DATA PARALLEL FPGA WORKLOADS: SOFTWARE VERSUS HARDWARE

Peter Yiannacouras, J. Gregory Steffan, and Jonathan Rose

Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road, Toronto, ON
email: yiannac,steffan,jayar@eecg.utoronto.ca

ABSTRACT

Commercial soft processors are unable to effectively exploit the data parallelism present in many embedded systems workloads, requiring FPGA designers to exploit it (laboriously) with manual hardware design. Recent research [1, 2] has demonstrated that soft processors augmented with support for *vector* instructions provide significant improvements in performance and scalability for data-parallel workloads. These soft vector processors provide a software environment for quickly encoding data parallel computation, but their competitiveness with manual hardware design in terms of area and performance remains unknown. In this work, using an FPGA platform equipped with DDR memory executing data-parallel EEMBC embedded benchmarks, we measure the area/performance gaps between (i) a scalar soft processor, (ii) our improved soft vector processor, and (iii) custom FPGA hardware.

We demonstrate that the 432x wall clock performance gap between scalar executed C and custom hardware can be reduced significantly to 17x using our improved soft vector processor, while silicon-efficiency is improved by 3x in terms of area-delay product. We modified the architecture to mitigate three key advantages we observed in custom hardware: loop overhead, data delivery, and exact resource usage. Combined these improvements increase performance by 3x and reduce area by almost half, significantly reducing the need for designers to resort to more challenging custom hardware implementations.

1. INTRODUCTION

The designer of an FPGA-based embedded system often has the difficult choice between designing custom hardware by hand using a hardware-description language (HDL) that is mapped directly to the FPGA fabric, or writing software in a high-level language such as C that targets a *soft processor*— a processor implemented using the programmable FPGA fabric and programmed using traditional sequential programming languages and software compilers. The performance of a soft processor is often sufficient for parts of the design allowing embedded systems designers to use them to reduce their time to market and exploit single-chip advantages without requiring specialized FPGAs with *hard processors*; however, the performance and area of current commercial soft processors is still significantly inferior to that of a custom hardware solution, meaning designers need to spend more time implementing hardware to meet their design constraints. As a result, we are motivated to improve soft processors to reduce FPGA design time.

Recent advancements in both academia [3, 4] and industry [5] have indeed expanded the applicability of soft processors by improving them over current commercial soft processors. In particular, recent work has proposed extending soft processors with vector processing capabilities [1,2] as a means of scaling performance for data-parallel workloads. Vector processing allows a single instruction to command multiple datapaths called *vector lanes*. On an FPGA the number of vector lanes can be configured by the designer, allowing them to use more FPGA resources to scale-up performance. However, the impact of soft vector processors depends on their ability to lure FPGA designers into software design by providing *good enough* performance/area to reduce the amount of needed manual hardware design. Thus, it is crucial to understand the perfomance and area gap between soft vector processors and custom hardware.

1.1. Measuring, Understanding, and Reducing the Gap

We measure the area and performance gap using several data-parallel benchmarks (primarily from the industry-standard EEMBC [6] embedded benchmark suites) of three platforms executing: (i) "out-of-the-box" C on a scalar soft processor; (ii) hand-vectorized-assembly on many configurations of the soft vector processor called VESPA (Vector Extended Soft Processor Architecture) [1]; and (iii) custom hardware hand-designed in Verilog. Our goal in this work is to use this measurement to quantify the competitiveness of recent soft vector processors and further improve them by leveraging our insights into the causes of the performance/area gap as

well as the circuit structures used to implement the benchmarks in hardware. Specifically we identify the following key advantages of custom hardware over VESPA, and we improve VESPA reducing the impact of each advantage.

Loop Overhead Loop control in custom hardware is generally implemented using a finite state machine (FSM) that executes in parallel with the loop computation, while in VESPA the control datapath must complete an instruction before the vector lanes can issue the following instruction, and vice versa. We reduce this advantage and improve VESPA by decoupling the control datapath from the vector lanes and exploiting instruction-level parallelism.

Data Delivery High performance custom hardware can often achieve near perfect delivery of data to functional units with no cycles wasted. In contrast, for soft processors including VESPA, data flows from memory through caches to registers and eventually to functional units. We improve data delivery in VESPA in two ways: (i) by tuning cache design, and (ii) by supporting prefetching.

