Low Power Interconnects for SIMD Computers

Mark Woh[1], Sudhir Satpathy[1], Ronald G. Dreslinski[1], Danny Kershaw[2],
Dennis Sylvester[1], David Blaauw[1], and Trevor Mudge[1]
[1]ACAL, University of Michigan, Ann Arbor, MI {mwoh,sudhirks,rdreslin,dennis,blaauw,tnm}@umich.edu
[2]ARM Ltd., Cambridge, UK danny.kershaw@arm.com

***Abstract*—**

Driven by continued scaling of Moore's Law, the number of processing elements on a die are increasing dramatically. Recently there has been a surge of wide single instruction multiple data architectures designed to handle computationally intensive applications like 3D graphics, high definition video, image processing, and wireless communication. A limit of the SIMD width of these types of architectures is the scalability of the interconnect network between the processing elements in terms of both area and power.

To mitigate this problem, we propose the use of a new interconnect topology, XRAM, which is a low power high performance matrix style crossbar. It re-uses output buses for control programming, and stores multiple swizzle configurations at the cross points using SRAM cells, significantly reducing routing congestion and control signaling. We show that compared to conventionally implemented crossbars, the area scales with the product of $input \times output$ ports while consuming almost 50% less energy. We present an application case study, color-space conversion, utilizing XRAM and show a 1.4x gain in performance while consuming 1.5-2.5x less power.

I. Introduction

Not so long ago, growth in computing was dominated by PCs in the form of laptops and desktops. Today, smartphones and handheld devices like the iPad and iPhone are already poised to outsell PCs within the next few years. The capability of these devices has increased rapidly and now provide features like high definition video, high bandwidth internet access, and 3D graphics all within the same die. These devices will continue to grow in terms of capabilities and performance while still needing to adhere to a strict power budget.

Continued scaling of VLSI technology due to Moore's Law allows us to integrate an increasing number of transistors on a single die. Both in industry and academia we have seen one of the most power efficient ways to utilize this transistor area is through integrating multiple processing elements (PE) within a die [1]. This is represented by many architectures in the form of increased number of single instruction multiple data (SIMD) lanes in processors [2][3], and the shift from multi-core to many-core architectures [4][5]. As we scale the number of PEs in these architectures, traditional on-chip interconnects such as buses and crossbars are unable to keep up with the bandwidth required to fully utilize the PEs. Moreover, traditional interconnects do not scale well in terms of power and area as the number of PEs increase. The National Science Foundation held a workshop trying to identify the most critical challenges facing on-chip interconnects and their findings showed that power and latency were the biggest challenges that needed to be solved [6]. Network-on-Chip architectures [7][8] show that the crossbar itself consumes between 30% to almost 50% of the total interconnect power. Another critical problem is that existing circuit topologies in traditional interconnects do not scale well, because of the complexity in control wire and control signal generation logic which directly effects the delay and power consumption. This area and power scaling problem is one of the critical bottlenecks that limits the increase in the number of PEs in the future.

This paper studies a low power and scalable crossbar design, called XRAM [9], which provides a solution to the bandwidth and scaling problem seen in low power SIMD architectures. XRAM implements novel circuit techniques to solve the scaling problem while providing high bandwidth. Unlike other interconnect networks [10][11], the XRAM is non-blocking and is able to perform all permutations and swizzle operations, including multicasting and broadcasting, illustrated in Figure 1. One circuit technique that helps solve the control complexity problem is to embed the interconnect control within the cross points of a matrix style crossbar using SRAM cells. This differs from the traditional technique where interconnections are set by an external controller. Other circuit techniques, like using the same output bus wires to program the cross point control, help reduce the number of control wires needed within the XRAM. Finally borrowing low voltage swing techniques that are currently used in SRAM arrays improves performance and lowers the energy used in driving the wires of the XRAM. Though these techniques help solve the performance and scaling problem of traditional intercon-

