A Testbed Architecture for Multi-Accelerator Applications

Riley Wood
Computer Architecture Lab
Tufts University
rjw245@gmail.com

May 13, 2016

Abstract

This work was conducted under the supervision of Professor Mark Hempstead as part of an independent study in hardware accelerators. This paper summarizes my research on hardware accelerators and details my work on the development of a testbed architecture to support multi-accelerator applications. Computer architects are preparing to confront a recent limitation on the number of transistors on chip that can be used at any given time. This "utilization wall" will limit the extent to which computers can benefit from increasing transistor counts. Specializing the transistors on chip is one solution which seems to be gaining traction among the research community and in industry. As researchers investigate this option, they have begun to develop new accelerator-rich architectures (ARAs) to handle the challenges posed by specialization. For researchers to conclusively evaluate and compare these architectures, they will need a catalog of applications which use many accelerators in taxing ways. This work focuses on the design of an architecture upon which such applications can be built and architectural choices can be evaluated. The architecture is developed for the Zynq-7000 line of systems-on-chip and serves as a baseline architecture for ARA research.

1 Introduction

The Motivation for Specialization It used to be that chip power consumption would remain constant as transistors grew smaller due to a principle known as Dennard scaling [6]. Starting around 2006, Dennard scaling began to fail due to the fact that sub-micron sized transistors are more susceptible to leakage current and other non-ideal phenomena [12]. This means that placing more transistors on chip incurs greater power consumption, yet modern computers have reached a limit on the amount of power they can consume while still being kept cool. For example, the power consumption of Intel CPUs plateaued back in 2006 [8]. To continue increasing the number of transistors on chip and stay within power budget, designers will have to selectively under-clock or turn off portions of the chip. Thus, as the number of transistors on chip grows, a smaller and smaller percentage of them will be usable at any given time. This *utilization wall*, as it is called by Venkatesh et al. [17], threatens to limit the extent to which computers can benefit from increasing transistor counts in the future.

Several solutions to the utilization wall have been proposed, including shrinking the number of transistors on chip, "dimming" regions of the chip, developing new, more efficient transistor technologies, and specializing groups of transistors.

The first of these, shrinking transistor counts, clearly means the end of Moore's Law scaling. This has been called the "most pessimistic" solution to the problem [16], and understandably, many in industry are hesitant to make such a big concession.

"Dimming" technologies make the transistors on chip more energy-efficient, thereby increasing the number of transistors that can be used while staying within power budgets. However, these techniques often cause a decrease in performance. Dynamic voltage and frequency scaling (DVFS) is a common technique to reduce the power consumption of transistors when high performance is not required. Some work has been done to make DVFS even more effective for modern architectures [9]. Other work achieves energy efficiency by making transistors more reusable. For example, DySER from the University of Wisconsin-Madison is an element of the processor pipeline that can be reconfigured to implement one of multiple functions [7]. Because DySER is reconfigurable, energy is only expended on the data path that is currently configured. This saves energy compared to architectures where idle data paths still consume power.

The third approach—the development of new transistor tech—addresses the failure of Dennard scaling by moving away from CMOS. Some of the work in this field includes tunnel field-effect transistors (TFETs) [14] which use quantum tunneling to implement a switch, and nano electro-mechanical switches (NEMS) [2] which exploit the power efficiency of nano-mechanical technology.

Lastly, specialization addresses power limitations by spreading computation across an array of specialized, more efficient blocks of transistors, called accelerators. Rather than have a monolithic processor do the majority of computation and offload a small portion to specialized hardware (as is the norm today), the vision for specialization in the future has computation bouncing between many specialized regions of the chip [16]. Because there are more transistors on chip than can be used at any one time, area can be "spent" on accelerators specialized for certain computations. These accelerators use fewer transistors than a general-purpose processor to implement specific functionality. Thus, on a fully specialized system, the computer will use a

minimal number of transistors for all tasks, thereby maintaining performance gains despite the utilization wall.