Exact Resource Usage A custom hardware implementation contains exactly the resources required to support the application: functional units support only the required operations, and datapath bit-widths exactly match those required. In contrast, soft processors such as VESPA are general-purpose and hence support a full instruction set (ISA) and the corresponding maximum bit-widths. We improve VESPA via support for subsetting the instruction set and reducing datapath bit-widths to match the application.

In this work we demonstrate that these improvements when combined provide 3x improved performance over the original VESPA and significantly broaden its design space. We also show that the performance gap between a scalar soft processor and custom hardware is 432x, and that our fastest VESPA implementation reduces this gap to 17x, while providing a performance-per-unit-area that is up to 3x that of the scalar processor. While the remaining gap is still large, these improvements allow soft vector processors to better compete with custom hardware, allowing designers to more often implement a software-programmable solution rather than having to design custom hardware.

1.2. Related Work

The most closely related work is by Yu *et. al.* [2], who demonstrate the potential for vector processing as a simple-to-use and scalable accelerator for soft processors, potentially scaling better than Altera's `C2H` [5] behavioral synthesis tool for three benchmarks. However, that work models a vector processor optimistically including using an on-chip one-cycle (latency) memory system. We compare a real vector processor to manual hardware design.

Hardt and Camposano [7] compare hardware circuits synthesized to 2μ CMOS to software on a SPARC processor with cycle performance estimated from static code analy-
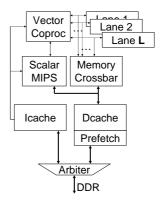

Fig. 1. VESPA processor block diagram.

sis. They find that hardware outperforms the processor by factors ranging between 24x and 44x for scalar workloads. Our work performs a similar comparison but between FPGA hardware and soft vector processors, while including the effects of clock frequency and latent memory. More recent work [8] has compared FPGAs to hard microprocessors but do not compare against soft vector processors.

1.3. Contributions

In this paper we make the following contributions: (i) using an FPGA platform with DDR memory we quantify and analyze the area/performance gaps for industry-standard benchmarks between a scalar soft processor, a parameterized vector soft processor, and hand-designed hardware implementations; (ii) we improve VESPA by targetting key advantages of hardware implementations—specifically by reducing loop overhead, tuning cache design, supporting data prefetching, and eliminating unused hardware; (iii) we show that our improved soft vector processor VESPA provides a powerful design space, spanning 5x in area and 11x in performance, with the fastest VESPA reducing the 432x scalar soft processor performance gap to 17x while improving silicon efficiency in terms of area-delay product by up to 3x.

2. VESPA

In our previous work on VESPA (Vector Extended Soft Processor Architecture) we implemented a parameterized vector processor in Verilog and explored its potential for scalability and customization. The following summarizes the VESPA architecture and parameters (old and new), further details can be found in [1].

Figure 1 shows a block diagram of the VESPA processor that consists of a scalar MIPS-based processor automatically generated using the SPREE system [3], coupled with a parameterized vector coprocessor based on the VIRAM [9]

Table 1. Configurable parameters for VESPA.

Parameter	Symbol	Values
Vector Lanes	L	1,2,4,8,16,...
Vector Lane Bit-Width	W	1,2,3,4,..., 32
Maximum Vector Length	MVL	2,4,8,16,...
Memory Crossbar Lanes	M	1,2,4,8,...L
Each Vector Instruction	-	on/off
DCache Depth (KB)	DD	4KB,8KB,...
DCache Line Size (B)	DW	16,32,64,...
DCache Miss Prefetch	DPK	1,2,3,...
Vector Miss Prefetch	DPV	1,2,3,...

vector instruction set. The scalar SPREE processor is a 3-stage pipeline with full forwarding and a 1-bit branch history table. The parameters of the VESPA system are listed in Table 1. The vector coprocessor consists of L parallel vector lanes where each lane can perform operations on a single element in a pipelined fashion. The width W of each vector lane datapath is 32 bits by default, but can be reduced for applications that require less than the full 32 bit-width. MVL determines the maximum vector length supported in hardware and is set to 64 for this study.