978-3-9810801-7-9/DATE11/

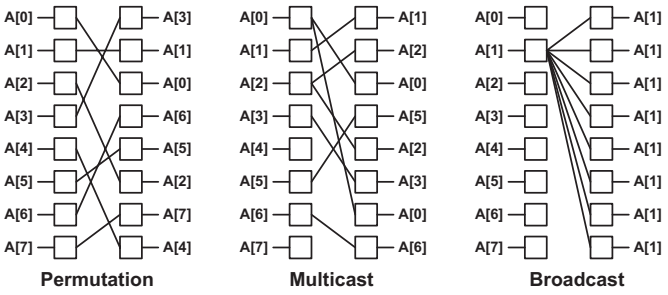


Fig. 1: Permutations are 1-to-1 mappings of input to output ports. Multicasts are mappings that allow individual inputs to connect to multiple outputs, however, no output may be connected to more than 1 input. Broadcast is a special form of multicast where a single input is connected to all the outputs. A swizzle interconnect allows for all three forms, including permutations, to occur in the network.

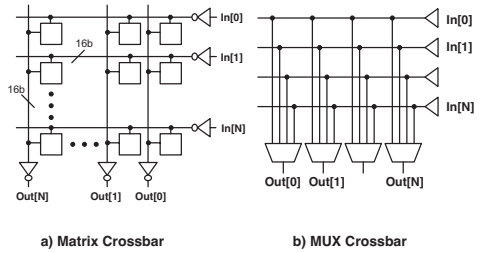


Fig. 2: Commonly Used Network Topologies

nects, one drawback is flexibility; the XRAM is only able to store a certain number of swizzle configurations at a given time. These configurations are not static and can be changed and reprogrammed at run-time. Though this may seem like a major drawback, for many applications only a small number of interconnect permutations is required. Furthermore, we show through a case study that the XRAM achieves 1.4x performance and consumes 2.5x less power in a color-space conversion algorithm.

The remainder of the paper is organized as follows. Section II will discuss the traditional crossbar approaches that have been used in on-chip interconnects and the problems that are faced as we scale. Section III will introduce and discuss the operation of XRAM, a low power and scalable crossbar network. Section IV will present a case study of the XRAM, and provide power and performance comparisons against other interconnect topologies. Finally, Section V will present our conclusions.

II. Related Works

Many techniques are currently used to implement permutation networks and swizzle networks. In this section we discuss traditional crossbar designs. Others solutions such as $Beneš$ [10] and $Banyan$ [11] networks are omitted due to space limitations.

Typically in modern designs when permutations are needed they are implemented using fully connected networks like crossbars. Crossbars are able to realize all permutations and swizzle operations. These networks are implemented in two different ways: MUX based crossbars or matrix style crossbars. Figure 2(a) and (b) show the topology of these two networks. MUX based crossbars utilize Nx1 MUXes to drive the outputs, while matrix style crossbars use switching elements such as transmission gates or tristate buffers to drive the output at each cross point. Current SIMD processors which have narrow SIMD widths, between 4 and 8, implement these style of crossbars. The problem with these networks is that as the number of SIMD processing elements increase, the complexity of these networks grows quadratically. In a MUX based crossbar, the data passes through $log_2(N)$ number of 2x1 MUXes which, combined with driving the wires, determines the critical path delay of the network. The area of the MUX based crossbar increases by $(2N-1)N^2$. As we start increasing N to large numbers, the area and energy consumption increase dramatically. With matrix style crossbars, the area increases N^3 with respect to the numbers of ports. This increases the wire capacitance that the input and output drivers need to drive, increases the delay but more importantly dramatically increases the energy consumption. Generating control for both of these types of networks is straight forward in that only a simple decoder is needed to select the interconnections. However, routing the wires for the control signals is very costly, increasing both area and power of the total network. Because of these drawbacks, these types of networks can only be used when the number of ports is relatively small.

