Volume 5, Issue 4 SEP 2016

COMMUNICATION OPTIMIZATION OF SPARSE MATRIX-VECTOR MULTIPLY ON GPUS AND FPGAS

Panikara Raju[1] rajupanikara@gmail.com[1]

L.Nishanthini[2] nishanthini@gmail.com[2]

[1]PG Scholar, Dept of ECE, Kommuri Pratap Reddy Institute of Technology, Ghatkesar, RangaReddy, Telangana, India.

[2]Assistant Professor, Dept of ECE, Kommuri Pratap Reddy Institute of Technology, Ghatkesar, RangaReddy,Telangana, India.

Abstract: Sparse matrix-vector multiplication (SMVM) is a crucial primitive used in a variety of scientific and commercial applications. Despite having significant parallelism, SMVM is a challenging kernel to optimize due to its irregular memory access characteristics. Numerous studies have proposed the use of FPGAs to accelerate SMVM implementations. However, most prior approaches focus on parallelizing multiply accumulate operations within a single row of the matrix (which limits parallelism if rows are small) and/or make inefficient uses of the memory system when fetching matrix and vector elements. In this paper, we introduce an FPGA-optimized SMVM architecture and a novel sparse matrix encoding that explicitly exposes parallelism across rows, while keeping the hardware complexity and on-chip memory usage low. This system compares favorably with prior FPGA SMVM implementations. For the over 700 University of Florida sparse matrices we evaluated, it also performs within about two thirds of CPU SMVM performance on average, even though it has 2.4x lower DRAM memory bandwidth, and within almost one third of GPU SVMV performance on average, even at 9x lower memory bandwidth. Additionally, it consumes only 25W, for power efficiencies 2.6x and 2.3x higher than CPU and GPU, respectively, based on maximum device power.

Keywords-sparse matrix vector multiplication, FPGA, accelerator, SPMV, SMVM, reconfigurable computing, HPC.

I. INTRODUCTION

Sparse matrix-vector multiplication (SMVM) has received significant attention due to its increasingly important application in scientific and commercial applications (e.g., computational fluid dynamics, computer vision, robotics, and structural engineering, among others). Although SMVM is a highly parallelizable algorithm, the irregular memory access patterns of real-world sparse matrices often restrict realizable parallelism. To address this problem, numerous studies have introduced specialized SMVM implementations for parallel microprocessors and graphics-processing units (GPUs).

Field-programmable gate arrays (FPGAs) are a compelling substrate for SMVM due to the availability of massive parallel resources (i.e., logic gates, on-chip memories) and a flexible interconnect to support fine-grained communication. Prior work has shown that FPGAs can perform similarly to GPUs [16], even with much lower peak memory bandwidth, and can exceed GPU performance with equivalent bandwidth [20]. Furthermore, for comparable or better performance, FPGAs consume a very small fraction of the GPU's power (e.g., 25W vs. 200W), which is a critical factor for supercomputers where energy costs can approach millions of dollars per month. FPGA performance and efficiency has been typically obtained by efficiently parallelizing multiply-accumulate operations within a single row of the matrix, while also leveraging FPGA specialized matrix encodings and accumulator architectures. In this paper, we introduce a novel FPGA accelerator for SMVM that addresses two key bottlenecks of previous approaches: 1) restrictions on exploitable parallelism, and 2) limited on-chip block RAM.

While prior approaches have shown promising performance, they can be difficult to scale due to limits on exploitable parallelism. Specifically, early works on accelerating SMVM focused mostly on exploiting parallelism within a single matrix row. For example, for an accelerator with 32 multipliers, if a given row of the matrix has less than 32 unprocessed nonzero values, the remaining multipliers will be wasted due to zero padding [21]. It is possible to begin processing the next row instead of using zero padding, but supporting an arbitrary number of rows with an arbitrary number of elements increases complexity significantly, limiting the clock rate. Ideally, an accelerator should be capable of processing elements from multiple rows of the matrix to maximize parallelism. Prior works implement support for dynamic scheduling across rows but have not demonstrated scalable performance or efficient utilization of memory bandwidth [13]. Furthermore, most prior works assumed Compressed Sparse Row (CSR) matrix encodings that are cumbersome to fetch across multiple rows for parallel processing because the matrix is encoded in a sequential, row-major fashion. This requires an entire row to be read from memory and buffered on-chip before the first element of the subsequent row can be fetched.