Accelerators are already being adopted in industry. The International Technology Roadmap for Semiconductors (ITRS) released a report in 2007 which predicted that systems-on-chip will contain nearly 1500 accelerators by the year 2022 [1]. Today, accelerators can be found in common devices like the iPhone [15] and the PlayStation 3 [10]. Many look to specialization as a promising answer to the utilization wall, and clearly the approach is gaining traction, but for accelerators to solve the problem entirely, they will need to become even more pervasive and numerous in future architectures. Integrating accelerators in greater numbers and moving away from large, general-purpose processors requires significant work in the development of new architectures.

The Challenges for Accelerator-Rich Architectures The shift toward specialized computation brings about new challenges for architectures in areas such as on-chip communication, memory organization, resource sharing between cores, and others.

As computation becomes less centralized, it will be important for accelerators to communicate with one another without the assistance of a central controller (which would become a bottleneck). This is also known as streaming. To stream data between accelerators, they must all agree upon a common interface. Architectures have been developed to help with this. They use different structures like FIFOs [11], DMA engines [3, 10], or shared L1 caches [5] to interface the accelerators with each other. These architectures all implement streaming slightly differently, and it would be interesting to know which implementation performs better under which circumstances.

Architectural researchers are also concerned with the organization of memory in many-accelerator systems. Accelerators are frequently designed with their own private memories, called scratchpads, as this ensures fast access to the data they need. The Cell processor is one example of an accelerator-rich architecture in which the accelerators own private memories [10]. Some researchers have argued that organizing memory in private blocks is wasteful when the number of accelerators on chip is large, and that instead accelerators ought to share memory [11, 4, 5].

Yet another concern is how a finite number of accelerators will be shared among cores of chip multiprocessors (CMPs). These architectures approach the issue in different ways, such as by approximating the wait time to use a busy accelerator [3], or by simply having as many copies of each accelerator as there are cores [18]. This last option becomes less feasible as the number of unique accelerators grows larger.

Clearly, specialization is not without its challenges. Even so, the research community is evidently hard at work developing solutions to these issues.

The Value of Accelerator-Rich Applications As we figure out how to beat the utilization wall, it will be necessary to compare and contrast architecture designs to determine which perform best for a particular metric. To do this, one can compare the performance of each when running an application that uses many accelerators.

There are a few examples of applications which employ hardware accelerators to meet requirements they otherwise could not running in software alone. The RoboBee, an insect-scale, flapping-wing robot, uses hardware accelerators to meet stringent energy constraints and real-time performance demands [21]. The Bee uses accelerators in ways which would challenge accelerator-rich architectures. For example, the Bee makes heavy use of accelerator streaming for image processing. In addition, the researchers behind the Accelerator Store architecture present an application to test their own system: an embedded security device which uses accelerators for JPEG compression, AES encryption, and Fast Fourier Transform (FFT) calculation [11]. They use the application to demonstrate how performance is affected when sharing memory among accelerators. Assessing how these applications and others like them perform on a range of different architectures would provide valuable insight into the trade-offs inherent in certain designs. This motivates the development of more accelerator-rich applications.

Requirements for the Testbed Architecture A prerequisite to the development of accelerator-rich applications is the architecture which supports them. Because the main goal of developing these applications is to test them on a variety of architectures, one should first focus on developing an architecture which is easily configurable. This will allow researchers to investigate different architectural parameters with ease.

In the Tufts Computer Architecture Lab specifically, PhD candidates Parnian Mokri and David Werner are working on breaking down an application into accelerators of optimal granularity, and designing new memory interconnect for accelerated systems, respectively. To facilitate Parnian's work, the architecture must allow the user to customize which accelerators are incorporated in the system. To facilitate David's work, the accelerator interconnect must be modular so that it can be replaced with a custom solution.