III. XRAM Fundamentals

XRAM is a matrix crossbar that leverages some of the circuit techniques used in SRAM arrays for improving area and performance. Figure 3 shows a system level diagram of XRAM. The input buses run horizontally

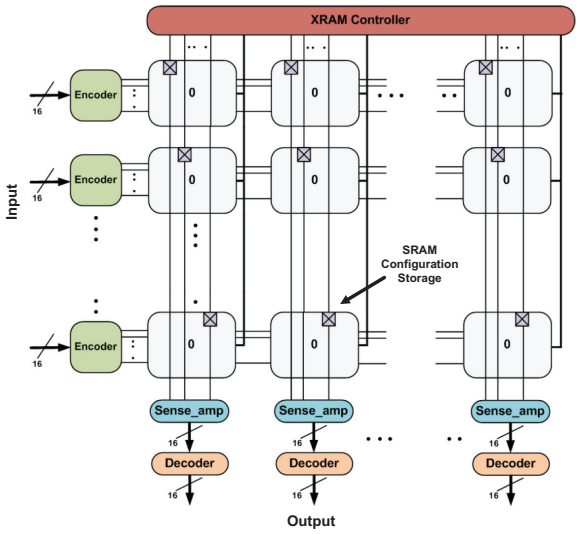


Fig. 3: XRAM is a low power high performance matrix style crossbar that re-uses output buses for control programming and stores multiple swizzle configurations at the cross points using SRAM cells. The XRAM controller sends the control line to pick which configuration SRAM cell to use. The encoder and decoder are the transition encoding/decoding logic used to minimize the switching power when the input and output bits do not change values.

while the output buses run vertically, creating an array of cross points. Each cross point contains a 6T SRAM bit cell. The state of the SRAM bitcell at a cross point determines whether or not input data is passed onto the output bus at the cross point. Along a column, only one bitcell is programmed to store a logic high and create a connection to an input. Matrix type crossbars incur a huge area overhead because of quadratically increasing number of control signals that are required to set the connectivity at the cross points. To mitigate this, XRAM uses techniques similar to what is employed in SRAM arrays. In an SRAM array, the same bit line is used to read as well as write a bit cell. Until the XRAM is programmed the output buses do not carry any useful data. Hence, these can be used to configure the SRAM cells at the cross points without affecting functionality. Along a channel (output bus), each SRAM cell is connected to a unique bit line of the bus. This allows for the programming of multiple SRAM cells (as many bit lines available in the channel) simultaneously.

Swizzle networks have traditionally been highly interconnect dominated, rendering a significant amount of logic space under-utilized. This gets aggravated in sub 100nm technology nodes because of poor scalability of interconnect wires in comparison to transistors. In a 128x128 port swizzle network with 16 bit channels fabricated using industrial standard libraries in an IBM 65nm technology, the silicon utilization is only 18%. XRAM mitigates this to some extent by re-using output channels for programming, resulting in improvement of silicon utilization to 45%. To further improve silicon utilization, multiple SRAM cells can be embedded at each cross point to cache more than one shuffle configuration. In 65nm, a 16-bit bus width allows six configurations to be stored without incurring any area penalty. Any one of these configurations can be selectively programmed or used to shuffle data. We find that many applications, especially in the signal processing domain, only utilize a small number of permutations over and over again. By caching some of the patterns that are most frequently used, XRAM reduces power and latency by eliminating the need to configure and reprogram the XRAM for those patterns.

XRAM operates in two modes: programming mode and transmission mode. In programming mode, the controller sends a one-hot signal onto each output bus. A global wordline is then raised high to program the XRAM. With 16-bit buses, a 16x16 XRAM can be programmed in a single clock cycle. Larger XRAMs are divided into multiple sections with independent wordlines and one section is programmed at a time. For example a 128x128 XRAM with 16 bit buses is divided into 8 sections using 8 independent wordlines to select each section—a total of 24 wires in each channel. To program a channel, in the first cycle all wordlines are raised high while sending an all zero code(*16'b0*) on the channel. This writes a logic low in all the bitcells in that channel. In the next cycle a one hot signal is sent while selectively raising only one wordline high to allocate the channel. For instance, to allocate the channel to *input 43*, the third wordline is raised high while sending *16'b0000100000000000* on the channel.

