Birth and Adolescence of Reconfigurable Computing: A Survey of the First 20 Years of Field-Programmable Custom Computing Machines

Kenneth Pocek*, Russell Tessier†, and André DeHon‡
**Los Altos, CA, Email: kpocek@ieee.org*
†*Dept. of Electrical and Computer Engineering, Univ. of Massachusetts, Amherst, MA, Email: tessier@ecs.umass.edu*
‡*Dept. of Electrical and Systems Engineering, Univ. of Pennsylvania, Philadelphia, PA, Email: andre@ieee.org*

***Abstract*—For 20 years, the International Symposium on Field-Programmable Custom Computing Machines (FCCM) has explored how FPGAs and FPGA-like architectures can bring unique capabilities to computational tasks. We survey the evolution of the field of reconfigurable computing as reflected in FCCM, providing a guide to the body-of-knowledge accumulated in architecture, compute models, tools, run-time reconfiguration, and applications.**

***Keywords*-FPGA, Reconfigurable Computing**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) were introduced in the mid-1980s (e.g. [1]) as a larger capacity platform for glue logic than their Programmable Array Logic (PAL) ancestors. By the early 1990s, they had grown in capacity and were being used for logic emulation (e.g. Quickturn [2]) and prototyping. Furthermore, the Splash [3], PAM [4], and CAL [5] research efforts were exploring the use of FPGAs as computing devices and were starting to demonstrate early success. By 1992, the notion of building a "custom" computing machine tailored to a particular compute task using the emerging capacity of these FPGAs looked attractive. With this motivation, the *Workshop on FPGAs for Custom Computing Machines* was conceived as a unique forum to bring together and promote the new community exploring how to use FPGAs as computational engines.

The original FCCM workshop evolved into a regular symposium and refined its name to the *International Symposium on Field-Programmable Custom Computing Machines* to reflect a broader agenda than just FPGAs. In 1993, few conferences included FPGA implementations of computations and there were significant speculative and radical elements to the conference. Nonetheless, FCCM quickly became the central conference for research exploring FPGAs for computing; that is, reconfigurable computing. While many conferences now include papers on FPGA computing applications, FCCM remains the premier forum for understanding temporal aspects of FPGAs—applications that change their configuration throughout their operational lifetime, or run-time reconfiguration (RTR). Central conference themes include characterizing the benefits of FPGA-based computations compared to alternative architectures, understanding what applications might benefit from FPGAs, exploring how reconfigurable architectures might evolve to better support computations, developing languages and tools for exploiting FPGAs, and sharing techniques and best practices for implementing designs on FPGAs.

In 2013, the idea of using FPGAs for computation is no longer radical. FPGA implementations of applications are now prevalent in domain-specific conferences for signal processing, cryptography, arithmetic, scientific computing, and networking. There are dozens of conferences and workshops that explore the use of reconfigurable platforms and two journals dedicated to the topic (i.e., *ACM Transactions on Reconfigurable Technology and Systems*, *International Journal of Reconfigurable Computing*). Commercial acceptance is growing as illustrated by numerous products that employ FPGAs for more than just glue logic.

In this article, we review the contributions of FCCM over its first 20 years. We identify and trace the evolution of major themes, provide some historical context, relate the development to broader technology trends, and offer some speculation on where reconfigurable computing may go next. The description that follows can be seen more as a guide to FCCM contributions than a stand-alone digest of the body-of-knowledge. As such, this is a broader companion to the endorsements in the FCCM20 volume of most significant papers from the first 20 years of FCCM. We start with a technology review to provide historical perspective on the evolution of the underlying technology and associated landscape (Sec. II). We then review the exploration and evolution of architectures for reconfigurable computing machines (Sec. III). Sec. IV reviews what the conference has examined about how we program these reconfigurable computers. We then review developments in tools to automate and optimize design for reconfigurable computers (Sec. V) and models and tools for RTR (Sec. VI). Sec. VII highlights important application domains. We conclude with some final observations in Sec. VIII.

A. Notation

Since all the remaining references are to papers that have appeared in FCCM, we use a short-hand citation of the form [NameYYYYpPPP], where "Name" is the last name of the first author, YYYY is the conference year of publication, and PPP is the page number. Papers selected for inclusion in the FCCM20 volume of most significant papers from the first 20 years of FCCM are denoted with a ⋆.

II. Technology Perspective

This is a 20 year retrospective. In human terms 20 years represents one generation. In semiconductor manufacturing technology terms 20 years represents over ten technology generations! This amazing evolution has driven computer technology to startling advances as shown in Table I. Today's average home personal computer (PC) has the power of earlier multi-million dollar supercomputers with much better storage, speed and graphics!

Let's first take a look at conventional PC technology. In 1993, only 40 million PCs were sold world-wide. Although the growth rate of PC sales has slowed significantly due to smart phones and tablet sales, the yearly sales are still slightly over 350 million units. The typical home PC in 1993 had a 60 MHz 486 microprocessor with 8 MB of RAM, a 250 MB hard drive, and 2 MB of video RAM. It cost nearly $4,000. It could barely play the newly-released game of Doom. Today's mid-range gaming PC is driven by an Intel Quad Core i5 microprocessor operating at 3.3 GHz. It has 8 GB of RAM, a 3 TB hard drive, and a graphics processing unit (GPU) like the GeForce GTX 560Ti with 384 simple compute cores operating at 1.6 GHz. This home PC system costs about $1,500 and plays modern, high-end computer games such as Skyrim or Halo 4 with cinematic quality!

Consider what was inside a typical 1993 supercomputer. Like the FCCM symposium, supercomputer performance ranking also started in 1993. This was the first year that the TOP500 Supercomputer Sites (www.top500.org) list of the 500 fastest supercomputers was published. Since then, the top 500 supercomputers have been ranked yearly based on Prof. J. Dongarra's LINPACK dense matrix solver. In 1993, the highest scoring machines were Thinking Machine Corp. Connection Machines (CM-5). A typical high-end CM-5 was the US National Security Agency (NSA) Machine (ranked fourth in 1993's Top500) that had 512 nodes with 2 TBs of disc storage and 10 GB of RAM, scored 65.5 peak gigaflops, and cost $25 million. Today's TOP500 winner is the IBM BlueGene/Q. This is the same family of supercomputers that powered Watson, which gained fame in 2011 for defeating human Jeopardy champions. BlueGene/Q has over 1.6 million cores with a 1.6 petabyte memory. It scored 20.1 petaflops on LINPACK, making it roughly five orders of magnitude faster than the 1993 NSA CM-5.

Finally, let's focus on the basic building block of reconfigurable computing, the FPGA itself. The XC4000 family was the flagship FPGA family of the then ten-year-old Xilinx Corporation in 1993. The XC4010 FPGA operated at a 70 MHz maximum clock rate and was fabricated in 0.6μm (or 600 nm) technology. Each XC4010 could hold a design complexity between 7,000 and 10,000 gate-equivalents. The Splash 2 super-FPGA machine architecture was a linear array of 16 XC4010s with an additional one used as a crossbar switch [Arnold1993p88]. Today's typical FPGA workhorse is Xilinx's Virtex 6 device that can be clocked in excess of 500 MHz and is fabricated in 40 nm technology. Its design complexity is more than a million gate-equivalents of logic and up to 8 MBs of dedicated RAM. Several hundred Splash 2 systems could fit into a single Virtex 6 FPGA. This is the product of those ten generations of semiconductor manufacturing technology evolution.

Table I
1993 vs. 2013 Technology Comparison

		1993	2013
Personal Computer	Clock Rate	60 MHz	3.3 GHz
	Peak Floating Point	3 MFlops	100 GFlops
	RAM	8 MB	8 GB
	Hard Disk	250 MB	3 TB
Super Computer	Peak Floating Point	66 GFlops	20 PFlops
	Cores	512	1.6 M
	RAM	10 GB	1.6 PB
FPGA	Clock Rate	70 MHz	500 MHz
	Gates	10 K	1 M
	Peak Floating Point	—	26 GFlops
	On-Chip RAM	1.6 KB	8 MB

III. Architecture and Technology

In 1993 general-purpose computers (PCs and workstations) were not fast enough for many tasks we wanted automated (e.g., video and image processing, cryptography, simulation and modeling, optimization). Supercomputers could solve some of these problems, but were only available to government laboratories, and even those with access to supercomputers wanted more processing power.

FPGAs offered the lure of hardware performance. It was well known that dedicated hardware could offer orders of magnitude better performance than software solutions on a general-purpose computer and that machines custom-built for a particular purpose could be much faster than their counterparts. However, building custom hardware was expensive and time consuming. Custom VLSI was the domain of a select few; the numbers that could profitably play in that domain were already small in 1993 and has shrunk considerably since then. Microprocessor performance scaled with Moore's Law, often delivering performance improvements faster than you could build a custom hardware design. FPGAs provided a path to the promise of hardware customization without the huge development and manufacturing costs and lead times of custom VLSI.

In the late 1980s it was still possible to provide hardware differentiation by assembling packaged integrated circuits in different ways at the printed-circuit board level. However, as chips grew in capacity and chip speeds increased, the cost of accessing functions off chip grew as well. There was an increasing benefit to integrating more functionality on chip. Opportunities for board-level differentiation decreased, increasing the demand for design customization and differentiation on chip. FPGAs provided a way to get differentiation without using custom VLSI fabrication.

The challenge then is how should we organize our computation and customization on the FPGA? How should the specialized-FPGA be incorporated into a system? How can the computation take advantage of the FPGA capabilities? As FPGAs grow in capacity and take on larger roles in computing systems than their initial glue-logic niche, how should FPGAs evolve to support computing, communication, and integration tasks?

