Efficient Reconfigurable Architecture for MIMD Streaming Execution Using Permutation Network

Chi Wen Cheng
Dep. of EE, National Taiwan University
Email: b99901041@ntu.edu.tw

Yu Sheng Lin, Shao Yi Chien
GIEE, National Taiwan University
Email: {johnjohnlys,sychien}@media.ee.ntu.edu.tw

***Abstract*—Reconfigurable architectures grant many circuits more flexibility as well as more efficiency. By dynamically reconnecting the datapath between calculation units, we can optimize the performance of many designs. Inspired by some prior works, we proposed a new MIMD Streaming (MIMDS) execution scheme on the aid of reconfigurable design, featuring high efficient stream processing. In this work, we also take the locality of programs into account when designing our reconfigurable architecture. Therefore, we use the permutation network [1] as our reconfigurable path, which provides less but enough reconfigurability, leading to less area cost and less power consumption. In this paper, we will take a commercial processor, C54x from Texas Instrument [2], as example, as well as detail the modification from the baseline C54x to our proposed MIMDS architecture. We show that with the extra ALUs and efficient datapath, C54x with MIMDS feature has overall 63% less execution cycles and 45% less memory access at most. Compared with traditional C54x, our design has only 12% area overhead. Besides, if we consider only configurable network, our permutation network saves 85% area compared to fully reconfigurable datapath while supports sufficient reconfigurability.**

***Keywords*—reconfigurable architecture, permutation network, digital signal processor, explicit datagraph execution**

I. Introduction

Reconfigurable architectures have become more and more important in modern architecture designs. Compared to fixed datapath in application specified IC (ASIC), reconfigurable architectures are able to dynamically reconnect different computation units. Many previous works such as [3]–[7] indicate that reconfigurable architectures are more powerful than fixed-datapath designs. For example, [3]–[5] demonstrate that reconfigurable datapath is powerful in camera pipeline along with streamed data. Besides, some people proposed the concept of explicit datagraph execution (EDGE) [8] for CPUs, which combines many individual instructions into a larger group known as a "hyperblock". EDGE CPUs usually has much more ALUs than traditional CPUs. A hyperblock can be easily converted to data dependency graph and directly mapped to ALUs (as nodes) and the connection (as edge) between them, enabling many instructions to execute in parallel. Also, we can view EDGE as multiple-instruction-stream-multiple-data-stream (MIMD) architecture.

Inspired by the camera stream processor and and EDGE, we proposed a new architecture that can handle multiple instructions and multiple streams, which we call the MIMDS architecture. Different from EDGE, our proposed MIMDS architecture focuses on and reinforces instructions in loops. With reconfigurable network, hyperblocks are pipelined and repeatedly executed on the hardware. The data go through the hyperblocks clock by clock as if the data are processed streamingly.

Furthermore, literatures on reconfigurable architectures usually do not mention much about their reconfigurable datapath. In the naive MUX-based design, for each possible input of calculation units, one MUX is used to select the signals from all outputs. Although it can support full connectivity, it wastes a lot of area, including big-area multiplexers and unnecessary control signals. Therefore, considering the locality of real programs, we proposed a new solution for reconfigurable architecture designs using *"A Permutation Network"* [1]. Although it has limited connectivity, we find that such limited connectivity is enough in our testcases. Moreover, the proposed permutation network has less area with many small switches (which will be introduced in II-C). Reasonably, we can infer that the permutation network is more cost-efficiently than traditional MUX-based networks.

We chose a commercial digital signal processor C54x from Texas Instrument [2] as our baseline, and proposed our MIMDS scheme with a permutation network. We reinforce the special repeat instruction, which is powerful for stream processing in the original C54x. On the aid of reconfigurable architecture, the performance is improved by explicit datagraph execution of streamed data, which also reduces unnecessary data memory accesses and saves more power accordingly.

To verify the performance, the corresponding C54x RTL model is synthesized with TSMC 90nm technology. Under different representative workloads, we show that C54x with reconfigurable datapath has overall 64% less execution cycle and at most 45% less memory accesses over traditional C54x with only 12% area overhead. Besides, our permutation network uses saves 85% area as MUX-based reconfigurable network while supports sufficient connectivity if we consider only configurable network.

The rest of the paper is organized as follows. We introduce important previous works in Section II. In Section III, we elaborate on our proposed processor scheme. We describe our evaluation methodology and experiment result in Section IV, and conclude in Section V.

