GPU-optimized parallel preconditioners for the element-by-element finite element method

Imre Kiss*, Zsolt Badics† and Szabolcs Gyimóthy*
*Budapest University of Technology and Economics, Egry József utca 18, H-1521 Budapest, Hungary
†Tensor Research LLC, Andover, MA, U.S.A.
Email: kiss@evt.bme.hu

***Abstract*—The utilization of several different preconditioners are investigated in conjunction with the parallel GPU version of the Element-by-Element (EbE) FEM. The EbE technique has long since been known to perform operations in Krylov-subspace iterative solvers without actually assembling the global system matrix. Since all the computations are performed on the element level, traditional preconditioning techniques that rely on the assembled matrix are not usable. Therefore efficient preconditioning methods that can be constructed and applied without the presence of the assembled matrix have been developed. Among these, the Hughes-Winget and the Gauss-Seidel EbE preconditioners will be investigated and compared to the diagonal Jacobi preconditioner. Each were implemented on a NVIDIA CUDA GPU in a fully parallel way. Another novelty of the implementations here is that the element matrices are not stored but rather recomputed in each iteration. The preconditioner is also generated on the fly, resulting in a minor overall memory consumption.**

***Index Terms*—CUDA, element-by-element, GPU FEM, preconditioning.**

I. Introduction

Our goal is to illustrate the efficiency of an application centric parallel method for the solution of a wide range of partial differential equations (PDEs) discretized by the finite element (FE) method. We demonstrate here the performance of the non-conventional finite element parallel method through solving the Laplace equation with spatially varying conductivity arising from a static current flow problem. The introduced technique exhibits highly improved locality and minimization of data movement compared to other FE implementations on GPUs.

Although several methods accelerating the FEM are already implemented on GPUs [1], [2], these usually suffer from the strict limitation of available memory. As large scale FEM problems need large storage capacity, the few GB of available memory on GPUs must be continuously cached to the global system memory through the relatively slow bus system.

In contrast, our acceleration technique, based on the element-by-element FE framework [3], provides a factor of ten times speed improvement for the ECG (electrocardiography) forward problem compared to top-notch CPU implementations for different discretization levels tested up to 30 million tetrahedrons. The present method utilizes the diagonal Jacobi preconditioner to facilitate the solution of the Laplace equation. There are however certain limitations when applying such a simple preconditioner. Most importantly, the high condition number of the system matrix may either lead to high iteration number or potentially fail convergence. When considering more efficient preconditioners, besides the mere performance attention must be payed to the computational demand of the underlying implementation as well, since these are subjected to be computed on the fly, in each iteration of the solver.

II. Aim of the work

As the gap between bus speed and computation density increases, codes based on the accelerator design, that is, execute only computation intensive parts on the GPU, will fall behind codes that take full advantage of it [3], [4]. The latter perform all the necessary computations on the GPU, contrary to the accelerator design, where operations are performed on the precomputed and transferred elements.

The aim of this paper is therefore to extend the scope of problems GPUs can effectively handle, by utilizing several different preconditioners that are suitable to the EbE implementation of the bi-conjugated gradient (BiCG) solver. Relying on the fact that it is cheaper to recompute element matrices and form the preconditioner than continuously cache the global system matrix between the GPU and the system memory, neither the element matrices nor the preconditioner is subjected to store, but are rather recomputed in each iteration.

III. Element-by-element method

A. Element-wise computations

The finite element assembling procedure relies on some functions by which the element matrix $\mathbf{A}_e$ and the RHS $\mathbf{b}_e$ appearing in the element level equation, $\mathbf{A}_e\mathbf{x}_e = \mathbf{b}_e$, are computed. These functions depend among others on the type of PDE to be solved as well as on the applied shape functions. The computed element matrices and RHS vectors are then assembled to form the global system matrix $\mathbf{A}$ and RHS $\mathbf{b}$. Let this assembly step be represented by an operator $\mathcal{M}$, which is defined differently for matrices and vectors, as follows:

$$\mathbf{A} = \mathcal{M}(\mathbf{A}_e) = \sum_{e\in\mathcal{E}} \mathbf{C}_e^T \mathbf{A}_e \mathbf{C}_e \quad (1)$$

$$\mathbf{b} = \mathcal{M}(\mathbf{b}_e) = \sum_{e\in\mathcal{E}} \mathbf{C}_e \mathbf{b}_e \quad (2)$$

where $\mathcal{E}$ is the set of elements: $1,\ldots,N_e$ and matrix $\mathbf{C_e}$ represents the transition between *local* and *global* numbering of the unknown variables for the e-th element. Contrary to the

sparse global system matrix $\mathbf{A}$, the matrix $\mathbf{A_e}$ of size $n_e \times n_e$ (n_e being the local degrees-of-freedom) is usually dense.