In transmission mode, incoming data is passed onto the output bus at any cross point storing a 1 using a precharge followed by conditional discharge technique. During the positive phase of clock, input data is launched and the bit lines are precharged to logic high. During the negative phase of clock, the bit lines are selectively pulled down if the data is high and the cross-point signals a connection. A bank of sense amplifiers evaluates the bit lines to retrieve data. The bit lines need not be pulled down all the way thereby saving power and improving performance. However, this technique results in power dissipation even with a non-switching input because the bit lines get precharged and discharged every cycle. To mitigate this, the incoming data is transition encoded at the input of XRAM. The original data is retrieved back at the output using transition decoders.

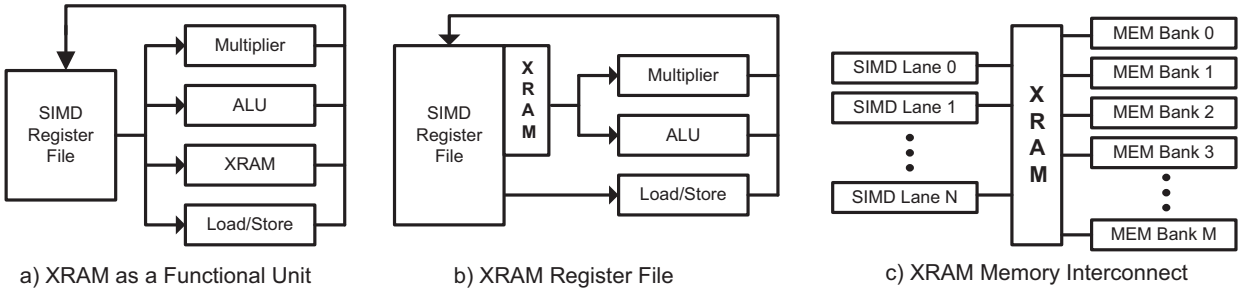


Fig. 4: Different Placements of XRAM within the Architecture. For the memory interconnect benefits occur when $M \geq 2N$.

A. Embedded Configurations in XRAM

The fully programmable implementation of XRAM discussed above is most suited for generic SIMD processors that run a variety of algorithms, each requiring a small subset of vector permutations. This will allow the host processor to program the XRAM before running the application, thereby improving runtime. If larger numbers of permutation operations are required, a one cycle programming penalty will be incurred to reprogram one of the configurations stored locally within the SRAM cell. However, this requires shuffle patterns to be stored in SRAM cells beneath the crossbar wiring. Hence, only a few (6 in 65nm implementation with 16 bit buses) shuffle patterns can be supported; technology scaling will increase the number of configurations stored. But for systems that only support a limited number of shuffle patterns in their ISA—application specific engines or generic SIMD processors like Ardbeg [12]—this may be sufficient.

B. Tradeoffs with Placement of XRAM in Processor Architectures

Figure 4 shows multiple placements of XRAM within a SIMD processor: as a separate functional unit, connected to the register file, and between the memory system and processor. Each placement has its benefits and drawbacks. In Figure 4(a), the XRAM is implemented as another SIMD functional unit, in Figure 4(b) all SIMD register file reads pass though the XRAM. Implementation (a) minimizes delay and power consumption when the re-arrangement functions carried out by the XRAM are infrequent. Implementation (b) improves the CPI performance of the whole machine because swizzle operations can occur every time a register is accessed, however, it increases the total power consumption because the XRAM is used even on instructions when no swizzle is necessary.

