Circuits and Systems, **2011, 2, 358-364**
doi:10.4236/cs.2011.24049 Published Online October 2011 (http://www.scirp.org/journal/cs)

Scientific Research

Polymorphic Computing: Definition, Trends, and a New Agent-Based Architecture

David Hentrich, Erdal Oruklu, Jafar Saniie
Department of Electrical and Computer Engineering, Illinois Institute of Technology Chicago, Illinois, USA
E-mail: *erdal@ece.iit.edu*
Received May 22, 2011; *revised June* 15, 2011; *accepted July* 15, 2011

Abstract

Polymorphic computing is widely seen as next evolutionary step in designing advanced computing architectures. This paper presents a brief history of reconfigurable and polymorphic computing, and highlights the recent trends and challenges. A novel polymorphic architecture featuring programmable memory event triggers and a new concept of control agents is proposed. This architecture can provide dynamic load balancing, distributed control, separated memory and processing fabrics, configurable memory blocks, and task-optimized computation.

Keywords: Polymorphic Computing, Reconfigurable Computing, Agents, Processing Fabric

1. Introduction

Microprocessor performance has advanced at a staggering pace during the past two decades. This can be attributed to:

1) Circuit architectural improvements,
2) Scaling of transistor sizes down, and
3) Scaling of clock frequency up.

Historically, each of the categories has contributed equally to the general performance increase [1]. Unfortunately, the rate of improvement in all of these areas is slowing or showing signs of slowing. Continual innovations in these areas are required to maintain the pace of improvement.

Polymorphic computing is a circuit architectural improvement technique that promises to improve overall computing performance. This work presents a definition of polymorphic computing, a brief history of the field of polymorphic computing, a summary of the current trends, a set of views on the current state of the field, and a novel polymorphic architecture.

2. Definition and History of Polymorphic Computing

The definition of a polymorphic computer is a computing machine that can dynamically arrange the underlying hardware computing architecture model in both time and space to match the computational demands of the moment. **Figure 1** shows how polymorphic computing sits in the set of all types of computation.

General computing is the set of all possible types of computation (*i.e.* any physical system that has a set of inputs and outputs). *Static computing* is the subset of general computing where the program (transfer function) is fixed in a device. Examples of static computing are simple NAND circuits, ASICs, and computers where the program is ROM'ed. On the other hand, *programmable computing* is composed of the set of computing devices where the program (transfer function) is intentionally changeable after manufacturing time. The programs may be hardware-based and/or software-based. *Reconfigurable computing* is the proper subset of general computing where the hardware configuration can be changed after manufacturing time. PLDs and FPGAs are the best examples of reconfigurable computing. *Polymorphic computing* is the proper subset of reconfigurable computing that includes devices that can reconfigure their hardware in time and space during runtime. FPGAs that can be configured only at startup time are reconfigurable computing devices, but not polymorphic computing devices. However, FPGAs that can be partially configured during runtime are both reconfigurable and polymorphic computing devices.

The history of polymorphic computing begins with the history of reconfigurable computing. The earliest thoughts for the creation of a reconfigurable computer were from Gerald Estrin of UCLA in 1960 [2]. His idea was to create a computer where the hardware could be

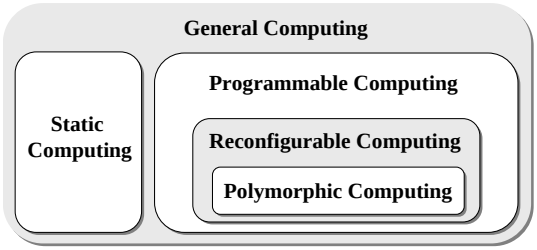

Figure 1. Categories of computing.

reconfigured to match the computational demands of the program. (In those days, a single computer filled an entire room and a single processor inhabited one or more equipment racks.) Research work on reconfigurable computing continued at UCLA under Estrin's leadership through the 1960s and 1970s [3]. Estrin's idea was about 40 years before its time and pre-dates the invention of the microprocessor.

