Fine-Grain Performance Scaling of Soft Vector Processors

Peter Yiannacouras, J. Gregory Steffan, and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road
Toronto, Canada
{yiannac,steffan,jayar}@eecg.utoronto.ca

ABSTRACT

Embedded systems are often implemented on FPGA devices and 25% of the time [2] include a *soft processor*—a processor built using the FPGA reprogrammable fabric. Because of their prevalence and flexibility, soft processors are compelling targets for customization—although current soft processors provide few architectural variations. Recent work has proposed augmenting soft processors with customizable vector processing support, enabling designers to easily scale performance by exploiting the data parallelism available in an application. However this approach provides only *coarse-grain* scaling, by successively doubling the number of vector datapaths for less than double the performance.

In this work we further augment soft vector processors with more fine-grain architectural modifications: we add support for (i) vector chaining and (ii) heterogeneous vector lanes, allowing the soft vector processor to be customized to not only the data-level parallelism available in an application, but to the functional unit demand. We evaluate the area and wall clock performance with full hardware implementations on state-of-the-art FPGAs and find that chaining can provide between 15-45% average performance for less area than doubling the lanes, and that heterogeneous lanes can save 6-13% area with little or no performance loss in some cases. Finally, we implement 1200 soft vector processors variants and find that the peak performance per area compared to our base vector processor can be increased by an average of 13% and up to 34% when choosing the best variant per application.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms

Measurement, Performance, Design

Keywords

Soft processor, FPGA, vector, SIMD, microarchitecture, VESPA, VIRAM, ASIP, SPREE, custom, application specific

1. INTRODUCTION

FPGAs are commonly used to implement embedded systems because of their low cost and fast time-to-market. Approximately 25% of FPGA designs contain a processor implemented in the FPGA reprogrammable fabric [2], such as the Altera Nios II or Xilinx Microblaze. These *soft processors* provide a software design environment for quickly implementing system components which do not require highly-optimized hardware implementations and can instead be implemented in a soft processor that is customized to achieve the desired performance/area/power. Current commercial soft processors are based on simple single-issue pipelines with few architectural variations, motivating research on configurable soft processor architectures that enable further customization.

While the customization of traditional *hard processors* has been thoroughly studied, the trade-offs on an FPGA substrate can be vastly different yet accurately measured—including area, clock speed, and power. As a result, several architectural axes have been recently studied in a soft processor context including: (i) single-issue in-order pipelines [17] which provide a limited design space; (ii) VLIW pipelines [11] which are limited due to port limitations on FPGA block memories; (iii) multi-threaded pipelines [6, 7, 13] and multiprocessors [14, 15] which exploit thread-level parallelism but require parallelization of the software; and (iv) vector processors [20, 21] which can scale performance by instantiating multiple *vector lanes* (the per-element datapaths of a vector processor) to exploit the data-level parallelism in an application. However, the flexibility of recently proposed soft vector processors is primarily limited to scaling the number of vector lanes by powers-of-two, to avoid division and multiplication operations in the control. For example, lane scaling provides only seven different configurations between a one-lane soft vector processor that consumes a fraction of the smallest FPGA device and a 64-lane configuration that fills one of the largest FPGA devices currently available—hence a system designer is provided with only very coarse-grain (powers-of-two) control over performance/area trade-offs when choosing an appropriate soft vector processor instantiation.

1.1 Fine-Grain Customization of Soft Vector Processors

In this work we extend soft vector processors with architectural features that allow for more fine-grain customization over current single-issue soft vector processors. Specifically we target the varying functional unit demand across applications by implementing (i) vector chaining, while parameterizing the number of vector instructions that can be simultaneously executed; and (ii) heterogeneous lanes, that parameterize the functional units that exist within individual lanes.

Vector Chaining A vector processor with support for vector chaining can begin execution of the element operations of one vector instruction before completing all the element operations of a previous vector instruction [10]. To support this simultaneous execution of multiple vector instructions, many operands must be read/written from/to the vector register file simultaneously. The conventional solution is to exploit a many-ported register file—but this design is not well-suited to an FPGA substrate since the block memories on FPGAs are normally limited to only two ports. Instead we propose to support chaining through a *banked* register file where the number of banks determines how many vector instructions can be in-flight. For some benchmarks, this results in significantly better utilization of the existing functional units and even motivates the replication of functional units in high demand.

Heterogeneous Lanes Typically the lanes in a vector processor are identical, requiring all functional units to exist even when data-parallel code uses only some of them. We introduce the ability to have a given functional unit supported in only a subset of the lanes, thus supporting heterogeneous lanes where some lanes are missing certain functional unit types. For those lanes, operations are time-multiplexed onto the lanes which do support the required functional unit. A designer can therefore create the exact number of desired functional units, similar to what would be done in a custom hardware design.

We evaluate these modifications using a full in-hardware implementation on a Stratix III FPGA executing data parallel EEMBC benchmarks. We show that with chaining we gain significant performance with more modest area cost than doubling the number of lanes. We show that heterogeneous lanes can provide area savings over homogeneous lanes with little or no performance degradation. Finally, compared to all previously possible configurations, we gain up to 34% performance-per-area after exhaustively searching the design space to minimize performance-per area on a per-application basis.

Our goal is to enable fine-grain customization of soft processors, allowing an FPGA-based embedded systems designer to use a few architectural parameters to specify a soft processor optimized to specific application and system design requirements; as a result, the amount of laborious hardware design is reduced. In the long term we envision that FPGA CAD tools will employ soft processor generators in conjunction with heuristics for automatically mapping applications to architectural configurations under a given performance/area constraint.