The benefit of placing an XRAM between the memory blocks and the processing elements, as shown in Figure 4(c), is that the system can cope with a larger number of memory banks than the number of processing elements. This is an interesting possibility because, as shown by Lawrie [13], access to rows, columns, diagonals, reverse diagonals, and blocks of a matrix cannot all be implemented if the number of memory banks is equal to the number of PEs. However, all these access patterns can be supported if the number of memory banks is twice the number of PEs. Because one of the major applications that SIMD processors can accelerate are signal processing algorithms, optimizing matrix data alignment for SIMD can improve performance dramatically.

IV. Case Studies

Case studies for the XRAM were done across several applications in the signal processing and graphics domain. These applications benefit from not only increase in processing elements but also the shuffle and multicasting operations. Comparisons to MUX based implementations are performed. Each of the case studies were synthesized using IBM65nm technology with Synopsys design compiler and place-and-routed using Cadence encounter. Energy dissipation were generated from the synthesis results. The FFT results were verified with a test chip fabricated in TSMC 65nm.

Figure 5 compares the results of XRAM and MUX based implementations of a SIMD processor that accelerates FFT operations. The XRAM enables us to build accelerators which contain many more processing elements running at a lower voltage and thus lower frequency. When performing at iso-throughput, the XRAM consumes almost 50% less total energy than the MUX based counterpart.

A. Color-space Conversion Hardware

Many smartphones and portable internet devices are based around systems like the Qualcomm Snapdragon [14] and ARM Mali [15] which integrate together a Mobile GPU and high definition video engine (HDVE). These systems typically are design where the HDVE decodes the video stream data and the mobile GPU

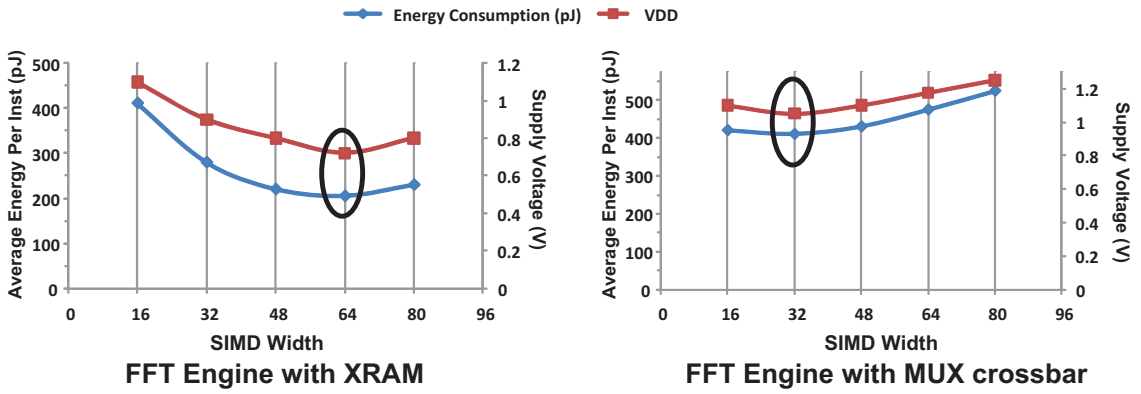


Fig. 5: Average energy per instruction and supply voltage for different width FFT engines at iso-throughput. The 64-width XRAM and 16-width MUX-based results are from TSMC 65nm Test Chip measurements. The other results are extrapolated from these values and guided by synthesis.