The industrial origins of reconfigurable computing lie in the creation of programmable logic devices (PLDs) in the 1970s. These early devices were simply fixed arrays of AND and OR gates where the connections could be configured (usually just once) after manufacturing time. The key ideas that emerged from PLDs were that hardware could be configured by the users rather than the manufacturer and that the configuration of the circuits could be generated using software. PALASM and ABEL are early examples of the languages and tools for generating custom circuits in PLDs [4].

The next big advance in reconfigurable computing was the invention of the field programmable logic array (FPGA) in 1984 by Ross Freeman (one of the founders of Xilinx, Inc.) [5]. The important innovations were:

1) a programmable signal routing fabric,

2) a flexible logic cell that could perform any logic function,

3) arranging the logic cells in a tiled manner, and

4) configuration of the configurable logic at device start-up time.

Freeman's seminal observation was that the transistor density penalties of configurable logic devices would be more than offset by the transistor manufacturing density increases described by Moore's law and the efficiencies realized in circuit design effort. FPGAs are the general template for reconfigurable computing and polymorphic computing devices.

Polymorphic computing is the next logical step in the advancement of reconfigurable computing. Essentially, the hardware architecture can be deliberately modified during runtime to improve performance. The performance improvements all come from exploitations of parallelization (coarse-grained and/or fine-grained parallelism). A general trend (but not a rule) is that polymorphic computers are similar to FPGAs with small microprocessors in the place of configurable logic blocks.

In the late 1990's and early 2000's, DARPA funded several promising polymorphic computing architectures: Raw, Smart Memories, TRIPS, and MONARCH. Of these, Raw and MONARCH have transitioned to commercial products.

The Raw architecture was developed at MIT and is a replicated series of identical tiles arranged in a grid [6] [7]. Each tile contains a programmable compute processor and programmable network interfaces. Of the four DARPA sponsored projects, Raw most resembles an FPGA with full processors in the place of logic blocks. A primary contribution of the architecture was to keep critical wire lengths small since the length of wires is not scaling down as quickly as transistor geometries and wire resistance increases as the wires get smaller [8]. This was accomplished by keeping the individual processors small and strictly limiting the network interfaces to only point-to-point connections between adjacent tiles. A company called Tilera now offers Raw architecture chips commercially. Thus far, the Tilera chips have seen success in network switches.

Smart Memories is a polymorphic computing architecture from Stanford University [9,10]. Each tile contains four processors. Each individual processor is paired with a private memory fabric that can be configured as standard addressable memory, cache, streaming memory, and (in a later version of the design) transactional memory [11]. Different combinations of these configurations are available simultaneously. The primary contribution of this project was the notion that memories, as well as processing units, can be configured to exploit parallelism to improve performance.

TRIPS is a polymorphic computing architecture from the University of Texas at Austin [12-14]. Its primary contributions were showing that a dataflow architecture [15] can be used as the basis of a polymorphic computer, showing how to implement a single processor in a fundamentally parallel fashion, and demonstrating that a von Neumann instruction set can co-exist with a dataflow instruction set. Dataflow instruction set architectures are an idea from the mid-1970's that allows data to be executed upon as soon as it is available. It inherently supports parallel execution. Dataflow is a concept that is still ahead of its time. It promises vast parallelization of programs, but no practical implementation of a dataflow machine has yet fully emerged.

MONARCH is a combination of the University of Southern California's Data IntensiVe Architecture (DIVA) RISC processor system [16-18] and Raytheon's Field Programmable Compute Array (FPCA) [19-21]. The FPCA is essentially a systolic array of arithmetic and

memory units, and is the portion of the architecture that makes MONARCH a polymorphic computer. MONARCH's primary contribution was the demonstration that a polymorphic computing fabric can be based on a simpler execution unit than a full microprocessor (like the other three DARPA funded architectures). MONARCH is currently in production and is used primarily for military signal processing applications. For a list of other polymorphic architectures, see also Hartenstein [22].

3. Current Trends in Polymorphic Computing

A study of the existing architectures reveals the following observations and trends in current polymorphic computer design:

1) Polymorphic hardware architectures strongly tend to be tile based. This allows designs to be scaled up by simply adding more tiles.

