

A Methodology for Generating Application-Specific Heterogeneous Processor Arrays

Stephen Craven, Cameron Patterson, Peter Athanas
Virginia Polytechnic and State University
{scraven, cdp, athanas}@vt.edu

Abstract

Hardware designers are increasingly turning to Single Chip Multi-Processors to achieve power and throughput goals. To further increase performance for a specific application the chip's array of processors can be tailored to their program, creating an application-specific multiprocessor. While much Hardware / Software Codesign research has been conducted on optimizing heterogeneous processing arrays, shortcomings exist in design scalability, simulation, verification, rapid prototyping, and the requirement of specialized skill sets.

To address these deficiencies, a design methodology is proposed targeting signal processing applications that maps a parallelized C program onto a homogenous array of processors linked by simple point-to-point connections. By individually optimizing each processor for its specific program the performance of the array is increased, while the common basic interface between processors eases optimization, implementation, debugging, and verification.

1. Introduction

The performance returns for traditional methods of extracting Instruction Level Parallelism (IPL), such as superscalar architectures, are receding as wire delays eat into the timing budgets of modern processors and power dissipation requirements become unmanageable. To combat these diminishing returns, designers are increasingly turning to Single-Chip Multiprocessors (SCMPs) [13] to improve performance, area, and power for server [8], supercomputing [1], and embedded applications [9]. The die size available in modern process technologies is quite large. Over 3,000 Arc600 32-bit RISC processor cores can fit on a single die, resulting in a throughput of over 600,000 MIPS and producing 25,000 MIPS / watt [3].

The major processor vendors AMD, IBM, and Intel all currently offer dual-core processor chips. The next generation gaming platforms for both Microsoft and Sony

utilize SCMPs; in the case of Sony's Cell processor, featuring an eight-processor chip [6]. This trend is evident in the mobile computing arena as well. Nintendo's DS handheld gaming device uses two different ARM processors. Cellular phones typically couple a Digital Signal Processor (DSP) on the same die as a microcontroller.

The smaller processor cores in an SCMP simplify design and verification costs through IP reuse, allow for higher clock frequencies due to shorter wire lengths, and provide better scalability due to the smaller processor granularity. These benefits, however, come at the expense of increased programming effort and reduced performance on single threaded applications. For mobile applications, where power consumption is of particular concern, having an array of smaller processors is advantageous as entire processors can be turned off to tailor the throughput of the array to the application's current requirements. Also, the shorter wire lengths found in these processors require less power to drive than the long wires found in Pentiums.

A major impediment to the wide acceptance and use of multiprocessing has been the communication latency between processors. Few applications can easily tolerate the hundreds of clock cycles of latency needed to transfer data across separate computers. However, when the processors are placed adjacent to each other on the same die, interconnect delay between them drops to less than the external memory latency. This reduced latency greatly expands the number of applications that can benefit from parallel implementations.

For desktop and server applications, SCMPs are created by tiling identical processors into a homogenous array. In the mobile computing world, SCMPs may have a variety of different processors running at different frequencies, and perhaps even different voltages, with the various parts of an application assigned to the processor resource best suited for the task. In each domain, further performance gains are possible by optimizing individual processors to their assigned task. By creating Application-Specific Instruction set Processors (ASIPs) area can be reduced by removing unused instructions and throughput increased through custom instruction creation

while retaining most of the flexibility of software programming [16].

While single processor development and programming is well established and methodologies exist for converting sequential programs into parallel processes [22], like most problems in Hardware / Software Codesign, design methodologies for application-specific multiprocessor arrays have remained a topic of intense research with few practical applications. HW / SW Codesign research generally focuses on optimizing a mixture of commodity processors and hardware co-processors. Most design methodologies ignore all possibilities of processor optimization beyond cache sizing.

Current HW / SW Codesign methodologies found in literature produce many design difficulties that have inhibited their acceptance. The input to these design flows is generally specified as a task graph or object-oriented code describing the application. The suitability of the resulting design is largely dependent on having an acceptable initial task flow graph, in terms of granularity and task division. However, the selection of an appropriate initial task-graph can be non-intuitive, particularly to those not versed in both hardware and software.

