Accelerating Sparse Approximate Inverse Preconditioners based on Matrix Entries on GPUs

Maryam Mehri Dehnavi, David M. Fernández, and Dennis Giannacopoulos
Department of Electrical & Computer Engineering, McGill University
3480 University Street, Montreal, Quebec H3A 2A7, Canada
maryam.mehridehnavi@mail.mcgill.ca, david.fernandezbecerra@mail.mcgill.ca, dennis.giannacopoulos@mcgill.ca

Abstract — The sparse approximate inverse preconditioner based on matrix entries (SPAI-ME) is developed to run in parallel on the NVIDIA graphic cards. The column per warp SPAI (CW-SPAI) algorithm proposed introduces new techniques to overcome GPU programming limitations and enhance the execution time of SPAI preconditioners on graphic cards. Speedups of up to 18 times are achieved on the GPU compared to CPU results and the convergence rate of the BiCGSTAB iterative solver is enhanced more than 33 times for some matrices using the preconditioner.

I. Introduction

In the past few decades the popularity of the finite element method (FEM) has considerably increased in electromagnetic simulations. The underlying linear system solution often dominates the overall execution time in such methods and as problem sizes grow larger and more complex, the design of efficient methods to accelerate the solution of such systems is inevitable. Iterative methods are a popular class of solvers for sparse linear systems, and the use of good preconditioners that can be efficiently computed in parallel are essential for such solvers in order to enhance their convergence rate [1].

Sparse approximate inverse (SPAI) preconditioners represent a class of methods tailored to improve the performance of preconditioners on parallel multiprocessors. The approximate inverse of A is computed and stored in the preconditioner M and then applied to the iterative solver. The computation of this preconditioner has been successfully implemented in parallel for distributed computing systems [1]-[5] but has not been accelerated on manycore architecture such as graphic processing units (GPUs).

Since the introduction of general purpose programming on graphic cards, GPUs have become an important computing resource for scientific computing [6]. With peak performances of up to 1.35 teraflops and easy to learn programming interfaces such as CUDA [6], NVIDIA GPUs are an attractive candidate to accelerate the computation of SPAI preconditioners. In this work we present a new method called column per warp SPAI (CW-SPAI), which proposes novel techniques to parallelize sparse approximate inverse preconditioners on GPUs.

II. The Sparse Approximate Inverse Preconditioner

The sparse approximate inverse is a popular class of preconditioners which can be computed by minimizing the frobenius norm of the residual matrix $AM - I$ [2]. Columns of M (m_i) are calculated independently by constructing the small system of equations $\hat{A}\hat{m}_i = \hat{e}_i$ and then scattered back to M to assemble the preconditioner. The hat over the variables indicates that rows and columns are deleted from their original matrix vectors according to a list N_i. Since the objective of this work is to parallelize SPAI preconditioners based on matrix entries (SPAI-ME) [2] the rows i in N are chosen based on a specified tolerance (τ) from the A matrix using the following formula [5]:

$$\left|A_{ij}\right| > (1-\tau)\max_j\left|A_{ij}\right| \qquad (1)$$

Parallel implementations of SPAI across multiple processors using MPI and OpenMP have been proposed in previous works such as [1]-[5]. The computation of SPAI is parallelized in these works by distributing the computation of columns of M amongst different processors. Techniques such as grouping communications [2], effective column partitioning [1] and dictionary based methods [3] enhance the performance of parallel SPAI on multiprocessors. The main disadvantage of such methods besides the high cost of multiprocessors is the distribution of A among different processors resulting in considerable number of intra-processor communications.

GPUs are an attractive hardware platform for parallelizing SPAI due to their massive multithreading, increased utilization of large number of cores and access to an on-chip global memory. While compute intensive kernels of iterative methods such as SMVM [8], [9] have been calculated on the GPU, to the best of our knowledge no work as been published on accelerating SPAI preconditioners on graphic cards.