II. Related Works

A. Reconfigurable Architectures

Reconfigurable architectures can process data efficiently in many aspects. For example, [3]–[5] have proven that the

reconfigurable architectures can be applied in camera pipeline, including color adjustment, noise reduction and feature detection. From observation, many operations of these image processing algorithm can be classified into few patterns. These papers show that many algorithms can be supported by only using few small hardware units such as window buffers, line buffers, ALUs and interpolation units. By merely reconnecting these units, many complex operations, such as demosaicing and SIFT [9], can be performed. Besides, these designs allow stream-in-stream-out dataflow scheme, which is consistent with camera pipeline, where pixels are streamingly read out from the sensor. In contrast, DSP or CPU must store the frames before the data can be processed, which requires large expensive on-chip SRAMs. As a result, compared to DSPs of similar performance, these reconfigurable circuits are 80~250x faster while consuming only 1/500~1/1000 power, and the normalized size are only 1/2~1/20.

Explicit datagraph execution (EDGE) also implicates the concept of reconfigurable architecture, and some of the most famous literatures are TRIPS [10] and Wavescalar [11], whose behaviour is like very long instruction word (VLIW) CPUs to some extent. Both EDGE and VLIW's ISAs support multiple inputs and outputs at once, so multi-bank memory is required in order to support such wide I/O. Their difference is their scale and the configurability. In VLIW, independent instructions are grouped together and executed concurrently. For example, one VLIW instruction may perform one add, one multiplication and one memory store because they use different hardware units. However, how many and what kind of instructions can be grouped are fixed by the specification of CPU. But in EDGE, usually one or more compatible basic blocks are grouped into a "hyperblock" (basic block roughly equals to a code segment enclosed in a pair of braces in C code). Then a hyperblock is converted to a data dependency graph by a compiler, and the resulting graph is directly mapped onto the ALUs and the routings between ALUs. EDGE have some advantages over other instruction level parallelism (ILP) mechanism. First, as long as a hyperblock can be mapped, the data can flow spontaneously in the graph composed of the ALUs and routings, as if they are executed on an application specified IC (ASIC). As a result, all possible ILP can be exploited so the execution time is just the shortest possible time in the data dependency graph. Moreover, most of the temporary data can traveling around ALUs without going out to caches or DRAMs, causing lower power consumption and higher speed.

B. TI C54x DSP

We chose Texas Instrument's digital signal processor, C54x, because it featured fast and streamed data execution even without the SIMD feature. Therefore, we would describe the important feature of its architecture as well as its DSP library.

1) Features: To begin with the architecture of C54x, we demonstrate with a common C code shows as follows:

Listing 1. A simple C code segment calculating inner product of two vectors

```
sum += (*ptr_a++)*( *ptr_b++);
```

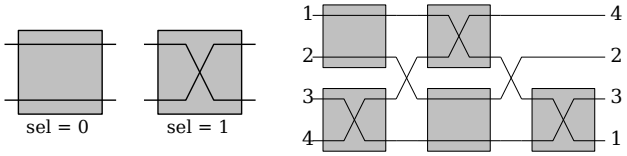
Imagine if we repeat this line many times, then this will calculate the inner product of vector `ptr_a` and `ptr_b`, and similar pattern also appears in signal filtering and matrix multiplication. For most of the processors, this would cost many extravagant instructions to handle the pointer's value, multiply and then add them into a temporary register. On the other hand, C54x's special instruction set and architecture takes such kind of operation for just one instruction (cycle). Moreover, another feature in C54x is its repeat instruction, which can execute a single instruction consecutively. Combining the C54x's powerful instruction and repeat functionality, streamed data from incremental SRAM address can be processed efficiently. For some long-latency instructions, repeat instruction can also hide their latency by filling up the pipeline with the data stream under this architecture.

Fortunately, we get the prototype C54x from the website Opencores [12]. To be frank, the C54x described in the rest of this paper is not absolutely the same as the commercial one. However, we support most of the important instructions so that we can utilize its compiler and code generator.

2) DSPLIB: C54x's DSP library describe many common and useful computation intensive functions which are highly optimized with C54x's special architecture. The functions of C54x's DSP library can be categorized into 8 different classes, including FFT, filtering and convolution, adaptive filtering, correlation, math, trigonometric, matrix function and miscellaneous. We would select some representative functions from each classes as our workloads and detail the evaluation methodology in Section IV.