The scalar processor and vector coprocessor share a single instruction stream fed by an instruction cache. The scalar processor and vector coprocessor are both in-order pipelines, but can execute out-of-order with respect to each other except for memory operations which are serialized to maintain sequential consistency. Both share a direct-mapped data cache with parameterized depth DD and cache line size DW. A crossbar routes each byte in a cache line to/from M of the L vector lanes in a given cycle. A full crossbar (M=L) can significantly reduce the clock frequency of the design when L is large; in such cases M can be reduced to restore the clock rate and save area, but more cycles will be spent moving data between the cache lines and vector lanes. The data cache is equipped with a hardware prefetcher configured with parameters DPK and DPV described in a later section.

Beyond our previous work, we compare the VESPA configurations to hardware for the first time, we added configurable caches and data prefetching, we explore the *complete* design space with our new robust design rather than individually for each parameter, and finally we make other non-parameterized architectural improvements (see Section 5.2).

3. MEASUREMENT METHODOLOGY

Our goal is to measure the area/performance gap between scalar soft processors, soft vector processors, and hardware, as well as to investigate techniques to reduce the gap. In this section we describe the components of our infrastructure necessary to execute, verify, and evaluate the FPGA designs. We describe our hardware platform, verification process, CAD tool measurement methodology, benchmarks, and compiler. We also discuss how hardware implementations of our benchmarks were created.

Soft Processor Platform We use the multi-FPGA Transmogrifier 4 (TM4) [10] to host the complete soft processor systems and to measure the number cycles for benchmark execution. The platform has four Altera Stratix EP1S80F1508C6 devices each with access to two 1GB PC3200 CL3 DDR SDRAM DIMMs clocked at 133 MHz (266 MHz DDR). We synthesize our processor systems onto one of the four Stratix I FPGAs connected to one of the DIMMs and clock the processor at 50 MHz. All instances of VESPA are fully tested in hardware using the built-in checksum values encoded into each benchmark. Debugging is performed using Modelsim and is guided by comparing traces of all writes to the scalar and vector register files. Note that because the Stratix I FPGAs on the TM4 are dated, we use this platform only for measuring benchmark cycle counts. For area and clock frequency measurements we use the CAD flow described below to target a faster Stratix III FPGA (which was unavailable to us) and achieves a clock speed of 130 MHz. While this faster clock speed would increase the memory latency observed by the processor, we believe that this would not significantly impact our results: the memory latency in our current system is already exaggerated by the fact that our DDR controller is hand-made and suffers many inefficiencies, including the use of a closed-page policy that requires every memory request to incur both row and column access latencies.

FPGA CAD Tools A key benefit of FPGA-based systems research is that we can obtain high quality measurements, including the area and clock frequency measurements provided by FPGA CAD tools. We use Altera's Quartus II 8.0 CAD software with register retiming and duplication enabled and with aggressive timing constraints. Through experimentation we found that these settings provided the best area, delay, and runtime trade-off. We perform eight such runs for each hardware design to average-out the non-determinism in the CAD algorithms. The relative silicon area of each tile in a Stratix II FPGA was supplied to us by Altera [11], which we then adjust to approximate the slightly modified FPGA architecture of the Stratix III. We report the silicon area consumed by a design in units of *equivalent ALMs*—the silicon area of a single ALM (Adaptive Logic Module—the basic programmable logic unit in the Stratix III) including its routing. For VESPA processors the areas we report include the instruction cache, data cache, data prefetcher, scalar processor, and vector processor.

Table 2. Benchmark applications.

Benchmark	Description	Source	EEMBC Suite	EEMBC Dataset#	Input size (B)	Output size (B)	Largest Vector Element	% VIRAM ISA Used
AUTCOR	auto correlation	EEMBC/VIRAM	Telecom	2	1024	64	32 bits	9.6%
CONVEN	convolution encoder	EEMBC/VIRAM	Telecom	1	517	1024	1 bit	5.9%
RGBCMYK	rgb filter	EEMBC/VIRAM	Digital Ent.	5	1628973	2171964	8 bits	5.9%
RGBYIQ	rgb filter	EEMBC/VIRAM	Digital Ent.	6	1156800	1156800	16 bits	8.1%
IP_CHECKSUM	checksum	EEMBC	Networking	-	40960	40	32 bits	8.1%
IMGBLEND	combine two images	VIRAM	-	-	153600	76800	16 bits	7.4%