III. Parallel SPAI-ME on GPUs

The NVIDIA GTX480 is classified as the new generation of graphic cards called Fermi [6]. This graphic card contains 480 scalar processors grouped into 15 clusters named streaming multiprocessors (SM) operating at 1.4 GHz. It has access to a 1.5GB global memory, 768KB of L2 cache, and a configurable 64KB memory divided between shared memory and L1 cache. To run an application on the GPU, compute intensive portions of the program should be sent to the GPU to run in parallel. Each GPU kernel launches thousands of threads grouped in to blocks. Several blocks are allocated to each SM depending on the amount of shared memory available and computed in clusters of 32-threads called warps.

The sparse approximate inverse based on matrix entries (SPAI-ME) is generated in 4 steps for each column m_i of M independently as follows:

1. Row indices I of $\hat{A}$ are chosen based on a user defined parameter τ and values of column i in A.
2. Row indices corresponding to columns I of A are loaded and matched to find J, the number of columns in $\hat{A}$.
3. $\hat{A}$ is constructed.
4. $\hat{A}$ is decomposed using Gram-Schmidt QR [1].
5. $\hat{A}\hat{m}_i = \hat{e}_i$ is solved using QR decomposition and a backward solve and m_i is then scattered back to M.

To parallelize SPAI-ME on graphic cards the above steps should be efficiently implemented on the GPU. Major limitations in accelerating SPAI on GPUs can be categorized as: a) memory restrictions *eg*. static memory space allocation, limited size of global and shared memory, long latency device memory accesses, b) and sequential and tedious to parallelize operations such as sorting and column matching during the constructions of $\hat{A}$, inner products in *QR* decomposition and the sequential solve in the last step.

The column per warp SPAI (CW-SPAI) algorithm proposed in this work, introduces new techniques to parallelize SPAI-ME on GPUs. CW-SPAI parallelizes the construction of M by allocating the computation of each column m_i to one warp. Every 256 thread is grouped in to one block and computes eight columns in parallel. Major contributions and advantages of the proposed algorithm to overcome some of the GPU programming limitations for accelerating SPAI are as follows:

- M is constructed in parallel by allocating the computation of each column m_i to one warp. By limiting the amount of shared memory used by each warp, the number of blocks executing in parallel per SM are increased to enhance parallelism.
- To coalesce memory accesses and hide shared and device memory communication latencies, column values and indices of A in steps 1 to 4 are loaded in parallel.
- Values satisfying the condition in step 1 are determined in parallel and columns are matched to $\hat{A}$ format simultaneously in steps 2 and 3.
- In smaller matrices the $\hat{A}$ is stored on shared memory to increase the speed of the *QR* decomposition.
- For denser matrices with a large τ parameter, the arbitrary dimension of $\hat{A}$ and the limited size of shared memory requires storing $\hat{A}$ in global memory. Since the size of $\hat{A}$ multiplied by the number of columns exceeds the size of global memory for such problems, the computation of M is divided between multiple kernels allowing for memory reuse.
- Columns of the R matrix in the *QR* decomposition are generated in parallel and dot products are computed using parallel reduction techniques [8]. The computed columns m_i are also scattered to global memory in parallel.
- The *cudaFuncCachePreferShared* [6] option is used to maximize the amount of shared memory on the GPU.
- While the iterative solver can run in double precision, due to higher peak performances for single vs. double precision computations on the GPU, M is calculated in single precision.

This proposed method is tested on two FEM matrices [7] namely s3dkt3m2 and s3dkq4m2 (Table I). The percentage of time CW-SPAI ($\tau = 0.9$) spends in each step of the SPAI algorithm is shown in Fig. 1. Due to the large number of nonzeros in both matrices the $\hat{A}$ matrix requires to be stored on global memory, increasing device memory accesses and the execution time of steps 2 to 4 in the algorithm (Fig. 1). The relative speedup of the proposed method on NVIDIA GTX480 compared to SPAI 3.2 [5] CPU (Intel Core2 Quad 2.4GHZ) results using optimization flags is presented in Table I. Since the SPAI preconditioner is a better approximate of A^{-1} with larger τ parameters [5], results are presented for τ equal to 0.7 and 0.9. As shown speedups of up to 18 times compared to CPU results are achieved using CW-SPAI.

