Efficient Multicore Sparse Matrix-Vector Multiplication for FE Electromagnetics

David M. Fernández, Dennis Giannacopoulos, and Warren J. Gross

Department of Electrical and Computer Engineering, McGill University, Montreal QC H3A 2A7, Canada

Multicore systems are rapidly becoming a dominant industry trend for accelerating electromagnetics computations, driving researchers to address parallel programming paradigms early in application development. We present a new sparse representation and a two level partitioning scheme for efficient sparse matrix-vector multiplication on multicore systems, and show results for a set of finite element matrices that demonstrate its potential.

Index Terms—**Finite element (FE), multicore, parallel computation, sparse matrices, sparse matrix-vector multiplication (SMVM).**

I. Introduction

THE trend towards solving increasingly complex computational electromagnetics problems has relied, in part, on continual CPU improvements; however, technological limitations have dictated the need to explore new alternatives. Increasingly, industry leaders (e.g., Intel, AMD, IBM) are displacing traditional single core CPUs with multicore architectures to improve performance, thus creating new programming opportunities and challenges; finite-element (FE) practitioners must explicitly consider parallel algorithms and techniques in order to exploit the full potential of emerging multicore CPUs.

Sparse matrix-vector multiplication (SMVM) is a kernel at the core of many widely used iterative solvers in FE applications, such as the conjugate gradient (CG) method. In fact, SMVM can be a dominant cost in obtaining FE solutions, thus making it an important kernel for parallelization on multicore CPUs. The SMVM kernel's main objectives are to compute Ax with the nonzero values of A and reduce its storage requirements. However, traditional sparse matrix storage formats used for SMVM are not necessarily well suited for efficient parallel processing, limiting the use of hardware optimization techniques and requiring a great deal of instruction overhead for their implementation [1]. For example, one drawback that limits attainable performance is the irregular memory access patterns to the elements in the vector x. The purpose of this contribution is to introduce a new sparse format for FE matrices coupled with a two level data partitioning scheme, to overcome this bottleneck and improve the speedup possible with multicore-based SMVM implementations.

II. Multicore SMVM Challenges

The size of modern FE computations is continuously driven by the need for larger and more accurate simulations, thus the importance of accelerating dominant computing kernels such as SMVM. Multicore processors have become the industry mainstream CPU platform, providing significant performance benefits without depending on higher clock frequencies that have traditionally fueled single core processor performance.

The key challenges that arise when accelerating the SMVM kernel on multicore systems can be summarized in three central tasks (addressed in this paper): selection of clever data structures [2] or sparse formats, sparse matrix partitioning and distribution schemes across parallel cores, and use of explicit parallel algorithms. In particular, the data format used is key to facilitating the other tasks; however, this selection might prove daunting because of the variety in sparse formats, which range from simple and direct representations to elaborate compressed ones that exploit different matrix properties [3]–[5]. One of the most popular of these formats is the compressed sparse row (CSR, also called AIJ or YALE format). In CSR, three vectors represent the sparse matrix, the first stores the matrix nonzero values in row order; the second, stores the column indices of the nonzero values; and the third, contains the indices of the first vector elements that begin a new row (including an additional index with the total number of nonzeros plus one). Many efficient libraries used to solve linear systems implement this format or some variation of it, as is the case with PETSc [6]. Fig. 1 shows the CSR format as well as the new *pipeline-matched* sparse (PMS) format proposed.

Single instruction multiple data (SIMD) processing has become an important source of parallelism widely available in most modern multicore environments. To exploit these resources, additional challenges arise such as: the use of architecture-specific SIMD instructions, data partitioning on SIMD boundaries, and memory alignment requirements, yet many sparse formats might not consider these factors. Another important issue is the irregular-indirect access to the x vector in SMVM (see line 4 in Algorithm 1). For large matrices this translates into long data access times due mainly to cache misses in general purpose processors (GPP). Software controlled memory hierarchies (e.g., Cell processor [7]) overcome the cache effect but may require previous knowledge of data access patterns. Among other challenges, the SMVM kernel requires additional data transfers (nonzero row/column indices) making it bandwidth (I/O) limited, and more control instructions increasing the processor work load per floating point computation. Finally, explicit parallel programing is required