Benchmarks The six benchmarks that we measure are listed in Table 2: five are from the industry-standard EEMBC collection [6], and one (IMGBLEND) was hand-made. AUT-COR, CONVEN, RGBCMYK, and RGBYIQ were hand-vectorized and provided by Kozyrakis and the Berkeley VIRAM project [9]. For these benchmarks we execute using the largest dataset with the EEMBC test harness uncompromised. We also manually extracted and vectorized the IP_CHECKSUM kernel from the Networking suite of EEMBC, and execute it using correspondingly-modified input data stream. Note that cycle counts are collected from a complete execution on our hardware platform as described above, and the vectorized code is never modified to support any specific vector configuration.

Compilation Framework Benchmarks are built using a MIPS port of GNU gcc 4.2.0 with the -O3 optimization level. Initial experiments with this version of gcc's auto-vectorization capability showed that it failed to vectorize key loops in our benchmarks, preventing us from automatically generating vectorized code. Instead we ported the GNU assembler to support VIRAM vector instructions allowing us to manually vectorize in assembly.

Area-Delay Product A system designer may care more about area than performance, or vice-versa, depending on the constraints of the design at hand. However, it is important to have an understanding of the overall performance-per-area of candidate designs motivating us to measure *area-delay product* as is traditionally done for digital circuits. We use the aforementioned equivalent ALMs for area and the wall-clock-time of benchmark execution as the delay (combining the cycle counts reported by real hardware with the maximum clock frequency reported by CAD tools).

3.1. Designing Custom Hardware Circuits

We model the performance of our hardware circuits optimistically while using area and clock frequencies from a real FPGA hardware design, achieved by manually converting each benchmark into a Verilog hardware circuit. While there are infinite variations of such hardware designs, we attempted to implement designs that maximize performance while simplifying this process with the following assump-

Table 3. Hardware circuit area and performance.

Benchmark	ALMs	DSPs	M9Ks	Clock (MHz)	Cycles
AUTCOR	592	32	1	323	1057
CONVEN	46	0	0	476	226
RGBCMYK	527	0	0	447	237784
RGBYIQ	706	108	0	274	144741
IP_CHECKSUM	158	0	0	457	2567
IMGBLEND	302	32	0	443	14414
AUTCOR-unroll	3699	256	0	244	143
CONVEN-unroll	67	0	0	476	98

tions: All input/output data starts/ends in memory and is transfered uninterrupted at the full rate of our DRAM device. We also idealize the control logic assuming it can make decisions in a single clock and accounts for negligible area. Finally we don't allow any value or value-range specific optimization in either the software or hardware. To summarize, we build only the datapath of the circuit under optimistic assumptions about the control logic and transfer of data. The resulting hardware circuits are tested in simulation using test vectors, and area and clock frequency are measured using the previously-described CAD flow. For each hardware circuit we compute the total number of cycles for execution as the sum of the pipeline latency plus cycles spent transferring data since the circuit computation is done in parallel with this transfer time. Overall we believe the hardware circuits are optimistic and certainly overcome the manual vectorization advantage in software.

As a result of forbidding value and value-range optimizations, we do not perform loop unrolling of non-vectorized loops, nor the equivalent in hardware. For example, benchmarks such as AUTCOR operate repeatedly on the same data set with the actual computation dependent on a parameter input which varies from 0 to 15. In hardware we can unroll that loop performing all 16 operations simultaneously. The benefit of unrolling a loop would be relatively small for VESPA which is an in-order single-issue processor, while hardware could readily exploit the exposed instruction level parallelism (ILP). The last two rows in Table 3 show the impact of unrolling in hardware for AUTCOR and CONVEN—the only two benchmarks where unrolling is useful in hardware. The unrolled circuits are not used in our results, but the performance impact can be large:

in the case of AUTCOR execution completes in 7.4x fewer cycles, although clock frequency is reduced and circuit area increases substantially.

4. COMPARING TO HARDWARE

In this section we compare the area and performance of the following three implementations of our benchmarks created via different design entry methods: (i) out-of-the-box C code executed on the MIPS-based SPREE scalar processor; (ii) hand-vectorized assembly language executed on many variations of our VESPA soft vector processor; and (iii) hardware designed in Verilog at the register transfer level as described in Section 3.1.

