The Journal Of Supercomputing manuscript No.
(will be inserted by the editor)

Scalability Analysis of AVX-512 Extensions

Juan M. Cebrian
Lasse Natvig · Magnus Jahre

Received: date / Accepted: date

Abstract Energy efficiency below a specific thermal design power (TDP) has become the main design goal for microprocessors across all market segments. Optimizing the usage of the available transistors within the TDP is a pending topic. Parallelism is the basic foundation for achieving the Exascale level. While instruction-level (ILP) and thread-level parallelism (TLP) are embraced by developers, data-level parallelism (DLP) is usually underutilized, despite its huge potential (e.g., Single-Instruction Multiple-Data (SIMD) execution). Companies are pushing the size of vector registers to double every four years. Intel's AVX-512 (512-bit registers) and ARM's SVE (up to 2048-bit registers) are examples of such trend.

In this paper we perform a scalability and energy efficiency analysis of AVX-512 using the ParVec benchmark suite. ParVec is extended to add support for AVX-512 as well as the newest versions of the GCC compiler. We use Intel's Top-Down model to show the main bottlenecks of the architecture for each studied benchmark. Results show that the performance and energy improvements depend greatly on the fraction of code that can be vectorized. Energy improvements over scalar codes in a single thread environment range from 2x for Streamcluster (worst) to 8x for Blackscholes (best).

Keywords Benchmarking · Vector · Efficiency · SIMD

Juan M. Cebrian
E-mail: jcebrian@ditec.umu.es

Lasse Natvig
E-mail: lasse.natvig@ntnu.no

Magnus Jahre
E-mail: magnus.jahre@ntnu.no

1 Introduction

Moore's law predicted in 1965 that the number of transistors will double every two years. This trend has slowed down since the introduction of 22nm technologies, being closer to 30 months nowadays. Nevertheless, providing power to all these transistors is a different story. Dennard scaling stated that the voltage and current required to operate a transistor should scale proportionally to its linear dimensions [12]. However, when process technologies go below 22nm, the benefit from voltage scaling becomes sublinear. This makes power and temperature a prime constraint on future many-core designs [7]. The International Technology Roadmap for Semiconductors report reflects this slowdown in voltage scaling [19]. Optimizing the usage of the available transistors under power and temperature constraints is an open critical issue.

Computer architects focus nowadays on parallelism to improve performance under a specific TDP. While instruction-level (ILP) and thread-level parallelism (TLP) have been extensively studied, data-level parallelism (DLP) is usually underutilized in CPUs, despite its huge potential. Vector architectures are designed to extract DLP by operating over several input elements with a single instruction [4,13]. Vector supercomputers have been present since the 1970s [26,11,13,5,30]. Originally, vector architectures exploited DLP with long vectors of thousands of bits. Nowadays, hardware manufacturers focus on short-vector designs (up to 2048-bits), although there are some exceptions (NEC's SX-Aurora features 16,384-bit vectors [24]).

The most widespread vector implementation is based on Single-Instruction Multiple-Data (SIMD) extensions. SIMD is available in microprocessors from different market segments, from high performance computing (HPC) to mobile phones [17,18,3,29,15,2]. Graphic Processing Units (GPUs) are architectures that also benefit greatly from DLP. A GPU uses a massive number of cores to execute sets of threads in a lock-step model.

Vector register size has seen an increase that doubled every four years [16]. The same authors state that doubling the register size can be significantly more energy efficient than doubling the number of cores. Companies like Intel [28, 18] and Fujitsu [32] have released commercial processors that work with 512-bit registers. Ultimately, the effectiveness of a vector architecture depends on its ability to vectorize large quantities of code [27].

This paper focuses on the evaluation of Intel's AVX-512 architecture in terms of energy efficiency and performance. The evaluation will also include Intel's Top-Down model information [31]. In this model, performance counter information is used to define the processor behavior for a given workload, and to detect hardware bottlenecks. The ParVec benchmark suite will be used for this analysis. ParVec is a modified version of PARSEC that exploits the SIMD-capabilities offered by modern processors [8,6]. This suite is extended with support for AVX-512 and other minor enhancements. The final source code is available via a git repository [9].