Codesign methodologies generally operate at a level of abstraction that, while greatly simplifying the analysis, glosses over the issue of HW / SW interfaces [7]. Implementing such a design requires detailed hardware and software knowledge to correctly design the lowest level interfaces.

Further complications exist in simulating and verifying a design with large hardware and software components. While much research has gone into this area, integrating an effective hardware simulation environment with one or more processor simulators can still be a daunting task, especially with a custom processor. Similarly, rapid prototyping tools for such designs are practically non-existent. While FPGAs can be used as prototyping platforms for smaller ASICs [10], modern processors are not easily amenable to rapid prototyping in an FPGA. Current solutions involve complicated prototyping platforms that simulate the processors in software and the hardware with attached FPGAs [18].

Applications implemented with a HW / SW Codesign flow may have a heterogeneous mixture of various processors, hardware co-processors, and communication buses. This irregular structure leads to scalability difficulties – to utilize an increased silicon budget the designer must return to the beginning of the design process.

In spite of more than a decade of dedicated research, no commercially successful automated HW / SW Codesign design flow exists. Typical industry design

flows resemble a Y-chart – hardware and software designers work separately on their respective parts after an initial HW / SW partitioning. This separation of hardware and software design increases design time and system complexity. Furthermore, the suitability of the final design largely hinges on the appropriateness of the initial HW / SW partitioning.

An automated method for generating an optimized application-specific multiprocessor array is presented that addresses many of the deficiencies of HW / SW Codesign flows. Starting solely from an application description written in a High Level Language (HLL), our method enables rapid prototyping, eases debugging and verification, greatly improves scalability, simplifies component interfaces, and eliminates the need for detailed knowledge of hardware. With the application described as a scalable parallelized C program, our methodology maps the application onto an array of soft processors connected by a simple, common interface. Soft processors are processors created out of the configurable logic in an FPGA. By iteratively applying optimizations to these processors the overall throughput of the array is improved while the common interface is maintained, improving scalability, verification, and debugging.

The remainder of this paper is organized as follows. Section 2 reviews existing research in Hardware / Software Codesign, SCMP designs, and ASIP design methodologies. Section 3 presents our design methodology for creating an optimized heterogeneous SCMP. A summary of the paper is given in Section 4.

2. Related Work

Much research has been conducted on SCMPs and only a few of the major projects are reviewed. Stanford's Hydra multiprocessor [5] integrates four MIPS processors linked by two buses. Specialized hardware performs thread-level speculation allowing sequential code to be executed across multiple processors. While impressive speedups were obtained the communication structure limits the number of nodes that can be integrated. Furthermore, a programmer is able to more effectively extract thread level parallelism than an automated, real-time approach.

MIT's RAW processor [21] is comprised of tiles of processors, each tile being an 8-stage MIPS processor with a floating-point unit. Data can move between processors with as little as a three-cycle latency; however, roughly one-half the die area is consumed by network-related functions. Furthermore, the large size of the processing nodes limits the number of processors per chip to 16 in their implementation.

Of the multiprocessor projects in literature, the computational fabric of Wolinski et al. [24] most closely resembles our array implementation. The computational fabric they present consists of 52 simple 16-bit computational nodes with redefinable instruction sets controlled by a 32-bit master processor. The interconnect between nodes allows for single-cycle communication between adjacent processors. Our design methodology aims to automate the production of a similarly connected master-controlled fabric of much more capable processors tailored to a specific application.

Over the past decade much work has focused on the HW / SW Codesign problem. For a review of this research see [23]. Most of the research focused on the use of commodity processors with no optimization and is of little relevance to our application. Sun et al.'s work on creating optimized arrays of heterogeneous ASIPs is perhaps most attune to the work proposed here. Sun et al. [20] developed an iterative flow for optimizing an array of ASIPs where the task scheduling and processor instruction set are optimized at each step. The result is a high level description of an optimized array of heterogeneous processors with an optimized scheduling routine. Sun demonstrates their architecture by implementing multimedia and encryption algorithms on two of Tensilica's Xtensa processors. Their results indicate a 1.4 – 3.1 times speedup over a non-optimized implementation.

