Portable, Flexible, and Scalable Soft Vector Processors

Peter Yiannacouras, *Member, IEEE*, J. Gregory Steffan, *Senior Member, IEEE*, and Jonathan Rose, *Fellow, IEEE*

***Abstract*—Field-programmable gate arrays (FPGAs) are increasingly used to implement embedded digital systems, however, the hardware design necessary to do so is time-consuming and tedious. The amount of hardware design can be reduced by employing a microprocessor for less-critical computation in the system. Often this microprocessor is implemented using the FPGA reprogrammable fabric as a *soft processor* which presently have simple architectures and moderate performance. Our goal is to scale the performance of existing soft processors hence expanding their suitability to more critical computation. To this end we propose extending soft processors with vector extensions to exploit the abundant data parallelism found in many embedded kernels. Such a *soft vector processor* can execute these kernels much faster than a single-core hence reducing the need for hardware implementations. We observe this improved execution speed through experimentation with vector extended soft processor architecture (VESPA) which is designed, implemented, and evaluated on real FPGA hardware. VESPA is shown to effectively scale performance up to 32 lanes, while providing substantial architectural flexibility to create a fine-grained design space. With these characteristics, and portability across FPGA devices, soft vector processors can provide exact-fit architectures which can efficiently and more easily implement data parallel workloads over custom FPGA hardware design.**

***Index Terms*—Customization, design space exploration, field-programmable gate array (FPGA)-based soft-core processors, processor generator.**

I. Introduction

FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs) are commonly used to realize digital systems because of the low non-recurring engineering (NRE) costs and fast time-to-market they provide relative to the creation of fully-fabricated VLSI chips. The logic density and size of modern FPGAs make them well-suited for hosting complete embedded systems including a microprocessor. But FPGA hardware design is difficult requiring the specification of a cycle-by-cycle description of the system in a hardware description language (HDL), achieving timing closure, and in-hardware debugging. Embedded system designers with aggressive time-to-market constraints often opt instead for microprocessor-based systems with their simpler sequential programming software-based design flows. Since FPGAs can provide increased performance and reduced energy over processors, we are motivated to simplify FPGA hardware design by leveraging the high-level programming and single-step debugging features of software design.

Microprocessors are already used in FPGA systems to interconnect the various system components and coordinate their operations, as well as to perform some light-weight computation. In some FPGAs since the Virtex II Pro [1], these processors can be implemented as *hard processors* implemented directly in silicon with the FPGA fabric around it. However because of the diversity of embedded systems, some need one processor, some need many, and some need none. This leads to device family fragmentation and inevitably more expensive FPGAs. Another option is the *soft processor* which is implemented **using** the FPGA programmable fabric, hence inheriting its end-user configurability. With this configurability soft processors provide the compelling opportunity of being architecturally customized to compete with hardware while providing a much easier software-based design flow. However, to be a viable alternative to HDL design, soft processors require significant performance scaling which is the main thrust of this work.

A. Scaling Soft Processor Performance

The architecture of current commercial soft processors are based on simple single-issue pipelines with few variations, such as the Xilinx Microblaze [2] with three-stage pipeline and the Altera Nios II [3] with up to five pipeline stages. Despite their ease of use relative to FPGA hardware design, they are predominantly used for system control tasks because of the large disparity between the performance possible on a soft processor versus FPGA hardware. To support more compute-intensive tasks on soft processors, they must be able to scale up performance by using increased FPGA resources. While this problem has been thoroughly studied in traditional *hard processors* [4], an FPGA substrate leads to different tradeoffs and conclusions. In addition, traditional processor architecture research favoured features that benefit a large application domain, while in a soft processor we can appreciate features which benefit only a few applications since each soft processor can be configured to exactly match the application it is executing. These key differences motivate new research into scaling the performance of existing soft processors while considering the configurability and internal architecture of FPGAs.

Recent research has considered several options for increasing soft processor performance. One option is to modify the amount and organization of the pipelining in existing single-issue soft processors [5], [6] which provide limited performance gains. A second option is to pursue VLIW [7] or superscalar [8] pipelines which are limited due to the few ports in FPGA block RAMs and

Manuscript received September 14, 2010; revised January 15, 2011; accepted June 01, 2011. Date of publication July 25, 2011; date of current version June 14, 2012. This work was supported by NSERC and Altera Corporation.

The authors are with the Edward S. Sr. Rogers Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: yiannac@eecg.utoronto.ca; steffan@eecg.utoronto.ca; jayar@eecg.utoronto.ca).

Digital Object Identifier 10.1109/TVLSI.2011.2160463

the available instruction-level parallelism within an application. A third option is multi-threaded pipelines [9]–[11] and multiprocessors [12], [13] which exploit thread-level parallelism but require complicated parallelization of the software. In this work we propose and explore vector extensions for soft processors which can be relatively easily programmed to allow a single vector instruction to command multiple datapaths. An FPGA designer can then independently scale the number of these datapaths in their design, referred to as *vector lanes*, to exploit the data parallelism in an application to increase performance. This scalability, coupled with architectural flexibility, and the portability of a vector architecture that can migrate between FPGA devices, can form a powerful FPGA design flow that is far simpler than hardware design while performing competitively.

Our goal is to provide this vectorized-software FPGA design flow as an additional tool for embedded system designers to more easily meet their performance and time-to-market constraints. Whether this design flow is more widely applicable to generic computing versus PCs and GPUs remains an open question that is beyond the scope of this work. Instead we explore the scalability, flexibility, and portability necessary for this design flow to be viable in an embedded FPGA context. This is investigated through the following research contributions made in this work: 1) we evaluate the design and implementation of our vector extended soft processor architecture (VESPA), a real hardware implementation of a parameterized soft vector processor; 2) we highlight its scalability on a set of embedded benchmarks derived from the industry standard EEMBC suite, illustrating its ability to convert FPGA resources into performance; 3) we provide a broad area/performance design space with fine-grain resolution allowing an FPGA designer to select a soft vector processor architecture that meets their precise application needs; and 4) we support automatic customization of VESPA through the removal of general purpose area overheads. The VESPA source code and compiler are freely distributed [14].

II. Background

A. Vector Processors

For the last half-century computer architecture research has sought to overcome the sequential execution limitations of traditional microprocessors by exploiting various forms of parallelism that exist within applications. One such form of parallelism is *data level parallelism (DLP)* where the same operation is performed over multiple data elements allowing multiple operations to be executed concurrently in parallel pipelines. This form of parallelization was used to create vector supercomputers in the 1960s [4], and is an important aspect of modern graphics processors. In addition, DLP continues to be exploited (albeit to a lesser degree) in virtually all mid to high performance microprocessors through SIMD extenstions: Intel's MMX/SSE/AVX, AMD's 3DNow!, MIPS's MDMX, IBM's Altivec, ARM's NEON, etc. While many variations in exploiting DLP exist, the vector processor approach continues to be influential in computer architecture [15].

The soft vector processor architecture described in this article was heavily influenced by previous research in vector processors, specifically by the VIRAM [16]–[20] vector processor which was used in the IRAM project [21] to explore the prospect of having memory and microprocessor together on the same chip. VIRAM was based on the T0 vector processor [22], [23] but was optimized for a high performance on-chip memory system, making it well suited for FPGA implementation where the speed degradation of the programmable substrate can be likened to a boost in memory performance. In addition VIRAM targeted the embedded application space which is also the target of current FPGA devices. The novelty in our research is in exploring these ideas in an FPGA context as an alternative to traditional HDL hardware design. Moreover this research uses a vector architecture designed from scratch for the FPGA fabric, and with built-in customization to support several vector architecture variations by simply changing its parameters.

B. Behavioral Synthesis

The goal of reducing hardware design time is one that is shared most notably by the behavioral synthesis community which has been an active research area for many decades and continues to have traction even in the FPGA context. The intractable nature of the behavioral synthesis problem including the pointer aliasing problem are often resolved by imposing restrictions to the `C` input and front-end resulting in a fragmented landscape of `C` variants [24]–[28]. A customized processor can however support the full ANSI `C` standard and offers additional advantages in modular development through libraries, intuitive single-step debugging, and a fluid design methodology by separately optimizing algorithm, compiler, machine language, and architecture. Recent research [29] even showed that soft vector processors can provide better performance scaling than a commercial behavioral synthesis tool.

C. Soft Processors

A soft processor is a processor designed for a reprogrammable fabric such as an FPGA. The two key attributes of soft processors are: 1) the ease with which they can be customized and subsequently implemented in hardware and 2) that they are designed to target the fixed resources available on a reprogrammable fabric. This differentiates soft processors from customizable processor cores such as Tensilica. The Actel Cortex-M1 [30], Altera Nios II [3], Lattice Micro32 [31], and Xilinx Microblaze [2] are widely used commerical soft processors with scalar in-order single-issue architectures that are either unpipelined or have between 3 and 5 pipeline stages. While this is sufficient for system coordination tasks, significant performance improvements are necessary for soft processors to replace the hardware designs of more performance-critical computations. Research in this direction is recent and ongoing, and summarized below.

Soft Single-Issue In-Order Pipelines—The SPREE system was developed to explore the architectural space of pipelined single-issue soft processors [5], [6], [32]. SPREE can automatically generate a Verilog hardware implementation of a processor from a high-level description of the datapath and instruction set. While this work succeeded in exploring the space and finding processor configurations superior to a mid-speed commercial

soft processor, it failed to extend the space since it did not consider higher-performance soft processor architectures. In this work we aim to significantly improve the performance of soft processors with vector extensions.

The LEON [33] is a parameterized VHDL description of a SPARC processor targetted for both FPGAs and ASICS with several customization options including cache configuration and functional unit support. LEON is a heavy-weight processor with a large area footprint and full support for exceptions, virtual memory, and multiprocessors. Performance can be scaled with multiple LEON cores but this requires parallelized code and replication of the entire processor.