The paper is organized as follows: Section 2 gives a brief introduction to vectorization, benchmarking and provides information about related work.

Section 3 describes our evaluation environment and methodology. Section 4 describes, for each benchmark, the performance and energy profiles as well as the Top-Down model results. Finally, Section 5 concludes the evaluation and suggests directions for further research.

2 Motivation and Related Work

Vector supercomputers dominated the HPC world from the early 1970s up to late 2004 (Earth Simulator). Vector processors differentiate from scalar processors in that they operate on one-dimensional arrays (vectors) rather than single elements. However, large registers have several drawbacks. Wider RAMs comes with a power cost. There are techniques for mitigating this, but over dimensioning the vector register file will cost power. In addition, if an application can not make full use of every register, then a hardware resource is being wasted. For this reason, fabricating chips with long vector registers for commodity processors has been considered senseless for a long time. The first widely-deployed desktop SIMD system dates from 1996, with Intel Pentium processors and the MMX extensions (64-bit registers). Revisions over the initial instruction set introduce new instructions to the repertory, as well as increase the register size. The newest AVX-512 (2016) supports up to 32 512-bit registers. Other companies such as ARM or IBM have their own SIMD instruction sets. SIMD architectures can be programmed using intrinsics, assembly, or automatic compiler vectorization.

SIMD instructions have a potential speedup proportional to the width of the SIMD registers. In addition, SIMD reduces instruction cache and pipeline pressure (less instructions need to be located, decoded and issued). As register size increases, many CPU bound applications may become memory bound due to the increased data requirements of SIMD units. There are, however, important disadvantages when using SIMD functionality. SIMD architectures increase the pressure on the data cache and memory subsystem, since they require more bandwidth to feed the SIMD ALUs. SIMD relies on large register files that increase power dissipation and chip area. SIMD codes are hard to generate automatically by the compiler. As a consequence, automatic vectorization of naively written code is usually limited by bandwidth, gather/scatters or tight data dependencies [20]. In such cases data conversions or algorithmic modifications are required.

In order to evaluate such architectures, computer architects usually rely on benchmarking. Ferdman *et al.* [14] introduce a benchmark suite (CloudSuite) for emerging scale-out workloads, and compares it against state of the art benchmark suites. However, none of these approaches analyze effects of vectorization on application energy efficiency. Molka *et al.* [22] discuss weaknesses of the Green500 list with respect to ranking HPC system energy efficiency. They introduce their own benchmark using a parallel workload generator and SIMD support to stress main components in a HPC system. Kim *et al.* [20] show how blocking, vectorization and minor algorithmic changes can speed up

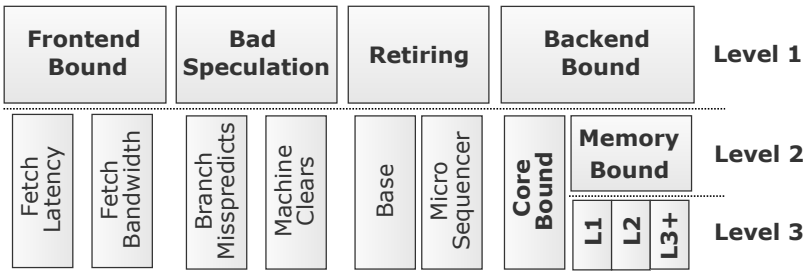

Fig. 1: Simplification of the Top-Down analysis hierarchy.

applications close to the best known tuned version. The RODINIA [10] and ALPBench [21] benchmark suites also offer limited SIMD support. BioParallel includes bioinformatic workloads, NU-MineBench measures data mining performance and PhysicsBench is used for computer game physics simulations. The Recursive Benchmark Suite [25], which includes recursive, task-parallel benchmarks that have been transformed in order to exploit DLP. Eight applications are included, ranging from microbenchmarks to kernels, that all have been manually vectorized with SSE (128-bit registers) AVX2 (256-bit registers) and AVX-512 (512-bit registers) intrinsics. Finally, ParVec is a modified version of PARSEC, that exploits the SIMD capabilities offered by modern processors [8,6].

