FPGA ACCELERATION OF SPARSE MATRIX-VECTOR MULTIPLICATION BASED ON NETWORK-ON-CHIP

H. Y. Jheng∗, C. C. Sun§†, S. J. Ruan∗ and J. Goetze†

†Dortmund University of Technology, Information Processing Lab
Otto-Hahn-Str. 4, 44221 Dortmund, Germany, §E-Mail: chichia.sun@tu-dortmund.de

∗National Taiwan University of Science and Technology
Microsystems Lab, Taipei 106, Taiwan

ABSTRACT

A new design concept for accelerating Sparse Matrix-Vector Multiplication (SMVM) in FPGA by using Network-on-Chip (NoC) is presented. In traditional circuit design on-chip communications have been designed with dedicated point-to-point interconnections or shared buses. Therefore, regular data transfer is the major concern of many parallel implementations. However, when dealing with the SMVM operation, which is the main step of most iterative algorithms for solving systems of linear equations, the required data transfers are usually dependent on the sparsity structure of the matrix and can be extremely irregular. Using a NoC architecture makes it possible to deal with arbitrary structure of the data transfers, i.e. with arbitrary structured sparse matrices. In this paper, a configurable interface is presented which can generate the pipelined SMVM calculator based on NoC architecture with size of $2\times 2, 4\times 4, \ldots, p\times p$ ($p\in\mathbb{N}$). The implementation is done in IEEE-754 single floating-point precision on the Xilinx Virtex-6 FPGA.

1. INTRODUCTION

Over the past 30 years, researchers and scientists have tried various approaches to mitigate the poor performance of sparse matrix computations. Despite these efforts, the sparsity of the matrices still dominates the performance of the Sparse Matrix-Vector Multiplication (SMVM) computations [1]. SMVM is used in many applications. For example, Finite Element Method (FEM) is a widely applied engineering analysis tool which is based on obtaining a numerically approximate solution for a given mathematical model of a structure [2, 3]. In general, iterative solvers, such as the Conjugate Gradient (CG) method, are almost dominated by SMVM operations (i.e. usually more than 95%). The CG method is the most popular iterative method for numerically solving systems of linear equations [4, 5]. Moreover, Google's PageRank (PR) Eigenvalue problem is considered to be the world's largest sparse matrix calculation. This algorithm is also dominated by SMVM operations where the target matrix is extremely sparse, and unstructured.

In the last decade, many researchers have dealt with the integration of pipelining and parallelism inherent in the SMVM computation in hardware designs. Sun et al. [6] proposed a SMVM design on FPGA containing many Processing Elements (PEs) with pipelined floating-point units. Gregg et al. built a specialized memory controller to accelerate the SMVM in [7]. Götze and Schwiegelshohn [8] presented a systolic algorithm which allows the parallel execution of SMVM on VLSI circuit. Williams et al. [9] used a multi-core environment, the heterogeneous x86 based quad-core CPU to speed up SMVM. Google's PR problem has also been investigated for acceleration with FPGA in [10, 11]. Conventional SMVM architectures are usually focused on a dedicated internal chip interconnection to forward vector components and nonzero matrix elements among several processors. For instance, the fat-tree style designs, which require pre-sorting and pre-ordering before input the data [12], will become extremely difficult when the matrix is very large and sparse.

To solve these challenges, a paradigm shift in on-chip interconnection to packet-based switch network is motivated. This new packet switching architecture is called Network–on–Chip (NoC) [13,14]. The basic idea of the NoC is that we regard a System-on-Chip (SoC) device as a micro network of components and the data are switched through the routers.

