eigv.	Time	Res.	Orth.	Eigv.
128	`mrrr_dp`	6.26	8.6e-7	6.6e-6	9.6e-7	3.84	5.0e-7	1.8e-5	3.0e-7
	`sstemr`	6.98	1.4e-6	1.0e-5	1.3e-6	6.79	1.4e-6	1.0e-5	1.3e-6
256	`mrrr_dp`	13.0	8.9e-7	8.0e-6	1.0e-6	8.34	4.9e-7	2.6e-5	2.8e-7
	`sstemr`	32.1	1.5e-6	1.0e-5	1.4e-6	31.8	1.5e-6	1.2e-5	1.5e-6
512	`mrrr_dp`	28.7	1.2e-6	9.8e-6	1.0e-6	19.2	8.5e-7	3.3e-5	3.0e-7
	`sstemr`	154.9	1.5e-6	9.4e-6	1.4e-6	152.7	1.6e-6	9.1e-6	1.5e-6
1024	`mrrr_dp`	60.2	3.0e-6	3.6e-5	1.2e-6	54.0	3.5e-6	8.1e-5	8.0e-7
	`sstemr`	656.1	2.8e-6	1.0e-5	1.6e-6	647.6	2.9e-6	1.4e-5	1.6e-6

Table 1. Timings and accuracy comparison between `mrrr_dp` and `sstemr` for random matrices with entries uniformly distributed on $[0,1]$ and $[-1,1]$.

In addition to timings, we also report three accuracy measures: the residual and the orthogonality of the eigendecomposition (1), and the distance of the computed eigenvalues from LAPACK's double-precision function `dsterf`, that implements the highly accurate QR algorithm.
Our experimental data demonstrate the potential of parallelizing the MRRR algorithm for data-parallel architectures. As shown in Fig. 1 and Table 1, `mrrr_dp` provides significant speedups over `sstemr` for matrices with $n \geq 128$ elements. The performance improvement increases linearly with the matrix size, reaching more than 10-fold speedups for $n = 1024$, which is still a modest problem size.
Table 1 also includes data on the accuracy attained by `sstemr` and `mrrr_dp`. The experiments provide evidence that our implementation is equivalent to `sstemr` in terms of residual and orthogonality, and that `mrrr_dp` returns eigenvalues one order of magnitude more accurate than `sstemr`.
The orthogonality of the eigenvectors computed by `mrrr_dp` degrades slightly as the matrix size increases, but remains of the same order of magnitude as that attained by `sstemr`. We believe that this problem can be alleviated by a more careful strategy for choosing the shift in the computation of the RRR for clustered eigenvalues.

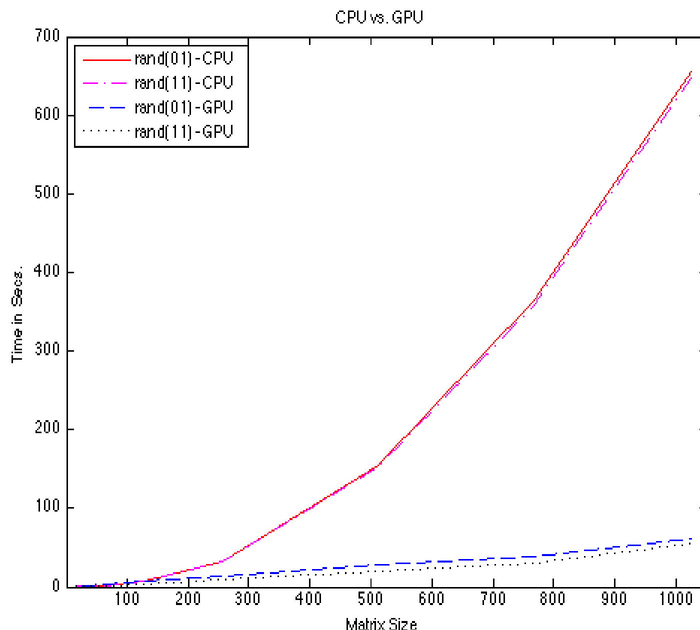

Fig. 1. Execution time for 30 random matrices for each matrix size $n \in [32, 1024]$.

5 Conclusion

We introduced mrrr_dp, a data-parallel version of the Algorithm of Multiple Relatively Robust Representations (MRRR) for the symmetric tridiagonal eigenvalue problem. Our experimental results demonstrate that the MRRR algorithm can be mapped efficiently onto data-parallel coprocessors: mrrr_dp retains the same accuracy of sstemr, LAPACK MRRR's implementation, while attaining significant performance speedups.
Our results are still preliminary: in the near future we will investigate a number of optimizations for both accuracy and performance. Also, we will release the constraints on the input matrix and further explore the tradeoff between speed and accuracy.

Acknowledgements

The first author thanks NVIDIA's London office, and in particular Mark Harris, for many helpful discussions and acknowledges support by the Natural Sciences and Engineering Research Council of Canada (NSERC). The second author gratefully acknowledges the support received from the

Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111.

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
2. Paolo Bientinesi, Inderjit S. Dhillon, and Robert A. van de Geijn. A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations. *SIAM J. Sci. Comput.*, 27(1):43–66, 2005.
3. Inderjit S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, EECS Department, University of California, Berkeley, 1997.
4. Inderjit S. Dhillon and Beresford N. Parlett. Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices. *Linear Algebra and its Applications*, 387(1):1–28, August 2004.
5. Inderjit S. Dhillon, Beresford N. Parlett, and Christof Vömel. The design and implementation of the mrrr algorithm. *ACM Trans. Math. Softw.*, 32(4):533–560, 2006.
6. Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
7. Mark Harris, Shubhabrata Sengupta, and John D. Owens. Parallel Prefix Sum (Scan) with CUDA. In Hubert Nguyen, editor, *GPU Gems 3*. Addison Wesley, August 2007.
8. Christian Lessig. Eigenvalue Computation with CUDA. Technical report, NVIDIA Corporation, August 2007.
9. NVIDIA Corporation. *CUDA Programming Guide*. NVIDIA Corporation, 2701 San Toman Expressway, Santa Clara, CA 95050, USA, first edition, 2007.
10. Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.
11. Vasily Volkov and James Demmel. Using GPUs to Accelerate the Bisection Algorithm for Finding Eigenvalues of Symmetric Tridiagonal Matrices. Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2007. Online available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-179.html.