Hardware Acceleration for Finite-Element Electromagnetics: Efficient Sparse Matrix Floating-Point Computations With FPGAs

Yousef El-Kurdi, Dennis Giannacopoulos, and Warren J. Gross

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

Custom hardware acceleration of electromagnetics computation leverages favorable industry trends, which indicate reconfigurable hardware devices such as field-programmable gate arrays (FPGAs) may soon outperform general-purpose CPUs. We present a new striping method for efficient sparse matrix-vector multiplication implemented in a deeply pipelined FPGA design. The effectiveness of the new method is illustrated for a representative set of finite-element matrices computed on our highly scalable and fully pipelined FPGA-based implementation.

Index Terms—**Conjugate gradient (CG) method, field-programmable gate array (FPGA), finite element (FE), hardware acceleration, parallel computing, sparse matrix striping.**

I. Introduction

FUELED by continual CPU performance improvements, finite-element (FE) practitioners perpetually strive to simulate increasingly complex electromagnetic systems. Solution via serial processing on current personal computers can yield impractical run-times due to the large number of degrees of freedom involved. Various approaches, such as parallel processing, hold promise for overcoming this barrier. Relatively lower cost alternatives recently gaining attention include solution acceleration via implementation in custom hardware such as FPGAs [1]. However, to realize the full potential of such approaches, the underlying algorithms must be inherently parallelizable.

Sparse matrix-vector multiplication (SMVM) is a kernel for many iterative numerical techniques, such as the conjugate gradient (CG) method, used to solve large, sparse linear systems arising in FE formulations. In fact, SMVM can be a dominant cost associated with obtaining FE solutions, which if implemented in custom hardware may lead to significant run-time reductions. However, sparse matrix storage schemes effective for software-based implementations are not "regular" enough to allow for efficient parallel manipulation in FPGA-based designs. To overcome this, various so-called striping algorithms have been proposed.

The purpose of this contribution is to introduce a new striping scheme for FE matrices that improves the parallel speed-up of our FPGA-based fully pipelinable SMVM accelerator (SMVM pipeline). The design comprises a pipeline of eight processing elements (PEs). Each PE contains a cascade of deeply pipelined floating-point arithmetic units (FPUs). To increase processor efficiency and maintain the peak floating-point performance of the SMVM pipeline, the sparse matrix should be represented in the least number of stripes possible.

Digital Object Identifier 10.1109/TMAG.2007.892459

The matrix-vector multiplication operation is defined by $Y = A \times X$, where A is the system's matrix and is usually square with dimension $(N \times N)$, while X and Y are dense vectors with dimensions $(N \times 1)$. The system's matrix A resulting from FE applications is large and sparse which causes typical CPU implementations of SMVM to suffer from low hardware utilization in addition to memory access bottlenecks. On the other hand, FPGAs have shown to produce higher hardware sustained performance for SMVM computation [2]–[4]. In addition, FPGA reconfigurable systems offer larger memory bandwidth making them more suitable for large FE computations.

II. Previous Work

Sparse matrix striping in general tries to meet either one of two major objectives; first, to produce the least number of stripes that cover the sparse matrix, and second, to obtain an efficient utilization of the parallelism features of array processors. Different forms of striping schemes have been previously developed in [5]–[7]. These schemes are used to compute SMVM on 1-D linear array processors. In this paper, we will present these striping schemes and compare their effectiveness in utilizing the computational capacity of a fully pipelined SMVM implementation to the efficiency of our proposed striping scheme, *pipelinable stripes*. We will also show that the pipelinable stripes scheme obtains higher computational throughput when used with the SMVM pipeline.

III. SMVM Pipeline

The SMVM pipeline is a linear array processor network shown in Fig. 1. The SMVM pipeline is constructed from identical PEs each containing a cascade of deeply pipelined FPUs that consist of a multiplier and an adder. The links between PEs are constructed from FIFO queues.