2 Technical Discussion

In this section, I will discuss the details of my testbed architecture. The block diagram of the architecture is shown in Appendix A. The architecture includes several key components:

- A general-purpose processor,
- a DMA engine,
- several AXI-Bus crossbars,
- and three Fast Fourier Transform (FFT) accelerators.

Three was the maximum number of FFT accelerators that could fit on the selected board's FPGA given the available DSP resources. Only one type of accelerator was integrated because of time constraints—given more time, I would have included a diverse group of accelerators. Regardless, with these building blocks I was able to construct and evaluate an architecture upon which applications can be built and accelerator research can be conducted.

Hardware Requirements This architecture is intended to be instantiated on a Zynq-7000 All Programmable System-on-Chip [19]. These SoCs contain a general-purpose ARM processor and reconfigurable FPGA fabric. A ZedBoard [20] with a Zynq-7020 was used to implement this infrastructure. Depending upon the available onboard FPGA resources, the infrastructure may be able to support more or fewer accelerators.

The FFT Accelerator The FFT accelerator is derived from the strided FFT algorithm found in MachSuite [13], a diverse suite of workloads that are well-suited for implementation in hardware. The algorithm computes a 1024-point, complex FFT using the Cooley-Tukey "butterfly" method. The C code from MachSuite was synthesized to RTL using Vivado HLS, a high-level synthesis tool. Directives were used to specify a 32-bit AXI interface for the synthesized hardware. It takes 4 input arrays: the real and imaginary parts of the input, and the real and imaginary twiddle bits. The accelerator uses local scratchpad memory to do computation in-place, overwriting the first two arrays with the FFT output.

Modular Interconnect To facilitate researching on-chip communication and data transfer, the interconnect between the system's accelerators has been isolated and labeled **MODULAR_INTERCONNECT**. The block diagram has been designed to allow this interconnect to be easily swapped out with a custom solution. For now, this block is simply an AXI crossbar which forces accelerators to contend for access to the bus. This leaves much room for improvement, as accelerators often require high memory bandwidth.

Top-Level Architecture I used Vivado v2015.4 to design the block diagram of the top-level architecture. The Zynq Processing System (**Zynq PS**) is the general-purpose processor. It is connected via an AXI crossbar to the AXI Central DMA engine (**CDMA**) and also to the **Modular Interconnect** which is home to the accelerators. One important design decision was to establish a hierarchy in the AXI bus. In this hierarchy, the **Zynq PS** is a master to the **Modular Interconnect** and the **DMA**, which in turn is a master to the **Zynq PS**'s main memory and the **Modular Interconnect**. As the only master to the **CDMA**, the **Zynq PS** must handle all DMA transfers. Because the **CDMA** and **Zynq PS** are both masters to the **Modular Interconnect**, either one can communicate with the accelerators in the system. With this arrangement, one can compare the **CDMA** and **Zynq PS** in terms of data transfer rate to and from the accelerators. The **Zynq PS** can also send control signals to the accelerators directly, rather than be forced to do so via DMA. Interrupt signals from the **FFTs** and the **CDMA** are concatenated and wired to the IRQ port of the **Zynq PS**. This allows for the possibility of asynchronous computation and data transfer.

3 Evaluation

It is important to validate my design to ensure its functionality. Here I discuss my test procedure, my findings, and my conclusions on how well my design meets the requirements. The architecture evaluated here is shown in Appendix A.

```
static void compare_results(double real_hw[FFT_SIZE], double
      real_sw[FFT_SIZE], double img_hw[FFT_SIZE], double
     img_sw[FFT_SIZE]) {
  int i;
  for(i=0; i<FFT_SIZE; i++){
    if(real_hw[i] != real_sw[i] or img_hw[i] != img_sw[i]) {
      xil_printf("ERROR: Different results at bin %d\n", i);
    }
  }
}
```

Listing 1: FFT Validation Function

3.1 Validating the FFT Accelerator

After synthesizing the strided FFT from MachSuite [13] in Vivado HLS, it is important to verify that the accelerator actually computes an FFT correctly. To proceed further, I must verify that the FFT accelerator's results exactly match those of the FFT in software. My results show that these expectations are met—the FFT hardware accelerators compute exactly the same results as the FFT in software.