While the results of Sun et al. are impressive, several issues are not addressed. Tensilica's Xtensa processors feature an impressive tool set and allow for configurability of the processor through instruction set extension, data level parallelism extraction through SIMD units, and instruction level parallelism extraction through multiple instruction issues (VLIW) – however, the Xtensa instruction set cannot be reduced beyond the initial processor, burdening the implementation with unneeded instructions and thus wasted silicon.

The configurability is also limited by the datapath width, possibly hampering the processor's maximum speed for applications using less than the minimum datapath width. Furthermore, both Sun's and Tensilica's optimization flows only allow optimization of the processor's instruction set, ignoring all possibility of creating a custom datapath implementation of a program's inner loop.

Finally, the output of the Sun et al. flow is a high level definition of the multiprocessor implementation. The engineer still must construct and debug the interface between processors and peripherals – an easy task perhaps for two processors, but a daunting one for dozens of nodes. The integration effort becomes significantly more difficult if different architectures are present. Simulating,

debugging, and verifying two Xtensa processors connected by a single bus is well within the capabilities of most design teams. The addition of IP hard cores, custom chip interfaces, external peripherals, and DSPs, all communicating over a shared bus or a network-on-chip, greatly complicates the situation. The interactions between components may not yield themselves to high-level simulations. The engineers implementing such a design require a thorough knowledge of all the hardware, software, and interfaces involved.

3. Design Methodology

The methodology proposed here begins by mapping a parallelized C program onto an array of soft processors on an FPGA like that shown in Figure 1. As many OpenFire processors as possible are placed in the available area on the FPGA. While various network topologies are possible, Figure 1 shows a 2-D mesh.

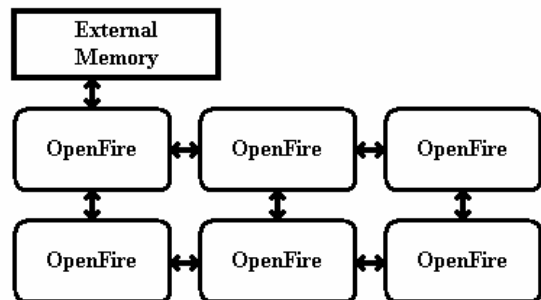


Figure 1. Initial homogeneous array

Upon completion of several iterations of our approach, the processors have been optimized in terms of speed and area. The resulting array, a hypothetical representation of which is shown in Figure 2, may include custom hardware datapaths, processors with extended instruction sets (shown in shades of gray), and more processing nodes due to area saved by decreasing datapath widths and removing instructions. The processor interfaces, however, remain unchanged.

The designer can apply numerous techniques to the creation of the scalable parallel input programs. Tools exist to ease in the development of parallel programs [3]. In this initial program creation the designer extracts task level parallelism from the application. While this manual parallelization task increases the designer's workload, it is believed that in the near future tools and techniques will exist to ease this burden. Also, as future architectures are moving towards multiprocessor implementations, the

designer will likely have little choice but to either write a parallel program or design a custom hardware core.

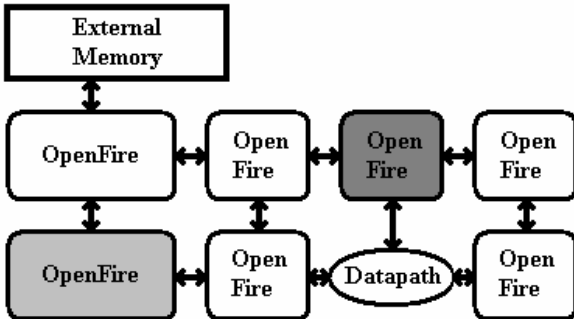


Figure 2. Array after optimization

Our method allows for rapid prototyping of the design even before optimizations have begun by simply mapping the C code onto an array of soft processors on an FPGA. The scalable nature of this input code allows for the application to benefit from the additional processors that can be added to the array if optimizations free up extra area by shrinking some processors.