The SMVM pipeline performs the overall SMVM computation in parallel by streaming the sparse matrix data along with the X and Y vectors through the processor array network. The matrix elements are grouped into specially ordered stripes and fed to each PE from the top, while the X and Y vectors are

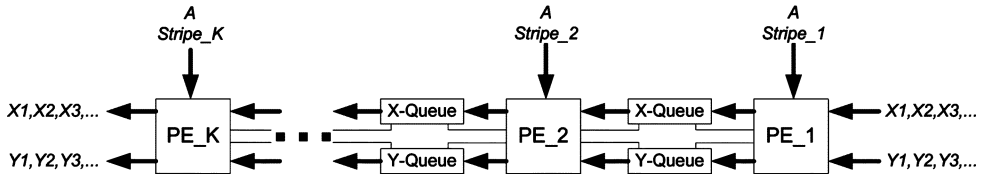

Fig. 1. SMVM pipeline is a fully pipelined 1-D linear array processor network.

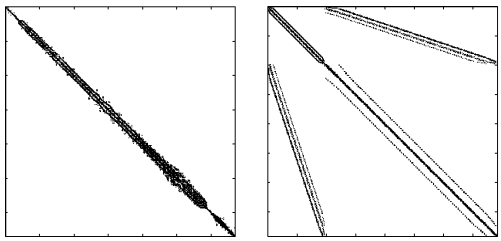

Fig. 2. Typical sparsity pattern for FE matrices [9].

fed into the pipeline horizontally from the same side. Each PE therefore, simultaneously computes a partial summation of

$$Y(i) \Leftarrow Y_{\text{prev}}(i) + X(j) \times A(i,j) \tag{1}$$

where $Y_{\text{prev}}(i)$ is the partially accumulated summation from a prior computation.

IV. Pipelinable Stripes

Striping is particularly suitable for FE matrices due to their special sparsity patterns. Melhem [8] establishes that FE matrices, when using specific mesh numbering techniques, can be covered by a number of stripes independent of the size of the problem. In other words, the number of stripes the FE matrices can have is bounded by a constant that is independent of the dimension of the matrix N. Since each PE is required to process a stripe, this implies that the computational resources needed by the SMVM pipeline does not indefinitely increase as the size of the FE matrix increases. For large FE matrices therefore, the SMVM pipeline scales well in terms of computational and memory resource requirements. Fig. 2 shows the typical FE matrix sparsity pattern for two example matrices.

A. Increasing-Order Stripes

To ensure correct operation of the SMVM pipeline, the stripe elements have to be ordered in an increasing order with respect to both row and column indices. Increasing-order stripes can fall into four distinct categories which are increasing order (IO), strictly increasing order (SIO), strict-columns increasing order (SCIO), and strict-rows increasing order (SRIO). To formulate these categories, we start by defining a stripe S_k as an ordered set of elements taken from the sparse matrix A as follows:

$$S_k = \{A(i_k, j_k)\} \quad \text{where } i_k, j_k \subseteq \{1, 2, \ldots, N\}.$$

TABLE I
Stripe Categories

Indices Relationship	Max Length	Stripe Category
$i_{k-1} \leq i_k, j_{k-1} \leq j_k$	$2N$	IO
$i_{k-1} < i_k, j_{k-1} < j_k$	N	SIO
$i_{k-1} < i_k, j_{k-1} \leq j_k$	N	SRIO
$i_{k-1} \leq i_k, j_{k-1} < j_k$	N	SCIO

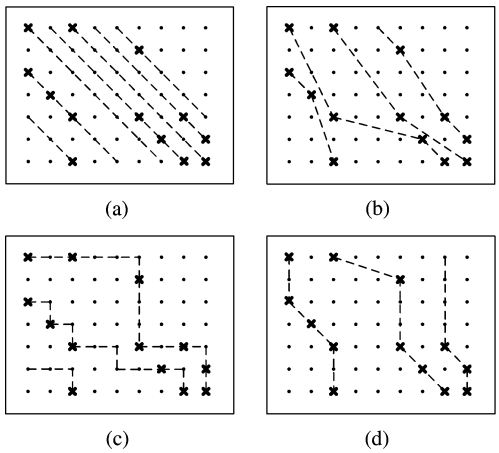

Fig. 3. Stripe formations. Pipelinable (SRIO) stripes generate the optimal number of stripes required for fully pipelinable hardware implementations. (a) Straight-diagonal stripes (seven SIO stripes). (b) Jagged-diagonal stripes (four SIO stripes). (c) Staircase stripes (three IO stripes). (d) Pipelinable stripes (three SRIO stripes).

Table I outlines the different striping categories.