Soft Multi-Issue Pipelines—Soft VLIW processors have been investigated [34], [35], [7] and shown to achieve significant performance gains up to 3-4× but scaling beyond this is significantly hindered by the limited amount of instruction parallelism in applications and the complications in achieving the register file bandwidth from the dual-ported FPGA block RAMs. The register file bandwidth issue is also a limiting factor in soft superscalar processors [8].

Soft Multi-Threaded Processors—A potentially promising method to fully utilize a processor pipline is to leverage multiple threads in soft multi-threaded processors [9], [11], [36]–[39]. Research into exploiting multiple threads in soft processors will become more fruitful as advancements in parallel programming are made in the microprocessor industry. However, to truly scale performance, multiple execution cores are required—as in multiprocessors.

Soft Multiprocessors—Scaling performance in soft multiprocessors requires extraction of the parallelism from applications. Doing this manually can be difficult and time consuming but has been used to create a soft multiprocessor version of a network processor which comes within 2.6× of the performance per area of a commercial Intel network processor [13]. Innovations for simplifying parallel programming from the computer architecture and languages community can be harnessed by soft processors. For example stream programming has been used as a front end to soft multi-processor systems [12] and shown that with 16 processors up to 5× performance can be gained. Customization of each processor core is then performed, the cores can even be transformed into custom hardware accelerators using behavioral synthesis [40].

Soft Vector Processors—Yu *et al.* [29], [41] first demonstrated the potential for vector processing as a simple-to-use and scalable accelerator for soft processors. In particular, through performance modelling the authors show that: 1) a vector processor can potentially accelerate data parallel benchmarks with performance scaling better than Altera's `C2H` behavioral synthesis tool (even after manual code restructuring to aid `C2H`) and 2) how FPGA architectural features can be exploited to provide efficient support for some vector operations. For example, the multiply-accumulate blocks internally sum multiple partial products from narrow multiplier circuits to implement wider multiplication operations. This same accumulator circuitry is used by Yu to efficiently perform vector reductions which sum all vector elements and produce a single scalar value. Also the block RAMs can be used as small lane-local memories for efficiently implementing table lookups and scatter/gather operations.

The work of Yu *et al.* was done in parallel with our own development of VESPA and its infrastructure, but it left many avenues unexplored. Its memory system consisted of only the fast on-chip block RAMs—memory systems with significant latency to off-chip DRAM were never explored. Also few customization opportunities in soft vector processors were examined beyond the number of lanes and the maximum vector length. The width of the lanes, multiplier, and memory were parameterized and were individually set for each benchmark. Finally, more sophisticated vector pipeline features such as vector chaining were never considered. Beyond the work of Yu, in this paper we offer a full and verified hardware implementation of a soft vector processor called VESPA, connected to off-chip memory, with GNU assembler vector support, and an evaluation of vectorized industry-standard benchmarks. This work more thoroughly explores the scalability, customizability, and architectural design space of soft vector processors.

III. VESPA: A Soft Vector Processor

In this section we introduce VESPA a soft vector processor architecture that is highly parameterizable, scalable and portable. While there are many forms of parallelism and various methods of exploiting them to scale the performance of soft processors, we focus on exploiting data level parallelism through vector extensions for several reasons. First, data level parallelism is shown to abundantly exist in embedded applications [16]. Second, we believe vector extensions are a good candidate for exploiting this parallelism because of their promising compiler support, simplified abstraction between software and hardware parallelism, and their suitability for FPGA implementation. A vector processor with all lanes operating in lockstep requires very little inter-lane coordination making the design scalable in hardware. Moreover, the architecture does not require any large associative lookups, many ported register files, or other structures that are inefficient to implement in FPGAs.

A. VESPA Architecture

VESPA is composed of a vector coprocessor attached to a MIPS [42] scalar soft processor generated by the SPREE system [5], [6]. A diagram including both components as well as their connection to memory is shown in Fig. 1. The figure shows the MIPS-based scalar and VIRAM-based vector coprocessor both fed by the same instruction cache. Both cores can execute out-of-order with respect to each other except for communication and memory instructions which are serialized to maintain sequential memory consistency. Vector instructions enter the vector coprocessor, are decoded into element operations which are issued onto the vector lanes and executed in lockstep. The vector coprocessor and scalar soft processor share the same data cache and its data prefetcher though the prefetching strategy can be separately configured for scalar and vector memory accesses. The sections below describe the vector instruction set,

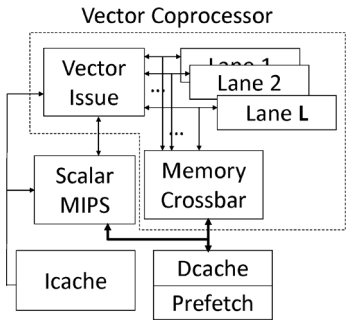

Fig. 1. VESPA processor system block diagram. The data cache is shared between the scalar processor and vector coprocessor. The memory crossbar selects between requests from the vector lanes to the byte(s) in the cache line.

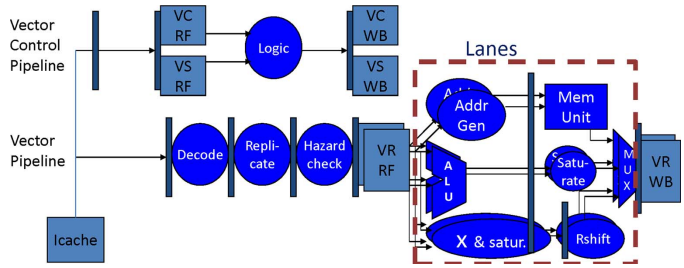

Fig. 2. VESPA architecture with two lanes. The black vertical bars indicate pipeline stages, the darker blocks indicate logic, and the light boxes indicate storage elements for the caches as well as the vector control (vc), vector scalar (vs), and vector (vr) register files.

the memory architecture, and the VESPA pipeline in more detail.

1) VIRAM-Based Vector Instruction Set: The instruction set architecture of the VESPA vector coprocessor is based on the VIRAM [20] vector instruction set and includes the integer, fixed-point, flag, and vector manipulation instructions. Three-operand instructions were excluded to avoid further complexities in the register file design. Floating point instructions were excluded since we target embedded applications which do not typically use them. The division instructions were excluded since this would require a large functional unit that is not used in our benchmark set.

2) Vector Memory Architecture: Fig. 1 also gives an overview of the VESPA memory architecture. Central to the memory architecture is the data cache shared between the vector and scalar processors. Each vector lane can request its own memory address but only one cache line can be accessed at a time. For example, lane 1 will request its address from the cache then each byte in the accessed cache line can be simultaneously routed to any lane through the *memory crossbar*. Thus, the spatial locality of lane requests is key for fast memory performance since it reduces the number of cache accesses required to satisfy all lanes. There is one such crossbar for reads and another for writes; we treat both as one and refer to the pair as the memory crossbar (with the bidirectionality assumed). This crossbar is the least scalable structure in the vector processor design and is a key factor in overall performance.

3) VESPA Pipelines: Fig. 2 shows the VESPA pipelines with each stage separated by black vertical bars. The topmost pipeline is the three-stage scalar MIPS processor discussed earlier. The middle pipeline is a simple three-stage pipeline for accessing vector control registers and communicating between the scalar processor and vector coprocessor. The actual vector instructions are executed in the longer seven-stage pipeline at the bottom of the figure. Vector instructions are first decoded and proceed to the *replicate* pipeline stage which divides the elements of work requested by the vector instruction into smaller groups that are mapped onto the available lanes; in the figure only two lanes are shown. The *hazard check* stage observes hazards for the vector and flag register files and stalls if necessary (note the flag register file and processing units are not shown in the figure). If there are two lanes, the pipeline reads out two adjacent elements for each operand, referred to as an element group, and sends them to the appropriate functional unit. Execution occurs in the subsequent stages after which results are written back to the register file.

TABLE I
CONFIGURABLE PARAMETERS FOR VESPA

	Parameter	Symbol	Value Range
Compute	Vector Lanes	L	1,2,4,8,16,...
	Memory Crossbar Lanes	M	1,2,4,8,...L
	Multiplier Lanes	X	1,2,4,8,...L
	Register File Banks	B	1,2,4,...
	ALU per Bank	APB	true/false
ISA	Maximum Vector Length	MVL	2,4,8,16,...
	Vector Lane Bit-Width	W	1,2,3,4,..., 32
	Each Vector Instruction	-	on/off
Memory	ICache Depth (KB)	ID	4,8,...
	ICache Line Size (B)	IW	16,32,64,...
	DCache Depth (KB)	DD	4,8,...
	DCache Line Size (B)	DW	16,32,64,...
	DCache Miss Prefetch	DPK	1,2,3,...
	Vector Miss Prefetch	DPV	1,2,3,...

B. VESPA Flexibility

VESPA is a highly parameterized design enabling a large design space of possible vector processor configurations. These parameters can modify the VESPA compute architecture (pipeline and functional units), instruction set architecture, and memory system. All parameters are built-in to the Verilog design so a user need only enter the parameter value and have the correct configuration synthesized with no additional source modifications. Table I lists all the configurable parameters and their acceptable value ranges. Note that many of the integer parameters are limited to powers of two to reduce hardware complexity. Further details about VESPA can be found in the thesis [43] or in its freely distributed source code [14].