3.1.1 Methodology

The results of each FFT accelerator were compared to the results of the MachSuite C code to check that the accelerator works correctly. The MachSuite C code was copied and pasted into the Xilinx SDK where it was compiled for the Zynq processor. The Zynq processor then computed seven FFTs using the same inputs: one in software; the other six on each of the three hardware accelerator instances, both using a DMA engine and not. Results from each were checked against results from software using the function in Listing 1. The results of the hardware FFT should be passed as arguments `real_hw` and `img_hw`, and the results of the software FFT should be passed as `real_sw` and `img_sw`.

3.1.2 Results

No differences arose between the results of the software FFT and any of the hardware-accelerated FFTs. This means that all three hardware FFTs correctly implement the strided FFT algorithm.

3.2 Examining Accelerator-Rich System Performance Relative to Software

With the previous test, I showed that the FFTs compute the correct results. With this test, I demonstrate the usefulness of the DMA engine and clock scaling in improving the performance of the FFT in hardware versus in software.

3.2.1 Methodology

The FFT was re-synthesized using Vivado HLS with a target clock frequency of 120MHz. 120MHz was selected because it is the maximum frequency at which the DMA engine can operate on the Zedboard. The timing portion of the FFT C-synthesis report is printed in Figure 1. The block diagram shown in Appendix A was synthesized with the peripheral clock (`FCLK_CLK0` from the Zynq processor) set to 100MHz and then 120MHz. At each frequency, execution time was recorded for the FFT in software, in hardware using the DMA engine, and in

Clock	Target	Estimated	Uncertainty
ap_clk	8.33	6.72	1.04

Figure 1: Timing characteristics taken from the C Synthesis Report for MachSuite's Strided FFT, generated by Vivado HLS. Targeting a clock frequency of 120MHz.

```
XTime tStart, tEnd;
XTime_GetTime(&tStart);
// Region of interest goes here
XTime_GetTime(&tEnd);
int t = (tEnd - tStart);
xil_printf("Region of interest took %d timer cycles.\n",t);
```

Listing 2: XTime example

hardware not using the DMA engine. The XTime library from Xilinx was used to time sections of code, as shown in Listing 2.

The number of timer cycles spent computing each critical section was recorded. Dividing by the timer clock rate yields the time in seconds. The following critical sections were selected for timing:

- Sending inputs to the FFT
- Computation of the FFT
- Retrieving results from the FFT

The software does not need to spend time sending or retrieving data since it operates on arrays already available to it in main memory. The FFT accelerator, on the other hand, requires data to be copied to and from its scratchpad memory. The DMA engine is used to move inputs and outputs between main memory and the FFT's scratchpad. When not using the DMA, the Zynq processor must manually copy the data. In all cases, the hardware implementation is synchronous, meaning the Zynq processor initiates each input/output copy one at a time and then must wait while each transfer takes place.

3.2.2 Results

Figures 2 and 3 show the time spent executing the FFT in software versus each step in hardware at two different clock rates and with and without a DMA engine.

Focusing on the accelerators, one will notice that using the DMA drastically speeds up execution. On average, the DMA speeds up all data transfer by approximately 16x as compared to the processor manually copying data to the accelerator. Unsurprisingly, the time spent computing the FFT is unaffected by the use of a DMA engine. Figure 3 gives a more close-up view of how the hardware accelerator with DMA compares to software.

When increasing the clock frequency of the FFT accelerator, DMA engine, and AXI interconnect from 100MHz to 120MHz, one can see that every step of the hardware implementation speeds up, whether using the DMA or not. This is because the DMA and the Zynq processor can each only transfer data as fast as the interconnect will allow.