1.2 Related Work

Vector processor architecture has been thoroughly studied both in multi-chip supercomputers and single-chip microprocessors [3,10,12], but much less so in the FPGA context. The FPGA design flow allows us to accurately measure area, clock speed, and real in-system performance of benchmarks. Finally, the reprogrammability of an FPGA leads us to appreciate architectural features that can benefit only a few applications, whereas historically these features would be deemed failures in a general-purpose context.

Yu *et. al.* [21] recently demonstrated the potential for vector processing as a simple-to-use and scalable accelerator for soft processors. In particular, the authors show that a vector processor can scale performance better than Altera's `C2H` behavioral synthesis tool. They also propose configurable architectural options such as lane-local memories and direct support for vector reductions which both exploit FPGA-specific features. However, their design does not support vector chaining nor heterogeneous vector lanes.

Hasan *et. al.* [8] designed a floating point FPGA-based vector processor for solving sparse sets of equations. Using this application a number of targetted customizations were performed including vector chaining. Follow on work [9] considered partial reconfiguration to create custom hardware units for an application. Cho *et. al.* studied a 16-way integer SIMD processor for multimedia kernels [4] and explored the effect various levels of banking in the memory. Our work considers soft vector processor architecture more generally and thoroughly explores a large design space.

In our own previous work [20] we implemented the *VESPA* FPGA-based vector processor in dated 130nm hardware and evaluated its performance scalability up to 16 lanes across various data parallel EEMBC benchmarks. No support for vector chaining or heterogeneous lanes was considered. Some parameterization of the instruction set was explored, but this customization can be orthogonal to any architectural tuning. Similarly, the memory system was explored, but its customization can be influenced by the scalar processor which often shares the cache. Customizing the vector processor compute architecture and pipeline through chaining and heterogeneous lanes can specifically target the vectorized workload and adds another much needed dimension of customizability to soft vector processors—in this work we build on VESPA to evaluate these features. In addition, we port the VESPA system to new 65nm FPGA hardware enabling the exploration of larger soft vector processor architectures.

1.3 Contributions

This paper makes the following contributions: (i) we implement VESPA on a state-of-the-art Stratix III FPGA device while accurately measuring area, clock frequency, and cycle performance of our modifications using full EEMBC benchmarks executed from off-chip DDR2 memory; (ii) we propose and evaluate an FPGA-specific implementation of vector chaining with the required register file bandwidth facilitated exclusively via banking; (iii) we propose and investigate heterogeneous vector lanes in a soft vector processor; and (iv) we exhaustively explore a design space of 1200 VESPA configurations and show that these modifications allow for more fine-grain architectural customization as well as better performance per area.

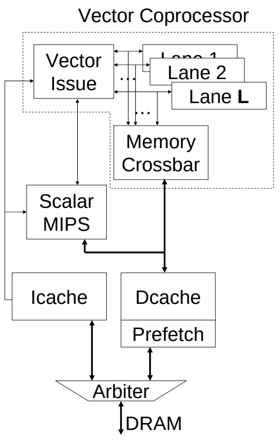

Figure 1: VESPA processor block diagram.

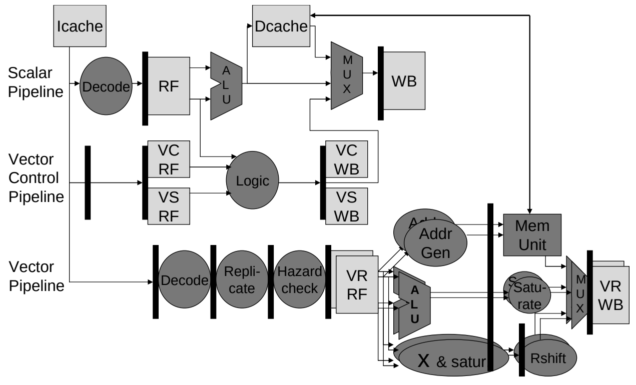

Figure 2: VESPA pipeline architecture.

2. IMPLEMENTING FINE-GRAIN CUSTOMIZATIONS

In this section we describe the parameterized base VESPA design implemented in Verilog then describe how we augment it with fine-grain customization features—namely support for both vector chaining and heterogeneous lanes.

2.1 Base VESPA Design

Figure 1 shows a block diagram of the VESPA processor that consists of a scalar MIPS-based processor automatically generated using the SPREE system [18], coupled with a parameterized vector coprocessor based on the VIRAM [12, 12] vector instruction set. Figure 2 the pipelines within VESPA. 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 with our newly added parameters marked with an asterisk. The top group of parameters configure the compute architecture, the middle group configures the instruction set architecture, and the bottom group configures the memory architecture. VESPA's architecture and parameters are as follows: 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.

Table 1: 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,...

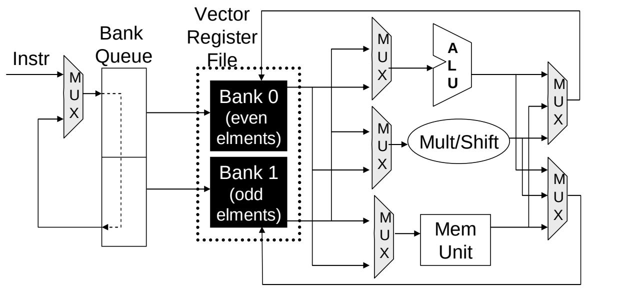

Figure 3: Vector chaining support for a 1-lane VESPA processor with 2 banks.