Using the above concept, for example the matrix-vector product, which is the basis of iterative solvers, can be reformulated in terms of element-wise computations as

$$\mathbf{A}\mathbf{x} = \sum_{e\in\mathcal{E}} \mathbf{C}_e^T \mathbf{A}_e \mathbf{C}_e \mathbf{x} = \sum_{e\in\mathcal{E}} \mathbf{C}_e^T \mathbf{A}_e \mathbf{x}_e = \mathcal{M}(\mathbf{A}_e \mathbf{x}_e). \quad (3)$$

This means that the product of an assembled global matrix and a vector is equivalent with the assembled vector of the elementary matrix-vector products. According to (2) the elementary contributions can be accumulated in a vector, the size of which is equal to the global degrees-of-freedom (DoF), hence only vectors have to be stored during the computations. Elementary matrix-vector products in (1) can be computed for each element separately, which enables parallel realization [5].

B. Element level preconditioners

The diagonal preconditioner The diagonal preconditioner is a particularly simple preconditioner, which is also known as *Jacobi acceleration*. Its form can be given as $\mathbf{W} = \mathrm{diag}(\mathbf{A})$. Its inverse hence can be computed straightforwardly, simply by taking the reciprocal of the elements. In a symmetric form, the preconditioned equation system can be written as

$$\mathbf{W}^{-\frac{1}{2}} \mathbf{A} \mathbf{W}^{-\frac{1}{2}} \, \mathbf{W}^{\frac{1}{2}} \mathbf{x} = \mathbf{W}^{-\frac{1}{2}} \mathbf{b}. \quad (4)$$

Since the preconditioner is only a vector of DoF length, it can be stored on the GPU and respective parts of it can be recalled and applied on each element matrix.

The HW-EbE preconditioner EbE type preconditioners were first introduced in [6] and hence are also called *Hughes–Winget* preconditioners, after the authors' names. A detailed analysis on this technique is given in [7], [8]. Using the definition of the assembly operation for the element matrix in (1) and assuming that $\mathbf{A}$ is positive definite, one may write the preconditioner as

$$\mathbf{A} \approx \mathbf{W}^{\frac{1}{2}} \left(\prod_{e=1}^{N_{el}} \mathbf{L}_e \right) \left(\prod_{e=1}^{N_{el}} \mathbf{D}_e \right) \left(\prod_{e=N_{el}}^{1} \mathbf{L}_e^T \right) \mathbf{W}^{\frac{1}{2}}, \quad (5)$$

where $\mathbf{W}$ is the same as in (4), implying that a Jacobi preconditioning is applied as a first step; $\mathbf{L}_e$ and $\mathbf{D}_e$ are resulting from the Crout factorization of the Winget regularized element matrix:

$$\overline{\mathbf{A}}_e = \mathbf{I} + \mathbf{W}^{-\frac{1}{2}} \left[\mathbf{A}^e - \mathrm{diag}(\mathbf{A}^e) \right] \mathbf{W}^{-\frac{1}{2}} = \mathbf{L}_e \mathbf{D}_e \mathbf{L}_e^T. \quad (6)$$