Finally, it is important to note that the software implementation is faster than hardware in all cases in terms of total execu-

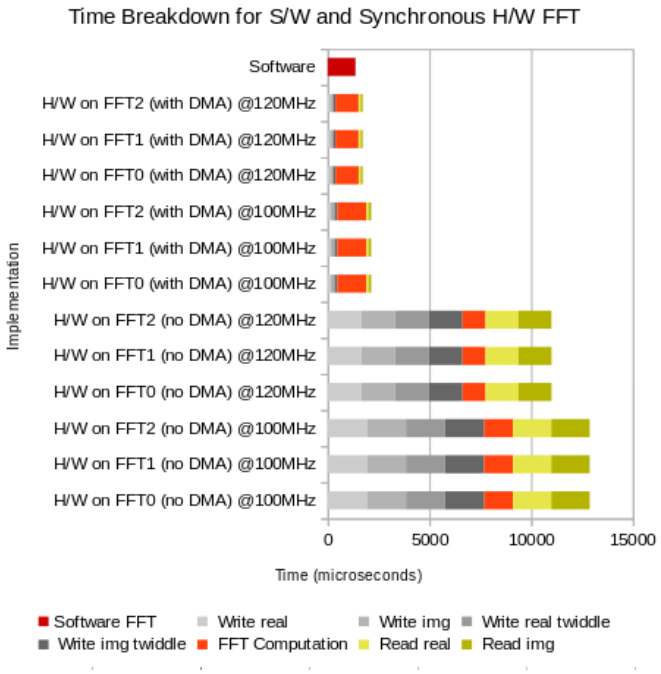

Figure 2: Time breakdown of an FFT done in software versus in hardware, showing all implementations (with/without a DMA engine, and with peripherals clocked at 100MHz and 120MHz). Each critical step is highlighted in a different color. Copying of input operands is highlighted in shades of gray, the actual computation in shades of red, and copying back results in shades of yellow.

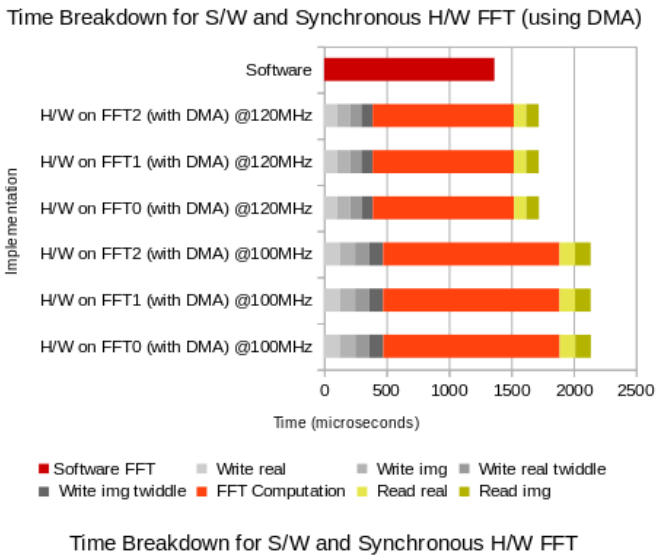

Figure 3: Close-up view of execution time for the hardware implementations with a DMA engine.

tion time. However, this includes the time spent copying data to and from the FFT accelerator. In Figure 3, one can see that at 120MHz, the actual computation (shown in red) is completed faster in hardware than in software. The synchronous data transfer which comes before and after computation lengthens execution time to the point where there is no speedup over the software implementation. Transferring data asynchronously could amortize this overhead by allowing the Zynq processor to do other useful work while data is moved by a DMA engine.