C. Permutation Network

Abraham Waksman's *"A Permutation Network"* [1] describes an n-input to n-output switching network, and the outputs of the network can be all possible permutations of the inputs. A building block of such a network is a simple 2-to-2 switch as Figure 1(a) shows, notice that it is a 2-to-2 permutation network itself. Each switch is controlled through a 1-bit signal. When the signal is 1, the switch would swap the two input signal. Otherwise, the switch wouldn't do anything, and just bypass the input signal directly to the output signal. A 4-to-4 permutation network is illustrated in Figure 1(b).



(a) The building block of the permutation network. Whether the 2 inputs are exchanged is determined the selection signal.

(b) A 4-to-4 permutation network and a possible configuration.

Fig. 1. Some illustrations of the permutation network.

Larger permutation networks can also be built from small permutation networks. For example, combining two parallel 4-to-4 permutation networks and 7 extra switches, we can build an 8-to-8 permutation network using 17 switches. Provided the target permutation, the control bits for the switches can be solve systematically in a recursive manner. To solve an 8-to-8 permutation network, we solve the 7 extra switches first and

then solve the two 4-to-4 permutation networks individually and recursively. Such a network can be proven to be optimal mathematically, and more detailed solution can be founded in [1].

Compared to the typical MUX-based reconfigurable network, we believe the permutation network is better in several aspects. The cost of area is the first topic. For example, an N-to-N network with MUX-based design has a degree of complexity of $\Theta(N^2)$ (N numbers of N-to-1 multiplexers). In contrast, a permutation network has a degree of complexity of $\Theta(N \log(N))$. Furthermore, each output of a MUX-based design has a pretty high fanout, while each output of a permutation network has at most 2.

III. Proposed Architecture

A. Modify the ISA

Memory operation, multiplication and addition occupy 82% instructions in the loops of C54x DSPLIB on average, so optimizing these instruction may yield large efficiency boost. Next we will illustrate our idea by a simple example.

Consider this case: if we want to multiply 2 complex vectors, assuming the real and imaginary part of the vectors are given as individual C pointers, the code will be looked like this:

Listing 2. A simple C code segment executing complex number multiplication

```
*oR = (*iR1)*(*iR2)-(*iI1)*(*iI2);
*oI = (*iR1)*(*iI2)+(*iI1)*(*iR2);
++oR, ++oI, ++iR1, ++iI1, ++iR2, ++iI2;
```

Imagine that if we repeat this segment many times, then we can have correct data stored in the memory space pointed by the initial `oR` and `oI`. The scheme is similar to the one we mentioned in II-B. In the inner product example, the DSP can repeatedly add or multiply two operands from SRAM and store the result to SRAM at each clock while moving the SRAM address pointers concurrently.

Our ISA tries to make the DSP's repeat functionality more general and is fully compatible to the original dataflow of the DSP. At each clock, the DSP can now receive multiple operands from and store multiple results to SRAM. Besides, the DSP can now perform not only single operations such as add or MAC but also compounded small instruction blocks. In this example, with this ISA, the DSP can now accept the real and imaginary parts of the two inputs and output the complex-multiplied results at each clock (with some latency). Equivalently we can say the DSP processes 4 streams and outputs 2 streams over the given repeating period, and this is why we call our architecture MIMD Streaming (MIMDS) execution. Such an ISA is especially useful in vector operation. In the next subsection we will describe how our architecture can support such MIMDS execution.

B. New Datapath in DSP

Multiplication and addition instructions occupy most of the processing time in C54x dsplib so we add some extra multipliers and adders to the original DSP. Many literatures such as [7] shows that ALUs are relative very small to data path in modern CPUs. That is, we can generously add ALUs as long as we have efficient datapath to well utilize them.

The reconfigurable routing is then added between these extra ALUs, but supporting full reconfigurability is wasting and unnecessary. For example, a basic MUX-based reconfiguration circuit such as [13] is shown in Figure 2. First, the input data are sent to some of the ALUs in the cluster (1). After that, data can be calculated and sent to other ALUs many times (2) before it goes to the output ports (3). However, in such scheme, the resulting configuration will allow long cascaded ALUs or even form ALU loops, which seldom appears in data dependency graphs in real programs due to locality of programs.