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 additional cycles will be required to move data between the cache lines and vector lanes. The data cache is equipped with a hardware prefetcher configured with parameters DPK and DPV. In this work we use only two prefetching configurations: off and prefetching of 8 times the current vector length (no prefetching is done for scalar instructions).

2.2 Supporting Vector Chaining

VESPA has three functional unit types, an ALU, a multiplier/shifter, and a memory unit, but only one functional unit type can be active at a time. Additional parallelism can be exploited by noting that for sufficiently long vectors, the first element-operations of one vector instruction are guaranteed to be independent of the last element-operations of another. Traditional 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 approach was shown to be more area-efficient than using many banks and few ports as in historical vector supercomputers [3]. But since FPGAs can not efficiently

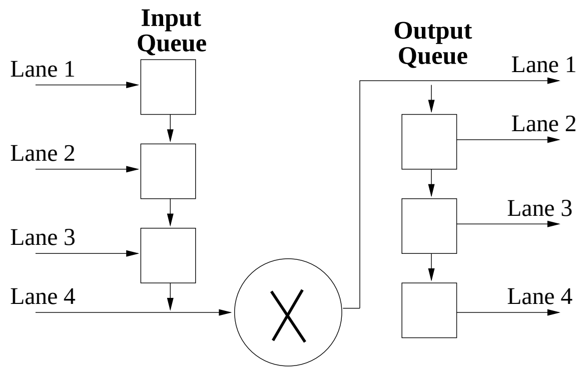

Figure 4: Heterogeneous lanes support for multipliers on a VESPA with 4 lanes and X=1.

implement a large many-ported register file, our solution is to return to this historical approach and use multiple banks each with 2 read ports and 1 write port (achieved by duplicating the register file). We partition the vector elements among the different banks allowing instructions operating on different elements to each use a register bank to feed their respective functional unit thus achieving the multiple vector instruction execution required for vector chaining.

Figure 3 shows an example of our implementation of vector chaining using two register banks to support a maximum of 2 vector instructions in flight for a 1-lane VESPA processor. Once resource, read-after-write, and bank conflicts are resolved, instructions will enter the Bank Queue and cycle between the even and odd element banks until all element operations are completed. During that time another instruction can enter the queue and rotate through the cyclical Bank Queue resulting in 2 element operations being issued per cycle. As each operation completes the result is written back to the appropriate register bank. Using a cyclical queue simplifies the control logic necessary for assigning a bank to an element operation, but causes a cycle delay for the few vector instructions which can not start on an even element (most vector instructions start with element 0).

The number of register banks B used 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 Bank Queue and multiplexing. VESPA can potentially issue as many as B instructions at a time, provided they each have available functional units. To increase the likelihood of this, VESPA allows replication of the ALU for each bank, the APB parameter enables or disables this feature. 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 VESPA's locking cache.

2.3 Supporting Heterogeneous Lanes

Increasing the vector lanes duplicates the default lane configuration, so all vector lanes are identical, or homogeneous. In this work we modify this assumption by allowing the multiplier units (also used for shifting) to appear in only some of the lanes. 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. Because of the frequency of ALU operations across the benchmarks and because of their relative size compare to the overhead, we do not support the elision of ALUs. Therefore VESPA requires that L is greater than or equal to the greater of X or M. This is a reasonable limitation since the multipliers are generally scarce, and the memory crossbar generally large, so reducing those units will have greater impact on area savings while being more likely to only mildly affect performance.

As shown in Figure 4 some area overhead is required to buffer operands and time-multiplex operations into the lanes which have the desired functional units, so the area savings from removing multipliers and shrinking the crossbar must offset this. In place of the missing functional units is a parallel-loading Input Queue which accepts input operands from all lanes with missing functional units and transfers them to the corresponding lanes that have the functional unit. The output is then loaded into an Output Queue which transfers the results back to the original lane.

3. MEASUREMENT METHODOLOGY

In this section we describe our infrastructure used for executing, verifying, and evaluating the new VESPA features. Specifically, we describe our hardware platform, processor system, verification process, CAD tool measurement methodology, benchmarks, and compiler.

Hardware Platform All processors are fully synthesized and implemented on an FPGA system. 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 currently available. We also use a 1GB DDR2-533 memory device for the storage of instructions and data in a program.

Processor System Each design consists of the VESPA soft vector processor with separate first-level direct-mapped instruction and data caches and the Altera DDR2 full-rate memory controller that connects to the DDR2 DIMM. The VESPA configurations are capable of 100-110MHz clock rates on the mid-speed 3S340C3 device. However we clock all designs at 100 MHz and the memory system at 266 MHz and then correct the wall clock time using the highest clock frequency achievable by that design on a faster 3S340C2. This allows us to model the performance of high-end FPGAs without owning them. The time dilation effects between the processor and memory from this correction generally do not affect the results significantly.

Testing All instances of VESPA are fully tested in hardware using the built-in checksum values encoded into each EEMBC benchmark. Debugging is performed using Modelsim and is guided by comparing traces of all writes to the scalar and vector register files. This trace is extracted from RTL simulation using Modelsim and compared against an analogous trace obtained from instruction-set simulation using the MINT [16] MIPS simulator augmented with the VIRAM extensions. Altera SignalTap II is used for in-hardware debugging.

Table 2: Benchmark applications.