C. VESPA Portability

The portability of soft vector processors is a major factor in whether FPGA vendors will adopt them in the future. Since FPGA vendors have many different FPGA devices and families, a non-portable hardware IP core would require more design effort to support across all of these devices. For this reason VESPA is fully implemented in synthesizable Verilog but was purposefully designed to have no dependencies to a particular FPGA device or family. In fact we ported VESPA from the Stratix 1S80 on the TM4 [44] to the Stratix III 3S340 on the DE3 and required zero source modifications. We do not port

TABLE II
VECTORIZED BENCHMARK APPLICATIONS

Benchmark	Description	Source	EEMBC Suite (Dataset)	Input size (B)	Output size (B)	Num Loops
AUTCOR	auto correlation	EEMBC/VIRAM	Telecom (2)	1024	64	1
CONVEN	convolution encoder	EEMBC/VIRAM	Telecom (1)	522	1024	1
RGBCMYK	rgb filter	EEMBC/VIRAM	Digital Ent. (5)	1628973	2171964	1
RGBYIQ	rgb filter	EEMBC/VIRAM	Digital Ent. (6)	1156800	1156800	1
FBITAL	bit allocation	EEMBC/VIRAM	Telecom (2)	1536	512	2
VITERB	viterbi encoder	EEMBC/VIRAM	Telecom (2)	688	44	5
IP_CHECKSUM	checksum	EEMBC (kernel)	Net (handmade)	40960	40	1
IMGBLEND	combine two images	VIRAM	(handmade)	153600	76800	1
FILT3X3	image filter	VIRAM	(handmade)	76800	76800	1

VESPA across different vendors or families, but there is nothing in VESPA's architecture preventing this.

D. FPGA Influences on VESPA Architecture

The VESPA architecture was influenced in a number of ways by the FPGA substrate. In this section we collect the key points and summarize them. First, the multiply-accumulate blocks are obvious choices for efficiently implementing processor multipliers. Their performance is still significantly less than an FPGA adder circuit leading to accommodations in the pipeline similar to hard microprocessors. However, the multipliers are also efficient [45] for implementing shifters since multiplexers are relatively expensive to implement in logic-elements on FPGAs. This shared multiplier/shifter functional unit means vector chaining on soft vector processors exhibits different behavior than traditional vector processors since vector multiplies and vector shifts cannot be executed simultaneously. Second, the block RAMs provide relatively inexpensive storage helping to justify the existence of caches even when they are not strongly motivated in our streaming applications. The low area cost of storage also helps motivate vector processors since the large vector register files required can be efficiently implemented if few ports are required. Finally, the fact that FPGA block RAMs have only two ports imposes architectural differences from traditional processors. For three-operand instruction sets such as MIPS, the register file must sustain two reads and one write per cycle. Since FPGA block RAMs have only two ports, a common solution is to leverage the low area cost of block RAMs to duplicate them. Additional ports are required to support multiple vector instruction execution. Section VI-D describes how VESPA employs banking to overcome the port limitations. This approach is reminiscent of vector processors before single-chip VLSI advancements, and marks a key architectural difference between VESPA and application-specific integrated circuit (ASIC) vector processors such as the T0 [23] and VIRAM [19]. In general, the lack of block RAM ports and expensive multiplexing logic make FPGAs less amenable to any architecture with multiple instructions in flight such as traditional superscalar out-of-order architectures.

IV. EXPERIMENTAL FRAMEWORK

We evaluate our proposed soft processor architecture enhancements with full hardware implementation of each VESPA variant on a real FPGA device connected to off-chip DDR2 RAM and executing EEMBC benchmarks. In this section we describe the individual components of this infrastructure.

Benchmarks—The benchmarks used in this study are predominantly from the industry-standard embedded microprocessor benchmark consortium (EEMBC) benchmark collection [46]. These benchmarks are widely used in the embedded systems domain, many of them were used in the VIRAM project which targeted the embedded domain. Altera's C2H synthesis tool and Nios II soft processor were also benchmarked with many of these same applications [47], suggesting their appropriateness for embedded FPGA workloads. Our infrastructure is capable of compiling and executing all EEMBC benchmarks uncompromised and with the complete test harness allowing us to report official EEMBC scores.

Our work on soft vector processors targets applications with data parallelism, so we assembled a corresponding subset of benchmarks in Table II. The top six are uncompromised EEMBC benchmarks vectorized in assembly and provided to us by Kozyrakis who used them during his work on the VIRAM processor [16]. The fifth benchmark is a kernel we extracted and hand-vectorized from the EEMBC IP_PKTCHECK benchmark. Since execution of this benchmark is independent of the data set values, we provide a hand-made data set of 10 arbitrarily filled 4 KB packets. Similarly FILT3X3 also uses two arbitrarily filled 320×240 images (one byte per pixel), while IMGBLEND uses one. These latter two benchmarks were provided to us from the VIRAM group as well. The last three columns of Table II show the input data size, output data size, and total number of loops not including nested loops. All loops were vectorized except for one loop in each of the FBITAL and VITERB benchmarks. Finally, note that all benchmarks were written to be independent of the maximum vector length supported in the vector processor. The instruction mix, vector length distribution, and percentage of vectorized code in each benchmark are summarized by Kozyrakis [19].

Software Compilation Framework—The benchmarks were compiled using GNU GCC 4.2.0 ported to MIPS to match the MIPS-based SPREE [48] scalar processors used throughout this work. Benchmarks are compiled with `-O3` optimization level. Auto-vectorization in GCC failed to vectorize key loops in our benchmarks, while commercial compilers with better auto-vectorization capabilities can not be ported to our instruction set. Instead of relying on auto-vectorization, we ported the GNU assembler found in `binutils` version 2.1.6

to support our VIRAM vector instruction set, allowing us to hand-vectorize loops in assembly.

FPGA CAD Software—A key value of performing FPGA-based processor research directly on an FPGA is the ability to attain high quality measurements of the area consumed and the clock frequency achieved—these are provided by the FPGA CAD software. In this research we use Altera Quartus II version 8.1 for FPGA synthesis, place, and route. There are many settings and optimizations that one can enable within the software, creating a wide range of synthesis results. In our work we use an over-constrained clock rate of 200 MHz and ensure that register retiming and duplication are enabled. These settings are similar to those suggested in previous research [32].

Measuring Area—Area is comprised mostly of the FPGA programmable logic blocks. Throughout this work we use the Altera Stratix III FPGA—its programmable logic blocks are referred to as *Adaptive Logic Modules (ALMs)*. In addition to ALMs, soft processors also make use of memory blocks and multiply-accumulate blocks. We measure the total silicon area consumed by the design and report it in units of *equivalent ALMs*. The silicon areas of each FPGA resource relative to an ALM (including its routing) was provided to us by Altera for the Stratix II: M512 10.6, M4K 22.6, MRAM 708.5, and DSP (9-bit) 7 equivalent ALMs. We extrapolated these areas for the Stratix III architecture: M9K 45.2, MRAM 354.25, and DSP (18-bit) 14 equivalent ALMs.

Measuring Clock Frequency—The clock frequency of a synthesized design is reported by the timing analysis tool in the FPGA CAD software. In addition to the CAD settings described above, the actual device targeted can affect these results due to different logic/circuit architectre, process technology, and speed bin. We report clock frequency from a Stratix III C2 (fastest speed grade) device, although we managed only to procure a slower C3 device.

DE3 Hardware Platform—All soft processors explored in this work are fully synthesized using the CAD flow described above and implemented in hardware on an Altera/Terasic DE3 board. Benchmarks are executed in hardware and report the precise number of clock cycles required to complete execution. We use the Terasic DE3-340 board equipped with a single Stratix III EP3SL340H1152C3 which is one of the largest state-of-the-art FPGAs available at the time this work was performed. The Stratix III is fabricated in a 65-nm CMOS technology and is connected to a 1 GB DDR2-533 MHz memory device which we use for the storage of program instructions and data. The Altera DDR2 memory controller connects the soft processor to the DDR2 DIMM and is clocked at the full-rate of 266 MHz.

Measuring Wall Clock Time—Implementation onto a real FPGA hardware platform enables accurate measurement of not just the execution cycles but also the wall clock time for executing a benchmark. Wall clock time considers both the cycle performance and clock frequency of a processor. For simplicity, we clock all designs at 100 MHz and measure wall clock time by multiplying the observed number of cycles with the highest clock rate achievable by a soft processor instance. The effect of this time dilation is negligible and quantified via simulation to reach 0.6% error for a 150 MHz design while VESPA's fastest variants are only 135 MHz.

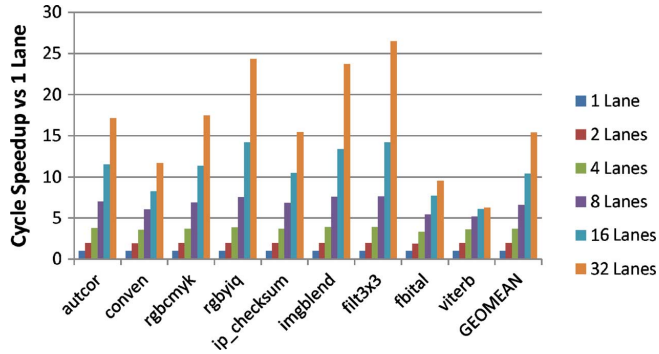

Fig. 3. Performance scalability as the number of lanes are increased from 1 to 32 for a fixed VESPA architecture with *full memory support* (full memory crossbar, 16 kB data cache, 64 B cache line, and prefetching enabled).

Testing—All soft processors were fully tested in hardware using the hard-coded checksum in each EEMBC benchmark. Debugging is performed using Modelsim and is guided by comparing traces of all writes to the register files to one obtained from instruction-set simulation using the MINT [49] MIPS simulator augmented with the VIRAM extensions.

V. Scalability of the VESPA Soft Vector Processor

With scalable performance, a soft vector processor can be used to leverage the computational power of FPGAs without complicated hardware design. VESPA's collection of architectural parameters can be used to achieve significant performance scaling, but a vector architecture's scalability is often defined by how well performance scales when only the number of lanes, `L`, is varied. The `MVL` (Maximum Vector Length) parameter determines the size of the longest possible vector operation and hence limits `L`. While `MVL` can exhibit some interesting application-dependent behavior [5], we fix `MVL` to be 128 for this section. In addition, we fix all other parameters and vary only `L` to measure VESPA's scalability.