In this paper, we present an FPGA accelerator for SMVM based on the NoC architecture (**SMVM-NoC**) to solve the problems, which arise from large sparse matrices with their extremely irregular structures. The basic idea of this design is to implement an on-chip internal network as the main transmission bone for the data transfers required for the SMVM computation. The presented architecture is able to handle large sparse matrices, especially it can deal with arbitrary sparsity structures. We have generated 4×4 and 8×8 pipelined SMVM-NoC calculators through a configurable interface, then implemented it on Xilinx Virtex-6 ML-605 FPGA. According to the implementation results, utilizing the packet-based forwarding functionality is beneficial concerning high capability of heterogeneous IP integration and flexibility.

This paper is organized as follows: In Section 2, an brief introduction to the SMVM operation, the design technique of NoC and the basic idea of proposed SMVM-NoC architecture is give. The hardware implementation is described in Section 3 in detail. In Section 4, our experimental results are given. Section 5 concludes this paper.

2. SMVM ON NETWORK-ON-CHIP

A typical SMVM operation can be expressed as follow:

$$A\cdot x = b, \tag{1}$$

where x and b are vectors of length n and A is a $n\times n$ sparse matrix. Many researchers have already utilized pipelining and parallelism to improve the performance. However, the performance is still determined by the sparsity of the matrix A [3].

© EURASIP, 2011 - ISSN 2076-1465

The growing complexity of multi–core architectures will soon require highly scalable communication infrastructure for the integration of more than thousand IP cores [15]. As the VLSI technology keeps shrinking down into nanoscale, the wire delays become more critical than gate delays causing performance degradation and synchronization problems [16, 17]. Moreover, the data synchronization issue with a single clock source will also become a critical problem for circuit synthesis [13, 18]. That means the issue of convergence for timing closure on the large SoC design is difficult to be solved. In order to solve these problems, NoC was presented as a new SoC paradigm to replace the traditional on-chip interconnections by packet based switch network architecture [19, 20].

In general, a typical multi-core system based on mesh style network consists of a regular $n \times n$ array of tiles. Figure 1 shows a 4×4 array which is connected by a two dimensional mesh topology. Each tile could be a general-purpose processor, a DSP, a customized IP or a subsystem. A Network Interface (NI) is embedded within each tile for connecting itself with its neighboring tiles. The data communication can be achieved by routing packets in the network.

SMVM is performed on this array using the packet-based switch network for the required data transfers. For instance, Figure 1 also shows a simple scenario. The circle symbols denote the nonzero elements of the sparse matrix, the square symbols denote the vector elements, and the rhombus symbols denote the results, respectively. First of all, the vector components x_j and nonzero matrix elements A_{ij} will be distributed to the corresponding PE according to the coordinate index i and j through the mesh network; i.e. vector x_1 arrives at the PE with A_{11}, A_{21}, and A_{51}. Now the partial products $b_i^{(j)} = A_{ij} \cdot x_j$ must be computed and then the $b_i^{(j)}$ are accumulated to form $b_i = \sum_i b_i^{(j)}$. Note that the number of nonzero elements n_z is usually larger than the number of PEs.

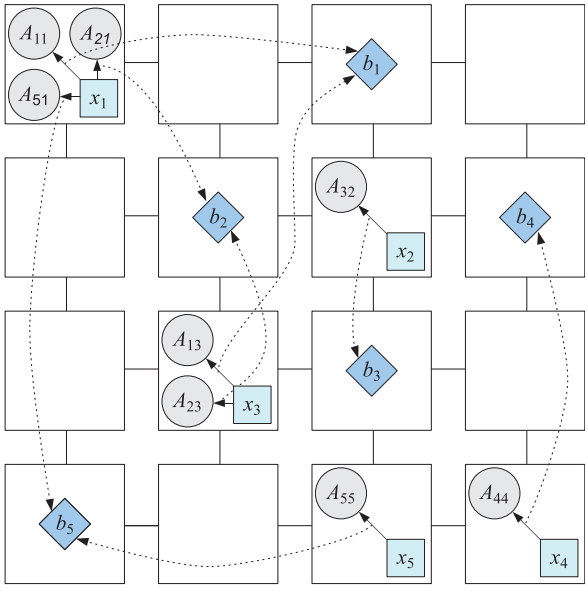

Figure 1: A simple example of direct mapping for parallel SMVM operations in NoC architecture