4 Project Execution Evaluation

My original task was to develop an application that uses many accelerators in a way that highlights their varied interactions in a many-accelerator architecture. Along the way, I discovered that designing the underlying architecture itself would take a significant amount of time and effort. I had to make decisions about how data would be moved between accelerators, how accelerators would interface with each other and the processor, how communication could be made asynchronous, and ultimately how to make my infrastructure most useful to other researchers in the Tufts Computer Architecture Lab. To make these decisions and implement them, I needed to learn a number of programs—such as the Xilinx SDK, Vivado, and Vivado HLS—and also research hardware accelerators, accelerator-rich architectures and applications, and general principles of architecture design. My work throughout the semester is catalogued in an online blog that can be found on the TCAL wiki, or at the following URL: `http://bit.ly/1R0IMJA`

Ultimately, I was unable to begin work on the application, but I have accomplished an important piece of this project by developing and validating the underlying architecture. To date, I have validated my design, highlighted areas for improvement and further development, and documented my work so that someone else can easily pick up where I have left off.

5 Conclusions

I have presented the need for applications that use multiple accelerators in varied data paths. With this work, I have successfully developed an infrastructure featuring a Zynq-7000 SoC processor which manages many accelerators. I have also demonstrated the extent to which a DMA engine can speed up accelerator workloads relative to software.

6 Future Work

The next step would be to build an application on top of this infrastructure. Aspects of the infrastructure could be tweaked to determine how various design decisions impact performance. Such tweaks could be making data transfer to and from the accelerators asynchronous, enabling streaming between accelerators, customizing the modular accelerator interconnect, and sharing the accelerators between the two cores of the Zynq PS.