A. Cycle Performance

Fig. 3 shows the cycle performance improvement for each of our benchmarks as we increase the number of lanes on an otherwise aggressive VESPA architecture with *full memory support*—i.e., a full memory crossbar, 16 kB data cache with 64-byte cache lines and hardware prefetching (while a variety of prefetching schemes are possible we prefetch the next $8 \times [\text{current vector length}]$ elements since this was shown reliable across our benchmarks as seen in our previous work [50], [51]). The figure illustrates that impressive scaling is possible as seen in the 27× speedup for FILT3X3 executed on 32 lanes. The potential for scaling soft processor performance for these benchmarks is also exemplified in the near 100% conversion of lanes to performance seen in the 2-lane configuration which performs 1.95× faster than 1-lane on average. Overall we see that indeed a soft vector processor can scale cycle performance on average from 1.95× for 2 lanes to 15× for 32 lanes.

Ideally, the speedup would increase linearly with the number of vector lanes, but this is prevented by a number of factors: 1) only the vectorizable portion of the code can benefit from extra lanes, hence benchmarks such as CONVEN that have a blend of

TABLE III
AVERAGE PERFORMANCE OF VESPA

Num Lanes	Clock (MHz)	Cycle Speedup	WCT Speedup
1	131	1.00	1.00
2	129	1.95	1.95
4	128	3.70	3.60
8	123	6.50	6.30
16	117	10.40	9.30
32	96	15.40	11.30

scalar and vector instructions are limited by the fraction of actual vector instructions in the instruction stream; 2) some applications do not contain the long vectors necessary to scale performance, for example VITERB executes predominantly with a vector length of only 16; 3) the movement of data becomes a limiting factor specifically for RGBCMYK, and RGBYIQ which access data streams in a strided fashion which requires non-ideal cache accesses, and FBITAL which uses an indexed load to access an arbitrary memory location from each lane. Indexed vector memory operations are executed serially in VESPA which severely limits performance when used often.

B. Clock Frequency

Table III shows the clock frequency for each configuration produced by the FPGA CAD tools as described in Section IV. As expected, the clock frequency decreases as lanes are added. The effect on wall clock time (WCT) is moderate at 16 lanes reducing the 10.4× cycle speedup to 9.3× in actual wall clock time speedup as seen in the right-most two columns of the table. At 32 lanes the clock frequency drops significantly reducing the 15× cycle speedup to 11× in wall clock time. The last column shows that despite the clock frequency reductions, the average wall clock time increases with more lanes up to 32. At 64 lanes the clock frequency is reduced to 80 MHz and the performance is worse than the 32-lane configuration. Because of these timing problems, the 64-lane configuration is not used in our study.

In both the 16 and 32 lane configurations, the critical path is in the memory crossbar which routes all 64 bytes in a cache line to each of the lanes. The `M` parameter can be used to reduce the size of the crossbar and raise the clock frequency, but the resultant loss in average cycle performance often overwhelms this gain and produces a slower overall processor as will be shown in Section VI-C3. The clock frequency reduction can instead be addressed by further pipelining the memory crossbar as well as additional engineering effort in retiming these large designs. Ultimately scaling to larger lanes requires careful design of a high-performing memory system, and may require a hierarchy of memory storage units as seen in graphics processors.

C. Area

Fig. 4 shows the area of each VESPA vector coprocessor (excluding the memory controller, system bus, caches, and scalar processor) on the x-axis and the normalized wall clock time execution plotted on the y-axis. The initial area cost of a vector coprocessor is considerable costing 2900 ALMs of silicon area due to the decode/issue logic, 1 vector lane, and the vector state with 64 elements in each vector (`MVL` = 64). As the vector lanes are increased a linear growth in area is eventually observed as

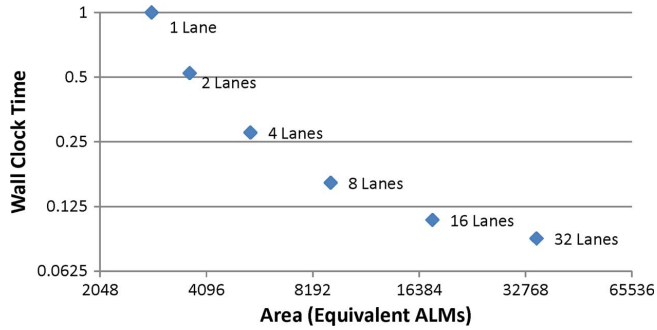

Fig. 4. Performance/area design space of 1-32 lane VESPA cores with full memory support. Reported area accounts only for the area of the vector coprocessor.

the constant cost of the state and decode/issue logic are dominated. This incremental area cost becomes quite substantial, for example growing from 8 to 16 lanes requires about 9000 ALMs. At 32 lanes the area reaches one third of the silicon on the Stratix III-340. Nonetheless performance scaling is still possible with 32-lanes and likely beyond with additional processor design effort.

VI. EXPANDING THE VESPA DESIGN SPACE

One of the most compelling features of soft processors is their inherent reconfigurability. An FPGA designer can ideally choose the exact soft processor architecture for their application rather than be limited to a few off-the-shelf variants. If the application changes, a designer can easily reconfigure a new soft processor onto the FPGA device. Thus, a highly parameterized processor core such as VESPA can be used to span a large design space while providing fine-grain architectural tuning. Ideally intelligent software can analyze code execution and automatically prescribe the most appropriate architecture.

This section explores VESPA's remaining architectural parameters individually. The previous section explored the number of lanes which can be tuned to match the amount of data parallelism in an application. VESPA's other architectural parameters can be used to exploit other program characteristics and similarly trade performance and area. These parameters and their resulting tradeoffs are discussed below.

A. Data Cache (`DD` and `DW`)

VESPA's data cache is direct-mapped with configurable capacity and cache line size [43], [50]. Increasing the capacity reduces the number of conflict misses, however due to the streaming nature of our benchmarks, this was not a significant factor in their performance. Instead, the cache line size was a key architectural option since with larger cache lines more lanes can retrieve their desired data in a single cache access. We explore the effect of cache depth and line size below.

Fig. 5 shows the average wall clock speedup across all of our benchmarks except VITERB[1] for each data cache configuration normalized against the 4 kB cache with a 16 B cache line size. We first note that the streaming nature of these benchmarks makes cache depth affect performance only slightly. For

[1]In VITERB the affect of cache lines are similar, but cache depth increases performance slightly but plateaus after 16 kB.

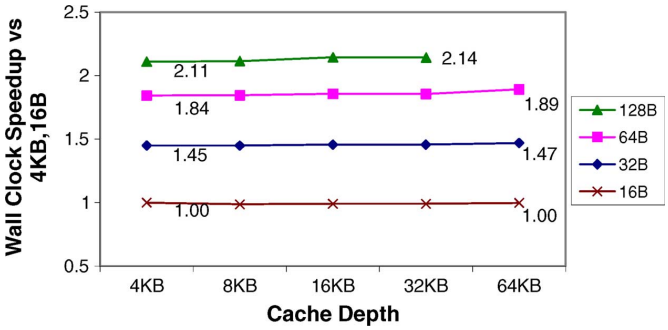

Fig. 5. Average wall clock speedup (excluding VITERB benchmark) attained for a 16-lane VESPA with different cache depths and cache line sizes, relative to the 4 kB cache with 16 B line size. Each line in the graph depicts a different cache line size.

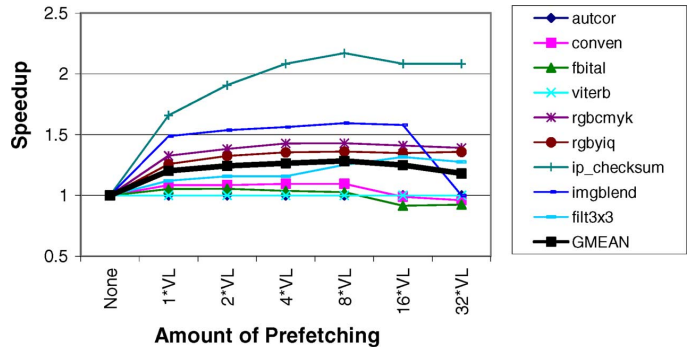

Fig. 6. Wall clock speedup achieved for different configurations of the vector length prefetcher.

the 16 B cache line configuration the performance is flat across a 16-fold increase in cache depth, while for 128 B this 16-fold growth increases performance from 2.11× to 2.14×. The cache line size plays a far more influential role on performance since it not only satisfies more per-lane requests in a single access, but also inherently prefetches neighboring data. Each doubling of line size hence provides a significant leap in performance spanning to more than double the performance of the 16 B line size with 128 B.

In terms of area, the results are similar: cache depth has a modest impact while cache line size has a dramatic impact. Increasing the cache depth increases only the FPGA block RAMs which do not significantly contribute to VESPA's area. However, increasing the cache line size results in a proportional increase in the memory crossbar size. Hence the 2× performance gained with 128 B cache lines is accompanied with a near 2× increase in area. Further details are available in Yiannacouras' dissertation [43].

B. Data Prefetching (DPV)

A hardware data prefetcher [52] can capture part of the benefit of larger cache lines without the excessive area costs associated with growing the memory crossbar. In VESPA we support sequential prefetching in hardware similar to conventional vector processor research [53] but in a reconfigurable context where the prefetching scheme can vary based on the application. A sequential prefetcher fetches the missed cache line plus the N neighboring cache lines. Determining a good value for N is application dependent, but in VESPA we also consider the vector length of the memory operation. That is, rather than prefetching a fixed number of cache lines, we prefetch a data set that assumes DPV executions of the same memory operation. For example, the $8 * VL$ configuration will multiply the vector length (a runtime parameter) by the access size (byte/half/word) and then by 8. A more complete description is available in a prior publication [50] which also shows the area cost to be less than 2%.