Fig. 3 shows a basic example of the four increasing-order stripe formations. The pipelinable stripes belong to the SRIO striping category which is expected to produce a lower number of stripes than the straight-diagonal [5] or the jagged-diagonal striping formations [6]. This is due to the fact that the condition required for forming SRIO is less restrictive than the condition required for forming jagged-diagonals or straight-diagonals. Therefore, each SRIO stripe can contain on average more elements than the jagged-diagonal or straight-diagonal stripe which reduces the overall SRIO stripe count.

In contrast, staircase striping may produce a lower number of stripes; however, they prevent the use of pipelined FPUs. This is because the IO or SCIO stripe may contain more than one element from the same row which creates a feedback line around the FPUs in the PE impeding the overall computational throughput of the SMVM pipeline. Amongst the previously mentioned striping formations, the pipelinable stripe formation

generates the optimal number of stripes suitable for fully pipelined, parallel, and efficient hardware implementations.

B. Stripe Ordering

Every two consecutive PEs in the pipeline must receive stripes ordered in a higher to lower relationship, so as to reduce the required SMVM pipeline queue sizes. Also ordered stripes do not cross which avoids data conflicts and improves the concurrency and parallelism of computations. The higher order relationship $(\geq_h)$ is defined as follows.

Definition 1: Given stripes S_k and S_p, S_p is said to be higher than S_k $(S_p \geq_h S_k)$, if $(i_p \geq i_k) \Longrightarrow (j_p \geq j_k), \forall i_k, j_k, i_p, j_p$.

C. SMVM Pipeline Queue Sizes

In order to ensure correct SMVM pipeline operation, it is necessary to determine the minimum queue sizes required for a given striping formation. To facilitate this, we start by defining the largest horizontal separation (LHS) value between two ordered stripes as follows.

Definition 2: Given stripes S_k and S_p, if $(S_p \geq_h S_k)$, then the largest horizontal separation (LHS) is computed by

$$\mathrm{LHS}(S_k, S_p) = \max\{\min_{i_p \geq i_k} (j_p - j_k - 1) \quad \forall i_k, j_k, i_p, j_p\}.$$

The minimum X-queue depth (X_{QD}) between any two PEs is determined by

$$X_{\mathrm{QD}}(S_k, S_p) \geq \mathrm{LHS}(S_k, S_p) + c \quad \text{when } S_k \leq_h S_p$$

where c is a small integer constant reflecting the latency response of the queue status signals. As for the queues on the Y stream, only small Y-queues are necessary to facilitate the implementation of deeply pipelined FPUs.

D. Pipelinable Stripes Algorithm

We present two basic algorithms that produce SRIO stripes. The first algorithm is called the bottom-up striping (BUS) algorithm which forms stripes by taking the element with the minimum column index that satisfies the SRIO property, shown in Table I, from each row starting from the bottom row and moving up toward the top row. The second algorithm is called the top-down striping (TDS) algorithm which forms stripes by taking the element with the maximum column index that satisfies the SRIO property from each row starting from the top row and moving toward the bottom row. Fig. 4(a) and (b) demonstates the behavior of both the BUS and TDS algorithms.

In order to ensure correct operation of the SMVM pipeline implemented with limited queue sizes, the BUS and TDS algorithms can easily be adapted to generate banded stripes. Banded striping can ensure an upper bound on LHS between every two adjacent stripes thus limiting the X-queue size requirement between the corresponding PEs. This can be done by dissecting the matrix along cut-lines. Since we are dealing with FE matrices, which have mostly diagonal sparsity structure, diagonal cut-lines are more appropriate. Fig. 4(c) and (d) demonstates a graphical representation of the banded-BUS (BBUS) and banded-TDS (BTDS) algorithms. The stripes generated by our algorithms are $(\geq_h)$ ordered.