Benchmark	Description	Source	EEMBC Suite (Dataset)	Input size (B)	Output size (B)	Largest Vector Element
AUTCOR	auto correlation	EEMBC/VIRAM	Telecom (2)	1024	64	32 bits
CONVEN	convolution encoder	EEMBC/VIRAM	Telecom (1)	517	1024	1 bit
RGBCMYK	rgb filter	EEMBC/VIRAM	Digital Ent. (5)	1628973	2171964	8 bits
RGBYIQ	rgb filter	EEMBC/VIRAM	Digital Ent. (6)	1156800	1156800	16 bits
FBITAL	bit allocation	EEMBC/VIRAM	Telecom (2)	1536	512	16 bits
VITERB	viterbi encoder	EEMBC/VIRAM	Telecom (2)	688	44	16 bits
IP_CHECKSUM	checksum	EEMBC (kernel)	Networking	40960	40	32 bits
IMGBLEND	combine two images	VIRAM	-	153600	76800	16 bits
FILT3X3	image filter	VIRAM	-	76800	76800	16 bits

FPGA CAD Tools A key value of performing FPGA-based processor research directly on an FPGA is that we can attain high quality measurements of the area consumed and the clock frequency achieved—these are provided by the FPGA CAD tools. We use aggressive timing constraints to maximize the CAD tool's effort for default optimization settings but with register retiming and register duplication enabled. Through experimentation we found that these settings provided the best area, delay, and runtime trade-off. We also performed 8 such runs for every vector configuration to average out the non-determinism in modern CAD algorithms. The relative silicon areas of each FPGA resource relative to a single Adaptive Logic Module (ALM) was supplied to us by Altera [5] for the Stratix II. We extrapolated this for Stratix III and used these equivalent areas to calculate the total silicon area consumed on the Stratix III measured in units of *equivalent ALMs*—the silicon area of a single ALM including its routing.

Benchmarks As listed in Table 2, the top six are uncompromised benchmarks from the EEMBC [1] industry-standard benchmark collection. The fifth is a kernel extracted from an EEMBC benchmark executing a hand-made data set, and the last two benchmarks were provided to us from the VIRAM group. Cycle counts are collected from a complete execution on our hardware platform.

Compilation Framework Benchmarks are built using a MIPS port of GNU `gcc` 4.2.0 with `-O3` optimization level. Experiments with this version of `gcc`'s auto-vectorization capability showed that it is in its infancy, preventing us from automatically generating vectorized code from key EEMBC program loops. Instead we use the GNU assembler ported to support VIRAM vector instructions. Hand-vectorized assembly EEMBC routines were provided to us by Kozyrakis who used them during his work on the VIRAM processor [12].

4. COARSE-GRAIN TRADE-OFFS: VECTOR LANES

In this section we show the powerful scaling afforded by soft vector processors and point out the gaps in the design space that need to be filled. Figure 5 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* (full memory crossbar, 16KB data cache with 64-byte cache lines and hardware prefetching). The figure verifies that impressive scaling still exists between 1 and 16 lanes on our state-of-the-art hardware platform. We also measure the effect of 32 lanes for the first time and notice that the performance scaling continues for benchmarks which have the available data parallelism. We see 10x average performance for 16 lanes, and 14x for 32 lanes with a peak of 24x.

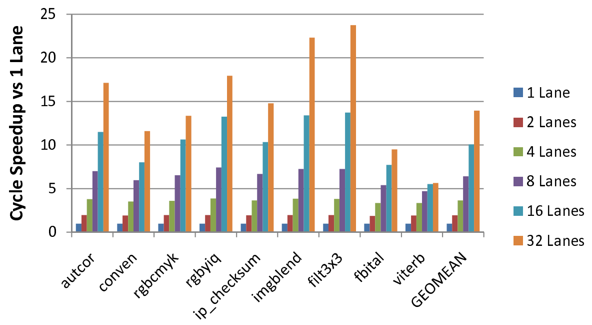

Figure 5: Performance scalability as the number of lanes are increased from 1 to 32 for a fixed VESPA architecture with *full memory support* (full memory crossbar, 16KB data cache, 64B cache line, and prefetching enabled).

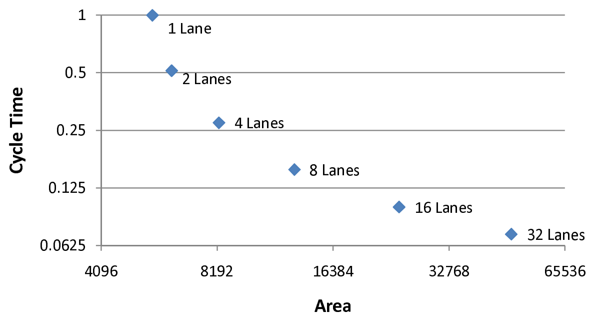

Figure 6: Performance/area design space of 1-32 lane VESPA cores with full memory support.

Figure 6 shows the area/performance space for these configurations and highlights the coarse-grain nature of using vector lanes to trade area and performance. The area cost of increasing the number of lanes can be substantial, for example growing from 8 to 16 lanes requires more than 10000 ALMs worth of silicon. While this powerful parameter allows VESPA to take leaps in the area/performance space, our new architectural parameters enable more fine-grain area/performance trade-offs as shown in the next section.

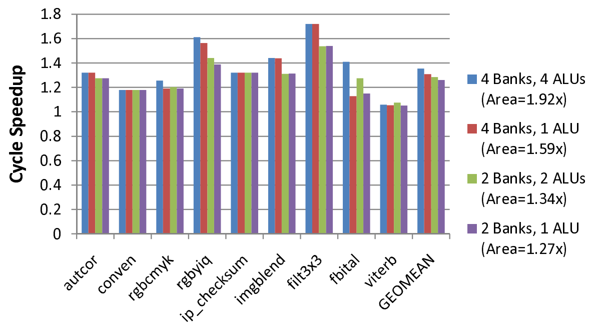

Figure 7: Cycle performance of different banking configurations across our benchmarks on an 8-lane VESPA with full memory support.