Another bottleneck of previous SMVM approaches is the need for replicated storage of the input vector using on-chip FPGA block RAM. Previous approaches parallelize multiplications by streaming matrix values from external memory, while reading a vector value, with one vector replica implemented in FPGA block RAM per multiplier. Although a replicated memory architecture is well-suited for small vectors, it becomes a bottleneck for highly parallelized implementations using large vectors, if they are to be stored entirely on-chip. For example, for an FPGA board with ~10 GB/s of external memory bandwidth and a clock of 100 MHz, an SMVM accelerator can potentially fetch 100 bytes and execute ~25 32-bit floating-point multiplications every cycle.

For a vector of 100,000 elements, previous approaches would require 10 MB of block RAM, which exceeds even the largest FPGAs. Furthermore, this bottleneck is rapidly becoming more significant due to the exponential growth of SMVM problem sizes [3]. Even for smaller vectors, replicating the vector limits usage of block RAM for other common purposes (e.g., external transfer buffers, buffers between pipelined tasks).

In this paper, we introduce a new Compressed Interleaved Sparse Row (CISR) matrix encoding that enables simultaneous multiply-accumulate operations on multiple rows of the matrix without the need for complex schedulers or load-balancers. We also introduce a Banked Vector Buffer (BVB) that supplies vector data at high bandwidth without requiring expensive replication of data, i.e., a single buffer services multiple computations simultaneously. We evaluate our accelerator using over 700 matrices from the widely used University of

Florida Sparse Matrix Collection [3] and compare our results with other platforms. We show that the FPGA performs within about two thirds of CPU SMVM performance, even though it has 2.4x lower DRAM memory bandwidth, and within almost one third of GPU SVMV performance, even at 9x lower memory bandwidth .Additionally, it consumes only 25W, with power efficiencies 2.6x and 2.3x higher than the CPU and GPU, respectively, based on maximum device power.

II. Literature Survey

Due to its importance in scientific and engineering applications, a considerable amount of effort has been devoted to maximizing the performance of SMVM. The authors describe techniques that increase instruction-level parallelism on superscalar RISC processors. In, the balancing of distributed storage of nonzero elements among parallel processor arrays is investigated. More recently, several new optimizations for multicore platforms have been proposed in the research literature. The potential advantages of explicit multi core programming. The emergence of GPU as a powerful multicore architecture in recent years has made it an attractive target for SMVM optimization. Several different approaches of SMVM on GPU have been presented, including. Some online toolkits and libraries for GPUs are also available. It is important to note that these GPU SMVM implementations achieve consistent speedups only for matrices with regular sparsity patterns.

FPGAs provide a relatively low-cost platform for parallelizing algorithms at the operand-level granularity. Consequently, a diverse selection of FPGA-based accelerators can be found in the research literature. FPGAs have great potential in coping with the irregularity of SMVM due to their easily pipelined ability, inherent parallelism and configurable architecture. The design of SMVM in employs double-precision floating-point multipliers and adders, and performs multiple floating-point and I/O operations in parallel. The results show that their design achieves over 350MFLOPS for all test matrices when the memory bandwidth is 8 GB/s. However, the performance of their design is greatly affected by the padding overhead. The smaller the overhead, the higher the performance. The implementation partitions the set of dot products across multiple Processing Elements (PEs).

