Advanced SIMD: Extending the Reach of Contemporary SIMD Architectures

Matthias Boettcher and Bashir M. Al-Hashimi
University of Southampton
Southampton, UK

Mbou Eyole, Giacomo Gabrielli and Alastair Reid
ARM Ltd.
Cambridge, UK

***Abstract*—SIMD extensions have gained widespread acceptance in modern microprocessors as a way to exploit data-level parallelism in general-purpose cores. Popular SIMD architectures (e.g. Intel SSE/AVX) have evolved by adding support for wider registers and datapaths, and advanced features like indexed memory accesses, per-lane predication and inter-lane instructions, at the cost of additional silicon area and design complexity.**

This paper evaluates the performance impact of such advanced features on a set of workloads considered hard to vectorize for traditional SIMD architectures. Their sensitivity to the most relevant design parameters (e.g. register/datapath width and L1 data cache configuration) is quantified and discussed.

We developed an ARMv7 NEON based ISA extension (ARGON), augmented a cycle accurate simulation framework for it, and derived a set of benchmarks from the Berkeley dwarfs. Our analyses demonstrate how ARGON can, depending on the structure of an algorithm, achieve speedups of 1.5x to 16x.

I. Introduction

Single Instruction, Multiple Data (SIMD) instruction set extensions are the norm in modern general-purpose microprocessors. They allow programmers and compilers to exploit data-level parallelism by relying on narrow hardware vectors, without resorting to external accelerators such as GPGPUs [1]. In recent years, chip vendors have pushed the boundaries of the SIMD paradigm by extending the range of supported data types, widening registers and datapaths, and introducing features like per-lane predication and indexed memory accesses that were previously exclusive to high performance computers [2]. While such features extend the applicability of SIMD to new domains, they impose a considerable cost in terms of silicon area, design complexity and power consumption.

This paper focuses on the analysis of advanced SIMD features with regards to their performance impact on a set of workloads that are: a) relevant for current and future applications from different domains, and b) hard to vectorize with traditional SIMD architectures, due to irregular computation and memory access patterns. To perform a quantitative analysis, we developed a benchmark suite inspired by the Berkeley dwarfs [3] (Sec. IV), and an experimental ISA (ARGON) derived by extending ARMv7 NEON with additional SIMD features and the ability to target different vector widths. Our key contributions are:

- A detailed analysis quantifying performance gains achievable by advanced SIMD features.
- A set of recommendations for future SIMD extensions to increase the vectorizabilty and datapath utilization

The following sections discuss related work (Sec. II), advanced SIMD features of interest (Sec. III), the analyzed benchmarks (Sec. IV), our evaluation methodology (Sec. V) and results (Sec. VI). Our key observations and conclusions are presented in Sec. VII.

II. Related Work

Several authors have investigated the limitations of SIMD and proposed techniques to speedup workloads which cannot be easily vectorized. Govindaraju et al. [4] analyze some of these restrictions and propose a more flexible accelerator and an augmented compiler to perform the transformations necessary for an efficient mapping. However, their approach is unable to tackle quintessentially irregular algorithms like sparse matrix-vector multiplication. The problem of endowing a compiler with sufficient intelligence in order to successfully map arbitrary programs to SIMD has been well-studied [5] and some more advanced approaches even rely on machine-learning [6]. However, it is generally accepted that such autovectorisation often leads to performance below that obtainable via manual optimization. Some SIMD extensions reduce programmer effort and improve scalability when approaching difficult algorithms by providing more flexible primitives such as indexed memory accesses [2]. Furthermore, several researchers have advocated the use of primitives performing inter-lane operations such as scans [7].