Table 1: Packet Format

Packet Format	
Bit	Functionality
51	Finish flag
50	OPCode:0⇒Multiplication and 1⇒Accumulation
49	DataType:0⇒Matrix and 1⇒Vector
40-48	Y–Coordinate
32-39	X–Coordinate
0-31	Pay-load

3. IMPLEMENTATION

In this section, a brief introduction of how to map the proposed design concept for parallel SMVM operations on NoC architecture is given. Figure 2 shows the block diagram of a 4×4 SMVM-NoC in Xilinx Virtex-6.

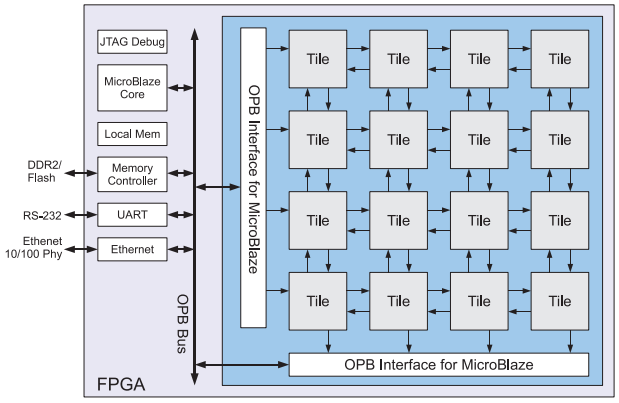

Figure 2: The system level view of a 4×4 SMVM-NoC in Virtex-6 ($p \times p$ is reconfigurable) [21]

3.1 Packet Format

The packet format that is used to communicate over the proposed NoC architecture is summarized in Table 1. The packet is mainly constituted by the header and the pay-load. The header contains three flags and two addresses. The three flags are named Finish, OPCode, and DataType. The Finish flag tells the routers that this packet contains the result and needs to be forwarded to the exit port. The OPCode informs the PE which floating-point operation is going to be performed (multiplication or accumulation), and the DataType flag represents the type of input data (matrix element or vector element). The objective of two coordinate addresses is to locate the position of the PE for the multiplier or the accumulator.

3.2 Switch Architecture

The switch is the most important component concerning the performance of SMVM-NoC. In [21], we built up a prototype SMVM–NoC platform, where the mesh network size is fixed to 4×4. However, in order to reduce area overhead and power consumption, a regular 5×5 crossbar switch is now divided into two 3×3 switches. This method can significantly reduce the complexity of arbitration since smaller and fewer arbiters imply fewer contentions [22]. This new switch

component for the local PE to communicate with its neighbor PEs consists of a set of input/output ports, dual-crossbar switches, four input FIFOs and controllers. It is illustrated in Figure 3.

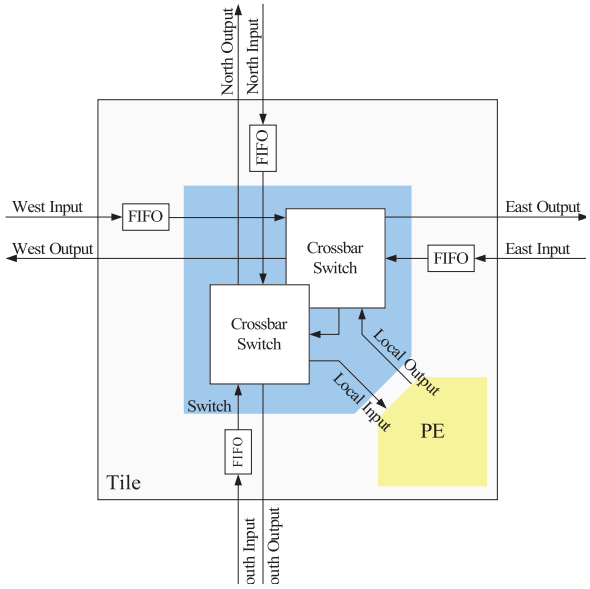

Figure 3: Detailed switch interconnection including two 3×3 crossbars, five I/O ports and four FIFOs