For initial investigations the open-source OpenFire processor [4] is being used. OpenFire is a 32-bit configurable RISC processor that is binary compatible with the Xilinx MicroBlaze Instruction Set, allowing the use of the MicroBlaze's mature toolset. A comparison of several configurable soft processor cores [14] found that the MicroBlaze had the highest performance per area. The OpenFire is of comparable performance, running 30% slower while requiring 13% less area than the MicroBlaze. OpenFire configuration options include the addition or removal of instructions, varying the width of the datapath, and adjusting memory sizes. As the OpenFire was designed specifically for configurable array research, it is not hampered by functions unnecessary in such an environment, such as interrupt and exception handling and peripheral bus interfaces.

The processors in the array communicate through a basic FIFO interface, similar to the MicroBlaze's Fast Simplex Link bus, to simplify optimizations and debugging as well as improve scalability. This interface consists of a FIFO buffer of configurable depth sized to the processor's datapath width. Data is written to one side of the FIFO and read from the other. This basic interface eliminates the need for interrupt-driven communication, simplifying the processor. This interface is well suited to signal processing and other streaming applications such as network applications, which exhibit a regular communication pattern. In such applications, data transfers generally are only conducted between nearest neighbors. The benefits of a simple processor

interconnect network have been recognized. An analysis performed by Tensilica [11] examined various communication schemes for arrays of configurable processors before settling on FIFO-like data queues for connecting processors.

For processors with datapath-intensive programs, this simple interface greatly eases the replacement of the processor with a custom hardware datapath. All processor optimizations occur inside these basic interfaces. As the optimization process does not change the communication between nodes, these constant interfaces provide an excellent observation point for debugging and verification.

By default it takes four cycles to move data through the link from one processor to a neighbor. However, optimizations exist to reduce this latency to a single cycle by extending the instruction set to allow for ALU-to-FIFO and FIFO-to-ALU operations. Such low inter-processor latencies allow for flow-through processing – an idea found in computational fabrics whereby data flows seamlessly from one processing node's ALU to its neighbor's just as if the two execution units were part of the same hardware datapath.

The proposed design flow, shown in Figure 3, begins with a scalable parallel C program written for the OpenFire processor. An analysis of the program allows for the selection of the major architectural parameters of the processors in the array. Through a simple simulation of the compiled program the minimum sizes of the instruction and data memories can be found. Due to the use of fast, on-chip memory for the processors, cache is not needed and the problem of sizing cache for optimum performance is avoided. The designer, familiar with the application's requirements, can select a smaller or wider datapath width as needed. The network topology is determined by the needs of the parallel program. These parameters serve as a starting point for optimizing the processor array and can be modified later.

After the initial processor structure has been determined, as large of an array of OpenFire processors is created as allowed by the area constraint. The interconnect topology depends on the application, but 2-D and ring structures are expected to be most common. The programs are then mapped to the processor array and simulated in the parallel programming environment CMPWare [3]. CMPWare was chosen for its extremely simple processor description process. After every optimization cycle the description of each processor or hardware datapath is updated inside CMPWare. The unique nature of CMPWare allows any combination of software processor or hardware datapath to be simulated. The level of simulation is instruction-accurate, each simulated instruction executing in a single clock cycle.

While not as precise as a cycle-accurate simulator, the reduced computational requirements allow for reduced simulation times, with up to 2 million cycles simulated per second.