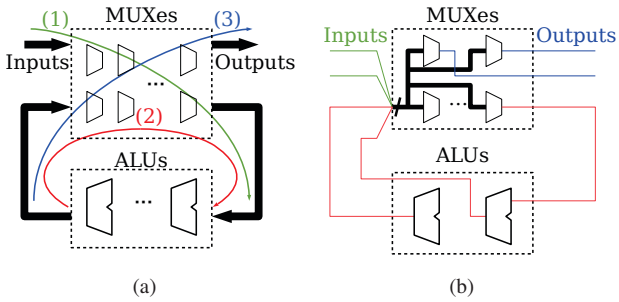


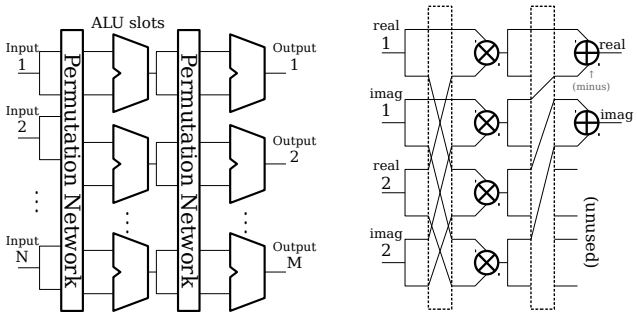
Fig. 2. A basic ALU cluster with reconfigurable architecture. (a) (a-1) Input data are sent to ALUs, (a-2) data are calculated and sent to other ALUs and (a-3) the data goes to the output ports. (b) is a more concrete illustration of the left one. In this figure we only show some representative connections.

Therefore our final design looks like Figure 3(a) and it is a 2-stage architecture. First the M data from SRAM are permuted and sent to the "ALU slots". The ALU slots are capable of holding one ALU which can be one of multiplier, adder, MAC or just bypass the data to next stage. The second stage are just the same as the first stage while it receives data from ALUs of previous stage and sends the ALU results to SRAM.

In our design, $N = M = 4$ and there are 4 ALU slots in each stage, enabling 8 ALUs to operate simultaneously. Also, there are 8 adders, 4 multipliers and 2 MACs for users to choose. Besides, although our design may lengthen the critical path, we add pipeline registers in the permutation network so that we can prevent from longer combinational critical path and make sure it meet the original timing constraint.

Because permutation networks only permute the original data, we duplicate each input twice into the permutation network to analogize registers which are read more than once in the programs. The number 2 is chosen because [14] shows that most of the temporary registers are accessed less than 3 times in programs.

As the preliminary assessment for applied research in future, we compile the network configuration offline now. For run-time compling the network in future work, we take refer to the methods in [15], in which either naive greedy manner or Recursive Dominator Split (RDS) is possible solution. We further reduce the number of switches in the permutation circuit because we duplicate the data. For instance, if 1 and 2 are the same data in Figure 1(b), then the switch connected directly to 1 and 2 will be useless and can be removed with



(a) Proposed 2-stage reconfigurable architecture: Each stage permutes inputs from the last stage to ALU slots. ALU slots can be configured to hold ALUs.

(b) The configuration of complex number multiplication: 4 streams are duplicated twice and the results come out from the two adders streamly.

Fig. 3. Proposed ALU cluster with permutation network.

relief. Eventually, totally 27 bits is used to represent one configuration in our design, 7 bits for the ALU slots and 20 bits for the permutation network. Such configurability is sufficient throughout all of the testcases we used. In the next part, we will elaborate how this circuit works by the same complex multiplication example.

Figure 3(b) shows a particular configuration for the complex multiplication example. First, we need to select 4 multipliers as first-column ALUs, and 2 adders as second-column. Then the first stage permutation network would permute the 8 signals of 4 streams into the 4 corresponding multipliers and the second stage would permute each outputs of the multipliers into the 2 adders. The 4 input streams are transformed into 2 output streams directly with such simple configuration.