Manuscript received October 07, 2008. Current version published February 19, 2009. Corresponding author: D. Giannacopoulos (e-mail: dennis.giannacopoulos@mcgill.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2009.2012640

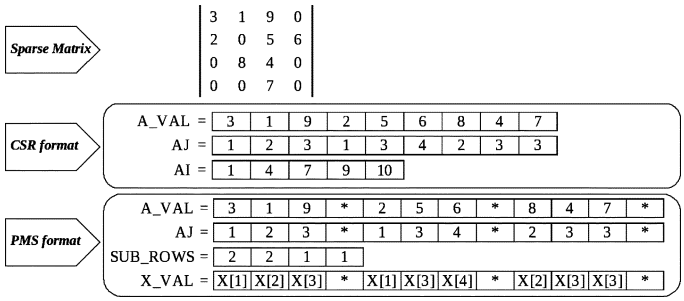

Fig. 1. CSR and PMS representations of an example sparse matrix. The PMS format is configured to contain two elements per sub-rows/vector. Star entries represent zero padding to match the two-way pipeline width.

to schedule tasks on the processing cores, a common challenge to parallel programming in distributed memory systems. The next section describes the new sparse format created as well as the partitioning technique used to address the three central tasks mentioned above.

Algorithm 1 Basic SMVM using CSR format.

1: **for** $i = 1$ to *number of rows* **do**
2: $b[i] \leftarrow 0.0$
3: **for** $j = AI[i]$ to $AI[i+1]$ **do**
4: $b[i] \leftarrow b[i] + A\_VAL[j] * x[AJ[j]]$
5: **end for**
6: **end for**

III. PMS Representation and Partitioning Technique

This section presents the two main contributions of this work, a new sparse matrix representation and a two-level partitioning scheme that facilitate data distribution, data processing, and expose SIMD parallelism to efficiently exploit the processing resources in multicore systems.

A. Pipeline-Matched Sparse Representation

The main goal of the new sparse format called PMS representation was to conform to the SIMD instruction set found in many modern multicore processors. The PMS format is configured using zero padding to match the SIMD pipeline-width in the target architecture (e.g., four single precision floating point values per SIMD register or four-way pipeline in the Cell processor), to efficiently exploit available processor parallelism. In the rare case where no SIMD units are available, PMS can be configured with a one-way SIMD size resulting in no zero padding. An additional goal, was to solve the irregular-indirect access to the x vector in the SMVM kernel and restrict the amount of data communication, since this kernel is bandwidth limited. This is achieved by including the elements of the x vector in the representation.

PMS is based on the compressed sparse row (CSR) format. It comprises four vectors (see Fig. 1) containing: 1) the nonzero elements of the A matrix, with zero padding (by rows) to match the number of SIMD pipeline-width of the target processor; 2) the column indices of the first array elements; 3) the number of pipeline-matched sub-rows per matrix row; and 4) the elements of x indexed by the second array. Once this representation is built, only the fourth vector need be modified to solve for different x vectors. The mapping of the x vector elements into the new format involves extra processing, but ultimately this work has to be done in the SMVM kernel regardless of the sparse format used. By doing this work in advance memory access patterns become regular (unit-stride access to the x vector elements is achieved), offering better locality and reduced cache misses on GPPs.

Compared to the CSR format, PMS requires the storage of an extra vector of size equal to the number of nonzeros in addition to the zeros used to match the pipeline-width of the target architecture. This extra memory usage yields benefits in terms of easier matrix and vector partitioning and subsequent data communication to the parallel computing cores as well as a regular computation. Partitions are easier to determine because of the vector boundaries defined by PMS and the proposed two-level partitioning scheme (explained in Section III-B). Also, because this format already contains the x vector there is no need to transfer it separately, in fact the amount of data transferred to the processing cores is similar to the amount required by the CSR format. Only three vectors need be transferred to compute the SMVM kernel using PMS A_VAL, SUB_ROWS, and X_VAL, whereas CSR requires an additional vector (the column indices). Regular computations are achieved by the layout of the x vector into the proposed format.