3 Experimental Methodology

Top-Down Model. This methodology is described by the authors as "a practical method to quickly identify true bottlenecks in out-of-order processors". It presents a multi-level summary of the hundreds of performance events available in Intel processors, to quickly and accurately identify dominant performance bottlenecks. From the many levels of detail available in this methodology, we will focus on the first and second levels (bold entries of Figure 1). The first level divides the status of the issued μops[1] into four categories: frontend-bound, backend-bound, retiring or bad-speculation. Retiring accounts for μops finishing normally and leaving the pipeline, while bad-speculation represents those μops squashed due to a mispredicted branch. If μops are neither allocated resources nor squashed, then it means they are stalled. Frontend-bound represents the ratio of μops stalled in fetch/decode, while backend-bound covers ready-to-issue μops that cannot continue along the pipeline due to resource unavailability. The second level of the model breaks down these four categories increasing the detail level. Backend-bound μops can be sub-categorized into CPU or memory bound, depending on the resources causing these stalls. Finally, from the third level breakdown, we will only focus on performance counter information related to memory bound applications. This model has

[1] Micro-operations.

Cache	Size	Sharing	Ways of associativity	Line size	Latency (cycles)
Level 1 Instruction	32KB	Private	8	64B	4
Level 1 Data	32KB	Private	8	64B	4
Level 2	1MB	Private	16	64B	12
Level 3	11MB	Shared	11	64B	68
Main Memory	64GB	Shared	-	-	DDR4@2333

Table 1: Memory hierarchy information for a 3.6GHz 7820X processor.

been adopted by the Intel's profiling production tool "VTune". The required performance counters are already featured in production systems (just eight new events in the PMU on Intel's Ivy Bridge and later models).

Evaluation Environment. We will analyze the vectorized benchmarks on an Intel Skylake-X 7820X processor. The 7820X processor has eight physical cores, with sixteen threads and its memory hierarchy is described in Table 1. It is built on a 14-nm process and uses tri-gate transistors (reduced leakage power). It offers support for SSE4.2, AVX2 and AVX-512 instructions.

The analyzed applications are based on ParVec 3.0b, that was extended to support AVX-512. This can be easily achieved by filling the "wrapper library[2]" included in the suite with the appropriate translation to AVX-512 intrinsics (Code 1). Codes are tested to compile using both GCC 7.3.0 and 8.2.0 with -O2 flag on a Ubuntu 18.04 distribution with Kernel 4.15. Results shown in the evaluation are for GCC 7.3.0, since it is the default version in our distribution.

```
1  float dist(Point p1, Point p2, int dim) {
2   int i;
3   _MM_TYPE result, _aux, _diff, _coord1, _coord2; -> __m512 ...
4   result = _MM_SETZERO(); -> _mm512_setzero();
5   for (i=0;i<dim;i=i+SIMD_WIDTH) { -> 16
6    _coord1 = _MM_LOADU(&(p1.coord[i])); -> _mm512_loadu_ps(&(p1.coord[i]));
7    _coord2 = _MM_LOADU(&(p2.coord[i])); -> _mm512_loadu_ps(&(p2.coord[i]));
8    _diff = _MM_SUB(_coord1, _coord2); -> _mm512_sub_ps(_coord1, _coord2);
9    _aux = _MM_MUL(_diff,_diff); -> _mm512_mul_ps(_diff, _diff);
10   ....
```

Code 1. Streamcluster dist function implemented with the wrapper library.