A. In the Beginning

The first FCCM in 1993 introduced a number of themes to address these concerns. Some of these proved to be highly robust, spawning rich subfields and impacting, or at least, anticipating the FPGA and computing industry.

1) Accelerators: A key idea from the beginning was that FPGAs could serve as generic, programmable hardware accelerators for general-purpose computers. Floating-point units were well known and successful at accelerating numerical applications. Many applications might benefit from their own custom units. The question is how should they be interfaced with the general-purpose CPU, and how should the reconfigurable computation be managed? In 1993 two papers looked specifically at these hybrid accelerators with one focusing on a single FPGA configuration for an application [Wazlowski1993p9], and one considering configurations that might be dynamically loaded during execution [French1993p50].

A host of papers contemplated architectures that integrated an FPGA or reconfigurable array on the same die with the processor [DeHon1994p31] [Albaharna1996p206] [Rupp1998p28] [Miyamori1998p2]. Key concerns included the visible architectural model for the array, how state was shared with the configurable array, how different timing requirements of the processor and array should be accommodated, and how to maximize the bandwidth and minimize the latency required to communicate with the array. [Wittig1996p126⋆] showed how the reconfigurable logic could be a programmable functional unit for a RISC processor. [Hauck1997p87⋆] continued the functional unit model and showed how to share state with the conventional register file and manage the reconfigurable array as a cache of programmable instructions. [Rupnow2007p261] explored reconfigurable functional unit design for integer code in scientific computing applications.

It was soon apparent that the processor would become a bottleneck if it had to mediate the movement of all data to and from the reconfigurable array. [Hauser1997p12⋆] showed how to integrate the array as a co-processor and how the array could have direct access to the memory system. A decade later, [Garcia2007p73] explored interfacing the reconfigurable logic with on-chip caches and virtual memory. [Vuletic2004p24] proposed a window in the virtual memory for communicating with the accelerator. Observing that a single port into a single cache could also be a bottleneck, [Choi2012p17] explored the impact of cache architectures on accelerator performance. [Rajamani1996p226] illustrated the benefits of a DMA connection.

FCCM papers have also explored the impact of accelerators beyond raw application performance. As the importance of energy for embedded systems became clear, [Stitt2002p143] showed that offloading tasks to a reconfigurable array can be effective at reducing the energy required for a computation. To simplify design representation, mapping, and portability, [Schmit2002p152] and [Levine2003p101] showed how a processor and an array could run the same machine-level instructions.

These designs at least anticipated and perhaps helped motivate commercial processor-FPGA hybrids. Xilinx offered a PowerPC on the Virtex2-Pro and now integrates ARM cores on their Zynq devices. Altera includes ARM cores on Cyclone V and Arria V SoC FPGAs. Stretch provided an integrated processor and FPGA device.

While attaching a reconfigurable array on the die with a processor is attractive, this integration is best suited to FPGA or processor vendors. Consequently, there remained much interest in high bandwidth integration of commodity FPGAs into existing systems. Pilchard [Leong2001p170⋆] showed how an FPGA could be interfaced to the memory bus to get higher bandwidth coupling than was possible with the PCI interface of the day. When Intel and AMD shared the specification of their processor interfaces with select partners, Nallatech and XtremeData provided FPGA modules that could directly communicate with the processor bus, reducing the latency of interacting with the FPGA. Intel now integrates an Atom processor and an Altera device in a multichip package.

2) Hardware/Software Partitioning: Determining which computation should be migrated to logic in these reconfigurable accelerator systems depends on many factors including the logic capacity of the FCCM, the amount of its attached memory, and the speed of the communication interface to the microprocessor. A host of papers have examined techniques to identify how to partition computation across heterogeneous resources. [Luk1994p82] suggested rules for identifying suitable data intensive image processing kernels for FPGA implementation. The same year, a HW/SW partitioner that automatically profiles a C application to locate critical sections was introduced [Jantsch1994p111]. Hardware size estimations were used to drive the dynamic programming-based partitioner. [Cardoso2012p192] allowed designers to include both an application specified in C and a series of execution performance requirements to guide hardware/software partitioning at compile-time. [Moore2007p229] used a run-time resource manager to bind functions to either hardware or software platforms based on the availability of hardware resources. The approach included a pre-compiled library of hardware components available for deployment in reconfigurable hardware.

Although significant research has taken place in developing hardware/software partitioning techniques over the past 20 years, most FPGA designers still manually select the computation that will be mapped to reconfigurable resources in heterogeneous systems. As designers move to more behavioral-level specification for designs over the next five years, the situation may change. The increased use of software profiling tools and hardware estimation for high-level design representations may also facilitate this process.

3) Hybrid Parallel Computers (attached Accelerators): Several papers [Raimbault1993p2] [Cuccaro1993p121] [Wazlowski1993p9] noted the value of adding reconfigurable logic to multiprocessors. The logic could be used for custom acceleration, similar to its use in single node machines, or for improving communication and synchronization. [Raimbault1993p2] and [Dhaussy1994p72] showed how the logic might be used for global reduce operations or synchronization for systolic computations. [Sass2007p127] suggested that integrated networking support on the FPGA provides cost-effective, low-latency communication. Cray integrated FPGAs into their XD1, SRC offered a parallel supercomputer with FPGA accelerators, and Convey Computer now ships a supercomputer with an FPGA acceleration board.

4) Board of FPGAs: From the beginning, boards with multiple FPGAs that form a large, customizable substrate have been a staple of FCCM [Casselman1993p43] [Bout1993p68] [Chan1993p152] [Milne1993p26] [Hogl1995p45] [Carrera1995p20]. [Darnauer1994p1] looked at building MCMs that integrated FPGAs. [Chan1993p152] explored how the FPGA should be interconnected. [Hauck1994p11] addressed how pins should be assigned, and [Babb1993p142⋆] and [Dahl1994p14] looked at how to avoid package pin limitations. [Li1995p61] explored using dynamic field-programmable interconnection devices on board-level systems to address limited package I/O. As FPGAs grew in size, such that many interesting applications could fit on a single FPGA and designing to fill an FPGA became a challenge of its own, the conference saw diminishing emphasis on multi-FPGA systems. Nonetheless, numerous companies now sell multi-FPGA system boards. Xilinx currently uses silicon interposers to economically compose multiple dies into large FPGAs (*e.g.*, Virtex-7 2000T).

5) Virtualization: The virtualization of FPGA resources has been a running theme since the first FCCM. [Babb1993p142⋆] virtualized the pins on the FPGA to exploit their potential bandwidth. This virtualization avoids a bottleneck at the boundary between FPGAs. [Brebner1997p77] considered virtualizing and reconfiguring portions of an FPGA. [Trimberger1997p22⋆] and [Scalera1998p78] introduced multicontext FPGAs that allowed the logic cells to rapidly switch between different logic functions. [Schmit1997p47⋆] explored a model for incrementally reconfiguring resources to virtualize array capacity.

FPGA-based Veloce logic emulators sold by Mentor Graphics are based on the techniques described in [Babb1993p142⋆]. Today's FPGAs include hardware support for high-speed serial links to address the chip I/O bandwidth bottleneck. Tabula now offers a commercial multicontext FPGA.

6) Organization: FPGAs give us freedom to organize our computation in just about any way, but what organizational patterns are actually beneficial for use on the FPGA? This was a theme in 1993 with papers exploring dataflow [Ling1993p33], VLIW [Iseli1993p17], and cellular [Milne1993p26] architectures. Exploration continued with dataflow [Hartenstein1994p139] [Cathev2006p121], VLIW [Zhang2000p3] [Kapre2009p37], and cellular [Marchal1994p66] [Margolus1997p2] [Durbano2004p156] [Kobori2001p120]. Later additions included vector processing [Weinhardt1999p52] and pipelines [Piacentino1999p82] [Laufer1999p200].

B. Emerging Themes

As Moore's Law continued and VLSI ICs grew in capacity, it became clear that FPGA arrays should be more than just fine-grained logic and chips should integrate more than just a processor and an FPGA array. Furthermore, as chip size grew, defects and faults emerged as a new challenge and an opportunity for field-programmable computing systems.

1) Coarse-grained: Early FPGAs contained fine-grained logic largely because of their limited capacity, their gate array roots, and their use as glue logic. As their application domain moved to address signal processing and computing problems, there was interest in more efficiently supporting larger, wide-word logic. Could we have flexible, spatially-configurable, field-programmable machines with coarse-grained computing blocks? Would a small amount of time-multiplexed sharing of these units be useful? [Mirsky1996p157⋆] explored using an 8b ALU-multiplier-register-file as a computing block that could be efficiently composed into a wide range of architectures (systolic, VLIW, SIMD/Vector, MIMD) by configuring instruction distribution. [Bakkes1996p118] explored using fixed floating-point units at the board level along with FPGAs to provide an early look at the heterogeneous integration of hardcore functional units. [Miyamori1998p2] explored using a 16b ALU-register-file array with VLIW control for multimedia applications and showed that it can be more compact than an FPGA accelerator achieving comparable performance.

[Ho2006p35] provided a model to explore hard logic integration in an FPGA array and explored coarse-grain hardware to support floating-point computations. [Kwok2005p35] examined register file architectures for coarse-grained reconfigurable architectures. [Parandeh-Afshar2010p229] reexamined how to optimize hardcore DSP blocks. Both Altera and Xilinx now embed hard logic DSP blocks that support wide-word addition and multiplication inside their fine-grained logic arrays.