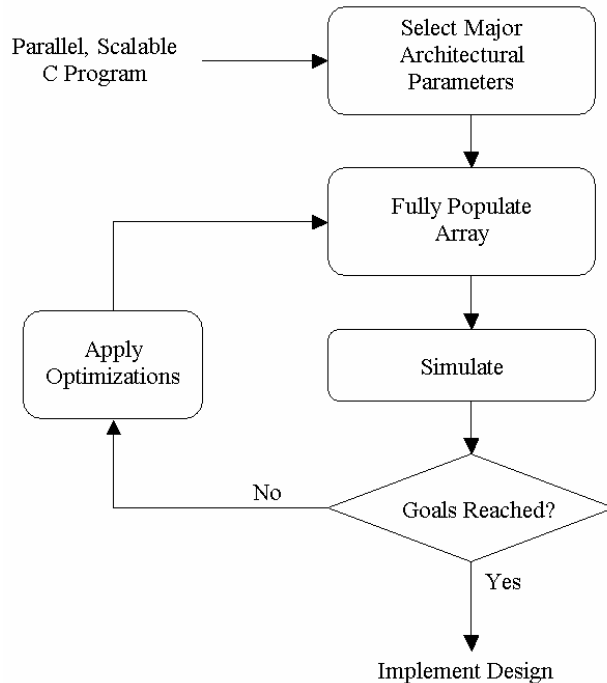


Figure 3. Simplified design flow

The advantage to using CMPWare, compared to either a fully cycle-accurate simulator or a more high-level simulation such as MESH [2], lies in CMPWare's ease of integration into an automated flow. To automatically generate cycle-accurate simulators for each processor at each stage of optimization and then integrate these simulators into a single working array would involve a significant investment in software development. If such a simulation environment is deemed advantageous in the future, it can easily be integrated with this design methodology. Higher-level simulators like MESH do not simulate each instruction but instead require processor characterizations. For high-level analysis and trade-off studies using commodity processors this approach is very advantageous. However, as our processors are modified at each iteration of the design flow, the overhead of recharacterizing each processor is deemed too high. Furthermore, while MESH is an excellent analysis tool, its simulation is at too high a level to be used as a debugging and integration tool.

A significant advantage of the proposed design methodology over other HW / SW Codesign flows is this tight integration of a lower-level simulator. Every iteration of our design methodology results in a functional

implementation. Furthermore, the designer can intervene in the simulation step of any iteration cycle to further optimize the software running on the processors.

Assuming performance goals are not met, our design flow applies successive optimizations to the processors until the desired throughput has been achieved. These optimizations, described below, first turn the processors into Application-Specific Instruction set Processors (ASIPs) before creating pure hardware datapath nodes where appropriate. After each round of optimizations the area of the resulting array is estimated and processors are added until the budgeted area is filled. Optimizations are accepted only if the throughput of the entire array is increased. For example, creation of a custom instruction could increase the throughput and size of a processor, resulting in an array of fewer but more capable processors that may have a lower total throughput than before the optimization.

Possible optimizations implemented in our methodology include datapath width adjustment, instruction removal, custom instruction creation through instruction fusion, Single Instruction Multiple Data (SIMD) unit insertion, dual instruction issues, separate communication interface controllers, micro-code replacement of instruction decode, custom hardware datapath construction, and direct ALU-to-interface and interface-to-ALU connections. Optimizations are applied in order of difficulty – simple improvements, such as instruction removal, are applied first.

Instruction removal seeks to decrease processor area and possibly increase the processor's maximum speed. Previous research [15] has shown area reductions of 30% and performance improvements of 24% by removing unneeded instructions on a similar processor. Experience with the OpenFire has shown that the critical path that sets the operating frequency involves the compare operation. If the compare instruction can be eliminated timing improves and area decreases. In some cases, such as multiplication, division, and barrel shifting, operations necessary for program execution can be performed in either hardware or software. If these instructions are implemented in hardware the possibility exists of removing the hardware and replicating the lost functionality in software. In these situations the extended execution times of the additional software functions are weighed against the decreased processor area to determine the benefit of removing these instructions.

Instruction set extension is accomplished through instruction fusion, possibly using a method from Sun et al. [19]. Commonly repeated groups of instructions are examined to determine if their fusion would significantly increase the critical path timing. Instructions considered for fusion do not have to be next to each other in the code,

as instruction reordering is used to enlarge the examined instruction groups. Once instructions have been fused, the instruction set is updated to include the new instruction. The OpenFire processor uses 32-bit instructions (although this can be enlarged) and has space for many new instructions. The assembly code is then modified to take advantage of the new instructions.