C. Banked SRAM

The C54x in the experiment uses two 1024x16 srams for instruction memory and data memory instead of 16kB sram in the commercial one. On the consideration of simplicity, we left the problem when it comes to function size larger than 2kB for future work. Furthermore, in order to support our MIMDS architecture's ability to process many different streams at the same time, we need to expand the memory bandwidth by partitioning data memory into many banks. As described before, our proposed scheme supports up to 4 streams in parallel, so the original memory is partitioned evenly into 4 smaller ones. That is, our in-processor data memory is with four 256x16 srams instead of one 1024x16 sram. In our trial, the bank conflict problem is avoided with manual data layout. Data in complex functions such as FFT is interleaved in sram banks to promise the data correctness. Our partition scheme is inspired from NVIDIA GPU's register file partitioning scheme [16] as Figure 4 shows. Such multi-bank register files architecture supports high bandwidth that GPU's bulk threads access. We use the similar idea in our memory partition preventing performance bottlenecked from memory bandwidth.

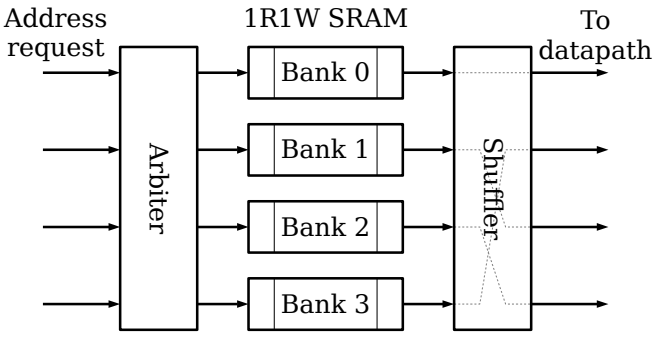


Fig. 4. Multi-bank data memory supporting wide I/O bandwidth.

TABLE I. C54X SYNTHESIS RESULT

Technology	TSMC 90nm CMOS		
Area (mm^2)	0.36		
Area overhead (mm^2)	0.0038	ALU units	0.028
		Reconf. network	0.010
Operating Frequency	230 MHz		
On-chip Memory	Instruction	(1024x16 dual-port SRAM)x1	
	Data	(256x16 dual-port SRAM)x4	

IV. EVALUATION

We select 5 representative functions in the DSPLIB as our workloads, including vector add (`add`), complex number multiplication (`cmul`), matrix multiplication (`mmul`), complex Fast Fourier Transform (`cfft`) and finite impulse response filter (`fir`).

To evaluate the performance of proposed architecture, these workloads are estimated with a hardware model. Both the baseline C54x and MIMDS architecture have their corresponding RTL model and synthesized with TSMC 90nm technology using Synopsys Design Compiler and SRAM are generated with TSMC 90nm memory generator. The synthesis result is shown in Table I. The area overhead is only 15% and our design can run with the same clock period because the architecture can be easily pipelined. Figure 5 shows the MIMDS execution cycles are averagely 63% less then the original C54x. Besides, the table shows that ALUs contribute 74% of the area overhead, which proves the efficiency of our datapath.

On the one hand, we observe that complex number multiplication outperforms the others significantly. In `cmul`, baseline architecture wastes lots of instructions to store the temporary value. We conclude that such 2-stage cascade instructions fit in our MIMDS architecture because such reconfigurable architecture reduce many redundant register access instructions and highly parallelize the datapath. On the other hand, we observe that as the workloads computation parameter grows, the performance boost are much higher, closer to its ideal parallelism bound.

What's more, we evaluate the data memory access counts in Table II, and find that at most 45% memory accesses are reduced by our MIMDS architecture, saving much SRAM access power. The access counts of `fir` and `mmul` remain the same because we only parallelize streams. Also, the `cmul` outperforms again, and the reason are similar to the one of its performance.

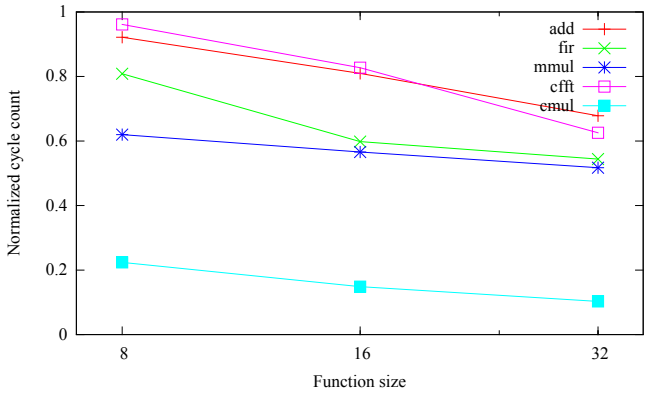


Fig. 5. DSPLIB execution time comparison: Normalized MIMDS execution cycles with respect to different workloads and loop count n.