2) Specialized Reconfiguration: Once we start adding more customized functional units to FPGAs, there is a danger that the FPGA becomes too specialized to a particular task. Nonetheless, if the domain is large enough, perhaps it does make sense to create a custom reconfigurable array for the domain. [Compton2001p111] showed how to optimize a reconfigurable array for a specific domain and illustrated the area savings in the DSP domain. [Eguro2003p111] showed several techniques for performing function allocation for a domain-customized architecture. [Phillips2005p203] described tools that automate the layout of domain-optimized reconfigurable arrays.

3) Integrated Memory: As FPGAs grew, it became important to integrate memory onto the FPGA die to avoid memory bottlenecks. This created opportunities to exploit high, application-customized bandwidth and raised questions of how to organize, manage, and exploit the memory. [Margolus1997p2] described an on-chip SRAM architecture for buffering and caching data from a high-speed off-chip DRAM. [Yu2006p76] more generally addressed how to use on-chip memories to provide efficient data reuse windows into large streaming computations. [Yan2002p195] explored managing distributed on-chip memories like a collection of small caches. [Diniz2003p207] showed how an FPGA accelerator could gather and filter data for a processor. [deLorimier2006p143] showed how organizing active computation around memory could accelerate graph processing. Xilinx and Altera now both offer significant amounts of on-chip, configurable memory on their FPGAs.

4) On-Chip Networking: The load-time configured interconnect in FPGAs is suitable for connecting together gates or performing systolic, pipelined, and cellular computations. However, as FPGAs started hosting more diverse tasks, including computation that used interconnect less continuously, it became valuable to explore disciplines for dynamically sharing limited on-chip FPGA communication bandwidth. Before there was enough logic to implement both routers and computation on an FPGA, [Yeh1995p56] built routing switches on the FPGA. [Lertora2005p45] used a packet-switched NoC to interconnect an ARM and three FPGA regions on a system-on-a-chip. [Kapre2006p205⋆] compared packet-switched and time-multiplexed overlay networks on an FPGA.

5) Defect and Fault Tolerance: We can exploit the homogeneous set of uncommitted resources on FPGA-like architectures to tolerate defects. [Culbertson1997p116⋆] pioneered the large-scale approach of identifying defects, both in the FPGA and in the interconnect between FPGAs, and mapping an application to avoid them. To avoid the high place and route time of traditional FPGAs, they designed their custom FPGA architecture to support fast design mapping [Amerson1995p32]. [Tahoori2004p176] showed how to efficiently test for defects in an FPGA or molecular-scale interconnect. [Emmert2000p165] exploited partial reconfiguration to interleave tests for failures during operation with quick reconfigurations to mask the new failures. [Xu2003p143] provided support for recovery from failures during the FPGA operational lifetime by communicating with a networked repair system that can recompile the bitstream to avoid new defects. Xilinx's EasyPath program exploits the ability to map around defects by matching partially-defective FPGAs with specific bitstreams and offers them at a reduced price.

Soft errors can change the configuration bits in SRAM-based FPGAs, leading them to perform incorrect functions. [Wirthlin2003p133] explored the susceptibility of FPGAs to radiation upsets. [Quinn2005p193] showed how to estimate upset rates for systems of various sizes and locations and pointed out that soft errors from radiation can be a concern even for airplanes and ground-based systems. [Caffrey2009p3] reported on experiences with radiation upsets in space on an experimental satellite. [Ichinomiya2010p47] and [Illias2010p73] explored online error detection and partial reconfiguration repair. [Schmidt2011p162] addressed how to monitor an accelerator and detect its failure. To minimize the impact of configuration upsets, Xilinx provides designs to scrub an FPGA bitstream and identify and correct configuration bit upsets.

C. Looking Forward

As technology continues to scale towards the atomic scale, energy and reliability are emerging as some of the biggest challenges facing the electronics landscape. In today's environment, an FCCM's ability to reduce energy is often as important as its ability to reach new levels of performance. Power concerns have slowed processor clock frequency scaling, making spatial architectures that exploit parallelism, as we've seen throughout the conference's history, increasingly attractive. Furthermore, power concerns are driving increasing interest in specialized accelerator architectures and heterogeneous computational organizations on a single chip—both areas that have seen significant development within this conference. Reconfiguration remains a promising way to address the reliability challenges of highly-scaled CMOS, ultra-low voltage CMOS, and post-CMOS computing substrates. These trends suggest that the FCCM body-of-knowledge is increasingly valuable to the design of all kinds of future computing systems.

IV. Languages and Compute Models

In 1993, programming for general-purpose computers and FPGAs was split into two very different domains. While procedural languages like C were generally used to target microprocessors, most FPGA application designers were still burdened with drawing schematics and writing Boolean equations. Hardware description languages (HDLs), such as Verilog and VHDL, were gaining a footing, but HDL synthesis at the time tended to produce designs that were larger and slower than hand crafted designs. In general, achieving reasonable design performance using the limited logic and routing resources in most devices required hand tuning and an understanding of the basic FPGA architecture.

Almost from the beginning, it was recognized that for FPGA-based machines to be accessible to the masses, new programming environments and models of computation would be needed. Since the many-core era was still a human generation away, the massive parallelism available in FCCMs required designers to consider succinct ways of expressing and exploiting massive parallelism long before these issues were mainstream in the general-purpose computing community. However, a desire to maintain familiarity for programmers of microprocessors created a dilemma: Should familiar processor-based languages and compute models be migrated to reconfigurable platforms or should whole new models be created? The rapid advancement of hardware synthesis tools in the 1990s provided increased accessibility to even more hardware designers, but it was clear that programming languages and models that could more easily express and extract parallelism were missing.

The challenge then is how to take advantage of the fine-grained parallelism and specialization offered by reconfigurable devices without requiring the application designer to explicitly define it all as gate-level hardware. Since not every type of application requires the same type of parallelism (fine-grained, coarse-grained, memory-intensive, etc.), a range of languages and models were needed, which today we often call domain-specific languages. As the capacity and diversity of FPGA resources grew, how could compute environments evolve to abstract away more of the low-level architectural details that initially required FPGA application designers to be hardware designers?

A. In the Beginning

The early FCCM conferences examined a number of ideas to push the programming realm of FPGAs beyond schematics and HDL code towards more familiar software programming languages. Many of these initial trends form the basis of contemporary mainstream FPGA and FCCM programming. [DeHon2004p13] provided a detailed taxonomy of many of these compute models from FCCM's first decade.

1) C-to-Hardware: A key goal in the early days of FCCMs was to make their programming environment as similar as possible to microprocessor-based systems to allow them to be accessible to a large population of programmers. A compiler that could convert familiar C code or code in another procedural language into hardware that could be synthesized into FPGA logic was an enticing prospect. Although the supported C constructs were typically significantly limited (e.g., no pointers or recursion), the amount of performance gained by specialization was significant. In the first FCCM, [Wazlowski1993p9] focused on converting chains of simple C operations (e.g. add, shift) in a function into HDL code that was then compiled to the gate-level using synthesis tools. This idea was then extended [Wo1994p147] to consider the use of a simple state machine to execute multiple sequential hardware operations for each extracted C block. Compiler features to support bitwidth specification and communication between the synthesized hardware and the rest of the circuit [Galloway1995p136] were soon added. [Peterson1996p178] described a system that mapped programs written in C to multi-FPGA platforms. Partitioning and scheduling were performed at the functional unit level to reduce hardware requirements and inter-FPGA communication costs.

Since FPGAs were so logic and interconnect limited in the 1990s, it was necessary to have accurate hardware estimates of area to successfully map C code to hardware. The use of macros [Gokhale1997p165] [Kulkarni2002p239] in the C-to-hardware mapping not only helped in area estimation, but introduced the idea of hardware design reuse to reduce design effort. Even with macros, these early compilers relied on the user-defined pragma statements in the code [Gokhale1998p126] to isolate code segments and define data storage locations in the reconfigurable logic. More recent efforts [Lau2006p45] not only focused on mapping C functions to hardware, but also on the optimization of the function's interface to external memory. This extension allowed for more advanced operations using pointers, the pipelining of memory transactions, and speculative execution. Other evolutions of the C-to-hardware approach included architecture-independent code generation from C [Villarreal2010p127] and support for bounded recursion in hardware [Greaves2008p3].

The research associated with reconfigurable computing has spawned a number of commercial procedural language-to-hardware offerings over the years, including Impulse-C, Catapult C, Handel-C, Altera's C2H, and AutoESL's AutoPilot (now Xilinx Vivado). A demand for higher productivity in the broader design automation community has also fueled a push to C- and system-level design and synthesis. Extensive improvements have led to increasing acceptance and a slow FPGA designer migration from design specification in HDL to specification in these higher-level languages.

2) Pipelined Computation and Communication: In 1993, lookup table-based FPGAs were recognized as desirable platforms for repetitive, datapath-oriented applications due to their inclusion of numerous flip flops. Image and signal processing applications that require little control or fine-grain synchronization can take advantage of a systolic compute model where computation and communication take place in lockstep and a pipelined model where high throughput is achieved using a chain of simple functions isolated by registers. The first two FCCMs included four papers that exploited extensive pipelining. In 1993, data sorting using fine-grained systolic computation was demonstrated [Luk1993p192], and the following year [Schmit1994p125] described the creation of systolic cells to compare nucleotides from a behavioral description. Also in 1993 it was shown that deep pipelines implemented in FPGAs could be used to perform repetitive operations on vectors [Guccione1993p79]. This approach was directly contrasted with a possible single-instruction, multiple data (SIMD) approach for the same problem. SIMD computing on the groundbreaking Splash 2 architecture was also described at the first FCCM [Gokhale1993p94]. In this architecture an instruction was globally distributed to 16 FPGAs, all performing the same operation. Data was then transferred between the devices in a systolic fashion.