performs video overlay of the menu system and post processing. Problems arise between the communication between the HDVE and GPU when we start going to high definition standards such as 1080p. In these systems interconnect bandwidth and memory storage are scarce and expensive commodities, so the HDVE and GPU typically read and write out a form of compressed pixel data—*YUV*. The problem is that current mobile GPUs operate on *RGB* data only [16]. In order to use *YUV* data, the GPUs have to perform color-space conversion on the *YUV* output of the HDVE and convert the data into *RGB* for subsequent processes. At this point the GPU can output the image directly to the display or perform another color-space conversion to compress the *RGB* data back into *YUV* data. Typically in desktop GPUs from Nvidia and ATI, power consumption is in the hundreds of watts, so the number of streaming processors is extremely large—typically in the hundreds. Performing color-space conversions for post processing on these GPUs utilizes very little of the total resources. In mobile GPUs, such as Imagination Technologies' PowerVR SGX [17] or ARM's Mali, there are less than ten's of stream processors because the power budget is in the hundreds of mW. Performing color-space conversion on mobile GPUs can saturate the whole GPU processing power. In addition, for video modes like 1080p real time rendering may be impossible.

In the literature, accelerators for color conversion have been designed to convert between *YUV* and *RGB* formats. However, these systems only support a specific set of *YUV* modes—such as *YUV 4:2:0* or *YUV 4:2:2* [18]. This is done because the pixel data is laid out differently in each format. Figure 6 shows a few different color-space modes. As you can see the pixel data itself can be in many different places,

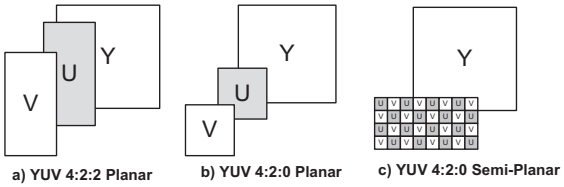
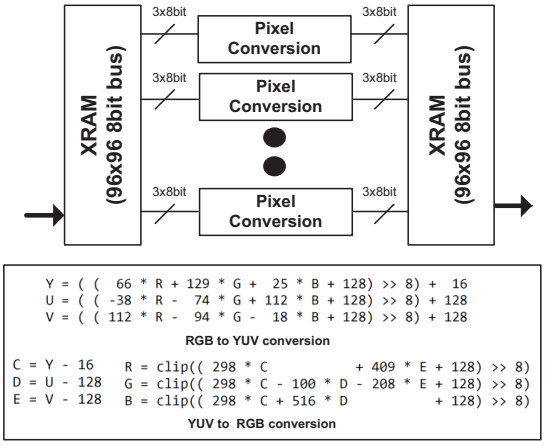


Fig. 6: Examples of different *YUV* modes. In each *YUV* mode the Y plane is always written out to memory then the U plane followed by the Y plane. In *YUV 4:2:0* semi-planar, the U and V data are interleaved together. The typical size of the Y plane is 16x16 pixels.



Operation of the Pixel Conversion Unit

	XRAM	MUX Based
Delay	1.001ns	1.727ns
Energy Consumption	64.7pJ	261pJ
Power	64.6mW	151.1mW

Fig. 7: Color Space Conversion Hardware

but only a single XRAM configuration is needed to perform a specific input color-space to output color-space conversion. This configuration can be programmed once and performed multiple times amortizing the one

cycle programming penalty across the entire conversion. However, the memory access pattern causes the color-space conversion accelerators to add increased amounts of contention into the memory system. Figure 7 shows our implementation of a color space conversion accelerator that can reduce the number of memory requests to the bus while also increasing performance. Because we need to buffer a larger number of pixels and then perform multicasting and permutations based on the color compression scheme, XRAM becomes a key enabler for this architecture. One example is to first convert *YUV 4:2:0* to *YUV 4:4:4* then perform the *YUV* to *RGB* conversion. After the pixel data is buffered, the system can efficiently process the pixels and convert them to the corresponding color-space. If a MUX based system is used the power consumption, area, and delay of the MUX would dominate the accelerator. These increases would lead to an alternative solution where more stream processors would be added to the GPU rather than using a color space conversion accelerator. Figure 7 shows that compared to the MUX based implementation of the color space conversion accelerator, we are able to perform 1.7x faster, and consume almost 2.5x less power. This allows the use of such an accelerator without the need to increase the number of processing elements.