In order to further improve the performance of the switch, the 3×3 switch is pipelined into 5 stages as shown in Figure 4. The five stages are named as Fetch, Routing, Channel Request, Channel Acknowledgment and Output, respectively. Fetch stage reads a packet when there is a packet available in the FIFO and forwards it to the next stage. Routing stage decomposes the header information for routing. Channel Request stage sends a channel request to the crossbar. The packet will be forwarded if desired channel is not occupied, otherwise it will be stalled until the requested channel is free. In an ideal situation, a three-port switch can transmit three packets in each clock cycle. There are two conditions that stall the pipeline, either output FIFO full or the requested channel has been occupied by another request. When a stall event happens, the switch will first move the packet in the last stage to a stall register since the Fetch stage cannot predict the stall event while reading the input FIFO. This might result in one more packet that has been read after stall happened. After the stall event is dissolved, the switch will retrieve the packet from the stall register.

3.3 Routing Algorithm

For simplicity and flexibility of the proposed SMVM–NoC hardware implementation, direct routing algorithm is selected (a.k.a. XY deterministic routing) for our NoC architecture design. In XY routing, the switches in the network are indexed by their XY coordinates. When a switch receives a packet from other switches, it will first extract the header information of this packet and arbitrate the direction, then transmit to next switch. Each packet is first routed along X coordinate and then along Y coordinate until the packet

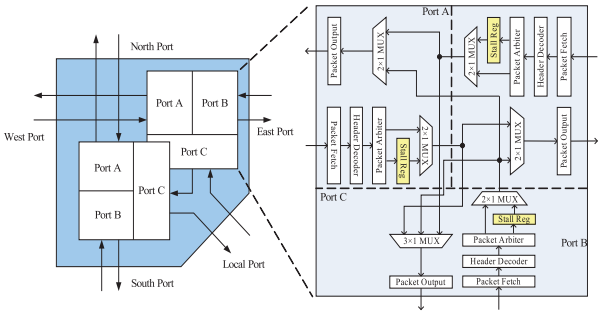

Figure 4: A 5-stage pipelined switch with two 3×3 crossbars, five I/O ports and four FIFOs.

reaches its destination.

3.4 Processing Element

The PE is designed to deal with the floating-point operations for SMVM–NoC as illustrated in Figure 5. The PE used in the proposed architecture contains a control circuit, a floating-point multiplier, a floating-point adder, a data arbiter, a 2×1 multiplexer, and four FIFOs. Multiplier and adder are used to perform multiplication and accumulation, respectively. The data arbiter receives the packet from the switch and extracts the header. Then it forwards the data to proper FIFOs or lookup tables. The multiplexer is used to select one of the two result FIFOs as data output. The Vector Table and the Sum Table can contain at most 16 matrix vector elements and 16 accumulation results for each.

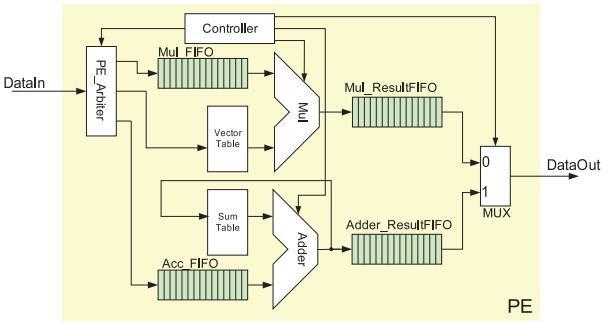

Figure 5: Schematic view of the PE

3.5 Data Mapping

When mapping a SMVM operation into the 4×4 SMVM-NoC, we first need to pack nonzero elements of sparse matrix and the corresponding vector elements with a proper header. Packing examples are listed in Table 2, where the two addresses (X–, Y–Coordinate) are separated into three sub parts. The lower 2 bits of the two addresses are used to indicate the horizontal and vertical positions of PE for multiplication. The third and forth bits are used to address the PE for accumulation operation. Note that these two address lengths should be long enough when the size p of SMVM-NoC increases. The higher 4 bits of X-Coordinate are used as an index for matrix Vector Table to cache 16 entries in PE