GS EbE preconditioner The Gauss–Seidel EbE preconditioner is based on the same decomposition as the EbE preconditioner. But instead of using a Crout factorization, the following decomposition is applied:

$$\widetilde{\mathbf{A}}_e = \mathbf{W}^{-\frac{1}{2}} \left[\mathbf{A}^e - \mathrm{diag}(\mathbf{A}^e) \right] \mathbf{W}^{-\frac{1}{2}} = \mathbf{L}_e + \mathbf{L}_e^T, \quad (7)$$

with $\mathbf{L}_e$ being a strictly lower triangular matrix. The preconditioner is then

$$\mathbf{A} \approx \mathbf{W}^{\frac{1}{2}} \left(\prod_{e=1}^{N_{el}} (\mathbf{I} + \mathbf{L}_e) \right) \left(\prod_{e=N_{el}}^{1} (\mathbf{I} + \mathbf{L}_e^T) \right) \mathbf{W}^{\frac{1}{2}}. \quad (8)$$

The advantage of this preconditioner is obviously its easy construction. In fact, if the matrix is initially scaled so that its diagonals are ones (Winget regularization), the preconditioner need not be explicitly constructed.

IV. Results

The test problem is a static conduction problem with inhomogeneous conductivity that mimics the ECG forward problem in a human TORSO [9]. The domain is discretized by tetrahedral elements and linear nodal shape functions are used. The computations were carried out on a HP-XW8600 workstation, having 32 GB memory, a NVIDIA GTX 480 GPU and a quad-core Intel Xeon X3440 CPU.

As a reference, a multi-CPU version of the BiCG method relying on the usual assembling technique with incomplete Cholesky preconditioning was implemented. The performance of the various element-level preconditioners on GPUs is compared to the performance of the CPU implementation. Table I shows the comparison for the diagonal preconditioner and the full paper will present the results for the other element-level implementations discussed in Section III-B.

Table I: Numerical resuls for the UTAH torso problem.

Test case	#1	#2	#3	#4
No. of tetrahedral	$560K$	$6,559K$	$18,884K$	$29,772K$
No. of unknowns	$91K$	$1,339K$	$3,836K$	$6,064K$
GPU implementation – BiCG w/ diagonal Jacobi (EbE)				
No. of iterations	322	671	978	1,111
Memory [MByte]	12	213	613	968
Runtime (1 GPU) [s]	2.8	40.3	171.3	303.8
CPU implementation – BiCG w/ incomplete Chol. (assembled)				
No. of iterations	79	248	338	391
Memory [MByte]	1,564	5,251	13,645	19,371
Runtime (4-cores) [s]	5.9	403	1,166	1,694

References

[1] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, pp. 917–924, July 2003.

[2] A. Cevahir, A. Nukada, and S. Matsuoka, "Fast conjugate gradients with multiple GPUs," in *ICCS 2009*, G. G. van Albada, J. Dongarra, and P. Sloot, Eds., 2009, vol. 5544, pp. 893–903.

[3] I. Kiss, S. Gyimóthy, Z. Badics, and J. Pávó, "Parallel realization of the element-by-element FEM technique by CUDA," *IEEE Trans. on Magnetics*, vol. 48(2), pp. 507–510, 2012.

[4] I. Kiss, S. Gyimóthy, and J. Pávó, "Acceleration of moment method using CUDA," *The International Journal for Computation and Mathematics in Electrical Engineering (COMPEL)*, vol. 31(6), pp. 1751–1761, 2011.

[5] G. F. Carey and B.-N. Jiang, "Element-by-element linear and nonlinear solution schemes," *Appl. Num. Meth.*, vol. 2 (2), pp. 145–153, 1986.

[6] T. J. Hughes, I. Levit, and J. Winget, "An element-by-element solution algorithm for problems of structural and solid mechanics," *Computer Methods in Applied Mech. and Eng.*, vol. 36, no. 2, pp. 241–254, 1983.

[7] A. J. Wathen, "An analysis of some element-by-element techniques," *Computer Methods in Applied Mechanics and Engineering*, vol. 74, no. 3, pp. 271–287, Sep. 1989.

[8] M. J. Daydé, J.-Y. L'Excellent, and N. I. M. Gould, "Element-by-element preconditioners for large partially separable optimization problems," *SIAM J. Sci. Comput.*, vol. 18, no. 6, pp. 1767–1787, Nov. 1997.

[9] R. MacLeod, C. Johnson, and P. Ershler, "Construction of an inhomogeneous model of the human torso for use in computational ECG," in *IEEE Medicine and Biology Society*. IEEE Press, 1991, pp. 688–689.