V. Conclusion

In this paper we evaluate a low power and scalable crossbar design, called XRAM, which provides a solution to the bandwidth and scaling problem seen in traditional interconnect implementations. XRAM implements novel circuit techniques to solve the scaling problem while providing high bandwidth. It is a non-blocking swizzle interconnect able to perform all permutations, multicasts, and broadcasts. XRAM solves the control complexity by embedding the cross point control within the cross points of a matrix style crossbar using SRAM cells rather than having it driven by an external controller. Other techniques like using the output bus to program the control helps reduce the number of control wires needed within the XRAM. Finally borrowing techniques that are currently used in SRAM arrays helps improve performance and lower the energy used in driving the wires of the XRAM.

We showed that compared to conventionally implemented crossbars, the area scales linearly with the product of the $input \times output$ ports while consuming almost 50% less energy. We present an application case study using XRAM and show that compared to conventional MUX based implementations, the XRAM improves performance by 1.4x and between 1.5-2.5x lower power for applications such as color-space conversion.

References

[1] S. Borkar, "Thousand core chips: a technology perspective," in *DAC '07: Proceedings of the 44th annual Design Automation Conference*. New York, NY, USA: ACM, 2007, pp. 746–749.

[2] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "Anysp: anytime anywhere anyway signal processing," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2009, pp. 128–139.

[3] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–15, 2008.

[4] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata, "Cell broadband engine architecture and its first implementation: a performance view," *IBM J. Res. Dev.*, vol. 51, no. 5, pp. 559–572, 2007.

[5] M. Gebhart, B. A. Maher, K. E. Coons, J. Diamond, P. Gratz, M. Marino, N. Ranganathan, B. Robatmili, A. Smith, J. Burrill, S. W. Keckler, D. Burger, and K. S. McKinley, "An evaluation of the trips computer system," in *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM, 2009, pp. 1–12.

[6] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, 2007.

[7] H.-S. Wang, L.-S. Peh, and S. Malik, "A power model for routers: Modeling alpha 21364 and infiniband routers," *IEEE Micro*, vol. 23, no. 1, pp. 26–35, 2003.

[8] A. Kumar, P. Kundu, A. P. Singh, L. shiuan Peh, and N. K. Jha, "A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator," in *in 65nm CMOS, ICCD-2007*, 2007.

[9] S. Satpathy, Z. Foo, B. Giridhar, D. Sylvester, T. Mudge, and D. Blaauw, "A 1.07 tbit/s 128x128 swizzle network for simd processors," in *IEEE Symposium on VLSI Circuits*, 2010.

[10] V. E. Benes, "Optimal rearrangeable multistage connecting networks," *Bell System Technical Journal*, vol. 43, pp. 1641–1656, 1964.

[11] C. P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. Comput.*, vol. 32, no. 12, pp. 1091–1098, 1983.

[12] M. Woh *et al.*, "From SODA to scotch: The evolution of a wireless baseband processor," in *Proc. of the 41st Annual International Symposium on Microarchitecture*, Nov. 2008, pp. 152–163.

[13] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. 24, no. 12, pp. 1145–1155, 1975.

[14] *Qualcomm Snapdragon*, Qualcomm Inc., 2010, http://www.qualcomm.com/products_services/chipsets/snapdragon.html.

[15] *Mali GPU OpenGL ES Application Development Guide*, ARM Ltd., 2010, http://infocenter.arm.com/help/topic/com.arm.doc.dui0363-/.

[16] *Render to Texture with OpenGL ES*, Texas Instruments, 2010, http://processors.wiki.ti.com/index.php/Render_to_Texture_with_OpenGL_ES.

[17] *SGX Graphics IP Core Family*, Imagination Technologies, 2010, http://www.imgtec.com/powervr/sgx.asp.

[18] *LF3370 Application Note*, Logic Devices Inc, 2001, http://www.logicdevices.com/support/appnotes/lf3370ycb.pdf.