A matrix mapping algorithm is critical in reducing the computation latency while minimizing the inter-PE communication. Also, since only on-chip Block RAM (BRAM) is used to store the matrix, the matrix size is constrained by the BRAM. This preprocessing of the input matrix and vector would lead to potentially large overheads for very big matrices. As the performance speedup due to the use of FPGA technology is a function of the percentage of time spent in SMVM in the accelerated application, this preprocessing can quickly amortize the overall performance benefit. The design has pipeline stalls and depends less on the matrix structure compared with a software approach. An efficient SMVM computation of very large sparse Finite-Element matrices.

Title 1: An I/O Bandwidth-Sensitive Sparse Matrix-Vector Multiplication Engine on FPGAs.

Sparse matrix-vector multiplication (SMVM) is a fundamental core of many high-performance computing applications, including information retrieval, medical imaging, and economic modeling. While the use of reconfigurable computing technology in a high-performance computing environment has shown recent promise in accelerating a wide variety of scientific applications, existing SMVM architectures on FPGA hardware have been limited in that they require either numerous pipeline stalls during computation (due to zero padding) or excessive input preprocessing during run-time. For large-scale sparse matrix scenarios, both of these shortcomings can result in unacceptable performance overheads, limiting the overall value of using FPGAs in a high-performance computing environment. In this paper, we present a scalable and efficient FPGA-based SMVM architecture which can handle arbitrary matrix sparsity patterns without excessive preprocessing or zero padding and can be dynamically expanded based on the available I/O bandwidth. Our experimental results using a commercial FPGA-based acceleration system demonstrate that our reconfigurable SMVM engine is highly efficient, with benchmark-dependent speedups over an optimized software implementation that range from 3.5x to 6.5x in terms of computation time.

Title 2: A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication

Sparse matrix-vector multiplication (SMVM) is a crucial primitive used in a variety of scientific and commercial applications. Despite having significant parallelism, SMVM is a challenging kernel to optimize due to its irregular memory access characteristics. Numerous studies have proposed the use of FPGAs to accelerate SMVM implementations. However, most prior approaches focus on parallelizing multiply accumulate operations within a single row of the matrix (which limits parallelism if rows are small) and/or make inefficient uses of the memory system when fetching matrix and vector elements. In this paper, we introduce an FPGA-optimized SMVM architecture and a novel sparse matrix encoding that explicitly exposes parallelism across rows, while keeping the hardware complexity and on-chip memory usage low. This system compares favorably with prior FPGA SMVM implementations. For the over 700 University of Florida sparse matrices we evaluated, it also performs within about two thirds of CPU SMVM performance on average, even though it has 2.4x lower DRAM memory bandwidth, and within almost one third of GPU SVMV performance on average, even at 9x lower memory bandwidth. Additionally, it consumes only 25W, for power efficiencies 2.6x and 2.3x higher than CPU and GPU, respectively, based on maximum device power.

Title 3: Communication Optimization in Iterative Numerical Algorithms: An Algorithm-Architecture Interaction

Trading communication with redundant computation can increase the silicon efficiency of common hardware accelerators like FPGA and GPU in accelerating sparse iterative numerical algorithms. While iterative numerical algorithms are extensively used in solving large-scale sparse linear system of equations and Eigen value problems, they are challenging to accelerate as they spend most of their time in communication-bound operations, like sparse matrix-vector multiply (SpMV) and vector-vector operations. Communication is used in a general sense to mean moving the matrix and the vectors within the custom memory hierarchy of the FPGA and between processors in the GPU; the cost of which is much higher than performing the actual computation due to technological reasons. Additionally , the dependency between the operations Hinders overlapping computation with communication. As a result, although GPU and FPGA are offering large peak floating-point performance, their sustained performance is nonetheless very low due to high communication costs leading to poor silicon efficiency.

We provide a systematic study to minimize the communication cost thereby increase the silicon efficiency. For small-to-medium datasets, we exploit large on-chip memory of the FPGA to load the matrix only once and then use explicit blocking to perform all iterations at the communication cost of a single iteration. F or large sparse datasets, it is now a well-known idea to unroll k iterations using a matrix powers kernel which replaces SpMV and two additional kernels, TSQR and BGS, which replace vector-vector operations.