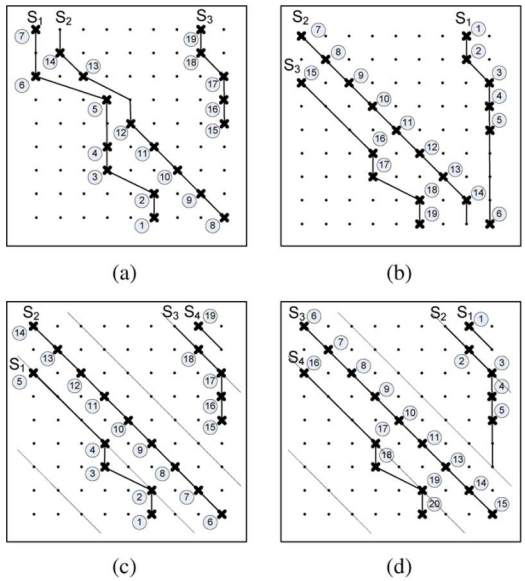

Fig. 4. BUS, TDS, BBUS, and BTDS stripe formations. Even though BUS and TDS stripes may differ, the number of stripes formed by both algorithms are identical. (a) BUS algorithm. (b) TDS algorithm. (c) BBUS algorithm. (d) BTDS algorithm.

The BBUS and BTDS algorithms are generalized versions of the BUS and TDS algorithms. For example, the banded versions can produce striping similar to the nonbanded versions if the band size is chosen to be greater than $2N+1$. Also, the banded bottom-up algorithm typically produces more stripes than the nonbanded one. But, since the band size used is large enough (current FPGA chips provide abundant internal SRAM memory to form long queues), the increase in the number of stripes when the banded algorithms are used will not be significant.

V. Performance Analysis

We can define an utilization factor $\mathcal{U} \in [0,1]$ for a given matrix. The utilization factor indicates how much of the peak SMVM pipeline performance can be achieved. Given that the striping scheme is either SIO or SRIO, $\mathcal{U}$ is computed as follows:

$$\mathcal{U} \geq \frac{\mathcal{D}}{N + \mathrm{LHS}} \tag{2}$$

where $\mathcal{D}$ is the stripe density which is found by

$$\mathcal{D} = \frac{\text{total number of nonzeros}}{\text{total number of stripes}}. \tag{3}$$

In other words, $\mathcal{D}$ describes the average number of nonzero elements per stripe. We can see from (3) and (2) that the utilization increases with increasing stripe density which is achieved by reducing the number of stripes representing the matrix.

VI. Results

The SMVM pipeline prototype is implemented with eight PEs and placed on one of the four Stratix FPGAs of the TM4 reconfigurable system [10]. The fully functional eight-PE prototype utilized only 30% of the logical resources and 40% of the internal memory resources (queues). The size of the FE matrix supported by the SMVM pipeline is only limited by the available

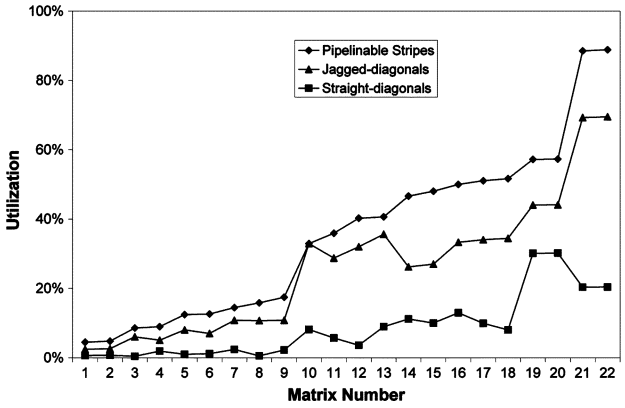

Fig. 5. Utilization results of the three striping schemes. Pipelinable stripes produce the highest utilization factors.

external memory resources for each FPGA in the TM4. This scalability advantage is the primary result of the stream-through design approach of the SMVM pipeline.

The SMVM pipeline peak performance is 1.76 Giga-FLOPS, and its maximum sustained performance is 86% of the peak performance (1.51 Giga-FLOPS), for our set of test FE matrices. In comparison with software implementations, Pentium 4 and Itanium 2 platforms obtained higher peak performance of 3.0 and 3.6 Giga-FLOPS respectively; however, their actual sustained performance was 14% (0.42 Giga-FLOPS) and 33% (1.18 Giga-FLOPS) of peak, which is lower than the SMVM pipeline's sustained performance [4]. In addition, the TM4 can more than quadruple the overall peak and sustained performance of the SMVM pipeline, when the four Stratix FPGAs are used for SMVM computation using our matrix partitioning scheme described below. The current SMVM pipeline performs the standard IEEE single-precision floating-point computations; however, if double-precision is required, the FPUs can be replaced with ones supporting double-precision arithmetic. Further details on the hardware implementation including the implications of limited external memory bandwidth on computational throughput and performance comparison with other computing platforms are provided in [2].