More sophisticated pipelined approaches followed as parallelizing compilers were developed for the general-purpose computing community and were migrated for use in FCCMs. One approach [Weinhardt1999p52] automatically generated pipelined coprocessors using loop unrolling that allowed for the execution of multiple loop iterations in parallel. This implementation represented one of the first automatic pipeline generators for FCCMs based on the widely-used Stanford University Intermediate Format (SUIF) compiler infrastructure for microprocessor systems. A similar approach [Maruyama2000p101] considered the use of internal FPGA memories and feedback paths in the creation of a balanced pipeline, while later work [Rodrigues2007p219] examined the use of dynamic data dependency analysis in allowing multiple pipelines to proceed simultaneously. Although the pipelining of data computation in reconfigurable compute machines is still important in today's systems, newer systems must also consider the time required for memory accesses, a bounding limitation for many applications. In general, contemporary commercial C-to-hardware compilers support pipelining operations involving loop unrolling and tiling.

B. Emerging Themes

As compiler technologies advanced and embedded memory blocks and fixed integer multipliers were added to FPGAs, more sophisticated design mapping systems for FCCMs were developed. The need for compile time reduction, design reuse, and more advanced treatment of external memory access and inter-functional unit communication led to significant advancement in languages and compute models.

1) Support for Integrated and External Memory: After the initial wave of languages and compute models in the early to mid-1990s that focused on efficient logic implementations, efficient memory usage moved to the forefront towards the end of the decade. [Babb1999p70⋆] introduced a parallelizing compiler that was optimized for memory access and communication time reduction, rather than computation clock speed. Another paper at the same FCCM [Gokhale1999p63] focused on the automatic allocation of data arrays to multiple memory banks. Both systems, which were based on SUIF, set the stage for over a decade of FCCM compiler research that explicitly considered both functional unit execution and memory accesses in optimizing performance. For example, the automatic creation of address generators and queue buffers for memory accesses was addressed the following year [Diniz2000p91], and a compile-time mapping approach for multiple FPGAs with external memories was outlined several years later [Ziegler2002p77]. The latter compiler infrastructure pipelined computation at a coarse level and considered the costs of inter-FPGA communication and memory access in functional mapping.

Rather than starting with a specification in a high-level language, a novel compiler [Zaretsky2004p37] converted processor assembly and binary codes to an intermediate form and then performed optimizations, such as loop unrolling, to optimize bandwidth to distributed embedded FPGA memory blocks. A more recent paper [Cheng2012p157] described a mapping approach for systems that contain multiple reconfigurable accelerators, each of which has a memory interface including a cache. Memory accesses were optimized based on pre-determined accelerator access patterns.

Advances in compiler technologies have encouraged FPGA companies to include specialized DRAM interfaces and large numbers of block RAMs in their devices. Convey Computer offers specialized scatter-gather external memory configurations for its multi-FPGA systems in an effort to more efficiently support random external memory accesses.

2) Object-oriented and Streaming Compute Models: The similarity between instantiated hardware modules and objects in object-oriented programming languages has led to numerous attempts to represent FCCM computation as parallel collections of objects. Early work using Smalltalk [Pottier1996p48] and Java [Chu1998p158] showed the ability to define object bitwidth and interconnection at a high level and verify their functionality in software. JHDL [Bellows1998p175] allowed for the definition of objects whose functionality can be dynamically changed. A more recent effort based on Scheme [Bachrach2008p13] automatically converted function calls between objects to wiring, eliminating the need for users to define bitwidths or types.

Although similar to pipelined implementations, streaming applications typically have coarse-grain compute objects that communicate with adjacent blocks via buffers or synchronized communication channels. This model is particularly useful for series of signal processing blocks that require minimal control flow or global synchronization. PamBlox and PamBlox II focused on the ability to define hierarchies of C++ objects [Mencer1998p167] and the use of signed representation and embedded block memories [Mencer2002p67]. These blocks could then be organized in streams. The Stream-C model [Gokhale2000p49⋆] introduced a series of communicating sequential processes that used small local memories. A follow-on project [Gokhale2004p186] used more explicit point-to-point synchronization for communication that was coordinated at compile time. The Stream-C language was later commercialized into the Impulse-C compiler. [Unnikrishnan2009p123] automatically generated a series of stream processors and buffered interconnections that were customized on a per-application basis. A recent paper [Neely2010p141] provided a simple language to specify and interconnect modules that use a streaming communications model. The wiring between the modules was determined automatically from the language specification. Perhaps the most comprehensive stream-based, object-oriented environment to date was the commercial Ambric model [Butts2007p55⋆]. In this model a simple processor executed one or more user-defined objects that communicated with objects in other processors via self-synchronizing channels. Additionally, a current commercial product, the Bluespec hardware synthesis system, is based on the manipulation of objects.

3) MATLAB: As the use of FPGAs in commercial products increased, so did the range of languages targeted to the devices. By the end of the 1990s, the use of MATLAB for developing and testing signal processing applications had become widespread. Since many of these signal processing designs were ultimately targeted to FPGAs, it was only natural that techniques to compile MATLAB to FPGA-based systems would be created. An initial effort [Banerjee2000p39⋆] explored techniques to map MATLAB to multiple types of platforms, including FPGAs. Precompiled FPGA modules were instantiated with some control (e.g., loop counters) synthesized from additional logic. The model was expanded the next year to include multiple FPGAs executing the same operations [Nayak2001p1]. Later work [Ou2004p47] examined techniques for implementing MATLAB functions in FPGAs in a power-efficient manner using parameterized designs. MATLAB support is now a staple in FPGA development tools (e.g., Altera's DSP Builder) and is widely used for the implementation of signal processing blocks in FPGA systems.

4) Many-core Inspired Models: The emergence of many-core processors and General-Purpose Graphics Processing Units (GPGPUs) in the past five years has allowed for the crossover of computing models from these domains to FCCMs. A defining aspect of these architectures is the presence of a large numbers of threads of control in the processing architecture. An initial effort in this space [Fort2006p131] examined the cost/benefit tradeoffs of the implementation of a multithreaded soft processor in an FPGA. This work was later extended in [Labrecque2008p195] to consider implementation issues associated with multiple multi-threaded soft processors on an FPGA. [Anderson2006p89] described a heterogeneous system in which threads could be assigned to either a standard CPU or an attached FPGA. Abstract data types were passed between the threads. A threaded system model where operating system functions, such as thread allocation, are implemented directly in FPGA hardware [Agron2010p39] was a natural progression from this idea. Recent interest in OpenCL and CUDA, multi-threaded GPGPU languages, led to an effort [Owaida2011p186] to map code written in OpenCL to multiple FPGA compute kernels that were customized to the application. The best paper at FCCM 2011 [Papakonstantinou2011p178] used sophisticated estimation techniques to select the appropriate amount of parallelism at multiple levels of granularity for CUDA programs mapped to FPGAs. Altera has recently announced software support for automatically mapping portions of OpenCL programs to their FPGAs.

As the use of multiple soft processors per FPGA has increased, so has interest in providing advanced shared memory protection and coherency techniques. The use of transactional memory, which supports atomic task execution involving shared memory, has been integrated into several reconfigurable systems. [Kachris2007p65] synchronized memory accesses for concurrent threads on multiple soft processors using the approach. A recent paper [Arcas2012p1] described a system to examine the performance of transactional memory in a multiprocessor prototyping environment built using FPGAs. The area cost of transactional memory in terms of FPGA resources was evaluated in both papers.

Over the past few years, massively distributed computing using thousands of processors in a "cloud" has gained increasing attention. Several FCCM papers have focused on reconfigurable implementations of MapReduce, a scalable algorithm that is widely deployed in such environments. In a MapReduce implementation, computation is split into simple transformations (map) followed by the aggregation of transformed data (reduce). In [Yeung2008p149] it was shown that the algorithm can be efficiently implemented in FPGAs and GPUs. The need for better operating system and programming support for cloud-based MapReduce which includes FPGAs and GPUs was later observed [Madhavapeddy2011p141]. Although no commercial products are yet available, Microsoft has remained an active investigator in examining the use of FPGAs to accelerate MapReduce. In 2011, Maxeler introduced MaxCloud, based on multiple Virtex 6 FPGAs, as an on-demand, high-availability cloud computing environment.

C. *Looking Forward*

Many of the challenges in terms of the automatic extraction of parallelism facing the many-core computing community are the same ones that have been present in the FCCM community for the past 20 years. Continued advances in both fields will help better define languages and compute models. The emergence of GPGPUs as compute devices and the use of massively multi-threaded OpenCL provide a rich opportunity for model sharing. The continued movement of hardware design from HDL to the behavioral level will also make resource estimation and memory access analyses more accurate, reducing designer effort, and subsequently, making the programming of FCCMs more accessible to the masses.

V. Tools

In 1993, the design, optimization, and debug tools for FPGA-based systems were very primitive by today's standards. While most general-purpose computer systems were supported by advanced profilers and debug infrastructures with single-stepping and breakpoints, users of FPGA-based systems primarily relied upon gate-level design manipulation to achieve desired performance and gate-level simulation to isolate design bugs. After bugs were located, design fixes required the same long FPGA compile times present today with an additional issue; in many cases FPGA device routing would fail due to limited available FPGA routing resources, necessitating even more design-level manipulations.