for multiplication, whereas the higher 4 bits of Y-Coordinate are used for Sum Table to buffer 16 entries for accumulation.

If we look at Figure 1 and Table 2 as an example, the nonzero elements A_{11}, A_{21} and A_{51} need to perform multiplications with the vector element x_1. The addresses of these three nonzero elements and one vector element should be mapped to the same PE. This means that the lower 2 bits in those packets have to be identical (emphasized in italic style). On the other hand, the multiplication results ($A_{21} \times x_1$) and ($A_{23} \times x_3$) need to be accumulated in the same PE so that the third and forth bits of A_{21} and A_{23} and sum index (emphasized in bold style) should be identical, too.

4. EXPERIMENTAL RESULT

The proposed SMVM-NoC platform has been first modeled in Verilog HDL and synthesized with Xilinx ISE 11.5 targeting on XC6VLX240T-1FF1156. Later the implementation has been verified on the Xilinx ML-605 development kit with Xilinx Platform Studio 11.5 as shown in Figure 2. The synthesized resource utilization reports are listed in Table 3. In Figure 6, an overall comparison is made between Pentium-4 PC and pipelined SMVM-NoC with different size of sparse matrix. A set of random matrices from the size of 16×16 to 256×256 with sparsity from 10% to 50% is tested. Obviously, the pipelining improves the performance. For example, when the matrix size is 128×128 , the performance of the proposed pipelined SMVM–NoC with the size of 4×4 can reach 580 MFlops. Therefore, it can obtain a speed up by a factor of 2.8 compared to the Pentium-4 using Matlab SMVM operation. The ideal performance of the SMVM-NoC platform with 4×4 mesh network can achieve a peak performance 8.8 GFlops in theory. Each PE can perform at most two floating-point operations in one clock cycle. The peak performance can be calculated as:

$$Performance_{peak} = FPOPC * PEC * f, \qquad (2)$$

where $FPOPC$, PEC and f are the Floating-Point Operations Per Cycle in one PE, the number of processor and the maximal clock frequency.

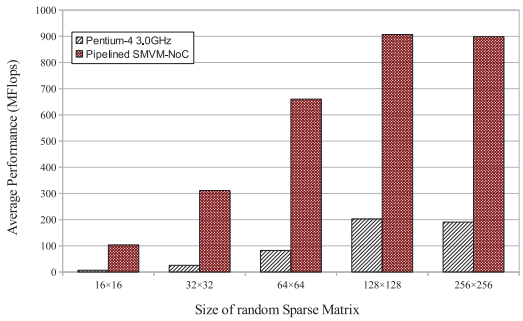

Figure 6: Performance analysis of different matrix size with random sparsity on the 8×8 SMVM-NoC (operating at 200MHz)

Figure 7 shows that the performance of SMVM-NoC is influenced by the sparsity of the matrix. Matrices containing around 6,500 nonzero elements with sparsity from 10% to 50% are tested. For the SMVM, the variance of the performance is smaller than the general processor.

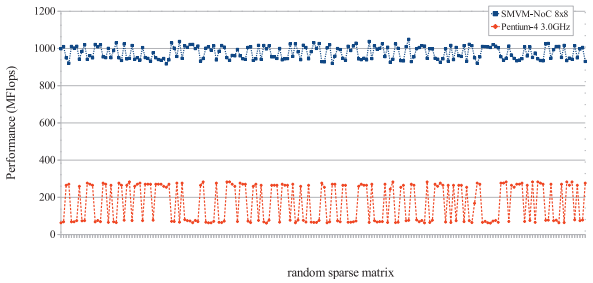

Figure 7: Influence of sparsity on different architectures with random sparsity from 10% to 50%.