Fig. 6 shows the variation in performance of a range of vector length prefetchers. Prefetching 8 times the vector length (8VL) number of cache lines performs best, achieving a maximum speedup of 2.2× for IP_CHECKSUM and 28% on average. Of specific interest is the $1VL$ configuration which prefetches the remaining elements in a vector miss and hence has zero cache pollution. This configuration has no speculation, it guarantees no more than one miss per vector memory instruction, and is ideal for heavily mixed scalar/vector applications and achieves 20% speedup on average across our benchmarks. Greater performance can be achieved by incorporating the right amount of speculation in the prefetching. The figure shows that adding speculation gains performance, but too much can undo the performance gains as seen in IMGBLEND where large prefetches of the input stream can pollute the cache and increase the miss rate.

C. Heterogeneous Lanes

In this section, we examine the option of reducing the number of copies of a given functional unit which is in low demand [54]. For example, a benchmark with vector multiplication operations will require the multiplier functional unit present, but if the multiplies are infrequent the application does not necessarily require a multiplier in every vector lane. In the extreme case a 32-lane vector coprocessor can have just one lane with a multiplier and have vector multiplication operations stall as the vector multiply is performed at a rate of one operation per cycle. We use this idea to parameterize the hardware support for vectorized multiplication and memory instructions as described below.

1) Supporting Heterogeneous Lanes: The VESPA vector datapath contains three functional units: 1) the arithmetic logic unit (ALU) for addition, subtraction, and logic operations; 2) the multiplier for multiplication and shifting operations; and 3) the memory unit for load and store operations. Increasing the vector lanes with the `L` parameter replicates all of these functional units, so all vector lanes are homogeneous. We provide greater flexibility by allowing the multiplier units to appear in only some of the lanes specified with the parameter `X`. Similarly the number of lanes attached to the memory crossbar can be selected using `M`. This allows for a heterogeneous mix of lanes where not all lanes will have each of the three functional unit types. A user can specify the number of lanes with ALUs using `L`, the number of lanes with multipliers with `X`, and the number of lanes with access to the cache with `M`.

2) Impact of Multiplier Lanes (X): The `X` parameter determines the number of lanes with multiplier units. The effect of varying X is evaluated on a 32-lane VESPA processor with 16 memory crossbar lanes (halved to reduce its area dominance) and a prefetching 16 KB data cache with 64 B line size. Each

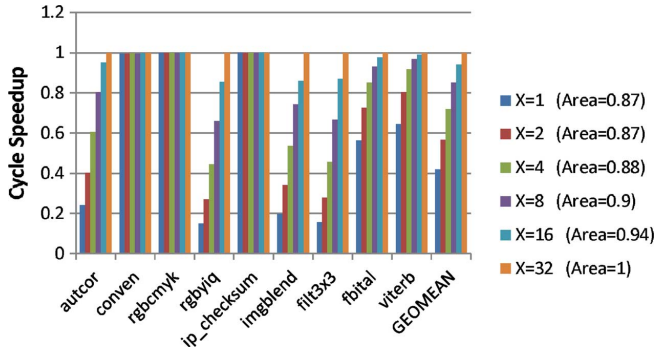

Fig. 7. Performance impact of varying X for a VESPA with $L = 32, M = 16, DW = 64, DD = 16$ K, and $DPV = 8 * VL$; area and performance are normalized to the $X = 32$ configuration.

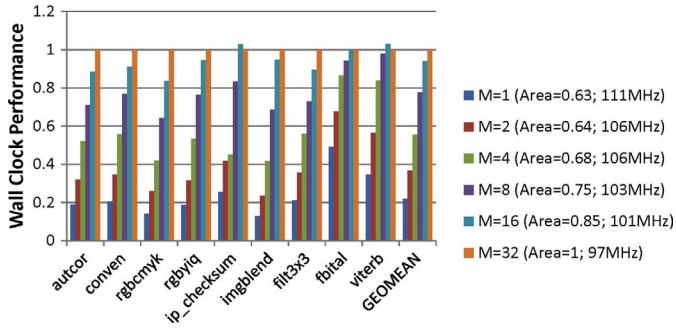

Fig. 8. Wall clock performance of various memory crossbar (M) values on a 32-lane vector processor with 16 KB data cache, 64B line size, and 8*VL prefetching. Performance is normalized against the full $M = 32$ configuration, as is area shown in parentheses adjacent to each processor's clock frequency.

halving of X doubles the number of cycles needed to complete a vector multiply with more than X elements. We measure the overall cycle performance and area and normalize it to the full $X = 32$ configuration. Note that clock frequency was unaffected in these designs.

Fig. 7 shows that in some benchmarks such as FILT3X3 the performance degradation is dramatic, while in other benchmarks such as CONVEN there is no impact at all. Programs with no vector multiplies can have multipliers removed completely with the instruction-set subsetting technique explored later in Section VIII-B, but programs with just few multiplies such as VITERB can have the number of multipliers reduced saving 10% area at the expense of 3.1% performance.

The area savings from reducing the multipliers is moderate starting at 6% for halving the number of multipliers to 16, and the savings asymptotically grow and saturate at 13%. Since the multipliers are efficiently implemented in dedicated FPGA multiplier blocks, their contribution to the overall silicon area is small, and the additional overhead for multiplexing operations into the few lanes with multipliers ultimately nullify the area savings. However, multipliers are often found in short supply, so a designer might be willing to accept the performance penalty if another more critical computation could benefit from using the limited FPGA multipliers.

3) Impact of Memory Crossbar (M)*:* A vector load/store instruction can perform in parallel as many memory requests as there are vector lanes, however the data cache can service only one cache line access per clock cycle. Extracting the data in a cache line to/from each vector lane requires a full and bidirectional crossbar between every byte in a cache line and every vector lane. Such a circuit structure imposes heavy limitations on the scalability of the design, especially within FPGAs where multiplexing logic is comparatively more expensive than in traditional IC design flows. Because of this, the idea of using heterogeneous lanes to limit the number of lanes connected to the crossbar can be extremely powerful.

The parameter, M, controls the number of lanes the memory crossbar connects to and hence directly controls the crossbar size and the amount of parallelism for memory operations. For example, a 16-lane vector processor with M equal to 4 can complete 16 add operations in parallel, but can only satisfy up to 4 vector loads/store operations in a cycle. The 16 memory operations are serialized through shift registers in groups of 4. Decreasing M hence reduces area and decreases cycle performance of vector memory instructions. Also, clock frequency can be increased by reducing M when the memory crossbar is the critical path in the design.

Fig. 8 shows the effect on wall clock performance and as the memory crossbar size is varied on a single 32-lane vector processor with 16 kB data cache, 64B line size, and 8*VL prefetching. Wall clock performance is normalized to the full memory crossbar $(M = 32)$ configuration as is the area shown in parentheses beside the clock frequency of each configuration. All configurations with $M <= 4$ are dominated by the $M = 4$ configuration since they have very similar areas. For $M = 8$ the area is reduced by 25% and performance by an average of 26%. A half-size memory crossbar with $M = 16$ provides an area savings of 15% with an average performance degradation of 6%. For FBITAL there is no performance degradation, while surprisingly for VITERB and IP_CHECKSUM there is a performance increase because of the clock frequency increase discussed below.

The memory crossbar can often be the critical path in determining the maximum clock speed of the processor. In the 32-lane base design of Fig. 8, the crossbar size limits the clock frequency to 98 MHz compared to the 131 MHz achievable on a 1-lane VESPA. Reducing the memory crossbar to $M = 1$ raises the clock frequency to 110 MHz but results in many additional cycles yielding a wall clock performance of one-fifth of the full memory crossbar. In the cases where there is little cycle degradation such as $M = 16$ for VITERB, IP_CHECKSUM, and FBITAL, the wall clock performance is the same or better than the full memory crossbar because of the gain in clock frequency. Thus the area savings from reducing M must be weighed against the impact on performance which can either degrade or improve.

*D. Vector Chaining in VESPA (*B *and* APB*)*

Our goal of scaling soft processor performance is largely met by instantiating multiple vector lanes using a soft vector processor. However, additional performance can be gained by leveraging a key feature of traditional vector processors: the ability to concurrently execute multiple vector instructions via vector chaining [55]. By simultaneously utilizing multiple functional units, VESPA can more closely approach the efficiency of a custom hardware design. In this section, VESPA is

augmented with parameterized support for chaining designed to be amenable to FPGA architectures.

1) Supporting Vector Chaining: VESPA has three functional unit types: an ALU, a multiplier/shifter, and a memory unit, but previously, only one functional unit type could be active in a given clock cycle. Additional parallelism can be exploited by noting that vector instructions operating on different elements can be simultaneously dispatched onto the different functional unit types, hence permitting more than one to be active at one time. Modern vector processors exploit this using a large many-ported register file to feed operands to all functional units keeping many of them busy simultaneously. This multi-port approach was shown to be more area-efficient than using many banks and few ports in historical vector supercomputers [23]. But since FPGAs cannot efficiently implement a large many-ported register file, our solution is to return to this historical approach and use multiple banks each with two read ports and one write port.

The number of register banks, `B`, needed to support vector chaining is parameterized and must be a power of two. A value of 1 reduces VESPA to a single-issue vector processor without vector chaining support eliminating the hardware associated with supporting vector chaining. VESPA can potentially issue as many as `B` instructions at one time, provided there is no bank conflict and the corresponding functional units are available. To increase the likelihood of this, VESPA allows replication of the ALU for each bank as enabled by the `APB` parameter. For example, with two banks and `APB` enabled, each vector lane will have one ALU for each bank and in total two ALUs. Since multipliers are generally scarce we do not support duplication for the multiply/shift unit, and we also do not support multiple memory requests in-flight because of the system's blocking cache.