While this approach can provide a Θ(k) reduction in the communication cost, the extent of the unrolling depends on the growth in redundant computation, the underlying architecture and the memory model. In this work, we show how to select the unroll factor k in an architecture-agnostic manner to provide communication-computation tradeoff on FPGA and GPU. T o this end, we exploit inverse-memory hierarchy of the GPUs to map matrix power kernel and present a new algorithm for the FPGAs which matches with their strength to reduce redundant computation to allow large k and hence higher speedups. We provide predictive models of the matrix powers kernel to understand the communication-computation tradeoff on GPU and FPGA.

We highlight extremely low efficiency of the GPU in TSQR due to off-chip sharing of data across different building blocks and show how we can use on-chip memory of the FPGA to eliminate this off-chip access and hence achieve better efficiency. Finally, we demonstrate how to compose all the kernels by using a unified architecture and exploit on-chip memory of the FPGA to share data across these kernels. Using the Lanczos Iteration as a case study to solve symmetric extremalei gen value problem, we show that the efficiency of FPGAs can be increased from 1.8% to 38% for small to-medium scale dense matrices whereas up to 7.8% for large-scale structured banded matrices. We show that although GPU shows better efficiency for certain kernels like the matrix powers kernel, the overall efficiency is even lower due to increase in communication cost while sharing data across different kernels through off-chip memory. As the Lanczos Iteration is at the heart of all modern iterative numerical algorithms, our results are applicable to a broad class of iterative numerical algorithms

III. Proposed System

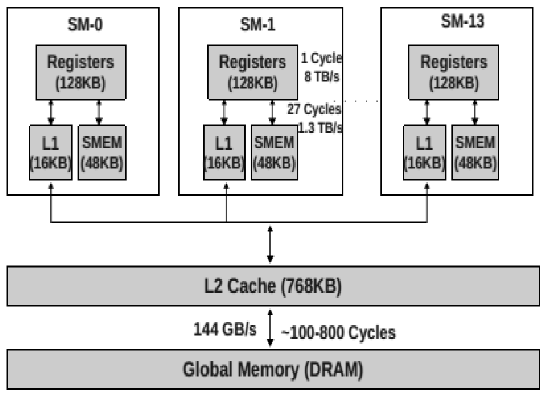


Fig 1 GPU Architecture

A simplified architectural description of the GPU is shown in Figure 3 highlighting memory hierarchy as well as capacity, bandwidth and latency of each memory. The GPU comprises 14 streaming multiprocessors (SMs) each operating at 1.15 GHz. Each SM has 32 floating-point cores capable of performing 1 single precision flop/cycle reaching a peak throughput of 1.03 TFLOPs for single-precision and 515 GFLOPs for double-precision. Tasks are scheduled on GPU as thread blocks. Each thread block can run independently on an SM without any communication with other SMs during a single parallel task.

Matrix Powers Kernel on a GPU

While mapping the matrix powers kernel on a GPU, we want to answer two important questions 1) How to optimally utilize the current GPU memory subsystem and pick the algorithmic parameter k to achieve the desired performance for a particular architecture.

2) How can current GPU architecture be changed to enhance the performance of iterative solvers we first present the current GF100 GPU architecture and then discuss different optimization techniques that lead to high throughput. We then present an analytical model to predict and understand the performance of the matrix powers kernel on any GPU device (model parameters are obtained using micro-benchmarks). We use the same model to select k for current GPU architectures and also make architectural projections to obtain a desired performance with future devices.

GPU Architecture

We select the Nvidia GF100 variant C2050 GPU, which is intended for high-performance numerical computing. A simplified architectural description of the GPU is high lighting memory hierarchy as well as capacity, bandwidth and latency of each memory. The GPU comprises 14 streaming multiprocessors (SMs) each operating at 1.15 GHz. Each SM has 32 floating-point cores capable of performing 1 single-precision flop/cycle reaching a peak throughput of 1.03 TFLOPs for single precision and 515 GFLOPs for double-precision. Tasks are scheduled on GPU as thread blocks. Each thread block can run independently on an SM without any communication with other SMs during a single parallel task, e.g. each SM can compute k SpMVs for a single block (ii) forq¼2.