Problems emerging from a lack of regularity in computation patterns and memory accesses are the bane not just of in-core accelerators but also of other data-level parallel structures. The high latency associated with data transfers and the large energy cost of underutilization often favor CPU-centric approaches. While some degree of non-linearity in memory accesses is tolerable in GPUs due to the inherent decoupling of processing from memory accesses [8], branch divergence within warps leads to low utilization and poor efficiency. One solution proposed by Sengupta [9] involves tracking concurrency by partitioning data into segments. Unfortunately, many approaches still suffer from the fact that the primitives they introduce are not general or expressive enough to be readily adopted by programmers [10]. Lee et al. [1] challenge this notion by performing an extensive design space exploration to find a design point which can be highly efficient and yet provide easier programmability. However, these authors neglect the subword-SIMD pattern considered in this paper which can be potentially more efficient by unifying otherwise separate decode, dispatch, and register read operations within a CPU. It also leverages efficiency gains in the design of scalar datapaths when performing control-flow dominated tasks.

III. Advanced Vector Features

Key challenges for the vectorization of general purpose code are irregular computation patterns and data dependencies. We increase NEON's applicability by extending it with:

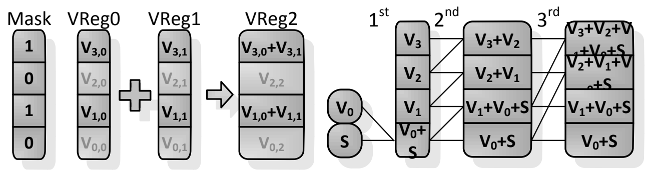
A **Vector Length Register (VL)**, to allow the deliberate underutilization of a SIMD datapath for loops that exhibit iteration counts indivisible by the hardware vector length, without the need for scalar fix-up operations.

A **Mask Register (VM)**, to enable per lane predication (Fig. 1a) and permutations of elements originating from two sources. Both use cases combine control- and data-plane computations to vectorize conditional statements such as $C[i] = (A[i] > B[i])\ ?\ A[i] : B[i];$.

Indexed Memory Accesses (Gather/Scatter), to allow the vectorization of data-dependent memory accesses. We use a scalar base address and a vector of 32-bit offsets independent from the underlying data type. This requires multiple instructions to service all elements of a vector register, but smaller offsets would limit the addressable memory region.

Scans, to perform cumulative operations over sequences of vector elements to identify specific elements within a data set (e.g. the minimum), or reduce multiple elements to a single outcome (e.g. sum). Fig. 1b illustrates the data-flow underlying the timing profile used as baseline for scans in Sec. VI. It respects inter-lane dependencies when computing up to N elements in $log_2N + 1$ cycles. This performance advantage over a serialized alternative is desirable; however, the energy consumed by the increased number of operations is concerning. Furthermore, as most floating point (FP) operations are not associative, their accuracy may be affected.

Segmented Scans, to allow arbitrary length segments within vectors to be processed in parallel. While VL allows scans to operate on a reduced number of adjacent elements, segmented scans can improve datapath utilization by effectively performing multiple independent scans within a single vector.



(a) Per Lane Predication using VM (b) Vector Scan Add

Fig. 1. Use Case for VM, and Timing of $log_2N + 1$ Scan Implementation

IV. Analyzed Benchmarks

ARGON attempts to widen the scope of traditionally multimedia and DSP focused SIMD ISAs. The benchmarks developed here are inspired by the Berkeley Dwarfs [3], a set of algorithmic methods capturing computation and communication patterns representative of future applications.

While the highest optimization level is employed to compile all analyzed implementations, the vectorized portions are hand-coded to leverage the newly implemented features. We refrain from using default library code when this would result in discrepancies between scalar and vector variants. Similarly we avoid system calls to minimize OS interactions and other system-level effects. Additional implementations to those introduced in Tab. I were investigated, but are not further discussed because they exhibited inferior performance or behaviors very similar to those presented.