2) No clear "best processing cell" type has yet emerged.

3) Critical circuit path lengths are intentionally limited to roughly the diameter of a tile. Smaller critical path lengths allow higher clock frequencies to be utilized.

4) Network links tend to be point-to-point connections between only directly adjacent tiles. This is an easily scalable network strategy for tile based arrays. It also supports the trend of limiting the length of critical paths in the system.

5) Processing elements are trending toward simpler designs compared to today's single-core and multi-core processors.

6) Configurable memories are an emerging trend.

7) Algorithms tend to be statically scheduled and placed in the computing fabric. Programs tend to be compiled and mapped to array elements at compile time, not runtime.

8) Polymorphic architectures require extensive compiler and software support.

9) Polymorphic computers support multiple different programming models. Their fabrics tend be configurable into SIMD units, pipelines, and systolic architectures.

10) There is a clear trend towards hybrid computer architectures. For example, Raw tiles are a combination of a compute processor and a network processor. MONARCH is a collection of RISC processors combined with a configurable systolic array.

11) Polymorphic architectures tend to have both current and future scaling strategies. Most designs have currently available options to connect multiple chips together. Additionally, tiled designs scale up easily by adding more tiles.

4. Issues and Views

With regards to current polymorphic architectures, the authors perceive that:

1) Dynamic processing balancing is not addressed by current architectures. Process mapping is currently handled as a design-time and compile-time problem.

2) Performance monitoring capabilities in current architectures are lacking. Monitoring is necessary in order to perform dynamic load balancing.

3) Processing control is either centralized or determined pre-runtime. These are generally non-scalable techniques. As systems get larger, centralized control will become a bottleneck. Distributed control will become necessary. Additionally, dynamic control will require runtime decisions.

4) The rule, "direct communication links are strictly limited to only between directly adjacent tiles" is too restrictive. This rule is in place in tiled architectures because it fits nicely with the tiling scalability strategy and it helps limit the length of critical path wire lengths. However, this can route some communication through disinterested tiles and increase communication latencies. This rule could be relaxed to allow direct communication between neighboring, non-adjacent tiles.

5) The single processors at the heart of the tiles may be too complicated. Most processing elements are pipelined. This could prove to be too complicated for processing elements.

6) Pure meshes of processor may not necessarily scale with mesh size. The current square array tiling strategy cannot be efficiently scaled indefinitely. All inputs and outputs must enter and exit, respectively, through the edges of the array. It is conceivable that arrays could be scaled so large that it would be rare for execution graphs to penetrate very far into the array before completing and being routed back out. Interior array elements may be underutilized. Other array geometries should be considered, such as rectangular arrays.

7) Debugging strategies must be built into polymorphic systems. Polymorphic computing systems are unavoidably complex. The ability to view machine state and capture problems as they occur is vital.

5. A New Polymorphic Architecture

Given the trends and views in the current state of polymorphic computing, a new polymorphic computing architecture is proposed (shown in **Figure 2**) in this work. At a high level, the architecture is partitioned into a configurable processing fabric and a configurable memory fabric connected via a crossbar bus. The crossbar contains a large number of channels and provides connectivity for the entire system, including external processors and the interrupt/event controller. In addition, the crossbar arbiter provides the ability to "park" channels on particular connections between the processing fabric and