Table 4 shows the area advantage and speedup of the hardware implementation versus the scalar SPREE processor in the first row and the slowest, the least area-delay, and the fastest configurations of VESPA in the remaining three rows. The limited number of multipliers in the Stratix 1S80 on the TM4 prevent us from evaluating soft vector processors with more than 16 lanes, but we expect further performance scaling on larger Stratix III based hardware platforms [1]. Focusing on the first row of the table, we observe that the scalar processor executing out-of-the-box C code is on average 6.7x larger than the hardware circuits and performs 432x slower. Not exploiting the available data parallelism is the primary cause of the under-performance. The area of the scalar processor is larger than each of the hardware implementations, suggesting that despite the time-multiplexed resources, the general purpose overheads cause the processor to be still larger than the spatially executed hardware. In an extreme case, CONVEN with its 1-bit datapath is 64x smaller than the scalar processor.

With respect to the hardware circuits VESPA is 13x to 64x larger and 192x to 17x slower. A more quantitative analysis follows in a subsequent section but it is clear that vector processing extensions to soft processors are motivated since the 432x scalar processor performance gap can be reduced down to 17x. Such a massive performance boost could help convert many components of an FPGA system into software executing on a soft vector processor rather than laboriously-designed custom hardware.

Figure 2 shows the area-performance design space of many near-pareto-optimal VESPA processors normalized against hardware. We observe that the VESPA design space is quite large, spanning 5x in area and 11x in performance with the 16 lane VESPAs providing the best performance at the cost of additional area. The figure identifies the number of lanes in each configuration which is the most dominant parameter in determining area and performance, but also being varied is the memory crossbar size M, the data cache depth DD, the data cache line size DW, and the data prefetcher DPV. These parameters will be discussed
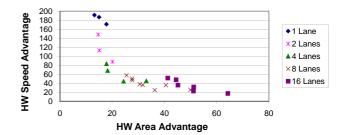

Fig. 2. Area-performance design space of VESPA processors normalized against hardware.

in a subsequent section, here they are used to show the fine-grained tradeoffs within VESPA. The tradeoffs are significant because VESPA could be a potentially large component in an FPGA system.

4.1. VESPA vs Scalar

Looking at the first two rows of Table 4, we can compare the scalar processor with a VESPA processor that has only a single lane and identical cache organization. The VESPA processors are at least 2x larger than the scalar since they are comprised of both a scalar processor and vector coprocessor. The hand-vectorized assembly executed on VESPA gains more than 2x average performance over the scalar out-of-the-box C code on scalar SPREE, even though there is no data parallel execution on the single-lane version of VESPA. This is partly due to a number of advantages in VESPA: (a) More efficient pipeline execution with few dependencies. (b) The large vector register file can store and manipulate arrays without having to access the cache or memory. (c) Amortization of loop control instructions. (d) Direct support for fixed-point operations, predication, and built-in min/max/absolute instructions in the VIRAM instruction set. (e) Simultaneous execution in the scalar processor and vector co-processor. (f) Manual vectorization in assembly versus the C-compiled scalar output from GCC.

Determining the exact contribution of each advantage is beyond the scope of this work, we instead perform some qualitative analysis. Closer inspection of CONVEN revealed the cause of the 9x performance boost seen on the single lane VESPA to be the repeated operations performed on a single array. In VESPA the large vector register file can store large array chunks and manipulate them without storing and re-reading them from cache as the scalar processor must. The other benchmarks are less impacted because of their streaming and low-reuse nature. The loop overhead amortization gained by performing 64 loop iterations (MVL=64) at once significantly impacts benchmarks with small loop bodies such as AUTCOR, CONVEN, IP_CHECKSUM, and IMGBLEND. The more powerful VIRAM instruction set with fixed-point support further reduced the loop bodies of