Energy estimation and performance counter information for the Top-Down model are accurately extracted via internal MSRs[3]. We use PAPI libraries (5.6.1) for such purposes [23]. We use PAPI's powercap module to retrieve per-processor energy information (whole package).

Executions were performed sequentially 10 times, using the native input sizes of the PARSEC benchmark suite. The results are reproducible and stable, with a relative standard deviation below 1%. The processor is heated to a steady temperature, (around 50C) before running the benchmarks, to minimize

[2] Set of macros used to generate the intrisics code using the C pre-processor.

[3] Model Specific Registers.

variability in the energy measurements. Systems run with minimal console configurations (runlevel 3), to reduce OS influence on the results. Turbo boost is disabled, and Scalar, AVX2 and AVX512 operating frequencies are forced to 3.6Ghz. This is important to note, since the default behavior of the 7820X Turbo Boost is to push scalar frequency to 4.5Ghz whenever possible, while AVX2 was limited to 3.8Ghz, and AVX512 to 3.6Ghz. We use "taskset" to prevent thread migrations between cores that may slow down application.

4 Evaluation

For every benchmark, we describe the impact of varying the number of threads and vector register size. More specifically, we report normalized speedup, energy reduction, instruction reduction (to our baseline, single-threaded scalar code). Then, for each combination of thread count and vector register size, we report the key parts of the Top-Down model as described in Section 3 to identify performance bottlenecks. IPC is also reported for a single thread version of each benchmark. The only energy MSR readable in the 7820X is the one that accounts power for the whole package. For this reason, energy scalability will be super-linear in the reported results, since idle cores are consuming power all the time for 1 and 4 thread tests.

Blackscholes. This benchmark shows a linear benefit in performance as register size increases (Figure 2-left). Scalability with thread count is also close to linear, and SMT has a marginal performance improvement. Energy efficiency peaks when using AVX-512 and running on 8 threads (24.5x better than baseline). The instruction count reduction is around 19x for AVX-512 (Figure 2-centre), meaning that most of the application code is vectorized. There is a super-linear component to instruction reduction since SIMD and scalar codes are slightly different. The SIMD code for exponential and logarithmic functions is based on SLEEF [1], while the scalar version is based on libc standard math library. By analyzing the Top-Down model (Figure 2-right), we can see that most of the time the application is stalled in the backend (around 70% for AVX-512).

Going one level deeper (Figure 3-left), we see that Blackscholes is mostly CPU-bound in our evaluation environment. Over 70% of backend stalls are due to core-related stalls, while 28% of stalls relate memory stalls (mostly L1D, as depicted in Figure 3-right). Since the IPC of Blackscholes is low (Figure 2-centre), we can conclude that the main limitation of this application is due to low ILP, either due to instruction dependencies, long latency instructions blocking the ALUs (45% of cycles), or a combination of both.

Since there is little to do in order to improve ILP in this code, vectorization becomes critical to improve its performance and efficiency. Regarding L1D stalls, using manual prefetching may be helpful to further push scalability.