By the first FCCM workshop it was recognized that better tools for design, test, and debug would be necessary to increase FCCM application designer productivity. Although standard FPGA synthesis and physical design tools are often a core component of this tool environment, an additional layer of tools that customizes the mapping of designs to the FCCM platform and bridges the debug gap between intermediate design results and the original design specification is needed to fully support an FCCM.

A strength of SRAM-based FPGAs is their support for the specialization of implemented design hardware based on a limited collection of compute operations, limited operand sizes, input data sets, and even specific constant parameters. These optimizations can provide area, performance, and energy benefits versus more generic hardware implementations.

These systems raised important questions about the way forward: Can the development tools for FCCM applications reach a level of support approaching microprocessor-based systems? Can tools that exploit compute specialization be developed to increase hardware efficiency? In the 1990s, improved register-transfer level and gate-level simulation environments and in-circuit debug tools provided by FPGA companies helped advance the debug issue somewhat, but more comprehensive tools were needed. The challenge then is how to make the tools for FCCMs easier to use and how to reduce the latency in the edit-compile-debug cycle to allow for increased designer productivity and more predictable performance results. Since FCCM platforms and applications vary widely, the scope of required tools is broad. As the application space of FCCMs grew, how could these tools assist compilation environments in unlocking the promise of reconfigurable computing?

A. *In the Beginning*

The early FCCM symposia highlighted a series of tools to assist designers in exploiting the functional specification of designs, especially in the context of heterogeneous and multi-FPGA systems. These early papers provided insights into the challenges and potential payoffs of successfully exploiting FCCMs. Although today's FCCM design implementations span a spectrum of system complexity, these early efforts helped develop important conceptual frameworks that led to improved FCCM usability as FPGA devices became larger and more diverse.

1) Tools for Functional Specialization: A significant benefit of FPGAs is the ability of the designer to specialize a specific FPGA circuit. For example, the multiplier hardware for a finite impulse response filter can be reduced if the filter coefficients are known and can be synthesized into the multiplier logic, reducing multiplier area and power consumption. In 1993, two papers examined the potential benefits of constant-based specialization. In [Foulk1993p163], constant data values were folded into logic to accelerate a text search application. It was argued that the FPGA hardware could be updated when the search keys were modified. A second paper [Lewis1993p60] examined the customization of logic to support logic simulation. A comparison was made to compiled-code, processor-based simulators that optimize sequential code for specific circuit parameters. Several papers [Singh1996p188] [Wang1997p145] explored the similar idea of partial evaluation. The latter paper presented a tool that used a priori information about application data sets to optimize functional units and memory accesses in the generated hardware. The paper that introduced the JHDL object-oriented language [Bellows1998p175] described the data-specific specialization of hardware objects using object constructors. Although today's powerful FPGA logic synthesis tools take advantage of data-dependent specialization when constants are specified by the user at compile time, standard interfaces for specializing computation at run time have not yet been developed.

2) Tools for Multi-FPGA Systems: In 1993, most reconfigurable computing systems either contained a large number of FPGAs (>10) or only one. The first two FCCM workshops highlighted several software tools used to map designs to large multi-FPGA systems. The software system for Splash 2 [Arnold1993p88] allowed users to specify individual FPGA programs in VHDL. A whole-system VHDL simulation environment provided a critical debug tool. The design compilation and simulation environment for PAM, another early multi-FPGA FCCM, was described the following year [Bertin1994p133]. A novel aspect of this system was the user's ability to include internal-FPGA placement information with the design specification. Eventually, a full set of high-level synthesis algorithms, including scheduling, binding and allocation, was used to target multi-FPGA systems [Duncan1998p106].

As FPGA device capacities increased rapidly after 2000, the number of devices used per multi-FPGA system generally dropped. However, large multi-FPGA platforms and their sophisticated software systems remain commercially viable. Convey Computer includes multi-FPGA design environments with their widely-used systems.

B. Emerging Themes

As FPGAs increased in size and their user community broadened, the need for more robust and comprehensive tool sets became apparent. Many of these tools focused on enhancing the productivity of the designer and improving the quality of the mapped result. Although still not as mature as FPGA synthesis, place, and route tools, a broad set of tools have been introduced.

1) Integrated Development Environments: The development of a reconfigurable computing application requires a number of steps including design specification, profiling, performance estimation, compilation, deployment, and testing. A number of integrated systems, primarily focused in specific application areas, have been introduced that facilitate the implementation of these tasks. [Wenban1996p28] described a data acquisition tool kit for FCCMs that included a compiler, debugger, linker and hardware/software interface. A full design suite [Hutchings1999p12⋆], including graphical design tools, was introduced a few years later. This integrated system included a graphical floorplanner, simulation models, schematic generator, and a compiler for the object-oriented JHDL language. Simulation support with a graphical waveform viewer was also provided. In the mid-2000s, when design power became a greater concern, power analysis tools were added to the tool set [French2006p185]. The CHAMPION system [Ong2001p10] was designed to allow for design specification using a graphical programming environment that allowed users to specify a dataflow of signal processing operations. A series of backend libraries were integrated into the system to allow for design mapping to a series of different FPGA-based boards. More recently, [Lavin2011p117] developed a series of custom FPGA synthesis tools for rapid prototyping and integrated them with Xilinx System Generator. This system provided extremely fast compile times through the use of pre-compiled macro blocks, significantly reducing the time required for the edit-compile-debug cycle.

While commercial FPGA computer-aided design (CAD) software in the early 1990s typically contained little more than synthesis and physical design tools, today's CAD packages, such as Xilinx IDE and Altera SoPC Builder, contain an integrated series of tools, similar to the systems described above. These tools allow for design expression, simulation, power analysis, compilation, and in-circuit testing, often using on-chip JTAG interfaces.

2) Debugging and Bitstream-based Design Modification Tools: Since FPGAs are generally quite large and often are difficult to analyze once they have been compiled to lookup tables and flip flops, the need for effective test and debug tools are critical for application development. In many cases, debugging can be performed at a high level via simulation, but in cases where it is difficult to isolate specific bugs, analysis is needed at the hardware-level after a bitstream has been loaded into the FPGA device. [Hemmert2003p228] described a source-level debugger that correlated circuit-level debug information with the original behavioral-level source code. The tool set supported useful features such as hardware breakpoints, watchpoints, and single stepping. A paper at the same FCCM [Slade2003p251] provided a similarly comprehensive view of debugging by offering the user the in-circuit implementation of a customized debug controller for each developed application. Users could interact with the FPGA under test via an application programming interface (API). In the late 1990s, Xilinx introduced a series of tools that significantly aided user access to low-level FPGA design information. The JBits tools gave designers the opportunity to directly read portions of an FPGA bitstream from a device, modify them without the need to perform time-consuming place and route, and then download them back to the device. One interesting debug approach [Graham2001p41] instrumented an FPGA design with a small logic analyzer that could be modified via bitstream manipulation.

Several other tools unrelated to design debug were created using bitstream manipulation tools. [James-Roxby2000p153] demonstrated that small changes to features in previously-compiled and implemented logic cores (e.g. constant values, pipeline registers) could be made via bitstream manipulation, providing specialization while avoiding long compilation, place, and route times. [Singh2001p91] demonstrated the rapid generation of an FPGA design using JBits. Bitstream construction was illustrated for highly-regular circuits that were aligned in repeatable patterns. A later paper [Megacz2007p45] targeting an Atmel FPSLIC device used bitstream manipulation to control a self-timed circuit. The ability to migrate partial bitstreams to a diverse set of alternative locations in the FPGA was explored in [Becker2007p35].

Over the years, many of the debug ideas explored at FCCM have been incorporated in Xilinx and Altera products. The availability of Xilinx ChipScope and Altera SignalTap allows for run-time downloading and analysis of FPGA design values. A successful commercial venture in the area of FPGA design debug, Veridae Systems (acquired in 2011 by Tektronix), provides the capability to instrument an FPGA using a small debug engine fashioned from FPGA logic. In general, FPGA companies provide little support for direct bitstream manipulation of contemporary FPGAs, although manipulations have been reported by researchers willing to expend non-trivial effort.

3) Tools for Precision Analysis: Unlike fixed microprocessors, FPGAs offer the flexibility of building arithmetic hardware with bitwidths that generate results that exactly match the desired precision of the application. Over the years, many FCCM papers have examined automated techniques to determine appropriate operand word lengths and intermediate compute value bitwidths to maintain a user-specified level of accuracy. These optimizations generally take place in the context of implementation of digital signal processing applications. Several tools developed for design time use [Chang2002p229] [Constantinides2002p219] [Chang2004p59] considered the selection of intermediate computation bitwidths in integer-based FPGA designs to maintain a desired precision. The effect of roundoff errors in intermediate bit-limited computations was considered in [Constantinides2003p81]. Subsequent work [Gaffar2004p79] considered bitwidth optimization for both fixed and floating-point FPGA implementations of an algorithm. Precision analysis was later extended [Osborne2008p129] to consider tradeoffs of design energy consumption and word length. [Boland2010p157] considered word-length optimizations for floating-point computations. Number representations were modified from standard IEEE formats to reduce implementation logic area. The implementation of both high-precision and low-precision datapaths in the same design implementation was explored in [Chow2011p17].