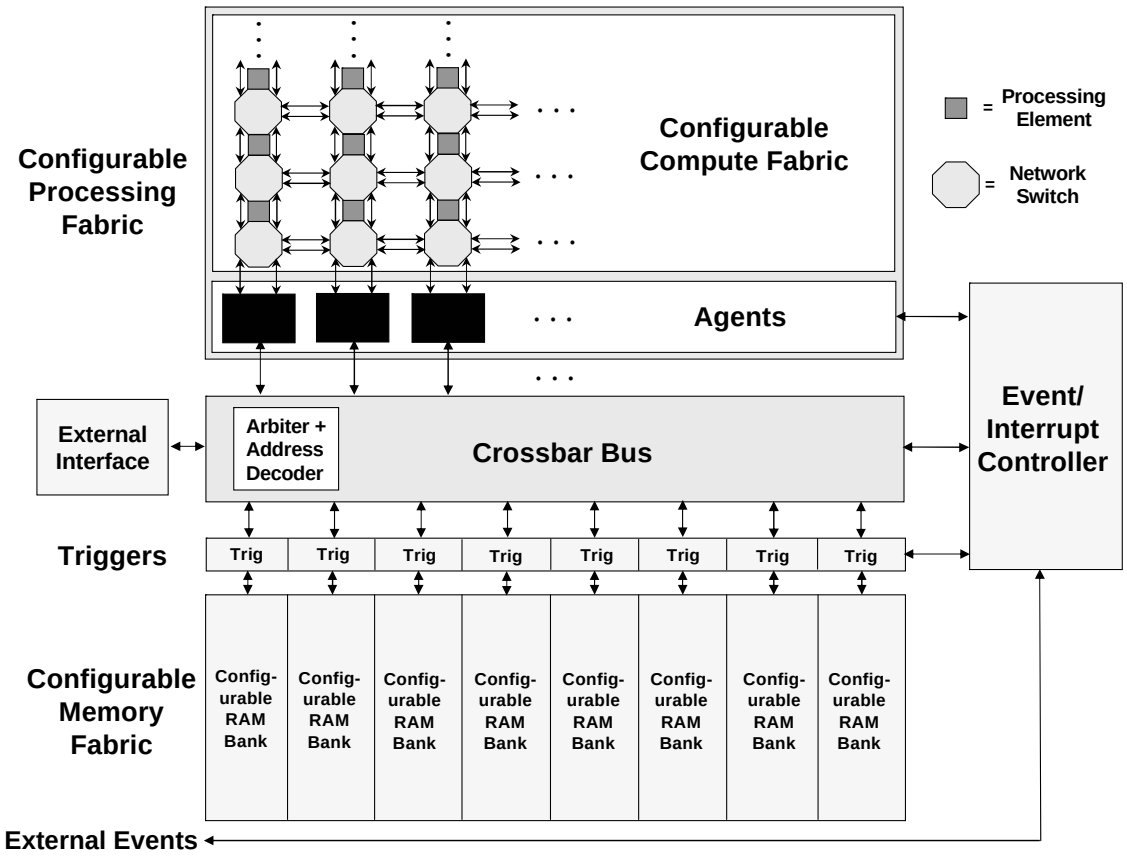

Figure 2. Proposed polymorphic architecture.

the memory fabric to support consistent and deterministic memory access times.

5.1. Memory Fabric

The memory fabric is composed of two types of components: independent, configurable memory banks and event triggers. Each memory bank has its own independent connection to the crossbar. In addition, each memory bank is associated with its own set of event triggers. From a configuration perspective, each memory bank can be selected as a traditional addressable memory, a streaming memory (FIFO and burst modes), a transactional memory [11], or a combination of the different modes. Caching has intentionally been omitted as an operational mode for deterministic memory access performance reasons. It is also expected that if a "wider" data path is needed, that the memory banks could be ganged together into a single "wider" memory bank.

The event triggers are mechanisms that associate arbitrary addresses within a memory bank with system events. They are akin to trace points and data breakpoints in modern embedded microprocessors. These enable writes to a particular memory location to be detected and communicated to interested entities throughout the system. The event triggers are a unique contribution of this work.

5.2. Processing Fabric

The processing fabric is abstracted into two partitions: a compute fabric and a layer of agents between the compute fabric and the rest of the system (*i.e.* the bus and the memories). The compute fabric is composed of a regular array of processing elements. These can be arithmetic logic units or simple microprocessors. The role of an element in the compute fabric is to receive operands, perform arithmetic operations upon them, and output the results. Memory accesses are not an intended role for compute fabric elements. The processing elements are intended to be workers with only a very local view of the system. They should only know how to do their assigned jobs, know on which of their ports to receive inputs, and know upon which of their ports to place their outputs.

