Towards scaling symmetric tridiagonal eigenvalue solvers on multiple nodes of CPUs-GPUs

Alberto Estrella Cruz, Jorge A. Rivera Rivera, Amilcar Meneses Viveros, Sergio V. Chapa Vergara
Computer Science Department, CINVESTAV-IPN, Mexico D.F., México.

Abstract

This poster presents the ongoing research on the hybrid parallel implementation on a cluster with multiple GPUs and CPUs of the divide-and-conquer eigenvalue algorithm. Divide-and-conquer algorithm is a numerically stable and efficient algorithm that computes all eigenvalues and eigenvectors of a symmetric tridiagonal matrix. A major drawback of implementing the algorithm on GPUs is that computation is limited by the amount of GPU memory. We overcome the issue by using multiple nodes with CPUs and GPUs to solve subproblems which fit on device memory in parallel, and merging those subproblems to get the whole result. Preliminary experiments show promising results. Our approach has better performance than state-of-the-art libraries even in sequential version. Furthermore, it exhibits a meaningful degree of accuracy with respect to eigenvectors orthogonality and egeinpairs quality.

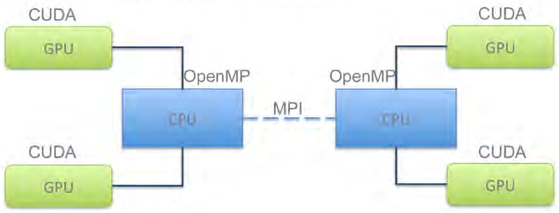


Fig. 1: Hybrid parallel architecture

Divide-and-Conquer Algortihm

The main idea behind the algorithm is to solve two smaller problems and join the solution with a rank one modification. Thus, we have two independent diagonalizations

$$T_1 = Q_1 \Lambda_1 Q_1^T \qquad \text{and} \qquad T_2 = Q_2 \Lambda_2 Q_2^T$$

The original matrix T can be rewritten as

$$T = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} + \rho v v^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T$$

The original matrix T has the same eigenvalues as the rank one modification M

$$M \equiv D + \rho v v^T$$

Whom eigenvalues Λ are given by secular equation

$$0 = 1 + \rho \sum_{i=1}^{n} \frac{v_i^2}{d_i - \lambda}$$

The eigenvectors of M are

$$y_i = (D - \lambda_i I)^{-1} v$$

Eigenvectors associated with the original matrix T are given by

$$q_i = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} y_i$$

To obtain an efficient and accurate implementation further details must to be taken into account. A better description of the algorithm can be found in [1, 2]

Implementation

We execute each phase of the algorithm on CPU, GPU, or both depending on the amount and suitability of the work.