Thereafter, the *pipeline-matched* representation renders three important benefits: 1) it enables exploitation of the SIMD units within CPU cores (low-level parallelism); 2) it provides natural boundaries for data partitions when exploiting parallelism across CPU cores (high-level parallelism); and 3) it offers regular data access patterns which result in more efficient computing kernel. The proposed format can be thought of as a compressed vector storage of the sparse matrix, with vectors the size of the target architecture pipeline-width. While this format was designed within the scope of FE applications, it can be used to represent other sparse matrix types regardless of their sparsity pattern, density, symmetry, or target application. This new representation could also be used in non conventional multicore architectures or reconfigurable hardware providing similar benefits.

B. Two-level Partitioning Technique

A two-level partitioning scheme was designed to distribute data in a shared memory multicore architecture taking into account the limited memory space available in different types of parallel computing cores. The data partitioning scheme was developed to generate *coarse* (*first level* partitioning) and *fine* grained (*second level* partitioning) partitions on the sparse matrix A as shown in Fig. 2. The coarse grained partitions were used to schedule row-blocks across parallel cores, while the fine grained partitions were used to determine the number of matrix chunks to stream within each processing core.

The fine grained partitioning allowed us to cope with the limited memory in the parallel processing cores (cache for GPP cores or local-space for cores with software controlled memory hierarchies, e.g., the SIMD cores in the Cell processor called SPEs [7]) and can be viewed as the cache blocking techniques used to reduce the effect of cache misses in GPPs. This second partitioning is key to applying streaming techniques that enable

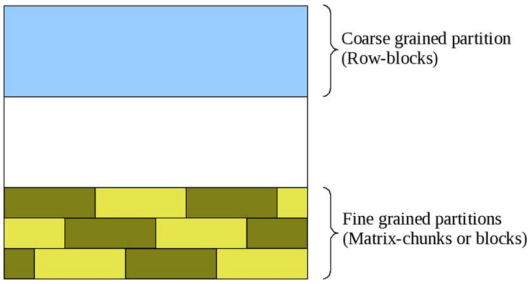

Fig. 2. Two-level partitioning scheme of a matrix. Coarse grained partitions generate row blocks, and fine grained partitions create smaller data sets to transfer in a block fashion.

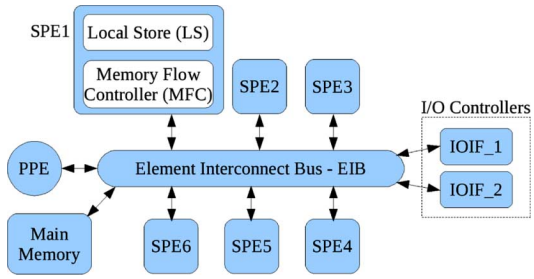

Fig. 3. Cell processor architectural overview.

overlapping communication with computations. As mentioned in Section III-A, there is no need to partition the x vector separately since it is already contained in the proposed format.

Overall, this partitioning scheme achieves load balancing under sparsity patterns that distribute nonzeros evenly among matrix rows, which is commonly the case in problems with fixed degrees of freedom that arise in FE computations. However, for very irregular sparsity patterns or when clusters of these multicore processors are considered a more sophisticated coarse grained partitioning scheme might become necessary to achieve load balancing and minimize communications between multicore chips. Important studies on sparse matrix partitioning based on graph and hypergraph partitioning with precise estimation of communication volume are presented in [8]–[11]. The study of these methods will be important when implementing efficient SMVM operations on massively parallel multicore clustered systems.

IV. Results

To examine the performance of the new PMS representation and partitioning scheme the SMVM kernel was implemented and tested on a Cell processor heterogeneous multicore processor installed with a 64-bit Fedora Core 6 Linux operating system. We also implemented a sequential SMVM kernel using the CSR matrix format in a latest generation Intel homogeneous multicore processor with a 64-bit Fedora Core 7 operating system, for validation and comparison purposes. The use of the Cell was motivated because it provides an efficient memory bandwidth usage, and it is capable of delivering a peak performance of over 200 GFLOPS for single precision floating point (SPFP) computations with a higher percentage of available peak performance (for kernels with regular memory access patterns) when compared to GPPs [12].