Partitioning Strategy-One Partition Per Thread Block

Each of the N_q blocks within the matrix powers kernel is mapped to a thread block and all these thread blocks are computed independently in parallel and in any order. The size of each block is b R_b where b R is the number of rows in each block. Computation that helps in avoiding communication. We assign each vertex to a single thread, which performs serial reduction to compute the dot product of the row with b components of vector x ðiÞ. If NT denotes the number of threads, we can represent partition size b Rand the total number of partitions as:

$$b_R = N_T - k(b-1)$$

$$N_q = \left\lceil \frac{n + b_R - 1}{b_R} \right\rceil .$$

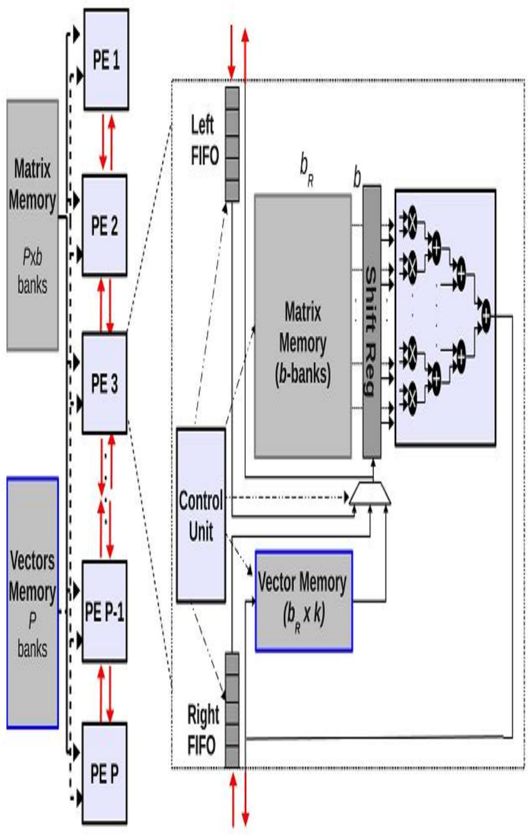


Fig 2. FPGA data-path for the matrix powers kernel

Modeling Performance To understand the performance of matrix powers kernel on FPGA, we use the same Log P model which comprises both computation as well as communication cost as the GPU. The model is exact due to the highly predictive nature of FPGAs as a computing platform.We show the

Volume 5, Issue 4 SEP 2016

parameters of the model. As FPGA has relatively larger on-chip memory compared to GPU, we intend to store the k vectors on-chip to be utilized by subsequent modules in communication avoiding iterative solver.

There are three stages in the matrix powers kernel on FPGA: loading the block, computing the sub-blocks in parallel and an optional stage for storing the k vectors back to the off-chip memory if they do not fit on chip. The latency ðLqÞof a single block is the summation of the latency of these three stages.

$$L_q = (lA_{glb2local} + lx_{glb2local}) + k \times l_{compute} + k \times lx_{local2glb}$$

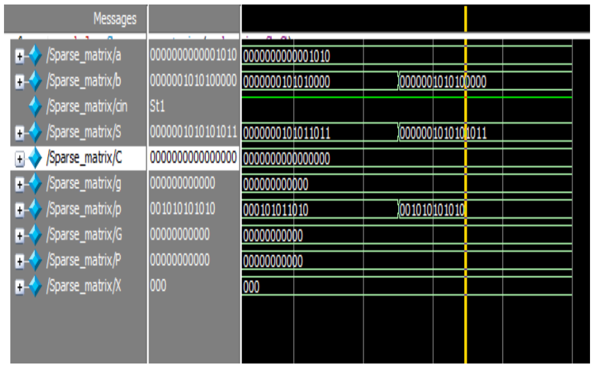
The overall latency L is then calculated by multiplying Lq with total number of blocks Nq. Sensitivity to Algorithmic Parameters select the algorithmic parameters for minimizing the runtime on GPU and FPGAs respectively. There are two algorithmic parameters, the partition size bR and the unroll factor k. While both parameters affect the surface to volume ratio but the impact of the partition size bR is marginal as compared to the unroll factor k. To show the performance variation with k, take aproblemsizeðn¼1MÞ and show both the communication and computation costs for band size equal that in GPU.