2) Impact of Vector Chaining: We measured the effect of the vector chaining implementation described above across our benchmarks using an 8-lane vector processor with full memory support (16 kB data cache, 64 B cache line, and prefetching of 8*VL) implemented on the DE3 platform. We vary the number of banks from 2 to 4 and for each toggle the ALU per bank `APB` parameter and compare the resultant four designs to an analogous VESPA configuration without vector chaining.

Fig. 9 shows the cycle speedup of chaining across our benchmarks as well as the area normalized to the 1 bank configuration. The area cost of banking is considerable, starting at 27% for the second register bank and the expensive multiplexing logic needed between the two banks. This 27% area investment provides an average performance improvement of approximately 26%, and in the best case is 54%. Note that if instead of adding a second bank, the designer opted to double the number of lanes to 16, the average performance gain would be 49% for an area cost of 77%. Two banks provide half that performance improvement at one third the area, and is hence a more fine-grain tradeoff than increasing lanes.

Replicating the ALUs for each of the two banks (two banks, two ALUs) provides some minor additional performance, except for FBITAL where the performance improvement is significant. FBITAL executes many arithmetic operations per datum making demand for the ALU high and hence benefiting significantly from increased ALUs and justifying the additional 7% area. Similarly the four bank configurations benefit only few benchmarks, while the area cost is significant at 59% with no replication and near 100% with replication. Since this configuration underperforms against a VESPA configuration with double the lanes and no chaining, we do not further study configurations with four or more banks. Note that instruction scheduling in software could better utilize the banks, but in many of our benchmarks only very little rescheduling was either necessary or possible, so we did not manually schedule instructions to exploit chaining.

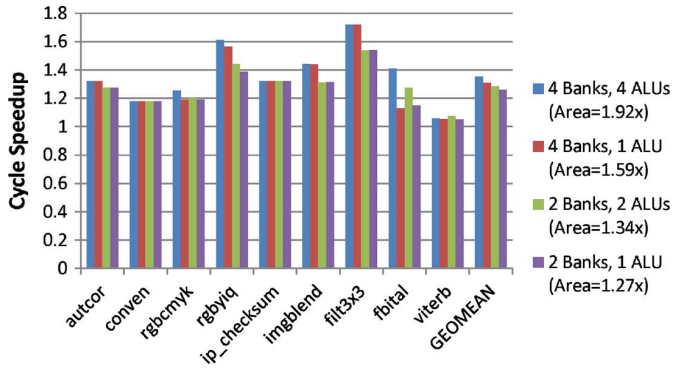

Fig. 9. Cycle performance of different banking configurations across our benchmarks on an 8-lane VESPA with full memory support. Area is shown in parentheses for each configuration. Both cycle speedup and area are normalized to the same VESPA without vector chaining.

VII. Exploring the VESPA Design Space

Using VESPA we have shown soft vector processors can scale performance while providing several architectural parameters for fine-tuning and customization. This customization is especially compelling in an FPGA context where designers can easily implement an application-specific configuration. In this section we explore this design space more fully by measuring the area and performance of hundreds of VESPA processors generated by varying all VESPA parameters shown in Table IV(a) and implementing each configuration on the DE3 platform. The configurations chosen were intelligently pruned to limit the number of inferior configurations, resulting in a 768-point design space. For example a configuration with 1 lane and a very wide cache performs insignificantly better than one with a smaller cache. Details about the derivation of our intelligently pruned design space can be found in [43].

Fig. 10 shows the area and wall clock time space of the 768 VESPA design points. This data includes the effect of clock frequency which decays only slightly throughout the designs up to eight lanes but is eventually reduced by up to 25% in our largest designs (the points in the bottom right of the figure). Since these largest designs are also the fastest, the maximum speed achieved is reduced considerably by the clock frequency reduction. The design space spans 18× in wall clock time instead of the 24× spanned in cycle count. This is in line with the 25% performance reduction expected because of the clock frequency decay. Additional retiming or pipelining can mitigate this decay, motivating a re-architected VESPA pipeline for many lanes or even a pipeline generator such as SPREE [32]. Nonetheless the de-

TABLE IV
VESPA DESIGN SPACE EXPLORATION. (A) EXPLORED SPACE OF VESPA PARAMETERS. (B) CONFIGURATIONS WITH BEST PERFORMANCE-PER-AREA

(a)

	Parameter	Symbol	Explored
Compute	Vector Lanes	L	1,2,4,8,16,32
	Memory Crossbar Lanes	M	L, L/2
	Multiplier Lanes	X	L, L/2
	Register File Banks	B	1,2,4
	ALU per Bank	APB	true/false
ISA	Maximum Vector Length	MVL	L*4,128,512
Mem	DCache Depth (KB)	DD	8, 32
	DCache Line Size (B)	DW	16, 64
	Vector Miss Prefetch	DPV	off, 7, 8*VL

(b)

Application	VESPA Configuration									Perf. per Area
	DD KB	DW B	DPV	APB	B	MVL	L	M	X	
Gen. Purpose	32	64	7	0	2	128	8	8	8	1
autcor	8	16	0	0	2	128	8	8	8	1.068
conven	32	64	8VL	0	2	128	8	8	4	1.054
rgbcmyk	32	64	8VL	0	1	64	16	16	8	1.071
rgbyiq	32	64	7	0	2	128	16	16	16	1.058
ip_checksum	32	64	7	0	2	32	8	8	4	1.087
imgblend	32	64	7	0	2	128	16	16	16	1.025
filt3x3	32	64	8VL	0	4	512	16	16	16	1.070
fbital	8	16	0	0	1	32	8	4	8	1.246
viterb	8	16	0	1	2	4	1	1	1	1.356
									GEOMEAN	1.110

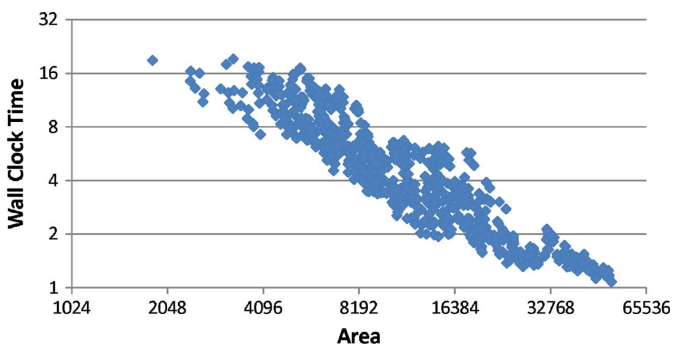

Fig. 10. Average wall clock time and vector coprocessor area of 768 soft vector processor variations across the pruned design space. Area is measured in equivalent ALMs, while the wall clock time of each benchmark is normalized against the fastest achieved time across all configurations and then averaged using geometric mean.

sign space is still very large and even the designs with reduced clock frequencies in the lower right corner of the figure provide useful pareto optimal design points. Overall the space spans a vast 28× in area and 18× in wall clock time, and does so with fine granularity allowing for precise area/speed tradeoffs.

Fig. 10 confirms that VESPA has the performance scalability and flexibility to meet a wide variety of area/performance constraints. This shows that soft vector processors can indeed be a powerful FPGA design tool which can complement traditional HDL hardware design and software soft processor design. In fact, our previous work [51] directly compares VESPA to these alternative design methods and shows that while a scalar soft processor is 432× slower than HDL hardware design, VESPA can scale to only 17× slower.

A. Per-Application Analysis

A key motivation for FPGA-based soft processors is their ability to be customized to a given application. This application-specific customization can aid FPGA designers in meeting their system design constraints. Since these constraints vary from system to system, in this section we examine the effect of optimizing the VESPA parameters for performance-per-area, a commonly used metric for measuring efficiency. To calculate performance-per-area we take the inverse of the product between area and wall clock time, both measured as described in Section IV. The measured area includes the complete processor system excluding the memory controller and host communication hardware; the wall clock time is measured on the DE3 platform.

Table IV(b) shows the VESPA configuration which achieves the best performance-per-area out of the 768 explored for each benchmark. The first row shows the general purpose configuration that achieves the best performance-per-area averaged arithmetically across all benchmarks. The per-application configurations can achieve up to 35.6% and an average of 11% better performance-per-area over the general purpose configuration, highlighting the benefits of tuning for a specific application. The selected configurations for each benchmark vary significantly from the general purpose eight-lane configuration with MVL = 128, 2 banks, 32 kB data cache, 64 B line size, and 7 cache line prefetching. Three of the benchmarks work best for the 8*VL prefetching strategy, while another three select the smaller cache which does not support prefetching. The number of lanes selected is typically 8 or 16 since 32-lane configurations require significant area and also suffer from a significant clock frequency degradation.

The VITERB benchmark is one of the least data-parallel applications and benefits most from exploiting a high degree of chaining on a one-lane soft vector processor. This configuration is certainly the most interesting as it differs the most from the general purpose. The best VITERB configuration has two banks and is the only one that benefits from enabling the APB parameter. The selected architecture is similar to a scalar processor except the vector instructions are issued up to two per cycle across the two ALUs, one multiplier, and the memory unit. Surprisingly, for this benchmark this is more efficient in terms of performance-per-area than any two-lane configuration.

The FBITAL benchmark achieves 24.6% better performance-per-area than the general purpose configuration. This is gained largely by the half-sized crossbar and the reduced vector state from the decreased MVL. Further area savings is gained by disabling chaining in this configuration. The FILT3X3 benchmark selects the largest configuration with 4 banks, MVL = 512 and 16 homogeneous lanes achieving 7% improved performance-per-area over the general purpose. The benchmarks with little or no multiply operations are seen to employ heterogeneous lanes reducing the number of lanes with multipliers. This is seen in CONVEN, RGBCMYK, and IP_CHECKSUM. Overall, these improvements highlight the value in matching the soft vector processor architecture to the application.