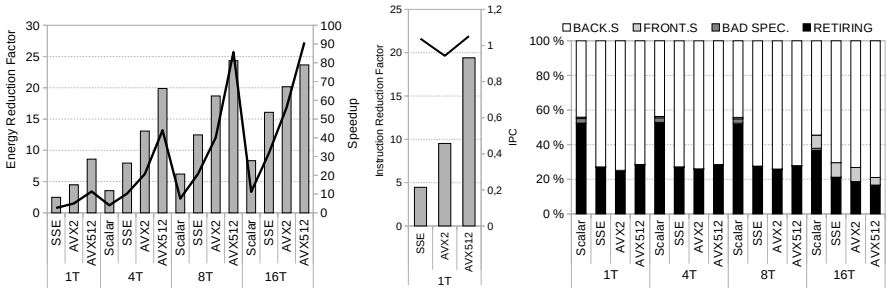

Fig. 2: Blackscholes energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

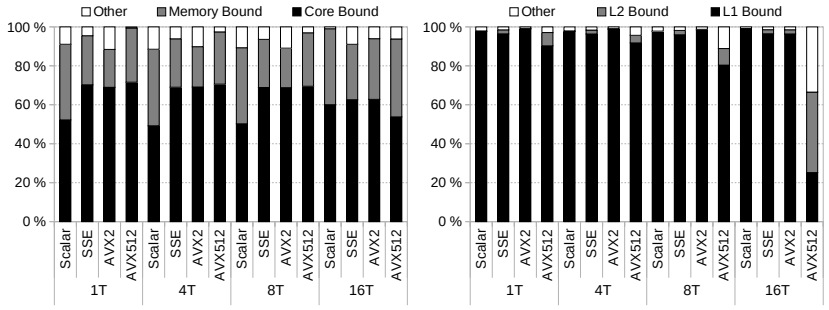

Fig. 3: Blackscholes backend stalls breakdown (left) and memory stalls breakdown (right).

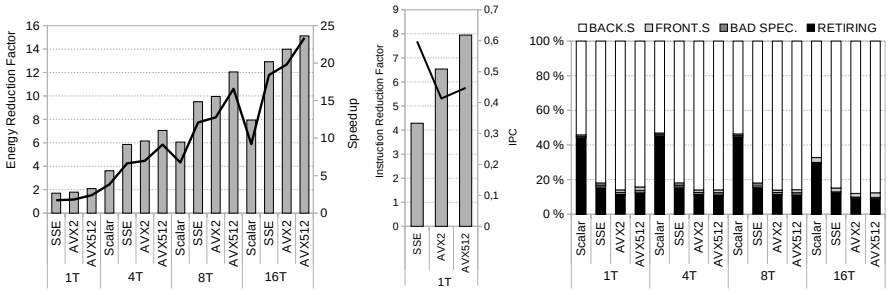

Fig. 4: Canneal energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

Canneal. Canneal performance scalability is very limited (Figure 4-left). In a single-thread environment, the AVX-512 implementation reaches a 2.4x speedup. Thread scalability is slightly sub-linear (6.75x for 8 threads scalar), and orthogonal to SIMD extensions (16.5x for 8 threads AVX-512). SMT significantly improves performance and energy efficiency of this benchmark. Speedup goes up from 16.5x with 8 threads to 23.4x with 16 threads when using AVX-512.

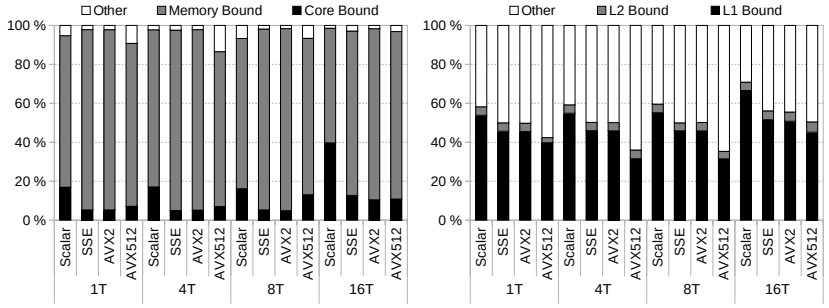

Fig. 5: Canneal backend stalls breakdown (left) and memory stalls breakdown (right).

Energy efficiency peaks when using AVX-512 and running on 16 threads (15.1x better than baseline). Regarding instruction count, the reduction is close to 8x for AVX-512 (Figure 4-centre), meaning that almost half of the executed instructions on the SIMD implementations are not vectorized. The first level of the Top-Down model (Figure 4-right) points to backend stalls as the main source of performance bottleneck (around 84% for AVX-512).