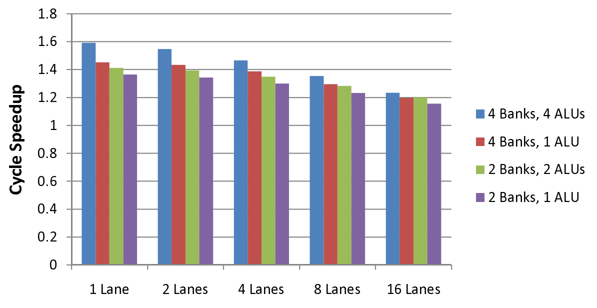

Figure 8: Cycle performance averaged across our benchmarks for different lane configuration all with full memory support.

5. FINE-GRAIN TRADE-OFFS

In this work we added parameterized vector chaining to VESPA, as well as heterogeneous lanes which allows a user to have `L` lanes of ALUs, `X` lanes of multipliers, and `M` lanes for the memory crossbar. These additions can be used to customize the vector core to the functional unit demands in an application.

5.1 Impact of Vector Chaining

We measured the effect of vector chaining via register file banking across our benchmarks using a vector processor with full memory support. 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 a single bank (no vector chaining) VESPA.

Figure 7 shows the cycle speedup of chaining across our benchmarks for an 8-lane vector processor, as well as the area normalized to the 1 bank configuration. The area cost of banking is considerable, starting at 27% for a second register bank and the expensive multiplexing logic needed between the 2 banks and functional units. The average performance improvement of this 27% area investment is 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 clearly a more fine-grain trade-off than increasing lanes.

Replicating the ALUs for each of the 2 banks (2 banks, 2 ALUs) provides some 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 greatly from increased ALUs and justifying the additional 7% area. Similarly the 4 bank configuration with no ALU replication benefits only few benchmarks, specifically RGBYIQ, IMGBLEND, FILT3X3. These benchmarks have a near equal blend of arithmetic, multiply/shifting, and memory operations and thus benefit from the additional register file bandwidth of extra banks. However the area cost of the 4th bank becomes significant at 59%. Finally, with 4 banks and 4 ALUs per lane the area of VESPA is almost doubled exceeding the cost of adding more lanes while performing slower; as a result we do not further study this inferior configuration. Though its peak performance is 4x that of the 1 bank configuration, our benchmarks and single-issue pipeline with locking cache can not exploit this peak performance.

Figure 8 shows that the speedup achieved from banking is reduced as the lanes are increased. Chaining allows multiple vector instructions to be executed if both the appropriate functional unit and register bank are available. But because only one instruction is fetched per cycle, chaining is only effective when the vector instructions are long enough to stall the vector pipeline, in other words, when the length of a vector is greater than the number of lanes. As the number of lanes increases, vector instructions are completed more quickly providing less opportunity for overlapping execution. In the slowest vector processor speedups from banking can average as high as 60% across our benchmarks, while in the fastest banking achieves only 23% speedup. The 1 lane vector processor represents a peak speedup achievable under extremely high load with long vector operations on the vector coprocessor.

The vector register file is comprised of many FPGA block RAMs. Given block RAMs with maximum width W_{BRAM} and total depth of D_{BRAM}, and using the parameters from Table 1, the number of block RAMs is equal to the greater of $L \cdot W \cdot B/W_{BRAM}$ or $32MVL \times W/D_{BRAM}$. For vector processors with many lanes, making the first expression greater, adding more banks increases the number of block RAMs used. For example increasing from 1 to 4 banks with no ALU replication on a 16 lane VESPA with `MVL`=128 adds 38% area just in block RAMs and 56% in total. On a design with many unused block RAMs this increase can be justified, moreover the added capacity of the block RAMs can be fully utilized by the vector processor with a corresponding increase in MVL.

Figure 9 shows the wall clock time versus area space of the no chaining (solid dots) configurations from 1 to 16 lanes, identical to Figure 6. We overlay two vector chaining configurations on the same figure and observe that the points with 2 banks appear about one third of the way to the next solid dot, proving that chaining can trade area/performance at finer increments than doubling lanes. Note that the 4 bank configurations are omitted since the area cost is significant and the additional performance is often modest compared to 2 banks. Since we have complete measurement capabilities of the area and performance we are able to identify that vector chaining in this case is indeed a trade-off and not a global improvement (it did not move VESPA toward the origin of the figure).

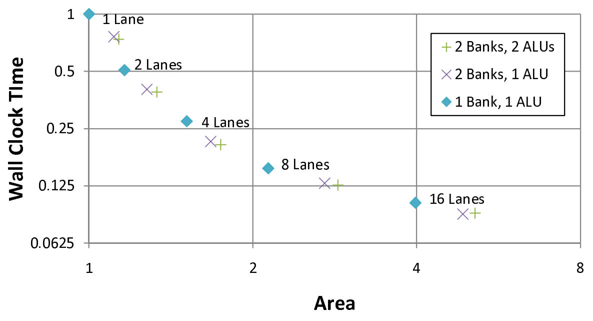

Figure 9: Performance/area space of 1-16 lane vector processors with no chaining (1 bank, 1 ALU), and 2-way chaining (2 banks, 1 ALU and 2 banks, 2 ALUs). Area and performance is normalized to the 1 lane, 1 bank, 1 ALU configuration.