The optimal value of k decreases as we increase the band size whereas in case of FPGA, it shows opposite trend. After a certain value of k, both computation and communication costs dominate on GPU. The computation cost increases due to Þ growth in redundant operations and as a result the larger the band size, the smaller the value of k whereas the communication cost increases with increasing value of k due to shared memory accesses.

IV. Results

The written Verilog HDL Modules have successfully simulated and verified using Modelsim6.4b and synthesized using Xilinxise13.2.

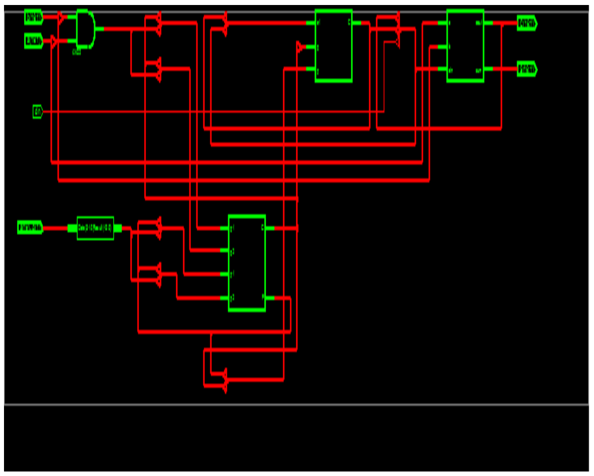
Simulation Results:



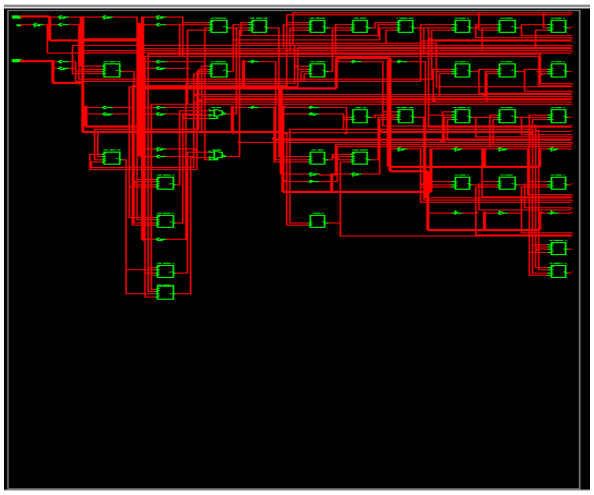
```
Messages
/Sparse_matrix/a     0000000000001010   0000000000001010
/Sparse_matrix/b     0000001010100000   0000000101010000   0000001010100000
/Sparse_matrix/cin   St1
/Sparse_matrix/S     0000001010101011   0000000101011011   0000001010101011
/Sparse_matrix/C     0000000000000000   0000000000000000
/Sparse_matrix/g     000000000000       000000000000
/Sparse_matrix/p     001010101010       000101011010       001010101010
/Sparse_matrix/G     00000000000        00000000000
/Sparse_matrix/P     00000000000        00000000000
/Sparse_matrix/X     000                000
```

Synthesis Results:

RTL Schematic:

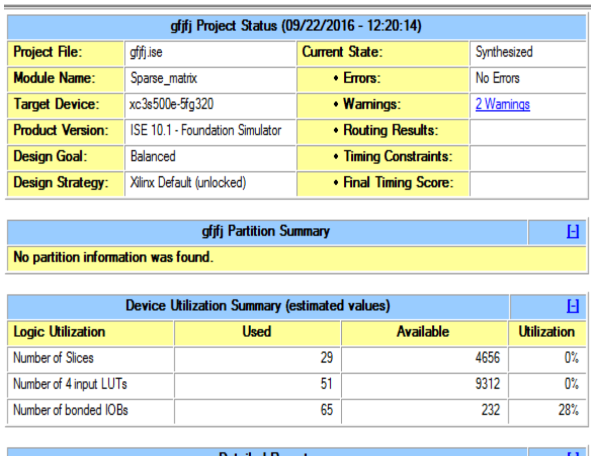


Technology Schematic:



Design summary:

Volume 5, Issue 4 SEP 2016



gfjfj Project Status (09/22/2016 - 12:20:14)			
Project File:	gfjfj.ise	Current State:	Synthesized
Module Name:	Sparse_matrix	• Errors:	No Errors
Target Device:	xc3s500e-5fg320	• Warnings:	2 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

gfjfj Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	29	4656	0%
Number of 4 input LUTs	51	9312	0%
Number of bonded IOBs	65	232	28%

CONCLUSION

In this paper, we introduce an FPGA-optimized SMVM architecture that uses a specialized CISR encoding to efficiently process multiple rows of a matrix in parallel, coupled with a highly banked buffer design that eliminates replication of buffered vectors, enabling larger vectors to be stored on-chip. We show that the presented architecture performs within about two thirds of CPU SMVM performance, even though it has 2.4x lower DRAM memory bandwidth, and within almost one third of GPU SMVM performance, even at 9x lower memory bandwidth. Additionally, our FPGA design consumes a maximum of 25W, for power efficiencies 2.6x and 2.3x higher than CPU and GPU, respectively. When supplied with the same memory bandwidth, we predict this FPGA architecture is 1.6x faster than the CPU and 2.7x faster than the GPU, and even more power-efficient (4.4x and 3.8x, respectively).

References

[1] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, “Optimization of sparse matrix-vector multiplication on emerging multicore platforms,” presented at the Supercomputing (SC), 2007.

[2] L. Zhuo and V. K. Prasanna, “Sparse matrix-vector multiplication on FPGAs,” presented at the Int. Symp. Field Programmable Gate Arrays (FPGA), Feb. 2005.

[3] , A. N. Langville and C. D. Meyer, Eds., Google’s PageRank and Beyond: The Science of Search Engine Rankings. Princeton, NJ: Princeton Univ. Press, 2006.

[4] M. M. Baskaran and R. Bordawekar, Optimizing Sparse Matrix-Vector Multiplication on GPUs, 2009, IBM Research Report, Tech. Rep. RC24704.

[5] A. Pinar and M. T. Heath, “Improving performance of sparse matrixvector multiplication,” presented at the Supercomputing (SC), 1999.

[6] E.-J. Im and K. Yelick, “Optimizing sparse matrix computations for register reuse in SPARSITY,” presented at the Int. Conf. Computational Science, 2001.

[7] General-Purpose Computation on Graphics Hardware [Online]. Available: http://www.gpgpu.org [8] SRC Computers, LLC [Online]. Available: http://www.srccomp.com

[9] Cray Inc. [Online]. Available: http://www.cray.com

[10] XD2000i Development System User Handbook, XtremeData Inc. [Online]. Available: http://www.xtremedata.com

[11] Convey Computer Corporation [Online]. Available: http://www.conveycomputer.com

[12] K. Underwood, K. S. Hemmert, and C. Ulmer, “Architectures and APIs: Assessing requirements for delivering FPGA performance to applications,” presented at the Supercomputing (SC) 2006.

BIOGRAPHIES

Panikara Raju is currently a PG scholar in ECE Department. He received B.TECH degree from JNTU. His current research interest includes Analysis & Design of VLSI System Design.

L.Nishanthini Currently working as Assistant Professor Department of Electronics and Communication Engineering in Kommuri Pratap Reddy Institute of Technology, Ghatkesar, RangaReddy,Telangana ,India Her current research interest includes VLSI Design.