V. Evaluation Methodology

The toolchain in Fig. 2 includes i) a unified database to represent ARGON instructions, ii) source files for intrinsics and equivalent C functions to allow functional verifications, and iii) a custom LLVM front-end, GNU Assembler and gem5 framework. Most components are automatically generated and parameterized to reduce turnover times and potential error sources. gem5 is augmented to support the features described in Sec. III, and its cache model extended with banks, ports, sub-blocks, and an accurate contention model. The latencies

TABLE I
Analyzed Benchmarks, corresponding Working Sets and Implementations

	Benchmark Name and Description	Working Set Params.	Implementation Specific Considerations		Utilized Features
AESEnc	Advanced Encryption Standard as established by NTIS in 2001 [11]; very computation-intensive; primarily focused on combinational logic; exhibits nonlinearly dispersed memory accesses	CTR (counter) mode; 128-bit keys	Scalar & ARGON	32-bit based version combining multiple round transformation steps in a single set of table lookups	indexed mem. acc.; VL (to a minor degree)
			NEON	Lack of support for indexed loads favors 8-bit based SIMD version	
BackProp	Backpropagation; used to train neural networks [12]; Parameters outside the default working set, did not significantly affect the observed performances, except for cases where the number of elements per layer exactly matched the vector width; these improved the results observed for the ARGON variant	3 input and 2 output nodes; 1 hidden layer including 5 neurons; 8 training sets; randomly initialized weights	Scalar	—	indexed mem. acc.; VL; VM; scans / segmented scans
			NEON	Lack of advanced features makes it not worthwhile	
			ARGON	Based on scans and VL; processes one neuron at a time	
			ARGON SegScan	Segmented scans to process multiple neurons in parallel; initialization phase to precompute indexes and segmentation masks	
BitAlloc	Bit Allocation; elemental step in compression and optimization algorithms; analyzed variant models the allocation of bits to carriers within a DSL transmission channel based on their SNR (16-bit integers)	256 carriers equivalent to working set provided by EEMBC TeleBench v1.1	Scalar	—	VM; scans; VL (to a minor degree)
			NEON	Lack of VM allows only partial vectorization	
			ARGON	Employs VM to remove control dependencies; almost fully vectorized	
PathFind	Dijkstra's Algorithm; fastest single-source shortest path algorithm for arbitrary directed graphs with unbounded non-negative weights [13]; inherently scalar nature limits vectorizability	New York City road map [14]	Scalar	Binary heap stored as consecutive array to simplify address calculations and speed up memory accesses	indexed mem. acc.; VL; VM; scans
			NEON	Lack of advanced features makes it not worthwhile	
			ARGON	d-heap; d equal to number of 32-bit elem. per vector	
SpMV	Sparse matrix-vector multiplication; integral building block for a variety of science and engineering applications; generally involves matrices with a large number of zero elements stored in a compressed format; analyses based on matrix sizes from 2*2 to 1000*1000 elements show that SegScan M1 consistently outperforms basic ARGON variant	randomly generated matrix in Yale format; 100*100 elements; 15% density	Scalar	—	indexed mem. acc.; VL; VM; scans / segmented scans
			NEON	Mimics indexed mem. accs. by using scalar loads	
			ARGON	Based on scans and VL; processes one row at a time	
			ARGON SegScan	Segmented scans to process multiple rows in parallel; segmentation masks computed within every iteration	
			ARGON SegScan M1	Assumes at least one non-zero element per row; avoids conditional branch used for scalar fix-up	

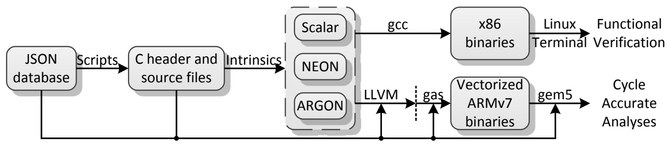


Fig. 2. Block Diagram of the ARGON Evaluation Toolchain

TABLE II
SIMULATION PARAMETERS OF THE BASELINE CONFIGURATION

Component	Parameter
Processor	single-core, OoO, 1 GHz, 40 ROB entries, 3 elem. fetch/decode/rename, 6 elem. dispatch, 8 elem. issue, LogNScans (Sec. VI-C), 256-bit SIMD
L1D cache	32 KByte, 2 cycle latency, 64 byte lines, 2-way set-assoc., 128-bit sub-blocks per line, m_2B_1P configuration (Sec. VI-D)
L2 cache	1 MByte, 12 cycle latency, 16-way set-assoc.
DRAM	512 MByte, 30 cycle latency