TABLE V
AREA ELIMINATION OPPORTUNITY AND RESULTS

Benchmark	Width Reduction		ISA Subset	
	Max Data Type Size	Norm. Area	%ISA used	Norm. Area
AUTCOR	32 bits	1.00	9.6%	0.55
CONVEN	1 bit	0.27	5.9%	0.70
FBITAL	16 bits	0.76	14.1%	0.85
VITERB	16 bits	0.76	13.3%	0.87
RGBCMYK	8 bits	0.67	5.9%	0.71
RGBYIQ	16 bits	0.76	8.1%	0.82
IP_CHECKSUM	32 bits	1.00	8.1%	0.43
IMGBLEND	8 bits	0.67	7.4%	0.81
FILT3X3	8 bits	0.67	7.4%	0.81
GMEAN		0.69		0.72

VIII. ELIMINATING FUNCTIONALITY

All architectural parameters explored so far traded area and performance while preserving functionality. But if a soft processor need only execute one application, or the FPGA can be reconfigured between runs of different applications, one can create a highly-customized soft processor with only the functionality needed by the current application. In this section we target and automatically eliminate two key general purpose overheads without affecting cycle performance: 1) the datapath width, since many applications do not require full 32-bit processing (we refer to this customization as *width reduction*); and 2) the hardware support for instructions which do not exist in the application (we refer to this as *subsetting* or *instruction set subsetting*).

Table V shows that there exists ample opportunity for such customization. The middle column shows that all benchmarks except AUTCOR and IP_CHECKSUM use less than 32-bit data types with CONVEN needing only 1-bit. The right-most column shows that all benchmarks use less than 15% of the vector instruction set. We can thus expect significant area savings from applying width reduction and instruction-set subsetting.

A. Impact of Vector Datapath Width Reduction (W)

Table V shows the effect of modifying the width in the third column using the `W` parameter on a 16-lane VESPA with full memory crossbar and 16 kB data cache with 64 B cache line size. Starting from a 1-bit vector lane width we double the width until reaching the full 32-bit configuration. The area is reduced to almost one-quarter in the 1-bit configuration with further reduction limited due to the control logic, vector state, and address generation which are unaffected by width reductions. A 2-bit width saves 68% area and is only slightly larger than the 1-bit configuration. Substantial area savings are possible with wider widths as well: a one-byte or 8-bit width eliminates 33% area, while a 16-bit width saves 24% of the area. On average the area is reduced by 31% across our benchmarks including the two benchmarks which require 32-bit vector lane widths. These area savings decrease the area cost associated with each lane enabling low-cost lane scaling depending on the width required by the application.

B. Impact of Instruction Set Subsetting

VESPA supports a *disable* parameter for each vector instruction, the activation of which removes hardware support for that instruction. Doing so automatically eliminates control logic and datapath hardware for those instructions, with whole functional units eliminated if all the corresponding instructions are disabled. In the extreme case, disabling all vector instructions eliminates the entire vector coprocessor unit. To perform the reduction we developed a tool that parses an application binary and automatically disables instructions not found.

The last column of Table V shows the area of the resulting subsetted vector coprocessors for each benchmark using a base VESPA configuration with 16 lanes, full memory crossbar, 16 kB data cache, and 64 B cache lines. Up to 57% of the area is reduced for IP_CHECKSUM which requires no multiplier functional unit and no support for stores which eliminates part of the memory crossbar. Similarly AUTCOR has no vector store instructions resulting in the second largest savings of 42% area despite using all functional units. The benchmarks CONVEN and RGBCMYK can eliminate the multiplier only resulting in 30% area savings while the remaining benchmarks cannot eliminate any whole functional unit. In those cases removing multiplexer inputs and support for instruction variations results in savings between 15%–20% area. Across all our benchmarks a geometric mean of 28% area savings is achieved.

C. Impact of Combining Width Reduction and Instruction Set Subsetting

We can combine these two customizations, width reduction and instruction subsetting, to be performed together in VESPA for a given application, thereby creating highly area-reduced VESPA processors with identical performance to a full VESPA processor. Since these customizations overlap, we expect that the savings will not be perfectly additive: for example, the savings from reducing the width of a hardware adder from 32-bits to 8-bits will disappear if that adder is eliminated by instruction set subsetting.

Fig. 11 shows the area savings of combining both the width reduction and the instruction set subsetting. For comparison, on the same figure are the individual results from width reduction and instruction set subsetting. The CONVEN benchmark can have 78% of the VESPA vector coprocessor area eliminated through subsetting and reducing the datapath width to 1 bit. Except for the cases where width-reduction is not possible, the combined approach provides additional area savings over either technique alone. Compared to the average 31% from width reduction and 28% from subsetting, combining the two produces average area savings of 47%. This is an enormous overall area savings, allowing FPGA designers to scale their soft vector processors to 16 lanes with almost half the area cost of a full general-purpose VESPA.

IX. CONCLUSION

The laborious hardware design required for efficient FPGA implementation can be mitigated by shifting some computation into a microprocessor which uses high-level programming abstractions. A *soft processor* is a compelling candidate for doing this, allowing designers to customize its architecture to

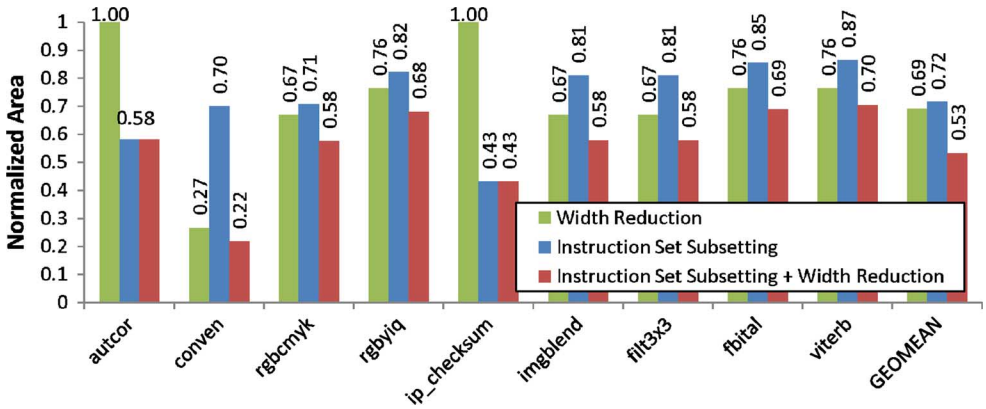

Fig. 11. Area of the vector coprocessor after eliminating both unused lane-width and hardware support for unused instructions. All configurations have 16 lanes with full memory crossbar and 16 kB data cache with 64 B cache lines. All area measurements are relative to the full uncustomized VESPA.

compete with a custom hardware implementation. In this work, we improve soft processors by allowing their architecture to respond to characteristics in the application. Specifically, we focus on the data level parallelism found in many embedded and multimedia applications. We show that soft vector processors are indeed scalable and can be used to efficiently scale performance for such applications, hence more likely justifying implementation of a computation in a soft vector processor rather than using manual hardware design. The inherent flexibility in soft processors was leveraged in two ways: 1) VESPA allows designers to precisely match their area/performance needs through its several architectural parameters which together were shown to finely span a broad 28× range in area and 18× in wall clock; and 2) VESPA was customized to reduce general purpose overheads by matching the precision and instruction support required by the application. With these characteristics and built-in portability across FPGA devices, a portable, flexible, and scalable soft vector processor can be used to implement computations on an FPGA with relative ease thereby simplifying FPGA design while still achieving high performance realizations.

REFERENCES

[1] Xilinx, San Jose, CA, "Virtex II Pro and Virtex II Pro X Platform FPGAs," DSO83 v4.7, 2007.

[2] Xilinx, San Jose, CA, "MicroBlaze," 2010. [Online]. Available: http://www.xilinx.com/microblaze

[3] Altera, San Jose, CA, "Nios II," 2010. [Online]. Available: http://www.altera.com/products/ip/processors/nios2

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture; A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann, 1992.

[5] P. Yiannacouras, J. Rose, and J. G. Steffan, "The microarchitecture of FPGA based soft processors," in *Proc. Int. Conf. Compilers, Arch., Synth. for Embed. Syst. (CASES)*, 2005, pp. 202–212.

[6] P. Yiannacouras, J. G. Steffan, and J. Rose, "Application-specific customization of soft processor microarchitecture," in *Proc. Int. Symp. Field Program. Gate Arrays (FPGA)*, 2006, pp. 201–210.

[7] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, "An FPGA-based vliw processor with custom hardware execution," in *Proc. ACM/SIGDA 13th Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2005, pp. 107–117.

[8] R. Carli, "Flexible MIPS soft processor architecture," Massachusetts Inst. Technol., Cambridge, Tech. Rep. MIT-CSAIL-TR-2008-036, 2008.

[9] B. Fort, D. Capalija, Z. G. Vranesic, and S. D. Brown, "A multithreaded soft processor for sopc area reduction," in *Proc. 14th Annu. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2006, pp. 131–142.

[10] R. Dimond, O. Mencer, and W. Luk, "CUSTARD—A customisable threaded FPGA soft processor and tools," in *Proc. Int. Conf. Field Program. Logic (FPL)*, 2005, pp. 1–6.

[11] M. Labrecque, P. Yiannacouras, and J. G. Steffan, "Scaling soft processor systems," presented at the IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM), Palo Alto, CA, 2008.

[12] D. Unnikrishnan, J. Zhao, and R. Tessier, "Application-specific customization and scalability of soft multiprocessors," presented at the IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM), Napa, CA, 2009.

[13] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer, "An FPGA-based soft multiprocessor system for ipv4 packet forwarding," *Field Program. Logic Appl.*, pp. 487–492, Aug. 2005.