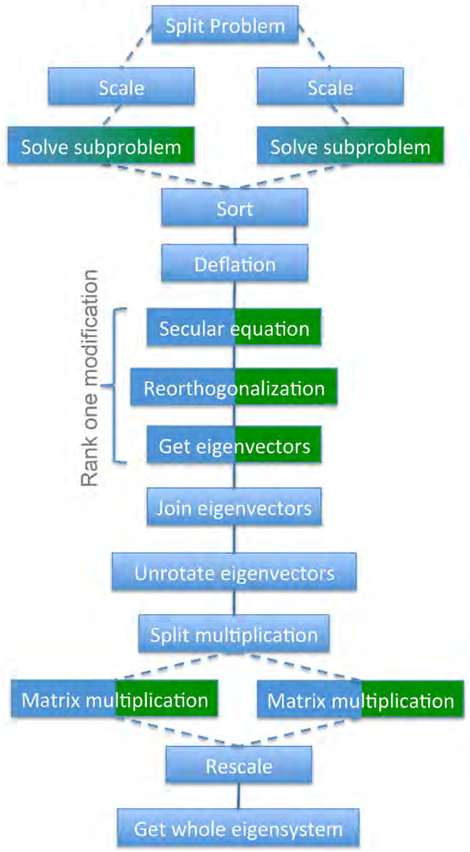


Fig. 2: Process flow

References

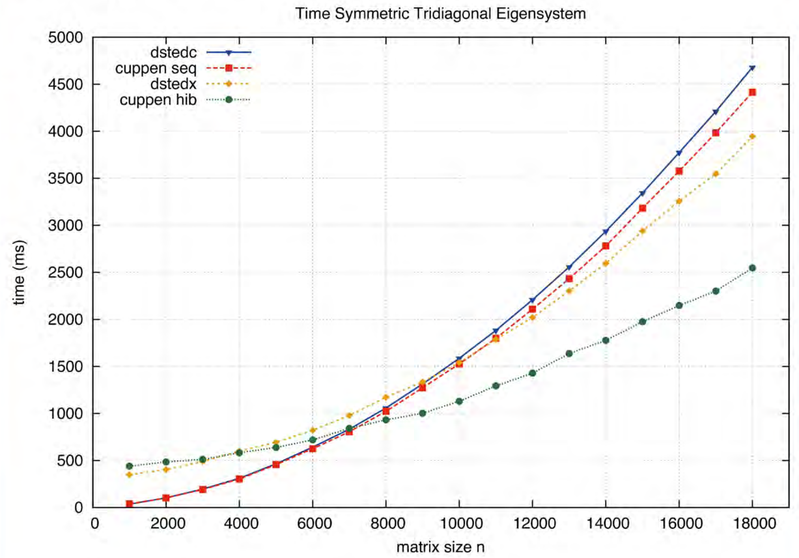
[1] Peter Arbenz. Chapter 4 The Cuppen's divide and conquer algorithm. In Lecture notes on solving large scale eigenvalue problems. Web site: `http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf` [Last accessed: 12 September 2014].

[2] Ming Gu and Stanley C. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal on Matrix Analysis and Applications*, 15(4):1266–1276, October 1994.

[3] Innovative Computing Laboratory (ICL) Team. Matrix Algebra on GPU and Multicore Architectures (MAGMA) Library, version 1.5.0. Web site: `http://icl.cs.utk.edu/magma/` [Last accessed: 12 September 2014].

Preliminary Results

To get these results we use an architecture with the following specifications: 2 six-core Intel Xeon X5675 processors at 3.07GHz, 12 GB of RAM, Nvidia Tesla 2070, Nvidia GeForce GTX 460, and MAGMA [3] compiled with Intel MKL.



dstedc: Divide-and-conquer sequential MAGMA routine
cuppen seq: Sequential implementation
dstedx: Divide-and-conquer parallel MAGMA routine (12 threads, 1 GPUs)
cuppen hib: Hybrid parallel implementation (2 nodes with 6 threads and 1 GPU)

Sequential Cuppen Errors

n	$\|AQ - Q\Lambda\|_F$	$\|QQ^T - I\|_F$
1000	3.541376936E-14	4.357052648E-14
2000	5.136740656E-14	6.104834842E-14
3000	6.948477673E-14	8.342782855E-14
4000	6.859582577E-14	8.527954316E-14
5000	8.323512655E-14	1.017648518E-13
6000	9.813712063E-14	1.200419097E-13
7000	9.031387285E-14	1.099637942E-13
8000	9.862009684E-14	1.207374579E-13
9000	1.072145891E-13	1.333045672E-13
10000	1.196421220E-13	1.444052865E-13
11000	1.278536693E-13	1.545681602E-13
12000	1.362339403E-13	1.653957160E-13
13000	1.181758579E-13	1.470576993E-13
14000	1.283036187E-13	1.566744735E-13
15000	1.326946596E-13	1.630397917E-13
16000	1.392612719E-13	1.728430409E-13
17000	1.462918336E-13	1.799072233E-13
18000	1.537716708E-13	1.883434610E-13

Hybrid Parallel Cuppen Errors

n	$\|AQ - Q\Lambda\|_F$	$\|QQ^T - I\|_F$
1000	7.503678357E-14	9.529264524E-14
2000	1.112517803E-13	1.358937426E-13
3000	1.172849143E-13	1.425196426E-13
4000	1.531046661E-13	1.904294287E-13
5000	1.401008049E-13	1.698887898E-13
6000	1.644446667E-13	2.019083607E-13
7000	1.924026676E-13	2.374194577E-13
8000	2.182177804E-13	2.693576798E-13
9000	1.769904245E-13	2.178262381E-13
10000	1.971301239E-13	2.406783725E-13
11000	2.128947481E-13	2.616874319E-13
12000	2.314818610E-13	2.860579610E-13
13000	2.540576202E-13	3.118560676E-13
14000	2.730416499E-13	3.331240237E-13
15000	2.908211089E-13	3.576597100E-13
16000	3.090302981E-13	3.799984033E-13
17000	2.376664933E-13	2.908348661E-13
18000	2.509467949E-13	3.087416228E-13