of vector FUs are parameterizable by a set of timing profiles (Sec. VI-C). The baseline configuration used for the following analyses, represents a high-end ARM A-class core and the corresponding memory hierarchy (Tab. II).

VI. EVALUATION RESULTS

A. Baseline Configuration

Fig. 3 shows speedups for the baseline configurations of all implementations introduced in Sec. IV relative to their scalar counterparts. Labels along the x-axis and colors identify specific benchmarks and implementations. The NEON variants of AESEnc, BitAlloc and SpMV show speedups of 0.8x, 1.3x and 0.9x, respectively. The first is limited by the lack of indexed memory accesses, which favors an 8- instead of a faster 32-bit version of the algorithm (Sec. IV). The second is only partially vectorizable due to the absence of per lane predication, and the third relies on scalar fix-up operations to gather/scatter vector elements and emulate scans.

ARGON implementations consistently outperform both, scalar and NEON variants. Utilizing VM to remove data dependencies from BitAlloc allows a high degree of vectorization yielding a speedup of 13.5x, which approaches the theoretical maximum of 16x for 16-bit operations on a 256-bit datapath. Although AESEnc exhibits a similar degree of vectorization, it is held back by its nonlinearly dispersed memory accesses, as indicated by the fact that more than 50% of its read requests are stalled due to an insufficient number of cache ports/banks. Furthermore, as we considered instructions like element-wise rotation as too algorithm specific, we rely on a slower combination of two shifts and one OR instead.

The ARGON versions of BackProp and SpMV are fully vectorized, too. However, with an average of only 3.5 and 6.0 elements active, respectively, they underutilize the available datapath. Employing segmented scans (SegScan) increases the achieved speedups from 1.3x to 2.1x and from 2.1x to 2.3x, respectively. Moreover, assuming a minimum of one element

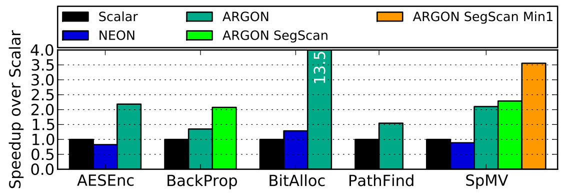


Fig. 3. Speedup over Scalar Implementations for Baseline Configuration

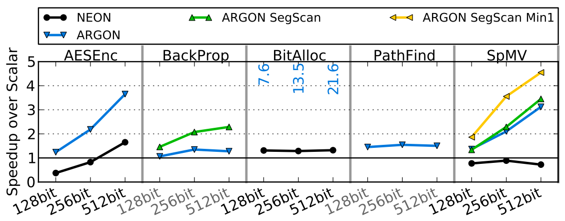


Fig. 4. Speedup over Scalar Impl. for different Datapath Configurations

in each row of SpMV (Min1), allows the elimination of a conditional branch (Sec. IV), yielding a speedup of 3.6x.

The datapath utilization of the inherently scalar PathFind algorithm depends on the underlying heap size; e.g. a complete 8 element vector within the decreaseKey() function requires $\sum_{i=0}^{7}(8^i) = 7,907,396$ nodes ($1 + 8 = 9$ for extractMin()). Fig. 3 shows that the vectorized version of this function still increases the overall performance of the algorithm slightly.