TABLE II. SRAM ACCESS COUNT REDUCTION OF DSPLIB WITH OUR MIMDS EXECUTION.

Workload	add	mmul	fir	cfft	cmul
Reduction ratio	30%	0%	0%	32%	44%

Furthermore, compared to naive MUX-based network, our proposed permutation network saves 85% area while preserving sufficient connectivity. In our trial, merely implementing the MUX-based network costs upto 20% area of the original C54x design. The building block of the permutation network is synthesized with two multiplexers in this works. Actually, the switch can be modelled with a higher density customized cell. We left this part as our future work.

V. CONCLUSION

Again, our work proves that reconfigurable architectures are powerful. We verify that by adding little reconfigurable wires to TI C54x DSP, the DSP can support both its original functionality and MIMDS execution simultaneously and seamlessly. Our work greatly reduces SRAM accesses and execution cycles with little area overhead compared to the original design. Besides, the reconfigurablility of our circuit is based on the permutation network, which prevents large MUXes and high fanout wires. Although with some limitations, the permutation network is sufficient owing to locality of programs, and is pretty enough in our testcases. Our result has a great impact because dataflow schemes similar to our MIMDS execution also appear in many aspects. For instance, in graphics pipeline, we can stream geometries into our processing unit to perform 3D transformation or lighting speedily. We believe our result open another door in the promising research area of configurable computing.

REFERENCES

[1] A. Waksman, "A permutation network," *J. ACM*, vol. 15, no. 1, pp. 159–163, Jan. 1968. [Online]. Available: http://doi.acm.org/10.1145/321439.321449

[2] C54x DSP. [Online]. Available: http://www.ti.com/lsds/ti/dsp/c5000_dsp/c54x/products.page?paramCriteria=no

[3] J. Chen and S.-Y. Chien, "Crisp: Coarse-grained reconfigurable image stream processor for digital still cameras and camcorders," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 9, pp. 1223–1236, Sept 2008.

[4] T.-Y. Cheng, L.-G. Chen, and S.-Y. Chien, "Crisp-ii: Coarse-grained reconfigurable image stream processor for image-processing and intelligent operations in qfhd video cameras," in *Solid State Circuits Conference (A-SSCC), 2012 IEEE Asian*, Nov 2012, pp. 209–212.

[5] T.-H. Chen, J. Chen, T.-Y. Cheng, and S.-Y. Chien, "Crisp-ds: Dual-stream coarse-grained reconfigurable image stream processor for hd digital camcorders and digital still cameras," in *Solid-State Circuits Conference, 2009. A-SSCC 2009. IEEE Asian*, Nov 2009, pp. 193–196.

[6] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The chimaera reconfigurable functional unit," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 206–217, Feb 2004.

[7] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: Balancing efficiency & flexibility in specialized computing," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 24–35, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485925

[8] D. Burger, S. Keckler, K. McKinley, M. Dahlin, L. John, C. Lin, C. Moore, J. Burrill, R. McDonald, and W. Yoder, "Scaling to the end of silicon with edge architectures," *Computer*, vol. 37, no. 7, pp. 44–55, July 2004.

[9] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

[10] D. Burger, S. Keckler, K. McKinley, M. Dahlin, L. John, C. Lin, C. Moore, J. Burrill, R. McDonald, and W. Yoder, "Scaling to the end of silicon with edge architectures," *Computer*, vol. 37, no. 7, pp. 44–55, July 2004.

[11] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "Wavescalar," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 291–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956546

[12] Opencores. [Online]. Available: http://opencores.org/

[13] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," *SIGARCH Comput. Archit. News*, vol. 3, no. 4, pp. 126–132, Dec. 1974. [Online]. Available: http://doi.acm.org/10.1145/641675.642111

[14] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 235–246, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024723.2000093

[15] E. Chan, R. Ng, P. Sen, K. Proudfoot, and P. Hanrahan, "Efficient partitioning of fragment shaders for multipass rendering on programmable graphics hardware," in *Proceedings of the conference on Graphics hardware 2002*. Eurographics Association, 2002, pp. 69–78.

[16] J. Choquette, M. Gautho, and J. Lindholm, "Methods and apparatus for source operand collector caching," Jun. 20 2013, US Patent App. 13/326,183. [Online]. Available: http://www.google.com/patents/US20130159628