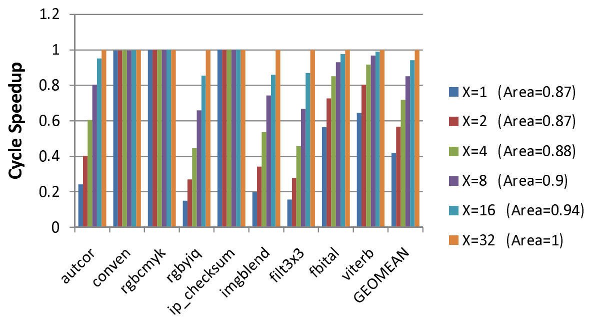

Figure 10: Performance impact of varying X for a VESPA with L=32, M=16, DW=64, DD=16K, and DPV=8VL, area and performance is normalized to the X=32 configuration.

Another option for fine-grain area/performance tradeoffs is to use lane configurations that are not powers of two, resulting in cumbersome control logic which involves multiplication and division operations. Since the control logic is often critical, and the additional area overhead significant, this approach would likely generate inferior configurations that, in terms of Figure 9, would form a curve further from the origin than the processors with lanes that are powers of two. Chaining, on the other hand is shown to directly compete with these configurations, and in Section 6 is shown to even improve performance per unit area. Note that instruction scheduling in software could further improve the performance of vector chaining, 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.

5.2 Impact of Heterogeneous Lanes

The X parameter determines the number of lanes with multiplier units. We evaluate the effect of varying X on a 32 lane VESPA processor with 16 memory crossbar lanes and a prefetching 16KB data cache with 64B line size. Each halving of X doubles the number of cycles needed to complete a vector multiply. We measure the overall cycle performance and area and normalize it to the full X=32 configuration.

Figure 10 shows that in some benchmarks such as FILT3X3 the performance degradation is dramatic, while in other

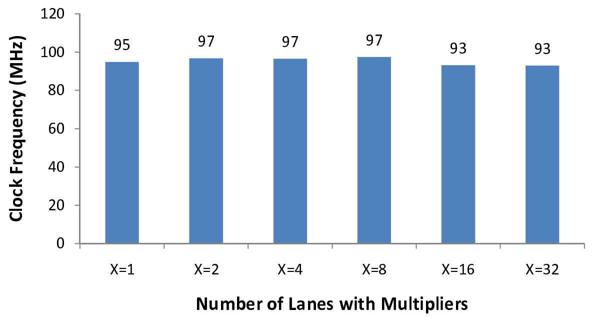

Figure 11: Clock frequency impact of varying X for a VESPA with L=32, M=16, DW=64, DD=16K, and DPV=8VL.

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 in [20], but programs with just few multiplies such as VITERB can have its multipliers reduced with very small performance penalties. The resultant saved area can then be used for other architectural features or components of the system.

The area savings from reducing the multipliers is small starting at 6% for halving the number of multipliers to 16, the savings asymptotically grow and saturate at 13%. Since the multipliers are efficiently implemented in the FPGA as a dedicated block, the contribution to the overall silicon area is small, and the additional overhead for multiplexing operations into the few lanes with multipliers ultimately cancel 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 system component could benefit from the multipliers.

Figure 11 shows the effect on clock frequency as the number of lanes with multipliers is varied. The large memory crossbar is primarily responsible for the limited clock frequency in the design. As a result removing multipliers has only a moderate impact on clock frequency. Specifically values of X between 2 and 8 achieve a 97 MHz clock over the original 93 MHz. The X=1 configuration achieves 95 MHz. All of the configurations match or improve on the clock of the full X=32 configuration. In general we do not expect the inserted logic for supporting heterogeneous lanes to degrade the clock frequency.

Overall the heterogeneous lanes mechanism provides an effective means of conserving FPGA dedicated multiplier block for applications which have low demand for them. The memory crossbar explored in [20] uses the same heterogeneous lanes mechanism to reduce the crossbar size for applications which exhibit low demand of the memory system. Together these parameters can be used to closely match the instruction mix in an application.

6. EXPANDING THE VESPA DESIGN SPACE

With the new banking and heterogeneous lanes parameters, the VESPA design space has grown significantly making it increasingly interesting to pursue customized architectures. In this section we explore that design space and quantify the additional benefit of our new parameters.

Table 3: **Explored VESPA architectural parameters.**

	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	128, 256
	Vector Lane Bit-Width	W	-
	Each Vector Instruction	-	-
Memory	ICache Depth (KB)	ID	-
	ICache Line Size (B)	IW	-
	DCache Depth (KB)	DD	8, 32
	DCache Line Size (B)	DW	16, 64
	DCache Miss Prefetch	DPK	-
	Vector Miss Prefetch	DPV	off, 8*VL

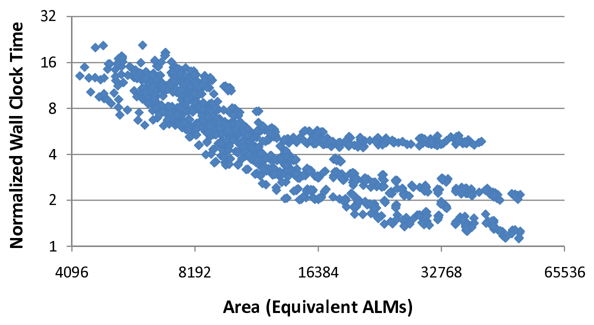

Figure 12: Average wall clock performance and area of 1200 soft vector processor variations.