Table 4. Area and performance advantage for hardware over various processors

Processor					Clock	Area ($A_{processor}/A_{hw}$)							Wall Clock Time ($T_{processor}/T_{hw}$)						
L	M	DD (KB)	DW (B)	DPV	(MHz)	AUTCOR	CONVEN	RGB-CMYK	RGB-YIQ	IP_CH-ECKSUM	IMG-BLEND	**GEO MEAN**	AUTCOR	CONVEN	RGB-CMYK	RGB-YIQ	IP_CH-ECKSUM	IMG-BLEND	**GEO MEAN**
Scalar		4	16	0	159	2.7	63.8	5.6	1.3	18.6	3.9	**6.7**	440.8	1899.6	267.7	549.1	163.9	322.5	**432.1**
1	1	4	16	0	141	5.3	125.3	10.9	2.6	36.5	7.7	**13.2**	224.8	211.7	204.9	205.9	114.3	214.7	**191.5**
8	8	16	64	8VL	139	14.6	344.2	30.0	7.1	100.2	21.1	**36.3**	32.8	30.1	27.2	25.7	12.4	25.8	**24.6**
16	16	16	64	8VL	122	25.9	610.0	53.2	12.7	177.6	37.4	**64.3**	23.8	24.4	18.5	16.4	8.8	16.0	**17.1**

Table 5. Hardware advantages over fastest VESPA.

Benchmark	Clock	Iteration Parallelism	Cycles per Iteration
autcor	2.6x	1x	9.1x
conven	3.9x	1x	6.1x
rgbcmyk	3.7x	0.375x	13.8x
rgbyiq	2.2x	0.375x	19.0x
ip_checksum	3.7x	0.5x	4.8x
imgblend	3.6x	1x	4.4x
GEOMEAN	3.2x	0.64x	8.2x

Fig. 3. Area-delay product versus area of VESPA processors normalized against hardware.

AUTCOR and RGBCMYK. Finally, the scalar disassembled GCC output did not appear significantly less efficient than the vectorized assembly for any of the benchmarks, leading us to infer that manual assembly optimization was not a disproportionally significant advantage for VESPA.

4.2. VESPA vs HW

By focussing only on loops we can decompose the performance difference between VESPA and hardware into the following categories: (i) the clock frequency; (ii) the number of loop iterations executed concurrently called *iteration level parallelism*; and (iii) the number of cycles required to execute a single loop iteration. For each of these components, the hardware advantage over the fastest VESPA configuration (see last row of Table 4) is shown in Table 5. The second column shows the hardware circuits have clock speeds between 2.2x and 4x faster than the best performing VESPA. This 3.2x average clock advantage can be improved through further circuit design effort in VESPA.

The third column of Table 5 shows that the iteration level parallelism exploited by the hardware is less than or equal to that exploited by VESPA which is 16 for all benchmarks since there are 16 lanes. But in the hardware circuits we matched the parallelism to the memory bandwidth, for example, the IP_CHECKSUM benchmark operates on a stream of 16-bit elements meaning in a given DRAM access only 8 elements can be retrieved from memory. The circuit is hence designed to have only 8-way parallelism while VESPA wastes cycles gathering data for its 16 lanes.

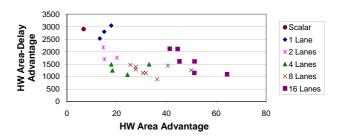The last column shows the speedup of a single it-

eration in hardware over VESPA and is calculated from the measured overall speedups in the last row of Table 4 divided by the aforementioned clock and iteration parallelism advantages. This component represents the inefficiencies inherent in our VESPA design as well as in any processor-style architecture. VESPA currently can sustain only one vector instruction in flight while known techniques such as vector chaining can be used to overlap execution of multiple instructions through a multi-ported vector register file and multiple functional units. The hardware circuit has the benefit of creating as many functional units as necessary and can keep them fed with data without the scaling limitations of a centralized register file.

Further improvements to VESPA's cycles per iteration are motivated since it remains the largest component and will further expose fundamental limitations in processor architectures. VESPA's vector extensions reduced the iteration parallelism hardware advantage from 10.3x for a scalar soft processor to 0.64x, proving that VESPA has greatly increased iteration parallelism leaving cycles per iteration as a key target for further reducing the performance gap.