C. Looking Forward

Although FCCM tools have improved dramatically, significant work remains in several areas. The movement of programming environments to the behavioral level provides ample opportunity for improved estimation of low-level resource use without the need for complete design synthesis. Compilation time is still a source of pain, and in some cases it may make sense to forgo full device resource usage to dramatically reduce compile times. The recent strong interest in GPGPUs makes it likely that an intermediate architectural point between these thread-parallel devices and the fine-grained parallelism, specialization, and reconfigurability of FPGAs will be developed.

The introduction of GPGPUs and energy-conscious design has also led to greater interest in precision analysis and optimization across the computing spectrum. Since most GPGPUs are limited to single precision, there is increasing interest in limiting the use of double-precision computation to cases where it is absolutely necessary, a viable alternative in the specialized hardware of FCCMs. Energy concerns are also driving wide-word processors to only use the portion of the datapath that is necessary.

A fundamental approach to energy-efficient design is shutting down resources that are not currently in use. Although contemporary FPGAs do not currently support dynamic intra-device region shutdown, FCCMs with multiple devices or future devices that do include this feature may provide a path for increased energy efficiency.

The use of heterogeneity in FCCMs will require additional tools beyond the current state of the art. At the same time, energy efficiency is driving increased interest in heterogeneous accelerator architectures. Such heterogeneity will provide new challenges regarding hardware/software partitioning and identifying which compute functions should be mapped to which resources.

VI. Run-Time Reconfiguration

SRAM-configured FPGAs can change their functionality during operation. To distinguish this use of reconfiguration from the cases where the configuration remains constant during an application, the former case has been dubbed Run-Time Reconfiguration (RTR). RTR fully exploits the hybrid nature of FPGAs, allowing for the implementation of hardware-like spatial computations and for hardware organization changes as needed during different phases of the computation. As a result, RTR creates a distinct requirement for application mapping to coordinate both the spatial and temporal aspects of the computation. The potential benefit of the approach is the specialization of the computation to the instantaneous needs of the application, reducing the size and energy required for the design.

While the benefits of fine-grained parallelism and specialization in FPGA-based systems were apparent even before the first FCCM conference as a result of the Splash and PAM projects, the broad usefulness and practicality of run-time hardware reconfiguration has remained an open question over the past twenty years. For some applications, it is possible to gain the advantage of customization without the need for run-time logic modification simply by reprogramming data memories or statically loading one of multiple pre-computed configurations. To fully demonstrate the benefit of RTR, it is necessary to identify cases where logic modification during run time is necessary.

Additional tools are required to exploit RTR applications, including a diverse set of compile-time and run-time tools to update the specialized hardware to changes in the computational requirements, computational functions, and instance-specific parameters. Since compilation, synthesis, place, and route are slow, running a full pass of mapping tools can defeat the potential benefits of RTR. To accelerate changes, vendors such as Algotronix and National Semiconductor introduced reconfigurable arrays that supported partial reconfiguration to reduce load times when the configuration of the entire array did not need to change. To exploit partial reconfiguration, applications must define and control which regions are reconfigured. This issue motivated the need for a variety of additional supporting tools, including software to assist designers in identifying similarly-sized design regions that could be dynamically swapped into a device using partial reconfiguration and software to manage the deployment of device configurations at run time.

Consequently, RTR posed a number of challenges that would have to be addressed if it were to become viable. Can models for capturing and expressing applications that exploit RTR be developed? It was necessary to find ways to avoid or mitigate long mapping times and reduce reconfiguration times. When can RTR be harnessed to provide net benefits for run-time specialization? Before the commercial development of tools for RTR could be considered (a process that continues to this day), a better understanding of applications that could benefit from RTR was essential.

A. *In The Beginning*

The call for papers for the first FCCM highlighted an interest in RTR by soliciting papers "...on all aspects of the use or applications of (usually multiple) FPGAs as (usually reconfigurable) computing elements in attached or special purpose processors or coprocessors...". In 1993, the only FCCM paper to address the topic [French1993p50] described the possibility of dynamically loading hardware functions at run time as different applications were executed. Although few specifics were presented, a list of potential functions that could be supported (e.g. arithmetic operations) provided insights for later efforts.

B. *Emerging Themes*

1) Run-time Reconfiguration Models and Design Tools: Over the years, a number of FCCM papers have addressed the development of tools to assist RTR. [Hadley1995p78] examined techniques to make simple changes to the hardware implementation of an artificial neural network, such as multiplier bitwidth variations and constant value updates, without requiring full design recompilation. Circuit characteristics that can be used to locate partially-reconfigurable regions were presented in [Luk1996p167]. Follow-on papers presented a tool that used these characteristics to create designs that were customized for RTR [Luk1997p56] and a tool that minimized the differences between subcircuits that were swapped for each other to reduce their RTR overhead [Shirazi1998p147]. The DISC processor [Wirthlin1995p99⋆] provided an RTR model where reconfigurable hardware modules were invoked as specialized instructions that were swapped into the processor as needed on a per-application basis. DISC also provided a one-dimensional placement model that allowed the modules to be assembled without requiring placement and routing tools to execute during run time. A comprehensive approach [Cardoso2001p31] considered the joint use of both dynamic reconfiguration and resource sharing in performing computation. Functional unit sharing and the scheduling of RTR were combined in a single algorithm to reduce the need for frequent partial reconfiguration. The energy consumed by the reconfiguration process and its impact on the effectiveness of reducing energy with RTR were examined in [Becker2010p55]. [Beckhoff2012p37] described a practical system for supporting RTR on low-cost commercial FPGAs. The tools included a floorplanner and a constraint generator to assist the designer in creating regions of dynamically-reconfigurable logic.

In addition to the tools and models described above, tools that indirectly support RTR have also been developed. [Hauck1998p138⋆] showed how the wildcard scheme on the Xilinx XC6200 can reduce bitstream size and load time, and [Li2001p147] showed how to compress Virtex bitstreams. [Li2000p22⋆] introduced the idea of managing a cache for frequently-used configuration bitstreams so that they can be quickly swapped into FPGA hardware.

FPGA companies have been slow to develop easy-to-use tools and interfaces that could aid an FPGA designer's use of RTR on a per-application basis. More recent Virtex devices do support a configuration interface (ICAP) that is writable from inside the device. To some extent, the limited RTR support to date may be a result of a perceived lack of consumer demand and the slow identification of applications that could benefit from the approach. The application space that can benefit from RTR has recently grown to the point that both Altera and Xilinx are producing devices with partial RTR capabilities, and Altera is actively working to improve user support for the feature.

2) Operating System Support for FCCMs: The need to swap specialized reconfigurable circuits at run time requires high-level run-time software management beyond what is required for a static hardware implementation. Such management must consider the size of the reconfigurable resources and the likely performance benefit and overheads of performing circuit swapping. Typically, software components must be added to the FCCM operating system to dynamically manage available hardware resources. Over the past fifteen years, a number of operating system components that attempt to satisfy this need have been developed. [Burns1997p66] described a virtual hardware manager that allowed for the run-time swapping of pre-placed and pre-routed macro blocks in an FPGA. A more comprehensive paper [Fu2005p149] almost a decade later addressed operating system support for dynamically placing portions of an application into FPGA logic. The FPGA was viewed as a coprocessor in a microprocessor-based system and its configuration was scheduled considering multiple software threads of execution. A follow-on paper [Fu2008p87] examined how frequently the scheduling of computation to reconfigurable resources should occur. Two similar approaches [Fahmy2009p55] [Bauer2012p208] considered the high-level treatment of hardware and software versions of tasks during operating system scheduling. Both approaches attempted to shield the user from the low-level details of the hardware system implementation. [Rupnow2009p63] examined techniques to deal with intermediate state in reconfigurable coprocessors when an executing thread is preempted.

FCCM operating system components can also serve other functions beyond RTR. For example, operating systems can be enhanced to save system energy and increase system flexibility. [Ou2005p139] described a series of energy-saving real-time operating system techniques for FCCMs, including dynamic system shutdown and reactivation of reconfigurable resources via clock manipulations. An operating system that allowed for the easy integration of new hardware modules into the application platform was described in [Ismail2011p170].

Recently, operating systems for platforms that include one or more microprocessors and reconfigurable resources have gained in popularity (e.g. MicroC/OS), although commercial operating system support for RTR management is very limited.

3) RTR Applications: As mentioned earlier, a limiting factor in the development of commercial RTR tools has been the slow identification of applications that can benefit from the approach in a substantial enough way to justify the required extra design steps and run-time management. [Lemoine1995p90] described an RTR accelerated search of the human genome. [Villasenor1996p70⋆] used full-chip RTR to support a set of specialized search patterns that were too large to simultaneously fit on a single FPGA. [Bondalapati1999p249] dynamically managed the precision of computation based on the precision of input operands through RTR. The efficient implementation of vector products using RTR constant multiplier trees whose values change over time was explored in [Benyamin1999p188]. [Liang2004p91] illustrated how RTR could be used to optimize the energy efficiency of a communications decoder by dynamically updating the hardware parameters of the implemented turbo code algorithm to adapt to changes in channel noise. [Garcia2009p243] dynamically updated the hardware of Kalman filters as the operating environment of a wireless sensor network evolved. [Dennl2012p45] examined the use of partial FPGA reconfiguration for a reconfigurable device that accelerated SQL database searches.

After many years of slow growth in the research community, FPGA RTR is gaining traction in the commercial networking and telecommunications domains where hardware configurations must change to match protocol needs. In these applications, portions of the design must stay active while reconfiguration takes place, encouraging revitalized vendor support for partial FPGA reconfiguration.