However, according to Figure 6 we currently could only obtain an average performance in the range of hundreds of MFlops from the experimental results. Therefore we further look into the packet distribution analysis between the PEs and the switches as illustrated in Figure 8. The problem is that most of the time the packets are stuck in the switches due to the traffic congestion and poor throughput of the switches. On the other hand, the switches transmitting packets over the network have much shorter latency compared to the PEs. Therefore, the switch is compromised with the PE on clock rate, which results in performance decrements. This caused a huge performance gap. In this regard, our future work will extend this architecture to speed up the clock rate of the switch by separating the clock signal into NI clock and PE clock (asynchronous NoC architecture). Furthermore, the pipeline stage of PEs can also be adjusted in order to reduce the packet injection rate.

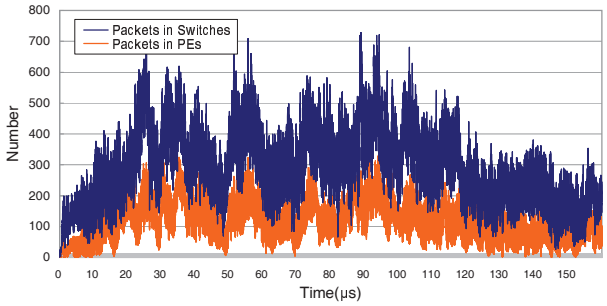

Figure 8: Analysis of the packet traffics for the 4×4 SMVM-NoC

5. CONCLUSION

In this paper, a new design concept for accelerating SMVM based on NoC in an FPGA was presented. Matrix-vector multiplications with various random sparse matrices in IEEE–754 single floating point precision has been tested on the Xilinx Virtex–6 FPGA. The advantages of introducing the NoC structure into SMVM computation are given by high resource utilization, flexibility and the ability to communicate among heterogeneous systems, such that more accelerators can be configured into a larger $p \times p$ array ($p = ,2,4,8,\ldots,2^k, k \in \mathbb{N}$). The synthesis results showed that the

Table 2: An example of packing vector elements and nonzero matrix elements into packet format

Finish Flag	Operation Code	Data Type	Y–Coordinate Sum	Acc	Mul	X–Coordinate Vector	Acc	Mul	Pay load
0	∅	0	∅∅∅∅	∅∅	*10*	*0110*	∅∅	*11*	x_1
0	∅	0	∅∅∅∅	∅∅	10	0001	∅∅	10	x_3
0	0	1	0111	11	*10*	*0110*	11	*11*	A_{11}
0	0	1	**0100**	10	*10*	*0110*	10	*11*	A_{21}
0	0	1	1010	10	*10*	*0110*	11	*11*	A_{51}
0	0	1	**0100**	10	10	0001	10	10	A_{23}

Table 3: Synthesis Results for pipelined SMVM-NoC Architecture in Xilinx Virtex-6 (XC6VLX240T-1FF1156)

Size	Logic utilization	Used	Available	Util.	Freq.	Peak Flops
4×4	Slice Registers	69,224	301,440	22%	275 MHz	8.8 GFlops
	Slice LUTs	77,540	150,720	51%		
	DSP48E1	80	768	10%		
8×8	Slice Registers	254,960	301,440	84%	199 MHz	25.47 GFlops
	Slice LUTs	318,076	150,720	211%		
	DSP48E	320	768	41%		

advanced FPGA with the chip-internal NoC network can provide a solution for sparse matrix computation to further accelerate many iterative solvers in hardware, such as solving systems of linear equations (CG Method), FEM problem and so on. Moreover, the NoC structure can receive data from and forward results to different entries simultaneously. This makes it possible to deal with very large sparse matrices with arbitrary sparsity structure of the matrix without interfering the performance by the sparsity of the matrix.