4.3. Area-Delay Product Gap

Figure 3 shows the area-delay of the scalar and VESPA processors relative to that of hardware, averaged across our benchmark set, and plotted against area. The figure demonstrates that VESPA can provide up to a 3.25x decrease in area-delay versus the scalar SPREE processor. Note that VESPA includes the same scalar SPREE processor, thus, adding the vector extensions significantly increase the performance-per-area of this processor. The VESPA
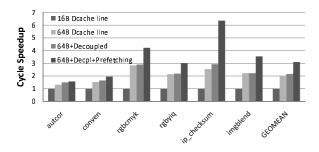
Fig. 4. Performance gained with improved VESPA architecture.

processor with the least area-delay product is still 892x worse than the hardware but is surprisingly not the VESPA design with the highest performance, instead it is the 8-lane, full memory crossbar vector processor with a 16KB cache, 64B line size, and data prefetching listed in the second last row of Table 4. While this area-delay gap is enormous, a significant part of it is due to area which in many cases may be well worth the general-purpose computing provided by the processor. Specifically, the processor can be used to implement many parts of an application: for example, if a designer wanted to execute all six of our benchmarks sequentially they can do so on a single instance of VESPA which would reduce the area gap by 6x compared to the instantiation of all six circuits.

5. REDUCING THE PERFORMANCE GAP

In this section we examine the performance advantages that hardware circuits have over VESPA and describe the architectural modifications that we use to mitigate the effects of those advantages: we examine different cache designs, the decoupling of certain pipelines within VESPA, and data prefetching These techniques directly tackled the cycles per iteration highlighted in our earlier results which included these improvements. Figure 4 shows the accumulated performance gains from these three improvements measured in cycle speedup since clock frequency did not change significantly. On average the cache, decoupling, and prefetching can be combined to increase performance by 3x over the previous VESPA, causing its 50x performance gap with hardware to be reduced to the 17x reported in Table 5.

5.1. Cache Design

Hardware circuits typically benefit from near-perfect delivery of data from the DRAM to the pipelined functional units, while for most processors data passes through levels of caches, then the register file, and finally to the functional units. Although we maintained this framework, we accomodated VESPA by tuning the cache, specifically the cache line so that ideally all vector lanes can be satisfied with a single

cache line request. The data cache line was parameterized and expanded from 16 bytes to 64 bytes, and accompanied with a corresponding growth in capacity to keep the FPGA block RAMs fully utilized. Our experiments show that this improved cache design results in 2x average performance gain as seen in Figure 4, due almost entirely to the expanded cache line rather than the capacity [12]. This performance gain comes with a 2x growth in VESPA area due primarily to the larger vector memory crossbar seen in Figure 1 which grows with the cache line size. The crossbar is necessary even without a cache, and since the cache storage is less than 6% of the area and is shared with the scalar processor, we are not motivated to investigate a no-cache solution.

5.2. Zero Overhead Loops

When comparing the hardware circuits to the vectorized loops, one glaring difference is the absence of the many control instructions required to manage a loop: in hardware a finite state machine (FSM) manages the loop in parallel with the computation. We modified VESPA by decoupling the three pipelines allowing vector, vector control, and scalar instructions to execute simultaneously and out-of-order with respect to each other. As long as the number of cycles needed to compute the vector operations is greater than the cycles needed for the vector control and scalar operations, the loop will have no overhead. While our previous work already decoupled the scalar pipeline, in this work we decouple the execution of the vector and vector control pipelines. The impact on performance for a 16-lane VESPA with 16KB data cache and 64B line size is shown in Figure 4. The technique improves performance by 7% on average and by 15% in the best case, while the area cost is negligible.

5.3. Data Prefetching

Another advantage of custom hardware is that it can overlap computation with memory accesses. We can do the same in VESPA by supporting hardware data prefetching where a cache miss translates into a request for the missing cache line as well as additional cache lines that are predicted to soon be accessed. Due to the predictable memory access patterns in our benchmarks simple sequential prefetching that loads the next DPK cache lines is effective, reducing the time spent servicing misses to just 4% of execution time [12]. Using the DPV parameter instructs VESPA to prefetch only for vector memory instructions with low strides and to prefetch either a constant or a multiple of the current vector length elements into the cache. All of these methods yield very similar results.

Figure 4 shows the 42% performance boost of our best overall prefetching configuration which loads 8 times the current vector length elements into the cache. By using
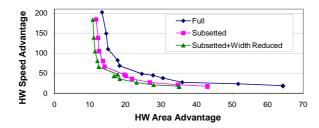
Fig. 5. Effect of instruction set subsetting and width reduction on the area and speed gap of VESPA processors versus hardware.

the vector length to determine the number of cache lines to prefetch, we guarantee no more than one miss per vector instruction regardless of the length of the vector. The cost of the prefetcher is less than 2% of the area due primarily to buffering dirty cache lines evicted by prefetched lines.