C. Looking Forward

For RTR to gain in popularity, standard models for describing or automatically extracting opportunities for RTR at higher levels of application design must be developed. Advanced tools that can identify, synthesize, and deploy similarly-sized design regions of an application for use in RTR will be needed as the RTR application space expands. Modifications in device architecture to allow easier access and interpretation of device configuration information are likely to positively influence this effort. The use of heterogeneity will also push the need for more complex operating systems to manage both resource mapping and accompanying run-time reconfiguration.

VII. Applications

Over the last twenty years a very wide range of applications have been implemented with FCCMs and presented at the FCCM conferences. Many of these applications can be organized into the five large classes we survey here.

A. Cryptography

Cryptography has been an important application for FPGA acceleration since the beginning of the conference [Cuccaro1993p121]. Bit-level manipulation is supported efficiently on FPGAs, and bulk encryption is often amenable to streaming data through a highly pipelined datapath. Because of drastically improved performance versus microprocessor implementations, encryption/decryption has often been used to demonstrate the potential benefits of an FPGA accelerator (e.g. [Hauser1997p12⋆] [Leong2001p170⋆] [Schmit1997p47⋆] [Vuletic2004p24]). Papers at the conference have shown how to implement DES, AES, IDEA, Elliptic Curve, and RSA encryption schemes.

1) Private Key Encryption/Decryption Acceleration: Early papers displayed the benefits of reconfigurable implementations of the Data Encryption Standard (DES) [Hauser1997p12⋆] [Leong2001p170⋆]. [Patterson2000p113] showed how to get even better performance using JBits to specialize the DES engine around the mode of operation and a particular key schedule. When the US National Institute of Standards and Technology (NIST) started a search for a DES replacement, [Dandalis2000p132] explored the FPGA implementation of the Advanced Encryption Standard (AES) candidate finalists and suggested a general architecture for a flexible FPGA-based cryptography engine. Once AES was standardized, [Drimer2008p99] showed how to exploit the dedicated DSP and memory blocks in modern FPGAs to achieve high AES throughput, achieving over 55 Gb/s on a Virtex 5. Several papers also explored the acceleration of the International Data Encryption Algorithm (IDEA) [Leong2000p122] [Schmit1997p47⋆] [Vuletic2004p24]. [Dollas2003p19] implemented the SCAN algorithm, which was designed specifically for image encryption.

2) Public Key: FPGAs have also been heavily used for public key encryption including elliptic curve cryptography. [Cuccaro1993p121] showed how attached FPGAs can accelerate modular multiplication. [Orlando1999p232] explored the implementation of Galois Field multipliers for elliptic curve cryptography. [Leung2000p68] and [Antao2009p193] provided compact microcoded implementations of elliptic curve cryptography, while [Jarvinen2008p109] described pipelined and specialized implementations to achieve higher throughput and lower latency.

3) Breaking Encryption: The high throughput of FPGAs on regular tasks makes them efficient at the kinds of brute force search necessary to attack encryption. [Tsoi2002p13] showed how to use FPGAs for brute force key search for Rivest Cipher 4 (RC4), as used in the Secure Sockets Layer (SSL), Secure Shell (SSH), and Wired Equivalent Privacy (WEP). The paper showed that a single 2002-era FPGA could recover a 40b key in 50 hours and that the approach could be trivially parallelized onto multiple FPGAs. [Simka2005p107] and [deMeulenaer2007p197] showed how to build highly-efficient FPGA hardware implementations of sieve operations that factor large numbers to attack the Rivest-Shamir-Adleman (RSA) encryption algorithm.

4) Strengthening Cryptographic Implementations: Papers have also explored how to support or enhance components of cryptosystems. [Tsoi2003p51] showed how to build compact true random number generators based on oscillator phase noise as well as compact pseudo-random number generators. [Kaps2010p273] described how to build AES implementations that were more resistant to differential power analysis attacks without excessive overhead. [Schafer2010p265] used FPGAs to sieve for Bent Boolean Functions that could be used to design better block and stream ciphers. The *Workshop on Cryptographic Hardware and Embedded Systems* has regularly included FPGA implementations since its inception in 1999.

B. NP-Hard Optimization Problems

Optimization problems are a large class of applications that require sizable amounts of computation. This is particularly true when the optimization is NP-hard. For these problems, the ability to evaluate options in heavily pipelined, parallel computations makes them attractive candidates for FPGAs. Furthermore, the problem statement and state for optimization problems is often small compared to the computation required, so they avoid running into memory or data transfer bottlenecks.

Boolean Satisfiability (SAT) is the canonical NP-complete problem. Both because of its own importance and the fact that other NP-hard problems can be recast as SAT, it is an important application target. It is particularly attractive because the evaluation of formulas can be cast directly as a combinational circuit that can be efficiently evaluated on an FPGA. Consequently, there is considerable parallelism and bit-level mapping efficiency to be exploited even before exploring pipelined, parallel evaluation. [Zhong1998p186⋆] introduced this approach at the conference and demonstrated three orders of magnitude speedup on many problem instances. The time to place-and-route the instance-specific design for the FPGAs presented a potential bottleneck that might undermine the performance gains. [Rashid1998p196] addressed this mapping time by partitioning the design into a fixed portion and a variable portion that is customized for the specific SAT expression under examination. In these early designs, the size of the SAT instance that the FCCM could attack was limited by the size of the platform. [Abramovic1999p306] and [deSousa2001p239] showed how to decompose large SAT problems into smaller ones that could be sequenced on the limited FPGA resources and how to avoid instance-specific mapping. [Fuess2008p119] extended the kind of pruning and state-enumeration search used in efficient SAT to solve the problem of explicit-state model checking.

Outside of SAT, several other NP-hard problems have been directly attacked. [Plessl2002p163] accelerated set covering, specifically as it shows up in logic synthesis. [Chan1997p175] showed how to accelerate the data structures required to support a router, and [DeHon2002p205] described how to use parallel, spatial hardware to perform the single-source, shortest path search which is at the core of FPGA routing. [Alves1999p168] explained how to solve nesting problems, and [Dollas1998p48] described how to accelerate the derivation of a Golomb ruler.

FPGAs have also been applied to efficient approximation algorithms for NP-complete problems. [Mavroidis2007p13] provided a 2-approximate solution to the Euclidean Traveling Salesman problem that was six-fold faster than the best known software solution at the time for small problem instances.

C. Pattern Matching

Pattern matching tasks, where it is necessary to identify specific patterns within a large data set, require a considerable amount of computation, but the computational tasks are generally very regular. As such, these tasks exploit the natural strengths of FPGAs. Matching applications that support target recognition within images, packet filtering in network data streams, and biological sequence identification in large genomic databases have all proven to be rich application classes for FCCMs.

1) Automatic Target Recognition: ATR uses image matching to locate and identify specific target objects. The ATR task demands a large amount of regular, bit-level computation that could not be performed in real-time on the leading microprocessors of the mid-1990s. Typical ATR implementations search for any of a large set of shapes within an image.

[Villasenor1996p70⋆] exploited the FPGA's ability to rapidly modify its gate-level logic through fast RTR. The FPGA could be specialized to allow a search through a series of match templates. New templates could be dynamically loaded into the FPGA as needed. As a result, the approach was able to achieve real-time matching on hundreds of 16×16 pixel target images. [Rencher1997p192] revised the design for the multi-FPGA Splash 2 [Arnold1993p88], improving performance and achieving a higher level of target sensitivity (i.e., more pixels per target). [Hemmert2001p199] avoided the large compile time necessary to generate unique bitstreams for a set of image operations by using a generic architecture and specializing only the LUT ROMs to the particular operation. [Bohm2001p209] and [Bohm2002p301] showed how the problem could be specified in the high-level SA-C language and could be automatically mapped to run in real time, 800-fold faster than a contemporary Pentium-based PC.

2) Bioinformatics: Deoxyribonucleic acid (DNA) sequence matching on FPGA accelerators has been of great interest since the first FCCM. [Hoang1993p185⋆] presented a classical, dynamic programming (DP) approach to the problem of homologous series detection (the matching of a target DNA sequence to a reference DNA sequence in a database) that outperformed conventional computers by several orders of magnitude. Software implementations of DP techniques, such as the Smith-Waterman algorithm, were challenged to keep pace with the exponential growth in the GenBank database of known DNA sequences. [Lemoine1995p90] extended the dynamic programming approach to match against the then-new Human Genome Database (3 billion base-pairs) in real-time using specialized configurations and RTR.

Ten years later it was apparent that the GenBank database was doubling in size every 18 months and that conventional DP approaches were too slow. The Basic Local Alignment Search Tool (BLAST) arose as a new heuristic approach that used approximate string matching to identify homologous series. This approach became the de facto standard for sequence matching. [Herbordt2006p217] presented a reconfigurable computing hardware implementation of both BLAST and the DP algorithm that scanned the reference genomic data in a single pass. This paper sparked a series of BLAST-oriented accelerators. [Jacob2007p95] accelerated the initial phase of BLAST, the "seed-generation." [Park2009p81] prefiltered the reference DNA database and produced a ten-fold speedup. [Datta2009p88] implemented the computationally intensive part of the BLAST algorithm with a reconfigurable accelerator.