B. Dependency on the Datapath Configuration

The simulation results obtained for a series of different datapath widths (Fig. 4) exhibit two distinct behaviors; i.e. performances either independent of, or proportional to the underlying datapath width. The former indicates a low degree of vectorization, or a frequently underutilized datapath. In particular, due to the limited number of network nodes, BackProp ARGON and ARGON SegScan saturate at 4.0 and 6.3 active elements, respectively. This also explains the poor performance of the NEON variants for BitAlloc and SpMV. The lack of VL and VM requires them to regularly fall back to scalar fix-up operations. Certain ARGON implementations exhibit linear gains from wider vectors. However, the energy cost and hardware complexity incurred by wider datapaths potentially outweigh the benefits of higher performance gains.

C. Dependency on Functional Unit Timings

Fig. 5 illustrates the effects of different timing profiles. The baseline (LogNScans) processes scans in $log_2N + 1$ cycles (Sec. III). However, certain algorithms may require FP operations to be executed in-order. The SerialScan profile shows that completely serialized scans impose a significant performance penalty particularly for SpMV, which iterates over a tight loop that is highly dependent on instruction latencies. Nevertheless, as all implementations still outperform their scalar counterparts, even high performance processors might prefer serial scans to avoid FP associativity issues and reduce energy costs by utilizing simpler functional units. Note that we did not consider VL when determining scan latencies; hence, they might be smaller if not all lanes are active.

The "Unpacked" profile estimates the contribution of CPU cycles required at the begin and end of each vector instruction

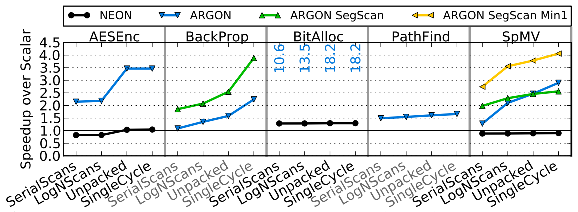


Fig. 5. Speedup over Scalar Implementations for different Timing Profiles

to route elements between their packed representation inside registers to individual datapath lanes. A comparison to the SingleCycle profile, which assumes one cycle latencies for all vector operations, shows that the integer based algorithms AESEnc and BitAlloc actually reach their theoretical peak performance. However, BackProp and SpMV are still limited by long latency FP operations.

D. Dependency on the Memory Model

To evaluate the impact of the underlying memory on the observed speedups we investigated all combinations of the following L1D cache parameters and identified the five distinct groups of configurations represented in Fig. 6.

- 1, 2, 4, 8 or 16 banks (1B, 2B, ...)
- 1 rd/wt (1P) or 1 rd/wt & 1 rd port per bank (2P)
- allowing only one access per cache line or merging accesses to the same 128-bit sub-block (m)

It can be observed that for these benchmarks multiple ports are more beneficial than multiple banks. In particular, the speedup of SpMV improves from 1.8x to 3.0x instead of 2.7x, when comparing 2B_1P against the 1B_2P setup. However, as additional ports dramatically increase cache energy consumption and access latency, contrarily to banking which effectively reduces those parameters, we decided to use an m_2B_1P implementation as baseline for our analyses. The prefix "m_" indicates the ability to merge accesses to the same 128-bit sub-block, as it is common practice in high end vector processors.

The full performance impact of memory accesses on particular implementations can be observed when comparing the baseline against the 1B_1P configuration. For instance, it reveals that the ARGON variants of BackProp and SpMV are computation-bound, whereas the corresponding SegScan implementations are memory-bound. For SpMV this actually allows ARGON to outperform ARGON SegScan. It is noteworthy that even for the 1B_1P model all ARGON variants outperform their scalar counterparts. This implies that gains due to an increased degree of vectorizability would even benefit low end systems, which might implement serialized versions of indexed memory accesses.