[14] Univ. Toronto, Toronto, ON, Canada, "VESPA," 2011. [Online]. Available: http://www.eecg.utoronto.ca/VESPA

[15] J. Gebis and D. Patterson, "Embracing and extending 20th-century instruction set architectures," *Computer*, vol. 40, no. 4, pp. 68–75, 2007.

[16] C. Kozyrakis and D. Patterson, "Scalable, vector processors for embedded systems," *IEEE Micro*, vol. 23, no. 6, pp. 36–45, Nov. 2003.

[17] C. Kozyrakis and D. Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks," in *Proc. IEEE/ACM Int. Symp. Microarch. (MICRO-35)*, 2002, pp. 283–293.

[18] C. Kozyrakis and D. Patterson, "Overcoming the limitations of conventional vector processors," *SIGARCH Comput. Archit. News*, vol. 31, no. 2, pp. 399–409, 2003.

[19] C. Kozyrakis, "Scalable vector media processors for embedded systems," Ph.D. dissertation, Elect. Eng. Comput. Sci., Univ. California-Berkeley, Berkeley, 2002.

[20] D. Martin, "Vector extensions to the mips-iv instruction set architecture (the v-iram architecture manual)," Univ. California, Berkeley [Online]. Available: http://iram.cs.berkeley.edu/isa.ps

[21] D. Patterson, K. Asanovic, A. Brown, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, C. Kozyrakis, D. Martin, S. Perissakis, R. Thomas, N. Treuhaft, and K. Yelick, "Intelligent ram (IRAM): The industrial setting, applications, and architectures," presented at the Int. Conf. Comput, Design (ICCD), Washington, DC, 1997.

[22] K. Asanovic, J. Beck, B. Irissou, B. Kingsbury, and N. Morgan, "The to vector microprocessor," *Hot Chips*, vol. 7, pp. 187–196, 1995.

[23] K. Asanovic, "Vector microprocessors," Ph.D. dissertation, Elect. Eng. Comput. Sci., Univ. California, Berkeley, 1998.

[24] C. Sullivan and S. Chapell, "Handel-c for coprocessing and co-design of field programmable system on chip," JCRA, Wilsonville, OR, 2002.

[25] S. McCloud, "Catapult c synthesis-based design flow: Speeding implementation and increasing flexibility," Mentor Graphics, Wilsonville, OR, White Paper, 2004.

[26] D. Pellerin and S. Thibault, *Practical FPGA Programming in c*. Upper Saddle River, NJ: Prentice-Hall, 2005.

[27] OSCI, "SystemC," 2011. [Online]. Available: http://www.systemc.org

[28] B. A. Draper, A. P. W. Böhm, J. Hammes, W. A. Najjar, J. R. Beveridge, C. Ross, M. Chawathe, M. Desai, and J. Bins, "Compiling sa-c programs to FPGAs: Performance results," in *Proc. 2nd Int. Workshop Comput. Vision Syst. (ICVS)*, 2001, pp. 220–235.

[29] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core cpu accelerator," in *Proc. Symp. Field Program. Gate Arrays*, 2008, pp. 222–232.

[30] MicroSemi (formerly Actel) Corporation, Mountain View, CA, "MicroSemi ARM cortex-M1," 2011. [Online]. Available: http://www.microsemi.com
[31] Lattice Semiconductor Corporation, Hillsboro, OR, "Lattice micro32," 2011. [Online]. Available: www.latticesemi.com/micro32
[32] P. Yiannacouras, "The microarchitecture of FPGA-based soft processors," Master's thesis, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2005.
[33] LEON SPARC, Goteborg, Sweden, "Gaisler Research," 2011. [Online]. Available: http://www.gaisler.com
[34] A. Lodi, M. Toma, and F. Campi, "A pipelined configurable gate array for embedded processors," in *Proc. ACM/SIGDA 11th Int. Symp. Field Program. Gate Arrays (FPGA)*, 2003, pp. 21–30.
[35] M. A. R. Saghir, M. El-Majzoub, and P. Akl, "Datapath and isa customization for soft vliw processors," in *Proc. Reconfig. Comput. FPGA's*, 2006, pp. 1–10.
[36] M. Labrecque and J. G. Steffan, "Improving pipelined soft processors with multithreading," in *Proc. FPL*, 2007, pp. 210–215.
[37] R. Moussali, N. Ghanem, and M. A. R. Saghir, "Supporting multithreading in configurable soft processor cores," in *Proc. Int. Conf. Compilers, Arch., Synth. for Embed. Syst.*, 2007, pp. 155–159.
[38] R. Dimond, O. Mencer, and W. Luk, "Application-specific customisation of multi-threaded soft processors," *IEE Proc.—Comput. Digit. Techn.*, vol. 153, no. 3, pp. 173–180, May 2006.
[39] J. Kingyens and J. G. Steffan, "A GPU-inspired soft processor for high-throughput acceleration," in *Proc. Reconfig. Arch. Workshop*, 2010, pp. 9–10.
[40] F. Plavec, Z. Vranesic, and S. Brown, "Towards compilation of streaming programs into FPGA hardware," in *Proc. Forum on Specification, Verification, Design Lang. (FDL)*, 2008, pp. 67–72.
[41] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Trans. Reconfig. Technol. Syst. (TRETS)*, vol. 2, no. 2, pp. 1–31, 2009, Art. No. 12.
[42] MIPS, Sunnyvale, CA, "MIPS technologies," 2011. [Online]. Available: http://www.mips.com
[43] P. Yiannacouras, "FPGA-based soft vector processors," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2009.
[44] J. Fender, J. Rose, and D. R. Galloway, "The transmogrifier-4: An FPGA-based hardware development system with multi-gigabyte memory capacity and high host and memory bandwidth," in *Proc. IEEE Int. Conf. Field Program. Technol.*, 2005, pp. 301–302.
[45] P. Metzgen, "A high performance 32-bit ALU for programmable logic," in *Proc. ACM/SIGDA 12th Int. Symp. Field Program. Gate Arrays*, 2004, pp. 61–70.
[46] EEMBC, El Dorado Hills, CA, "The embedded microprocessor benchmark consortium," 2011. [Online]. Available: http://www.eembc.org
[47] D. Lau, O. Pritchard, and P. Molson, "Automated generation of hardware accelerators with direct memory access from ansi/iso standard c functions," in *Proc. FCCM*, 2006, pp. 45–56.
[48] P. Yiannacouras, J. G. Steffan, and J. Rose, "Exploration and customization of FPGA-based soft processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 266–277, Feb. 2007.
[49] J. E. Veenstra and R. J. Fowler, "MINT: A front end for efficient simulation of shared-memory multiprocessors," in *Proc. 2nd Int. Workshop Model., Anal., Simulation Comput. Telecommun. Syst. (MASCOTS)*, 1994, pp. 201–207.
[50] P. Yiannacouras, J. G. Steffan, and J. Rose, "Improving memory systems for soft vector processors," in *Proc. Workshop Soft Process. Syst. (WoSPS)*, 2008, pp. 19–24.
[51] P. Yiannacouras, J. G. Steffan, and J. Rose, "Data parallel FPGA workloads: Software versus hardware," presented at the FPL, Grenoble, France, 2009.
[52] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 174–199, 2000.
[53] J. W. C. Fu and J. H. Patel, "Data prefetching in multiprocessor vector cache memories," *SIGARCH Comput. Archit. News*, vol. 19, no. 3, pp. 54–63, 1991.
[54] P. Yiannacouras, J. G. Steffan, and J. Rose, "Fine-grain performance scaling of soft vector processors," in *Proc. Int. Conf. Compilers, Arch. Synth. Embed. Syst. (CASES)*, 2009, pp. 97–106.
[55] R. M. Russell, "The cray-1 computer system," *Commun. ACM*, vol. 21, no. 1, pp. 63–72, 1978.

Peter Yiannacouras (M'09) received the B.A.Sc. degree in engineering science, and the M.A.Sc. and the Ph.D. degree from the Electrical and Computer Engineering Department, University of Toronto, Toronto, ON, Canada, in 2003, 2005, and 2009, respectively.

He was with Intel Microarchitecture Research Labs in 2006, Nokia Research Center in 2009, and is currently with Altera Corporation, since 2010. His research interests include processor and system architecture, and embedded processing.

J. Gregory Steffan (SM'08) received the undergraduate and M.S. degrees from the University of Toronto, Toronto, ON, Canada, and the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, in 1995, 1997, and 2003, respectively.

He is an Associate Professor with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto. His research interests include computer architecture and compilers, reconfigurable computing, and distributed and parallel systems.

Prof. was a recipient of the Ontario Ministry of Research and Innovation Early Researcher Award (2007), a Siebel Scholar (2002), an IBM CAS Visiting Scientist and Faculty Fellow. He is a senior member of the ACM.

Jonathan Rose (F'09) is a Professor with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. From 1986 to 1989, he was a Post-Doctoral Scholar and then a Research Associate with the Computer Systems Laboratory, Stanford University, Stanford, CA. He spent the 1995–1996 year as a Senior Research Scientist with Xilinx, San Jose, CA, working on the Virtex FPGA. In October 1998, he co-founded Right Track CAD Corporation, which delivered architecture for FPGAs and packing, placement, and routing software for FPGAs to FPGA device vendors. He became a Senior Director of the Altera Toronto Technology Centre from May 2000 to April 2003, after the acquisition of Right Track, sharing responsibility for the development of the architecture for the Altera Stratix, Stratix II, Stratix GX, and Cyclone FPGAs and associated software. His research covers all aspects of FPGAs including their architecture, computer-aided design, field-programmable systems, soft processors, and graphics, vision bio-informatic and mobile applications of programmable hardware.

Dr. Rose is the co-founder of the ACM FPGA Symposium. He served as Chair of the Edward S. Rogers Sr. Department of Electrical and Computer Engineering from 2004 through 2009. He is a Fellow of the IEEE, ACM, and Canadian Academy of Engineering, and is a Foreign Associate of the American National Academy of Engineering.