The agents have a more global view of the system. They are the "department managers" in the overall computational enterprise. Their duties are to monitor for events that are relevant to their assignments, perform

relevant memory accesses (both input and output), deliver/sequence inputs into the compute fabric, receive outputs from the compute fabric, perform their own transform functions to the data, and route the results to the appropriate system location (which could be the memory fabric, another agent, or an external location accessible via the crossbar). The agents are expected to be full microprocessors with their own local memories.

One of the reasons for distinguishing between agents and processing elements is to recognize the distinction between I/O bound algorithms and compute-bound algorithms. The act of memory accesses (I/O) and the act of computation are fundamentally different operations. Load/store computing architectures (RISC) have long recognized this distinction by providing completely separate instructions for memory accesses and arithmetic operations. For example, search algorithms are typically memory-bound operations and matrix operations are typically compute-bound algorithms. Concurrent execution opportunities in both algorithm types can usually be exploited to realize performance gains. In the case of a search algorithm, most of the effort is in the memory accesses. The search can typically be partitioned into smaller concurrently executing jobs with an agent assigned to each independent search effort. In this case, there is typically little reason to involve the processing elements. The search may be executed with only a collection of agents. On the other hand, matrix operations require more arithmetic operations than memory accesses and are easily parallelized. In this example, it is expected that there would be one or more agents assigned to the memory accesses and quite a few processing elements assigned to exploit concurrency in the arithmetic.

Another reason for distinguishing between agents and processing elements is geometric. Most processing elements are typically buried in the interior of a compute fabric. They tend to be relatively distant from the memories and memory access mechanisms (buses). The elements that are in the most convenient place to access memory are the elements on the edge of a compute fabric. Consequently, the edge elements need to be burdened with memory access circuitry. However, interior elements don't necessarily need to be burdened with memory access circuitry if they are directly given their inputs by neighboring elements. In this case, interior elements can be made smaller and simpler than the fabric periphery elements.

Yet another reason for distinguishing between agents and processing elements is managerial. A polymorphic system is expected to dynamically monitor loading and algorithmic demands, and continually adjust its architecture appropriately. Agents participate in this activity in a distributed manner. However, not every element in the fabric needs to participate in this activity and be burdened with this capability (*i.e.* extra circuitry).

The compute fabric itself (not including the agents) is composed of two types of elements: processing elements and network switches. Within the compute array itself, the roles of processing and routing are logically separated. Consequently, there are two types of elements within the array: processing elements and network elements. (Note, the agents are considered part of the overall processing fabric, but they are not included in the compute fabric subset.)

5.3. Data and Control Networks

There are two classes of networks within the compute fabric: a data network and a control network. All processing data are intended to be transmitted on the data network and all configuration, management, monitoring, and debug information are intended to be transmitted on the control network.

The data network is a configurable routing fabric composed of one or more channels that may be either circuit-switched or packet-switched depending on the implementation. All data network routing is performed by the network switches. The data network connectivity strategy is shown in **Figure 3**. The network switches are connected with their neighboring network switches and neighboring processing elements. However, processing elements are only connected with the neighboring network switches in the north and south directions. Processing elements are not directly connected together because they do not participate in data network routing.

The control network is a static network composed of individual buses terminated on the system crossbar (see **Figure 4**). Collectively, the buses map all agents, network switches, and processing elements into a global address space. Each "column" in the processing fabric will get its own control bus with the agent acting as the bus arbiter.

5.4. Polymorphism

The proposed system is polymorphic because the agents and processing elements can be reconfigured into different models. For instance, a pipelined processor might be created with an agent and a single column of processing elements; a SIMD unit could be created by arranging several agents and processing units together in a parallel fashion; and a systolic array could easily be configured from the processing fabric. Collectively, the array could be configured to cooperatively work on a single problem, or partitioned into independent subunits to work on different problems utilizing different processing models.