REFERENCES

[1] Morris GR, Prasanna VK. Sparse Matrix Computations on Reconfigurable Hardware. Computer. 2007 Mar;40(3):58–64.

[2] Elkurdi Y, Fernández D, Souleimanov E, Giannacopoulos D, Gross WJ. FPGA architecture and implementation of sparse matrix-vector multiplication for the finite element method. Computer Physics Communications. 2008 Apr;178:558–570.

[3] deLorimier M, DeHon A. Floating-point sparse matrix-vector multiply for FPGAs. In: International Symposium on Field-Programmable Gate Arrays; 2005. p. 75–85.

[4] Hestenes M, Stiefel E. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards. 1952 Dec;49(6):409–436.

[5] Golub GH, Van Loan CF. Matrix computations (3rd ed.). Johns Hopkins University Press; 1996.

[6] Sun J, Peterson G, Storaasli O. Sparse Matrix-Vector Multiplication Design on FPGAs. In: IEEE Symposium on Field-Programmable Custom Computing Machines; 2007. p. 349–352.

[7] Gregg D, McSweeney C, McElroy C, Connor F, McGettrick S, Moloney D, et al. FPGA Based Sparse Matrix Vector Multiplication using Commodity DRAM Memory. In: International Conference on Field Programmable Logic and Applications; 2007. p. 786–791.

[8] Gotze J, Schwiegelshohn U. Sparse matrix-vector multiplication on a systolic array. In: IEEE International Conference on Acoustics, Speech and Signal Processing; 1988. p. 2061–2064 vol.4.

[9] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: ACM/IEEE conference on Supercomputing; 2007. p. 1–12.

[10] McGettrick S, Geraghty D, McElroy C. An FPGA architecture for the Pagerank eigenvector problem. In: International Conference on Field Programmable Logic and Applications; 2008. p. 523–526.

[11] Zhuo L, Prasanna VK. Sparse Matrix-Vector multiplication on FPGAs. In: International Symposium on Field-Programmable Gate Arrays; 2005. p. 63–74.

[12] Kapre N, DeHon A. Optimistic Parallelization of Floating-Point Accumulation. In: IEEE Symposium on Computer Arithmetic; 2007. p. 205–216.

[13] Benini L, Micheli GD. Networks on Chips: A New SoC Paradigm. Computer. 2002 Jan;35(1):70–78.

[14] Bertozzi D, Benini L. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. IEEE Circuits and Systems Magazine. 2004 Sep;4(2):18–31.

[15] ITRS. 2009 Edition, System Drivers. International Technology Roadmap for Semiconductors; 2009.

[16] Kahng AB. Scaling: More than Moore's law. IEEE Design Test of Computers. 2010 May;27(3):86–87.

[17] Saleh R, et al. System-on-Chip: Reuse and Integration. Proceedings of the IEEE. 2006 Jun;94(6):10501069.

[18] Hemani A, Jantsch A, Kumar S, Postula A, Oeberg J, Millberg M, et al. Network on a Chip: An Architecture for Billion Transistor Era. In: Proceeding of the IEEE NorChip Conference; 2000. p. 24–31.

[19] Vitullo F, L'Insalata NE, Petri E, Saponara S, Fanucci L, Casula M, et al. Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip. IEEE Transactions on Computer. 2008 Sep;57(9):1196–1201.

[20] Wolf W. The future of multiprocessor systems-on-chips. In: Design Automation Conference; 2004. p. 681–685.

[21] Sun CC, Götze J, Jheng HY, Ruan SJ. Sparse Matrix-Vector Multiplication Based on Network-On-Chip. Advances in Radio Science. 2010;8:289–294.

[22] Kim J, Nicopoulos C, Park D, Narayanan V, Yousif MS, Das CR. A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. In: International Symposium on Computer Architecture; 2006. p. 4–15.