Appendices

A Architecture block diagram

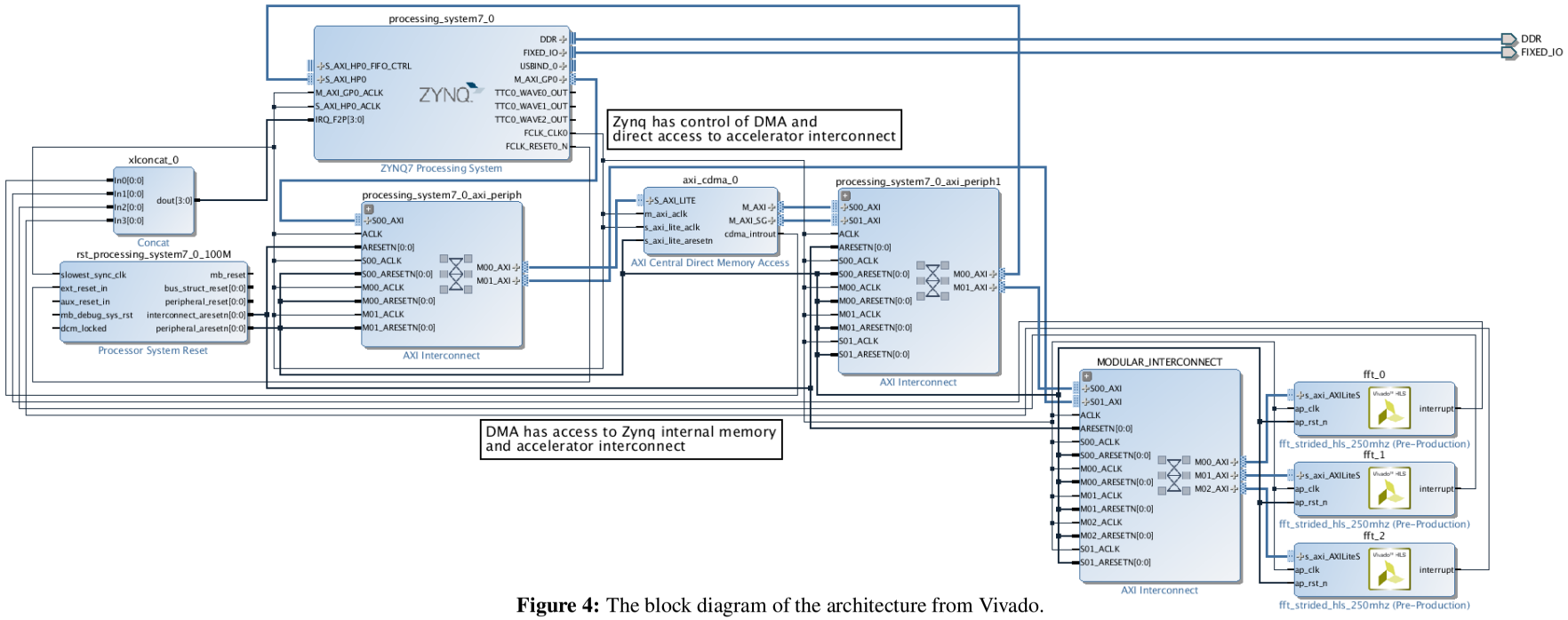

Figure 4: The block diagram of the architecture from Vivado. Composed of a Zynq processor, AXI Central DMA, AXI crossbars, and three FFT accelerators synthesized by Vivado HLS.

References

[1] International technology roadmap for semiconductors 2007 edition system drivers, 2007.

[2] F. Chen, M. Spencer, R. Nathanael, C. Wang, H. Fariborzi, A. Gupta, H. Kam, V. Pott, J. Jeon, T. J. K. Liu, D. Markovic, V. Stojanovic, and E. Alon. Demonstration of integrated micro-electro-mechanical switch circuits for vlsi applications. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 150–151, Feb 2010.

[3] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture support for accelerator-rich cmps. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 843–849, June 2012.

[4] J. Cong, M. A. Ghodrat, M. Gill, C. Liu, and G. Reinman. Bin: A buffer-in-nuca scheme for accelerator-rich cmps. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 225–230, New York, NY, USA, 2012. ACM.

[5] M. Dehyadegari, A. Marongiu, M. R. Kakoee, S. Mohammadi, N. Yazdani, and L. Benini. Architecture support for tightly-coupled multi-core clusters with shared-memory hw accelerators. *IEEE Transactions on Computers*, 64(8):2132–2144, Aug 2015.

[6] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[7] V. Govindaraju, C. H. Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 503–514, Feb 2011.

[8] J. Hruska. The death of cpu scaling: From one core to many and why were still stuck, Feb 2012.

[9] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, Feb 2008.

[10] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, May 2006.

[11] M. Lyons, M. Hempstead, G. Y. Wei, and D. Brooks. The accelerator store framework for high-performance, low-power accelerator-based systems. *IEEE Computer Architecture Letters*, 9(2):53–56, Feb 2010.

[12] A. McMenamin. The end of dennard scaling, Apr 2013.

[13] B. Reagen, R. Adolf, Y. S. Shao, G. Y. Wei, and D. Brooks. Machsuite: Benchmarks for accelerator design and customized architectures. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 110–119, Oct 2014.

[14] A. C. Seabaugh and Q. Zhang. Low-voltage tunnel transistors for beyond cmos logic. *Proceedings of the IEEE*, 98(12):2095–2110, Dec 2010.

[15] Y. S. Shao, S. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks. Toward cache-friendly hardware accelerators. *Proc. Sensors to Cloud Architectures Workshop (SCAW), in conjunction with HPCA 2015*, 2015.

[16] M. B. Taylor. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1131–1136, June 2012.

[17] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, 2010. ACM.

[18] C. F. Webb. Ibm z10: The next-generation mainframe microprocessor. *IEEE Micro*, 28(2):19–29, March 2008.

[19] Xilinx. Zynq-7000 all programmable soc. `http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`.

[20] ZedBoard.org. Zedboard. `http://zedboard.org/product/zedboard`.

[21] X. Zhang, M. Lok, T. Tong, S. Chaput, S. K. Lee, B. Reagen, H. Lee, D. Brooks, and G. Y. Wei. A multi-chip system optimized for insect-scale flapping-wing robots. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*, pages C152–C153, June 2015.