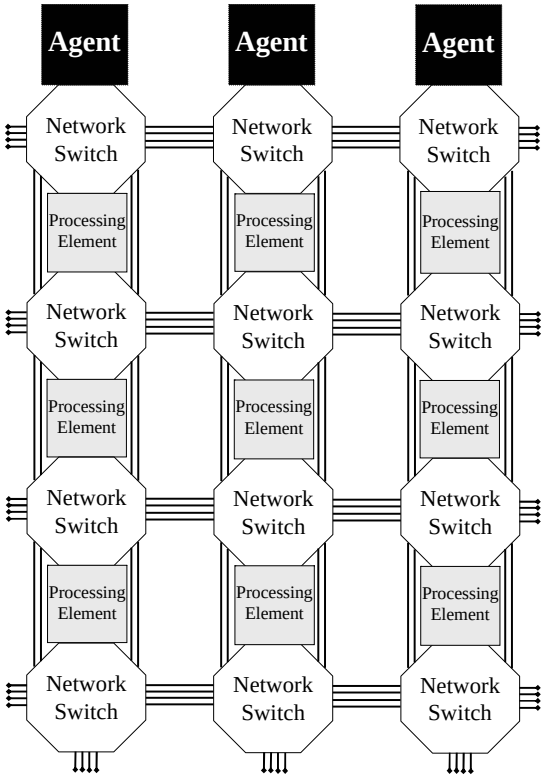

Figure 3. Data network.

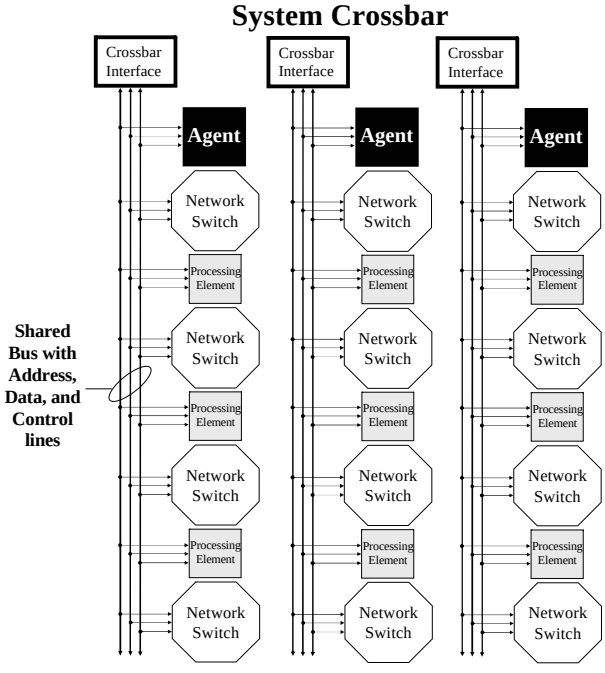

Figure 4. Control Network.

The monitoring and load balancing capabilities of the system are largely software-based. However, a series of programmable counters are also built into the agents and processing elements to facilitate system monitoring.

6. Conclusions

Overall, this work presents a definition of polymorphic computers, briefly sketches the history and development of the field, presents a list of trends occurring in the field, lists a series of views on the current state of field, and presents a novel polymorphic architecture. The significant contributions of the architecture are a clean partitioning between memory and computation, a method for globally detecting system events (the triggers), the concept of an agent, and a clear division of roles between agents and processing elements.

7. References

[1] D. Harris, "Skew-Tolerant Circuit Design," Morgan Kaufmann Publishers, Waltham, 2001.

[2] G. Estrin, "Organization of Computer Systems—The Fixed Plus Variable Structure Computer," *Proceedings of the Western Joint Computer Conference*, New York, 3-5 May 1960, pp. 33-40.

[3] G. Estrin, "Reconfigurable Computer Origins: The UCLA Fixed-Plus Variable (F+V) Structure Computer," *IEEE Annals of the History of Computing*, Vol. 24, No. 4, 2002, pp. 3-9. doi:10.1109/MAHC.2002.1114865

[4] M. A. Baker and V. J. Coli, "The PAL20RA10 Story—The Customization of a Standard Product," *IEEE Micro*, Vol. 6, No. 5, 1986, pp. 45-60. doi:10.1109/MM.1986.304713

[5] R. Freeman, Configurable Electrical Circuit Having Configurable Logic Elements and Configurable Interconnects, US Patent No. 4,870,302, 26 September 1989.