Instead of generating compiler tools and recompiling the source code for each optimization, the original compiled binaries are modified to take advantage of the added instructions as in [12]. While a retargetable compiler is desirable to ease code modifications on the optimized array, the additional effort of integrating an efficient retargetable compiler is left as a future research topic.

To allow for the creation of custom datapaths, both as a coprocessor option to the OpenFire and as a processor replacement, two avenues are being explored. In the first case, the automated design flow would identify potential blocks of code for implementation as custom datapaths based on profiling and metrics such as number of iterative loops and control statements, in a manner similar to [17]. The designer then, at his or her discretion, recodes the block in MATLAB and utilizes Xilinx's System Generator [25] to perform MATLAB-to-RTL translation. Alternatively, it may be possible for the processor to be optimized into a datapath controlled by an FSM or microcode controller. If the processor hardware was significantly reduced through instruction removal and the program size reduced through instruction fusion, then possibilities exist to replace the register file with one or more FIFOs and replace the fetch and decode stages with a simplified controller capable of making branch decisions in parallel with datapath computations. Under either datapath creation option, the constant FIFO interface between nodes facilitates the replacement of the processors by custom cores by eliminating complex interface protocols. Little, if any, additional logic is needed by a hardware datapath to communicate via these interfaces.

The optimization iterations continue until the performance goals have been met or all optimizations have been attempted. If the performance goals still are unmet, the initial area constraint can be expanded and the process restarted from the best results obtained during the previous iterations.

A preliminary investigation of this approach [4] focused on datapath sizing as the only optimization. Implementing a median image filter on an array of OpenFire processors, it was discovered that by decreasing the datapath width to 16-bits the processor's area was reduced by 37%. This in turn allowed us to add

additional processors to the FPGA and increase the overall throughput by roughly 60%.

In spite of the potential and promising initial results of this methodology, hurdles remain. Obtaining high resource utilization of an FPGA's configurable logic requires placement constraints to keep routing from becoming untenable. Currently, placement constraints for each processor are added by hand, but an automated scheme is planned. While much successful work has been conducted on ASIP optimization, the goal of optimizing the processor into a hardware datapath is at the early stages of development and is thus untested. However, multiple avenues exist to address these hurdles and the potential benefits of this approach justify their exploration.

4. Conclusions

The issues with HW / SW Codesign methodologies are many and have prevented their widespread use. A new methodology for creating heterogeneous arrays of ASIPs that offers an attractive alternative to traditional HW / SW Codesign flows has been presented. Starting from a high-level C description of the application mapped to a processor array that is iteratively optimized, the difficult issue of initial HW / SW partitioning is avoided. Furthermore, simple FIFO interconnect structures of the network simplify the optimization process and assist debugging. Hardware datapaths can be created to augment or replace the processor all within this constant interface. Finally, tight integration of the design flow with a customizable simulator simplifies verification and integration. Future work will focus on methods for optimizing the processor into a datapath as well as completing the implementation of this methodology.

5. References

- [1] Bright, A., et Al., "Blue Gene/L compute chip: Synthesis, timing, and physical design," *IBM J. Research and Development*, vol. 49, no. 2, Mar. 2005, 277-287.
- [2] Cassidy, A., Paul, J., and Thomas, D., "Layered, Multi-Threaded, High-Level Performance Design", *Proceedings of Design, Automation, and Test in Europe Conference*, Munich, Germany, Mar. 3-7 2003.
- [3] CMPWare, Inc., "An Introduction to Configurable Multiprocessing", Available for download at: <http://www.cmpware.com/Docs.php>.
- [4] Craven, S., Patterson, C., and Athanas, P., "Configurable Soft Processor Arrays Using the OpenFire Processor,"

Proceeding of the International Conference on Military and Aerospace Programmable Logic Devices, Washington, D.C., Sept. 7-9, 2005.

[5] Hammond, L., et al., "The Stanford Hydra CMP", *IEEE Micro*, vol. 20, no. 2, Apr. 2000, 71-84.

[6] Hofstee, H., "Power Efficient Processor Architecture and The Cell Processor", *Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture*, San Francisco, California, Feb. 12-16, 2005.