We vary all combinations of the explored parameter values listed in the last column of Table 3 and implement each architectural configuration. Note that the instruction cache was not very influential so it is not explored here; we also do not perform any modifications which reduce the functionality of the vector processor (e.g. lane width reductions or instruction disabling). A total of almost 1200 designs were explored, with the new parameters expanding the number of design points by 8x.

Figure 12 shows the average performance and area of all the VESPA configurations. The design space spans a total of 15x in area and 19x in performance providing a wide range of design points for an FPGA embedded system designer to choose from. Moreover, VESPA now provides more fine-grain coverage of this design space. The pareto optimal points in the figure approximate a straight line providing steady performance returns on area investments until the high-area designs begin suffering from significant clock frequency decay. Additional retiming or pipelining can mitigate this decay. Currently clock frequencies range from 134 MHz to 91 MHz. The maximum area is over 50000 equivalent ALMs, which accounts for more than one-quarter of the silicon area of the device. In terms of just logic implemented in Stratix III ALMs, the largest design consumes more than one-third of the available ALMs in the device.

As area is increased, two branches emerge: the topmost (slowest) being the designs throttled by a small 16B cache line size, and the middle branch throttled by cache misses without prefetching. With both larger cache lines and prefetching enabled the fastest and largest designs in the bottom branch can trade area for performance competitive with the smaller designs. The two top branches are examples of inferior VESPA configurations. In other words, these configurations are dominated in both area and speed by other VESPA configurations. We are therefore motivated to prune some of these configurations and explore more VESPA parameter values. Specifically we exclude configurations with:

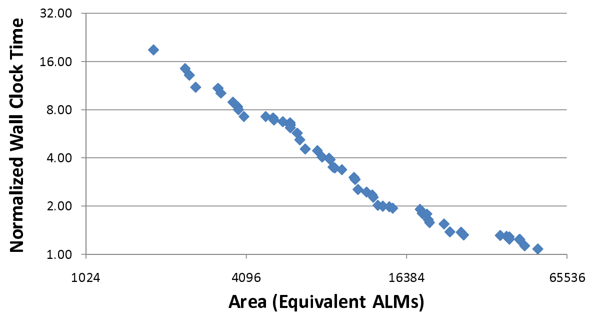

Figure 13: Average wall clock performance and area of pareto-optimal soft vector processor configurations from 768 pruned and expanded design space.

1. (`L` $<$ 8) and (`MVL` $=$ 512) – Configurations with few lanes can seldomly justify the area for such large amount of vector state.
2. (`L` $>=$ 8) and (`DW` $= 16B$) – Configurations with many lanes require wider cache lines as seen in Figure 12.
3. (`L` $>=$ 8) and (`DPV` $=$ 0) – Configurations with many lanes require prefetching as seen in Figure 12.
4. (`DD` $= 8KB$) and (`DW` $= 64B$) – Configurations which do not fully utilize their block RAMs waste area and are matched or dominated by configurations which do.
5. (`DD` $= 32KB$) and (`DW` $= 16B$) – Configurations with extra block RAMs used only to expand the cache depth which was shown to be ineffective in accelerating benchmark performance [19].

We also add the values `DPV`=7 and `MVL`=4*L to the exploration. With these new values and the above exclusions, the design space is reduced to 768 configurations. We can further reduce this design space to a much smaller set of pareto-optimal configurations which dominate all other configurations on average across this benchmark set. Note that certain configurations can benefit a specific application without being pareto-optimal on average.

Figure 13 shows the wall clock performance and area design space of the pareto-optimal VESPA configurations after pruning and adding the new `MVL` and `DPV` values. The `MVL` value of 4*L instructs VESPA to implement a maximum vector length that is four times the number of lanes. Typically the maximum vector length is large to enable long vectors, but for a single lane configuration this only adds significant area overhead without significant performance. The 4*L value allows for much smaller configurations to be explored. The smallest pareto-optimal configuration is 1838 equivalent ALMs which is signficantly smaller than any of the previously explored configurations in Figure 12.

The pareto-optimal points span an area of 28x and wall clock times of 18x. This range of areas is much larger than the 15x seen previously because of the reason described above. The performance range however has been reduced slightly from 19x to 18x. This reduction is due to the eliminated low-performance configurations which are dominated by the faster pareto-optimal configurations. The peak performance is the same in both cases, but the worst-case performance for the pareto-optimal points is less than for the full 1200 point design space.

6.1 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 more quickly and more easily. We therefore analyze the configurations from the original 1200 point design space which achieve the best performance-per-area on a per-application basis. Using this analysis we quantify the benefit of selecting a custom VESPA configuration over the best overall configuration.

Table 4 shows the VESPA configuration with the best performance-per-area for each benchmark selected out of the 1200 explored designs. Surprisingly, all chosen configurations have 2 banks, and FBITAL has the ALU per bank parameter enabled. Four of the chosen configurations also make use of the `X` parameter to instantiate half as many multipliers as lanes. The second-last column shows how much the performance-per-area has improved because of these new parameters compared to choosing the best configuration without the new parameters. We achieve up to 34% (average of 13%) better performance-per-area compared to the VESPA configurations without chaining and heterogeneous lanes.

The selected configurations for each benchmark vary significantly. For example `fbital` achieves a peak performance-per-area with a small 4 lane VESPA with aggressive chaining and small cache. Most benchmarks benefit from advanced memory systems, but none benefit from more than 16 lanes due to the clock frequency decay discussed earlier. The table also shows the best overall configuration across all benchmarks, which we refer to as the *general purpose* or GP configuration and is shown in the last row of the table. The general purpose VESPA uses chaining through 2 register banks and as a result increases its performance-per-area by 14% over the general purpose VESPA before adding these parameters. The last column shows the performance-per-area benefit of choosing a custom VESPA configuration for each benchmark compared to using the general purpose VESPA. On average a 12% performance-per-area benefit is seen, with a peak of 44%, motivating research into automatically selecting a configuration for a given application. Note that processors with similar performance-per-area can have drastically different performance and area profiles, amplifying the variation.