6. REDUCING THE AREA GAP

In hardware, we implement only the functional units required by the application and match them to the bit-width of the data operands. VESPA is equipped with parameters that allow it to perform similar application-specific customizations. The vector lane width W can be used to reduce the datapath for benchmarks which do not require the 32-bit processing VESPA provides. For example, CON-VEN requires only a 1-bit datapath (see Table 2) and its implementation in hardware gains a large area advantage over VESPA because of it. Using the W parameter we can reduce the lane width to 1-bit and reduce VESPA's area by half—vector state, control logic, the 32-bit address space, and the scalar processor limit further reduction. Note our previous work [1] limited the lane width to multiples of 8. VESPA also supports the individual disabling of each vector instruction which automatically eliminates hardware support for that instruction. This feature allows us to subset the instruction set to that used by the application shown in Table 2.

Figure 5 shows the effect of instruction set subsetting as well as the combined effect of subsetting and width reduction on the set of pareto optimal points in our VESPA design space. We see that compared to the full VESPA processor the area is significantly reduced, in the best case by 45%, and some performance is even gained from the higher clock speeds which reach as high as 153 MHz on the smaller customized VESPA processors. The points move closer to the origin as VESPA sheds general purpose overheads and begins to resemble a dedicated hardware part. It is interesting to note that after trimming this area, the 16 lane VESPA with full memory crossbar, prefetching, and 64B line size has the smallest area-delay product which is 561x worse than hardware; a substantial improvement over

the 892x for the full-size 8 lane VESPA discussed earlier, and 5.15x better than the scalar soft processor.

7. CONCLUSIONS

Our comparisons have demonstrated that C code executing on a scalar soft processor performs on average 432x slower and is 6.7x larger in area than custom FPGA hardware. The VESPA soft vector processor now provides a large design space of vector processors that, relative to hardware, ranges from 192x slower and 13x larger to 17x slower and 64x larger. This large space allows a designer to choose the area/performance of a system component without laborious hardware design, and can drastically reduce the 432x scalar soft processor performance gap to 17x for data parallel workloads. In addition, VESPA is shown to have 3x better area-delay product than our scalar soft processor. Finally, by eliminating hardware in VESPA which is not used by the application, we can reduce the area of VESPA by up to 45%, resulting in a 5.15x reduced area-delay product than that of a scalar soft processor. In summary, the quantified gap and improved soft vector processor can significantly reduce the need for embedded designers to resort to more challenging manual hardware design.

8. REFERENCES

[1] P. Yiannacouras, J. G. Steffan, and J. Rose, "Vespa: Portable, scalable, and flexible fpga-based vector processors," in *CASES'08: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 2008.

[2] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core cpu accelerator," in *Symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2008, pp. 222–232.

[3] P. Yiannacouras, J. G. Steffan, and J. Rose, "Application-specific customization of soft processor microarchitecture," in *FPGA'06: Proceedings of the International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM Press, 2006, pp. 201–210.

[4] R. Dimond, O. Mencer, and W. Luk, " CUSTARD - A Customisable Threaded FPGA Soft Processor and Tools ," in *International Conference on Field Programmable Logic (FPL)*, August 2005.

[5] D. Lau, O. Pritchard, P. Molson, and C. Altera Santa Cruz, "Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions," *Field-Programmable Custom Computing Machines*, pp. 45–56, 2006.

[6] "The Embedded Microprocessor Benchmark Consortium," http://www.eembc.org, EEMBC.

[7] W. Hardt and R. Camposano, "Trade-offs in hw/sw codesign," in *Workshop on Hardware/Software Codesign*. ACM, 1994.

[8] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of fpgas over processors," in *Symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2004, pp. 162–170.

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

[10] 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 *IEEE International Conference on Field Programmable Technology*, 2005, pp. 301–302.

[11] R. Cliff, "Altera Corporation," Private Comm, 2005.

[12] P. Yiannacouras, J. G. Steffan, and J. Rose, "Improving memory systems for soft vector processors," in *WoSPS'08: Workshop on Soft Processor Systems*, 2008.