[7] Jerraya, A., and Wolf, W., "Hardware/Software Interface Codesign for Embedded Systems", *IEEE Computer*, vol. 38, no. 2, Feb. 2005, 63-69.

[8] Kalla, R., Sinharoy, B., and Tendler, J., "IBM Power5 Chip: A Dual-Core Multithreaded Processor", *IEEE Micro*, vol. 24, no. 2, Mar. 2004, 40-47.

[9] Karim, F., Mella, A., Nguyen, A., Aydonat, U., and Abdelrahman, T., "A Multilevel Computing Architecture for Embedded Multimedia Applications", *IEEE Micro*, vol. 24, no. 3, May. 2004, 56-66.

[10] Krupnova, H., and Saucier, G., "FPGA-Based Emulation: Industrial and Custom Prototyping Solutions", *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications*, Villach, Austria, Aug. 27-30, 2000.

[11] Leibson, S., and Kim, J., "Configurable Processors: A New Era in Chip Design", *IEEE Computer*, vol. 38, no. 7, Jul. 2005, 51-59.

[12] Lysecky, R., and Vahid, F., "A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning", *Proceedings of Design, Automation, and Test in Europe Conference*, Munich, Germany, Mar. 7-11, 2005.

[13] Olukotun, K., Nayfeh, B., Hammond, L., Wilson, K., and Chang, K., "The Case for a Single-Chip Multiprocessor", *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, Massachusetts, Oct. 1-5, 1996, 2-11.

[14] Mattsson, D., and Christensson, M., "Evaluation of Synthesizable CPU Cores", Master's Thesis, Chalmers University of Technology, 2004.

[15] Peddersen, J., Shee, S., Janapsatya, A., and Parameswaran, S., "Rapid Embedded Hardware/Software System Generation", *Proceedings of 18th International Conference on VLSI Design*, Kolkata, India, Jan. 3-7, 2005.

[16] Rowen, C., *Engineering the Complex SOC*, Prentice Hall, Upper Saddle River, New Jersey, 2004.

[17] Salice, F., Vecchio, L., Pomante, L., and Fornaciari, W., "Partitioning of Embedded Application onto Heterogeneous Multiprocessor Architectures", *Proceedings of the 2003 ACM Symposium on Applied Computing*, Melbourne, Florida, 2003, 661-665.

[18] Slomka, F., Dorfel, M., Munzenberger, and R., Hofman, "Hardware/Software Codesign and Rapid Prototyping of Embedded Systems", *IEEE Design & Test of Computers*, vol. 17, no. 2, Apr. 2000, 28-38.

[19] Sun, F., Srivaths, R., Raghunathan, A., and Jha, N., "A Scalable Application-Specific Processor Synthesis Methodology", *Proceedings of the International Conference on Computer-Aided Design*, San Jose, California, Nov. 9-13, 2003, 283-290.

[20] Sun, F., Srivaths, R., Raghunathan, A., and Jha, N., "Synthesis of Application-Specific Heterogeneous Multiprocessor Architectures Using Extensible Processors", *Proceedings of 18th International Conference on VLSI Design*, Kolkata, India, Jan. 3-7, 2005.

[21] Taylor, M., et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs", *IEEE Micro*, vol. 22, no. 2, Apr. 2002, 25-35.

[22] Trinder, P., Hammond, K., Loidl, H., and Jones, S., "Algorithm + Strategy = Parallelism", *Journal of Functional Programming*, vol. 8, no. 1, Jan. 1998, 23-60.

[23] Wolf, W., "A Decade of Hardware/Software Codesign", *IEEE Computer*, vol. 36, no. 4, 2003, 38-43.

[24] Wolinski, C., Gokhale, M., and McCabe, K., "A Polymorphous Computing Fabric", *IEEE Micro*, vol. 22, no. 5, 2002, 56-68.

[25] Xilinx, Inc., "Xilinx System Generator v7.1 User Guide", Available for download at: http://www.xilinx.com/products/software/sysgen/app_docs/user_guide.htm.