With the exception of VITERB the benchmarks are characteristically similar: streaming-oriented across a large data set with little data re-use. With greater benchmark diversity we expect the improvements in selecting a per-application configuration to be significantly higher. Nonetheless, these improvements highlight the value in matching the soft vector processor architecture to the application.

7. CONCLUSIONS

We implemented VESPA in hardware using the Stratix III FPGA on a Terasic DE3 board which is equipped with a DDR2 DIMM. We verified the increased scalability of VESPA across a set of benchmarks mostly from the industry standard EEMBC suite, and noted the coarse-grain area/performance trade offs within it.

We proposed an FPGA-specific implementation of vector chaining facilitated exclusively through register file banks since FPGA block RAMs have only two access ports. We parameterized the number of banks in the architecture, as well as allowed duplication of the ALU for each bank and observed more fine-grain trade offs could be achieved. Performance gains averaged between 15% and 45% with area costs significantly less than doubling lanes. We further augmented VESPA with the ability to implement heterogeneous lanes, allowing separate specification of the number of lanes with ALUs, multipliers, and memory units. This provided further fine-grain control over the architecture allowing each functional unit to exactly meet its demands. Between 6-13% area can be saved compared to homogeneous lanes, in some benchmarks, with little or no performance degradation.

We observed that our modified VESPA spans an enormous 28x space in area, and 18x in performance. The addition of our new parameters results in up to 34% (average 13%) increased peak performance per area achievable on a per-application basis over the previous VESPA. Our new architectural parameters further motivate the customization of soft processors to their application providing up to 44% (average 12%) better performance per area for choosing a unique configuration for each benchmark versus selecting one configuration for all benchmarks. This motivates further design space expansion and eventual inclusion of algorithms to map applications to processor configurations in FPGA CAD tools.

8. REFERENCES

[1] The Embedded Microprocessor Benchmark Consortium. http://www.eembc.org.

[2] T. Allen. Altera Corporation. Private Communication, 2009.

[3] K. Asanovic. *Vector Microprocessors*. PhD thesis, University of California-Berkeley, 1998.

[4] J. Cho, H. Chang, and W. Sung. An fpga based simd processor with a vector memory unit. *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4 pp.–, 21-24 May 2006.

[5] R. Cliff. Altera Corporation. Private Communication, 2005.

[6] 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.

[7] B. Fort, D. Capalija, Z. G. Vranesic, and S. D. Brown. A multithreaded soft processor for sopc area reduction. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 131–142, Washington, DC, USA, 2006.

[8] M. Hasan and S. Ziavras. Fpga-based vector processing for solving sparse sets of equations. In

Table 4: Configurations with best performance-per-area for each benchmark

Benchmark	VESPA Configuration									Performance per Area	
	DD	DW	DPV	APB	B	MVL	L	M	X	vs old VESPA	vs GP
AUTCOR	8KB	16B	0	0	2	7	8	8	8	1.03	1.19
CONVEN	32KB	64B	8VL	0	2	7	8	8	4	1.07	1.05
RGBCMYK	32KB	64B	8VL	0	2	7	8	8	4	1.10	1.05
RGBYIQ	32KB	64B	8VL	0	2	8	16	16	16	1.21	1.00
IP_CHECKSUM	32KB	64B	8VL	0	2	7	8	8	4	1.21	1.05
IMGBLEND	32KB	64B	8VL	0	2	8	16	16	16	1.14	1.06
FILT3X3	32KB	64B	8VL	0	2	8	16	16	16	1.34	1.11
FBITAL	8KB	16B	0	1	2	7	4	4	4	1.05	1.44
VITERB	32KB	64B	8VL	0	2	7	4	4	2	1.01	1.18
									GEOMEAN	1.13	1.12
General Purpose (GP)	32768	64	8VL	0	2	7	8	8	8	1.14	1

Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on, pages 331–332, April 2005.

[9] M. Z. Hasan and S. G. Ziavras. Runtime partial reconfiguration for embedded vector processors. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 983–988, April 2007.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture; A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[11] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster. An fpga-based vliw processor with custom hardware execution. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 107–117, New York, NY, USA, 2005. ACM.

[12] C. Kozyrakis and D. Patterson. Scalable, vector processors for embedded systems. *Micro, IEEE*, 23(6):36–45, 2003.

[13] M. Labrecque, P. Yiannacouras, and J. G. Steffan. Scaling Soft Processor Systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'08).*, Palo Alto, CA, April 2008.

[14] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer. An fpga-based soft multiprocessor system for ipv4 packet forwarding. pages 487–492, Aug. 2005.

[15] D. Unnikrishnan, J. Zhao, and R. Tessier. Application-Specific Customization and Scalability of Soft Multiprocessors. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'09).*, Napa, CA, April 2009.

[16] J. E. Veenstra and R. J. Fowler. MINT: a front end for efficient simulation of shared-memory multiprocessors. In *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '94).*, pages 201–207, Durham, NC, January 1994.

[17] P. Yiannacouras, J. Rose, and J. G. Steffan. The Microarchitecture of FPGA Based Soft Processors. In *CASES'05: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 202–212. ACM Press, 2005.

[18] 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*, pages 201–210, New York, NY, USA, 2006. ACM Press.

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

[20] 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.

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