A. Testbed Description

The Cell system used was composed of one GPP called PPE, and 6 SIMD processors called SPEs clocked at 3.2 GHz with 256 MB Rambus extreme data rate (XDR) DRAM global memory as shown in Fig. 3. The PPE was used for administrative and control tasks while the SPE cores were used as the main computing resource. SPEs are high performance 128-bit SIMD cores with software controlled memory hierarchy, a four-way SPFP SIMD pipeline, and limited hardware support for branch prediction. They have a large register file (128–128 b registers), and a 256 kB on-core software managed memory called local-store (LS). A distinct difference of the SPEs from other processor is that transfers to/from LS and main system memory must be explicitly programmed by the user. The Cell optimized implementation was executed on the SPE cores using SPFP (since the four-way SIMD single precision computations are fully pipelined in this version of the Cell), referred to here as the Cell-SPE implementation. A reference implementation using the CSR matrix format was also done on the PPE, referred to as the Cell-PPE implementation. The Cell implementations were done using the Cell SDK 3.0. The GPP platform used was an Intel Core2 Quad 2.40 GHz CPU with 4 MB of L2 cache per core-pair and 4 GB of global DRAM, and was compiled using GCC version 4.1.2.

B. Testbed Optimizations

The Intel and Cell-PPE implementations were optimized using the "-O2" compiler flag which provided the best performance results for all the conducted tests. The Cell-SPE accelerated version of the SMVM kernel was implemented using specific vector intrinsics [7] for the SPEs. This mainly involved intrinsics to control the asynchronous DMA transfers between SPE LS and the Cell main memory; and specific intrinsics to perform SIMD multiplications and additions on 4 SPFP elements simultaneously, thus capitalizing the four-way SIMD pipeline in the SPEs. A multibuffering technique was used to transfer the A matrix and x vector data (fine grained partitions) from main memory to SPEs LS, while overlapping communication with the computations. The PMS format was configured to generate four-element SPFP or sub-rows (per matrix row) to match the SPE pipeline width. To minimize the overhead effect of the SMVM's control statements in the SPEs, simple conditional instructions were substituted with bit-selection intrinsics, thus eliminating the corresponding branch occurrences in the code. When ever this was not possible, branch hint instructions were used to reduce the impact of misprediction latency.

C. Results Discussion

The results for different FE matrices with varying sizes and different sparsity patterns taken from the Matrix Market database [13] are shown in Fig. 4. These results demonstrate that as the number of nonzeros grow the performance of the SPE implementation outperforms that of the Intel CPU. This type of performance is in agreement with the results presented in [14]. For the largest test case ($\sim$2.5 million nonzeros) the Cell-SPE SMVM kernel was 2 times faster than the Intel CPU and nearly 14 times faster than the Cell-PPE. When scaling the number

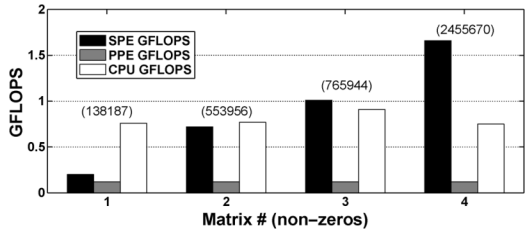

Fig. 4. Performance results in GFLOPS for six SPEs, the PPE, and the Intel CPU.

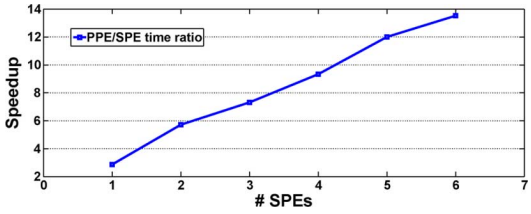

Fig. 5. Speedup achieved by scaling from 1 to 6 Cell-SPEs.