[6] A. Agarwal, "Raw Computation," *Scientific American*, Vol. 281, No. 2, pp. 60-63. doi:10.1038/scientificamerican0899-60

[7] M. B. Taylor, *et al*., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, Vol. 22, No. 2, 2002, pp. 25-35. doi:10.1109/MM.2002.997877

[8] R. Ho, K. W. Mai and M. A. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, Vol. 89, No. 4, 2001, pp. 490-504. doi:10.1109/5.920580

[9] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," *Proceedings of the* 27*th International Symposium on Computer Architecture*, Vancouver, 14 June 2000, pp. 161-171.

[10] A. Firoozshahian, A. Solomatnikov, O. Shacham, Z. Asgar, S. Richardson, C. Kozyrakis and M. Horowitz, "A Memory System Design Framework: Creating Smart Memories," *Proceedings of the* 36*th Annual International Symposium on Computer Architecture*, New York, 2009, pp. 406-417.

[11] M. Herlihy and J. E. B. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures," *Proceedings of the* 20*th Annual Symposium on Computer Architecture*, San Diego, 16-19 May 1993, pp. 289-300. doi:10.1109/ISCA.1993.698569

[12] D. Burger, S. W. Keckler, K. S. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald and W. Yoder, "Scaling to the End of Silicon with EDGE Architectures," *IEEE Computer*, Vol. 37, No. 7, 2004, pp. 44-55.

[13] R. McDonald, D. Burger, S.W. Keckler, K. Sankaralingam and R. Nagarajan, "TRIPS Processor Reference Manual," Technical Report, Department of Computer Sciences, The University of Texas at Austin, Austin, 2005.

[14] 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 (Extended Technical Report)," Technical Report TR-08-31, Department of Computer Sciences, The University of Texas at Austin, Austin, 2008.

[15] J. B. Dennis and D. P. Misunas, "A Preliminary Architecture for a Basic Dataflow Processor," *Proceedings of the* 2*nd Annual Symposium on Computer Architecture* New York, 1975, pp. 126-132. doi:10.1145/642089.642111

[16] J. Draper, J. Sondeen, S. Mediratta and I. Kim, "Implementation of a 32-Bit RISC Processor for the Data-Intensive Architecture Processing-in Memory Chip," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, San Jose, 17-19 July 2002, pp. 163-172. doi:10.1109/ASAP.2002.1030716

[17] J. Draper, J. Sondeen and C. W. Kang, "Implementation of a 256-Bit Wide-Word PROCESSOr for the Data-Intensive Architecture (DIVA) Processing in Memory (PIM) Chip," *Proceedings of the* 28*th European Solid-State Circuits Conference*, *Florence*, 2002, pp. 77-80.

[18] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim and G. Daglikoca, "The Architecture of the DIVA Processing-in-Memory Chip," *Proceedings of the* 16*th International Conference on Supercomputing* 2002, New York, 2002, pp. 14-25.

[19] J. Granacki, and M. Vahey," MONARCH: A Morphable Networked Micro-ARCHitecture," Technical Report, USC/Information Sciences Institute and Raytheon, Marina del Rey, May 2003.

[20] J. J. Granacki, "MONARCH: Next Generation SoC (Supercomputer on a Chip)," Technical Report, USC/Information Sciences Institute, Marina del Rey, February 2005.

[21] K. Prager, L. Lewins, M. Vahey and G. Groves, "World's First Polymorphic Computer—MONARCH," 11*th Annual High Performance Embedded Computing Workshop* 2007, 2007. http://www.ll.mit.edu/HPEC/agendas/proc07/agenda.html

[22] R. Hartenstein, "A Decade of Reconfigurable Computing: A Visionary Retrospective," *Proceedings of the Conference on Design, Automation and Test in Europe* 2001, Munich, 13-16 March 2001, pp. 642-649. doi:10.1109/DATE.2001.915091