VII. Observations and Conclusions

There are several lessons to be learned from this study. First, the combination of VL, VM and indexed memory accesses can yield significant performance gains on high-end CPUs, as well as on low-end systems which serialize scans and implement simple caches. In a heterogeneous environment such as big.LITTLE^{TM}, this greatly benefits the bigger core without diminishing its companion's performance/energy trade-off. Second, given a sufficiently high number of elements to operate on, wider datapaths can yield substantial speedups. Conversely, low-end systems may employ narrow datapaths to improve energy efficiency while maintaining moderate performance gains. Given current technology, the best trade-off seems to be a SIMD width of 256-bit.

Furthermore, our results indicate that merging L1D cache accesses to the same 128-bit sub-block yields higher gains than having additional ports on the same number of banks. From a performance/energy perspective, this suggests the use of multiple single-ported banks that do support merging. Another interesting observation is that segmented scans enable algorithms to achieve higher datapath utilizations by collapsing nested loops. However, their computational overhead favors data sets that allow operations on at least two segments concurrently. Limiting ARGON to a single VL and VM to save encoding space has an adverse impact on the density and performance of vectorized code. In particular, tight loops that have multiple assignments of VL/VM suffer from latencies and dependencies introduced by frequent updates of said registers. As a compromise, we suggest an additional encoding bit to activate/deactivate VL/VM for individual instructions. Finally, we show that pipeline latencies incurred by the packing/unpacking of vector elements diminish achievable speedups. Based on this, we suggest a hybrid packing scheme optimized for 32-bit elements; e.g. a 256-bit wide vector register comprising up to four 64-bit or eight 32/16/8-bit elements.

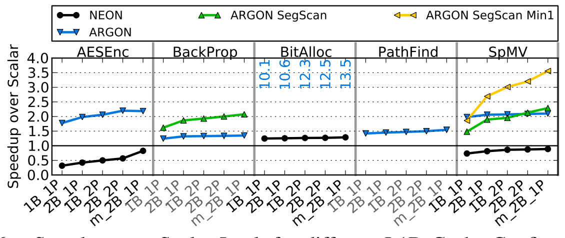


Fig. 6. Speedup over Scalar Impl. for different L1D Cache Configuration

References

[1] V. W. Lee *et al.*, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *ISCA'10 Proc.*, 2010.

[2] "Intel Architecture Instruction Set Extensions Programming Reference," Intel Corporation, Tech. Rep. July, 2013.

[3] K. Asanovic, R. Bodik, and B. Catanzaro, "The landscape of parallel computing research: A view from berkeley," EECS Departmentm, University of California, Berkeley, Tech. Rep., 2006.

[4] V. Govindaraju, T. Nowatzki, and K. Sankaralingam, "Breaking SIMD shackles with an exposed flexible microarchitecture and the access execute PDG," in *PACT '13 Proc.*, 2013.

[5] D. Nuzman and R. Henderson, "Multi-platform Auto-vectorization," in *CGO '06 Proc.*, 2006.

[6] K. Stock, L.-N. Pouchet, and P. Sadayappan, "Using machine learning to improve automatic vectorization," *ACM Trans. Arch. Code Opt.*, vol. 8, no. 4, Jan. 2012.

[7] G. Blelloch, S. Chatterjee, J. C. Hardwick, J. Sipelstein, and M. Zagha, "Implementation of a portable nested data-parallel language," *J. Parallel Distrib. Comput.*, vol. 21, no. 1, Apr. 1994.

[8] "NVIDIAs Next Generation CUDA Compute Architecture: Fermi," NVIDIA, Tech. Rep., 2013.

[9] S. Sengupta, "Efficient Primitives and Algorithms for Many-core architectures," Ph.D. dissertation, University of California, 2010.

[10] G. Caragea, F. Keceli, A. Tzannes, and U. Vishkin, "General-purpose vs. GPU: Comparison of many-cores on irregular workloads," in *HotPar Proc.*, 2010.

[11] National Technical Information Service (NTIS), "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," Tech. Rep., 2001.

[12] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, 1986.

[13] E. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, 1959.

[14] C. for Discrete Mathematics & Theoretical Computer Science. 9th DIMACS Implementation Challenge - Shortest Paths.