Fig. 5 shows the increased utilization factors as a result of using our pipelinable striping scheme over other striping schemes. This gain in utilization results in higher sustained computational performance. As expected, the utilization graphs of an extensive set of test matrices [9] striped using the three striping schemes indicate that the utilization as a result of SRIO striping is always equal to or higher than the jagged-diagonal utilization which, in turn is higher than the straight-diagonal utilization. This is due to the fact that the SRIO striping produced a lower stripe count than in the jagged-diagonal and the straight-diagonal schemes.

The number of PEs that can be implemented in the SMVM pipeline is not limited by the number of stripes the matrix can form; in fact, further scaling of the number of PEs can be obtained by horizontally partitioning the matrix and processing each partition in parallel by an independent SMVM pipeline. The partitioning scheme is especially favorable for FPGA reconfigurable systems that contain more than one FPGA chip

TABLE II
MATRIX PARTITIONING EFFECT ON UTILIZATION

Matrix Number	N	Non-zeros Count	Non-Partitioned Utilization	Partitioned Utilization
6	16614	1.09E06	12.66%	29.06%
12	13668	183394	40.26%	46.51%

such as the TM4. Unlike CPU-based implementations, communication overhead is not a limiting factor to the scalability of the SMVM pipeline. By virtue of the local interconnects between the PEs in the SMVM pipeline, scalability is only limited by the available FPGA resources. Table II shows the partitioning effect on utilization factors for two selected matrices that had low initial utilization results. The utilization due to partitioning can either increase or remain the same if the number of rows partitioned is large compared to the maximum LHS in the partition. This is due to the fact that the number of stripes within each partition will either be less than or equal to the number of stripes for the unpartitioned matrix.

VII. CONCLUSION

The highly scalable SMVM pipeline architecture which can obtain high-performance results for SMVM computation of very large sparse FE matrices was demonstrated. We developed our own pipelinable striping scheme, improving the overall utilization of our hardware design. In addition to providing an efficient and scalable utilization of the logical resources of the FPGA, the stream-through architecture inherently facilitates an efficient iterative implementation of the SMVM computation as required by iterative solvers such as the CG method.

REFERENCES

[1] H. Kawaguchi, K. Takahara, and D. Yamauchi, "Design study of ultra-high-speed microwave simulator engine," *IEEE Trans. Magn.*, vol. 38, no. 2, pp. 689–692, Mar. 2002.

[2] Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Sparse matrix vector multiplication for finite element method matrices on FPGAs," in *Proc. IEEE Symp. Field Programmable Custom Computing Machines*, 2006, pp. 293–294.

[3] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in *Proc. 2005 ACM/SIGDA 13th Int. Symp. on Field-Programmable Gate Arrays*, 2005, pp. 63–74.

[4] M. DeLorimier and A. DeHon, "Floating-point sparse matrix-vector multiply for FPGAs," in *Proc. 2005 ACM/SIGDA 13th Int. Symp. Field-Programmable Gate Arrays*, 2005, pp. 75–85.

[5] S. Y. Kung, *VLSI Array Processors*, ser. Information and System Science, T. Kailath, Ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

[6] R. Melhem, "Parallel solution of linear systems with striped sparse matrices," *Parallel Comput.*, vol. 6, no. 2, pp. 165–184, Feb. 1988.

[7] L. S. Heath, S. V. Pemmaraju, and C. J. Ribbens, "Processor-efficient sparse matrix-vector multiplication," *Compute. Math. Appl.*, vol. 48, pp. 589–608, 2004.

[8] R. Melhem, "Determination of stripe structure for finite element matrices," *SIAM J. Numer. Anal.*, vol. 24, no. 6, pp. 1419–1433, Dec. 1987.

[9] Matrix Market, Jun. 2004 [Online]. Available: http://math.nist.gov/MatrixMarket/

[10] The Transmogrifier-4 (TM4) Project, 2005 [Online]. Available: http://www.eecg.utoronto.ca/~tm4/

Manuscript received May 1, 2006 (e-mail: dennis.giannacopoulos@mcgill.ca).