TABLE I
SMVM GFLOPS AND SU RESULTS COMPARISON FOR SIX SPES WITH RESPECT TO THE PPE AND THE CPU

Matrix Name(#)	cavity26(1)	e40r5000(2)	fidapm37(3)	s3dkq4m2(4)
non-zeros	138187	553956	765944	4 2455670
SPE GFLOPS	0.19	0.69	1.00	1.56
PPE GFLOPS	0.12	0.12	0.12	0.12
PPE/SPE SU	1.70	5.92	8.18	13.53
CPU GFLOPS	0.76	0.77	0.91	0.75
CPU/SPE SU	0.26	0.90	1.10	2.06

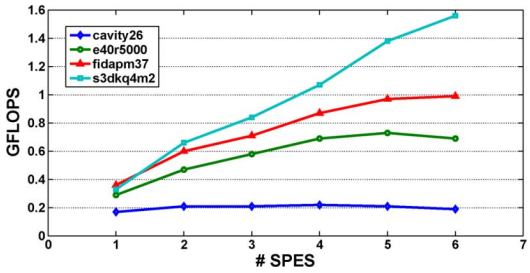

Fig. 6. Performance scaling GFLOPS with number of Cell-SPEs.

of SPEs with the new representation we obtained a superlinear speedup (SU) compared to the Cell-PPE version for the larger test matrix as shown in Fig. 5.

Average performance results obtained for each test matrix on the different platforms are presented in Table I. Fig. 6 demonstrates that the proposed format and partitioning scheme show linear scaling when incrementing the number of SPEs; on the other hand, for small matrices and as the number of SPEs increases communication and control overheads hinder performance. Nonetheless, this is the expected behavior of parallel systems, SU can only be achieved when sufficient work-load exists to overcome communication and control overheads.

V. CONCLUSION

Motivated by the need to solve bigger and more complex FE problems and meeting the trend towards the multicore parallel processing paradigm we proposed a new sparse format coupled with the two-level partitioning scheme. Linear scaling with the number of parallel cores and matrix nonzeros has been demonstrated and up to 14 times speedup for the best testcase. Efficient solutions to the data partitioning, load distribution, and SIMD processing challenges on multicore systems are the most significant problems solved by this work. The next step towards building a complete FE solution is to integrate this kernel into a CG solver. Ultimately, this work represents an essential building block towards achieving efficient parallel CG algorithms in modern multicore CPUs.

ACKNOWLEDGMENT

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Y. El-Kurdi *et al.*, "Hardware acceleration for finite element electromagnetics: Efficient sparse matrix floating-point computations with FPGAs," *IEEE Trans. Magn.*, vol. 43, no. 4, pp. 1525–1528, Apr. 2007.

[2] H. Magnin and J. L. Coulomb, "A parallel and vectorial implementation of basic linear algebra subroutines in iterative solving of large sparse linear systems of equations," *IEEE Trans. Magn.*, vol. 25, no. 4, pp. 2895–2897, Jul. 1989.

[3] S. Yousef, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: SIAM, 2003, p. 528.

[4] B. Richard *et al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. Philadelphia, PA: SIAM, 1994, p. 117.

[5] P. Fernandes *et al.*, "An evaluation of speedup in conjugate gradient routines with a mathematical vector library," *IEEE Trans. Magn.*, vol. 27, no. 5, pp. 4214–4216, Sep. 1991.

[6] B. Satish *et al.*, PETSc Users Manual Revision 2.3.3, Argonne National Laboratory, IL, May 2007, p. 190. [Online]. Available: http://www.mcs.anl.gov/petsc

[7] *Cell Broadband Engine Programming Handbook*. New York: IBM, Apr. 2007, p. 877, Version 1.1.

[8] B. Hendrickson *et al.*, "An efficient parallel algorithm for matrix-vector multiplication," *Int. J. High Speed Comput.*, vol. 7, no. 1, pp. 73–88, 1995.

[9] B. Hendrickson and T. G. Kolda, "Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel computation," *SIAM J. Scientific Comput.*, vol. 21, no. 6, pp. 2048–2072, 2000.

[10] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proc. IEEE 20th Int. Parallel Distrib. Process. Symp.*, 2006, p. 20.

[11] Ü. Catalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse matrix vector multiplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 7, pp. 673–693, Jul. 1999.

[12] S. Williams *et al.*, "The potential of the cell processor for scientific computing," in *Proc. 3rd ACM Int. Conf. Comput. Frontiers*, 2006, pp. 9–20.

[13] "National Institute of Standards and Technology,", Gaithersburg, MD, "Matrix market," Mar. 2007 [Online]. Available: http://math.nist.gov/MatrixMarket/

[14] S. Williams *et al.*, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *Proc. ACM/IEEE Conf. Supercomput.*, 2007, p. 12.