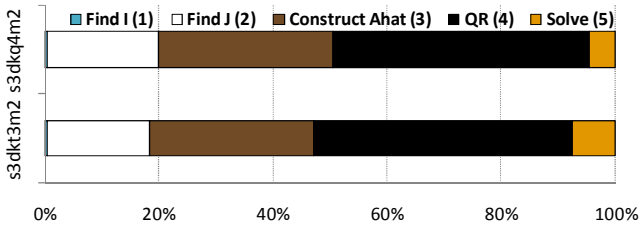

Fig. 1. Percentage of average execution time of the SPAI steps in CW-SPAI.

TABLE I
MATRIX PROPERTIES AND SPEEDUP (SU) OF GPU ACCELERATED CW-SPAI COMPARED TO SPAI 3.2 [5] CPU RESULTS

Matrix Name	nnz	nnz/col	SU ($\tau = 0.7$)	SU ($\tau = 0.9$)
s3dkt3m2	3,843,910	21	18	10
s3dkq4m2	4,911,340	27	9	8

The effects of using SPAI-ME preconditioners to enhance the convergence rate of iterative solvers are presented in Table II. Compared to a diagonal preconditioner, the generated SPAI preconditioner with τ equal to 0.9 and error tolerance of 1e-7 reduces the number of iterations in the BiCGSTAB iterative solver 7 and 10 times for s3dkq4m2 and s3dkt3m2 respectively. When the tolerance is decreased to 1e-8, using the produced SPAI preconditioner, s3dkt3m2 converges in 296 iterations while with a diagonal preconditioner the desired tolerance is not achieved within 10,000 iterations.

TABLE II
ITERATION COUNTS FOR BICGSTAB USING THE SPAI-ME PRECONDITIONER VS. DIAGONAL PRECONDITIONERS

Matrix Name	s3dkq4m2 tol=1e-7	s3dkq4m2 tol=1e-8	s3dkt3m2 tol=1e-7	s3dkt3m2 tol=1e-8
Diagonal	527	1074	1478	>10000
SPAI	73	132	135	296
SU	7	8	10	>33

In the long version of the paper, implementation details of the CW-SPAI algorithm will be presented and time consuming sections of the code (Fig. 1) will be further optimized to run faster on the GPUs. Methods of accelerating the BiCGSTAB iterative solver on graphic cards will also be introduced.

IV. REFERENCES

[1] P. Raghavan *et. al.*, "Parallel hybrid preconditioning: incomplete factorization with selective sparse approximate inversion," *SIAM Journal on Scientific Computing*, vol.32, issue.3, pp. 1323-1345, 2010
[2] P. Gonzalez *et. al.*, "Parallel sparse approximate preconditioners applied to the solution of BEM systems," *Engineering Analysis with Boundary Elements*, vol.28, Issue.9, pp. 1061-1068, 2004.
[3] T. Huckle *et. al.*, "An efficient parallel implementation of the MSPAI preconditioner," *Parallel Computing*, vol.36, Issue.6, pp. 273-284, 2010.
[4] G. Gravvanis, "High performance inverse preconditioning," *Archives of Computational Methods in Eng.*, vol.16, issue.1, pp. 77-108, 2009.
[5] SPAI3.2, www.computational.unibas.ch/software/spai/spaidoc.html
[6] NVIDIA CUDA, http://developer.nvidia.com/object/cuda.html.
[7] Matrix market, http://math.nist.gov/MatrixMarket/, 2007.
[8] M. Mehri Dehnavi *et. al.*, "Finite element sparse matrix vector multiplication on GPUs," *IEEE Trans. on Mag.*, vol.46, no.8, pp. 2982-2985, 2010.
[9] M. Mehri Dehnavi *et. al.*, "Enhancing the Performance of Conjugate Gradient Solvers on GPUs," *IEEE Trans. on Mag., to appear.*