During recent years the focus in bioinformatics has shifted, driven by the rise of high-throughput Next Generation DNA Sequencers that generate up to a million DNA fragments a day, each of which contain up to several hundred base-pairs ("short-read" sections). These sequencers have the potential of producing personal DNA genomes at a low cost that identify genetic-based disease markers and abnormalities for a given individual. These short-read sequences are aligned against a reference DNA sequence to get the best (and fastest) alignment and to identify potential mutations (additions, deletions and/or substitutions of base-pairs) in the target DNA being analyzed. [Alachiotis2011p226] used a combination of software and reconfigurable hardware to achieve a two order of magnitude speedup over a software-only solution. [Olson2012p161], recognized as the best paper in FCCM2012, tackled the problem of matching millions of short read sequences against a reference DNA sequence through the hardware-based acceleration of the popular BLAT-like Fast Accurate Search Tool (BFAST) software algorithm. [Preusser2012p169] investigated a highly parallel (thousands of search engines per FPGA) systolic design, and [Tang2012p184] used a hash table technique to identify and place the short-read sequences. The best paper at FCCM2010 [Jacob2010p87] examined the use of RNA folding. Other bioinformatics-related FCCM papers included [Bakos2007p85], [Martinek2010p79] and [Mahram2012p177].

The rapid advances in DNA sequencers and the resultant generation of terabytes of genomic data have set the stage for many more FPGA-based accelerators aimed at specific bioinformatics problems.

3) Network Intrusion Detection Systems: Over the last twenty years the Internet has grown from a few technology-aware early adopters to hundreds of millions of daily users. Unfortunately, the rise of malicious Internet traffic has kept pace and has grown in sophistication. Viruses, worms, Trojans, and zombie bots are just a few examples of these hostile Internet packet riders. Today's firewalled PC user has anti-virus and anti-malware software to defend against attacks inside the computer. Network Intrusion Detection Systems (NIDS) use regular expressions to identify malicious packets and prevent them from being delivered to host computers. [Sidhu2001p227⋆] introduced a technique for compactly implementing regular expression Non-deterministic Finite Automata (NFAs) on FPGAs. [Hutchings2002p111⋆] showed how this regular expression matching could be applied to the SNORT NIDS rule-set database to implement hundreds of state machines on a single FPGA. The resulting acceleration engine ran at over 600 times the speed of SNORT in software.

The growth of Internet speeds from 100 megabits per second (Mbps) to 1 gigabit per second (Gbps) to over 10 Gbps challenged the early researchers in network-based NIDS. [Bellows2002p121] examined the offloading of network processing from the host CPU to the network interface card to achieve low Gbps rates. Another issue was the scanning of the full contents of the Internet data packet itself at Gbps data rates. [Moscola2003p31] implemented a real-time network packet scanning Internet firewall that could review packet contents, drop the detected hostile packets and inform network administrators. [Lee2003p39] used firewall access rules encoded in a high-level language and downloaded them after conversion to a lower level representation. [Cho2004p125] sustained a filtering rate of over 3 Gbps while examining the payload of each packet in depth. [Baker2004p135] optimized the rule-based data in software for a reconfigurable packet analyzer. [Clark2004p249] and [Sourdis2004p258] explored different hardware architectures to scan for hostile data packets at a 10 Gbps rate.

The NIDS work intensified as the Internet became more and more hostile. The SNORT rules grew in size to encompass the ever expanding number of viruses and worms. [Cho2005p215] furthered earlier work with a dual reconfigurable processor approach; one processor optimized the rules database in real-time and the other scanned each Internet packet against it. Using a similar approach, [Attig2005p225] developed a software-based front end rule processor to support all the features found in the SNORT rules while handling 10 Gbps data rates. [Singaraju2005p235] developed a new NIDS architecture based on a CAM-based cellular processor. [Jedhe2008p43] explored the problem of packet classification for high speed routers which was driven by the rise of streaming video over the Internet.

FPGA-based networking systems are uniquely positioned to provide solutions in this ever-changing, ever-faster Internet environment. The acceptance of FPGA solutions to NIDS and other networking data and control plane issues are frequently seen at networking conferences such as the *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, the *IEEE International Conference on Communications*, the *IEEE International Conference on Computer Communications*, and the *IEEE International Conference on High Performance Switching and Routing*.

D. Floating-Point Arithmetic

In 1993, when FCCM started, FPGAs were too small to implement full floating-point (FP) datapaths. FPGAs were known to be viable only for integer or scaled integer formats due to the small gate-equivalent counts of early 1990s FPGAs. Early attempts at FP on FPGAs, such as [Shairzi1995p155], used non-IEEE 754 FP with limited precision. As FPGA capacity grew, researchers investigated different implementations for FP square roots [Li1997p226], FP adders and FP multipliers [Louca1996p107]. [Liang2003p185] and [Wang2003p195] explored tradeoffs in FP implementation throughput, latency and area.

With the advent of the modern, large gate-equivalent FPGAs and FP libraries, FP arithmetic on an FPGA has become much easier and more practical to use. It cannot challenge the peak capacity of GPGPUs, which contain hundreds of FP-capable processors, but [Underwood2004p219⋆] showed that FP FPGA arithmetic may surpass the FP performance of the conventional PC microprocessor on memory-bound, double-precision (DP) FP dense matrix operations, vector dot product, matrix-vector multiply, and matrix-matrix multiply operations. This research sparked a flurry of DP FP research on FPGAs. It was extended to DP FP FFTs [Hemmert2005p171], DP FP eigenvalue solvers [Huang2010p95], and quad-precision FP multipliers [Jaiswal2012p25]. The availability of high-quality, open-source, flexible FP libraries [Wang2006p249][Tsoi2003p51] that supported a wide range of operations (division, square root, accumulation), precision, and pipelining has also driven greater exploitation of FP on FPGAs. [Kapre2009p37] showed that FPGA implementations of SPICE-level device model evaluation could run orders of magnitude faster than microprocessors.

Floating-point FPGA implementations of applications now appear in conferences such as the *Supercomputing Conference* and the *IEEE International Parallel and Distributed Processing Symposium*.

E. Molecular Dynamics

The three-dimensional modeling of physical phenomena on a discrete, particle-by-particle basis is another great challenge with an insatiable need for computation. Models were developed for the N-body problem [Lienhart2002p182] [Tsoi2004p68], fluid flow, systems of charged particles, and plasmas where large groups of discrete particles interact with each other in a complex manner.

In molecular dynamics (MD) simulations, a large number of particles (thousands) are modeled at the atomic level using Newtonian mechanics. The forces on each particle are summed and then integrated using the classical equations of motion. Among the many applications of this technique is the modeling of biological molecules, including protein folding. [Azizi2004p197⋆] demonstrated that an FPGA clocked at 100 MHz had a 20-fold performance advantage over a GHz microprocessor. [Scrofano2006p23] replaced the original fixed-point arithmetic with single-precision, floating-point arithmetic, and used a hybrid software-FPGA approach where the FPGA accelerated only the compute intensive portion. The approach was able to model clusters with over 50,000 particles. [Gu2007p117] modeled molecules with a combination of forces, including localized or bonded forces (covalent, hydrogen bonds) and non-bonded forces (van der Waals, Coulombic) and divided the system into cells or grids. Instead of modeling all the particles during every time step, [Herbordt2008p248] achieved additional speedup by modeling each molecule on a discrete event basis and using a priority queue to sort events along the timeline.

The simulation of molecular dynamics will continue to grow as more focus is put on protein folding and the more general goal of understanding biological processes at the molecular level. In the future it will be necessary to understand how a 2D-encoded structure of amino acids folds into a repeatable, biologically-active 3D protein molecule.

VIII. Conclusion

In twenty years, reconfigurable computing has grown from a wild, exploratory idea to a viable alternative to Application-Specific Integrated Circuits (ASICs) and fixed microprocessors in our computing systems. With increasing energy and reliability concerns, there is every reason to believe its importance will grow as technology scales. As we move to larger chips with heterogeneous fabrics and accelerators, reconfigurability will continue to merge into the mainstream computing infrastructure.

Over the past twenty years, the FCCM community has pioneered accelerator architectures, hardware specialization, and RTR models and support. FCCM has collected ample evidence of the superior performance and energy efficiency of FPGAs compared to processors over a broad range of applications. Furthermore, it has covered significant advances in the capture of applications, C-to-gates compilation, and streaming models and architectures. With the possible exception of RTR, all of these contributions are central to today's mainstream computing trends.

Having survived its adolescence and looking to a healthy prime, there is still much to learn and a long way to go before the field begins to mature. Ease-of-use, ease-of-debug, accessibility, and a slow edit-compile-debug cycle remain challenges that must be addressed to achieve broader appeal. Hybrid architectures and RTR, while attractive, need better abstraction models and support. Demonstrations and patterns of effective use in large-scale, complex applications are still needed, as is a better understanding of the relative strengths and weaknesses of reconfigurable architectures, GPGPUs, and massive many-core chips.

Acknowledgment

Jeff Arnold, Jonathan Babb, Paul Chow, Miriam Leeser, Wayne Luk, and Viktor Prasanna provided valuable corrections, feedback, and insights for this article.

References

[1] W. S. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable logic array," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1986, pp. 233–235.

[2] J. Varghese, M. Butts, and J. Batcheller, "An efficient logic emulation system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 171–174, June 1993.

[3] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and using a highly programmable logic array," *IEEE Computer*, vol. 24, no. 1, pp. 81–89, January 1991.

[4] P. Bertin, D. Roncin, and J. Vuillemin, "Programmable active memories: A performance assessment," DEC Paris Reserch Laboratory, 85, Av. Victor Hugo, 92563 Rueil-Malmaison Cedex, France, PRL Report, June 1992.

[5] T. Kean and J. Gray, "Configurable hardware: Two case studies of micro-grain computation," *Journal of VLSI Signal Processing Systems for Signals, Image and Video Technology*, vol. 2, no. 1, pp. 9–16, 1990.