Digging deeper into the backend issues, Figure 5-left shows that around 7% of stalls are due to core issues and 10% due to other processor resources (e.g., reorder buffer, reservation stations, store buffer, etc). However, the gross performance penalty comes from the memory subsystem (around 83%). More specifically, around 40% of the stalled cycles happen during L1D cache misses, 5% during L2 cache misses, and the remaining 55% from other levels of the memory hierarchy (Figure 5-right). Due to these huge stalls, the IPC of Canneal is really low (Figure 4-centre). L1D stalls could be reduced using manual prefetching, but the performance scalability would still be limited by the data movements performed by the application. Note that this is an iterative clustering algorithm, so as long as the whole dataset cannot fit in the caches, there will be performance degradation. However, SIMD can help to reduce overall energy requirements. By increasing register size we can perform the same computations in less time. Since this time reduction is useless, we can run cores at a lower frequency, burning less power, while still overlapping computations with data movements along the memory hierarchy.

Streamcluster. Figure 6-left shows the performance scalability of Streamcluster. The speedup of this application reaches 2.9x for single threaded AVX-512, and 20x for 16 threads AVX-512. Scalability with thread count is sub-linear, and SMT achieves an improvement of up to 16% in performance.

The best energy efficiency is obtained using AVX-512 and running with 16 threads (7.3x better than baseline). However, the difference is minimal when running on 8 threads. The instruction reduction factor is slightly under 5x for AVX-512 (Figure 6-centre). This is far from the ideal 16x reduction factor, meaning scalar code will soon dominate the overall performance of the application (Amdahl's law). If we analyze the results of the first level Top-

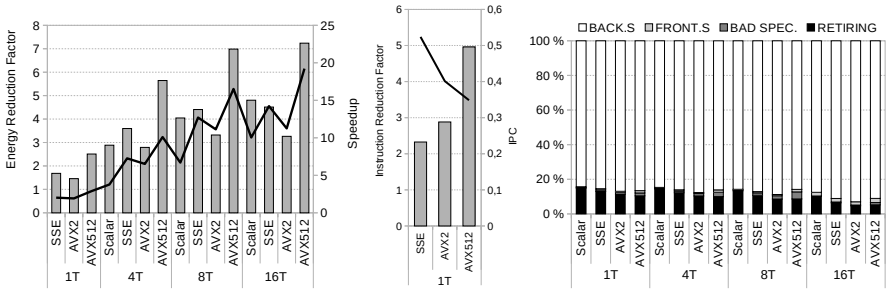

Fig. 6: Streamcluster energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

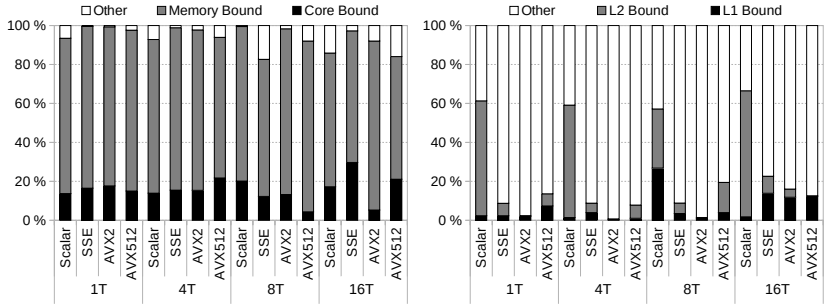

Fig. 7: Streamcluster backend stalls breakdown (left) and memory stalls breakdown (right).

Down model (Figure 6-right), almost 93% of the time the processor is stalled on the backend, and around 2% on wrong path execution.

Figure 7-left breaks down the backend stalls, clearly showing that Streamcluster is highly limited by the memory subsystem. Indeed, memory stalls account for over 82% of the total backend stalls when using AVX-512. Furthermore, the memory stall analysis reveals that only 12% of these stalls relate to the combined L1 and L2 misses, while the vast majority of stalls correlate to other levels of the memory hierarchy (Figure 7-right). The negative effects on performance from memory stalls drop the IPC below 0.5 (Figure 6-centre).

L1D stalls could be reduced using manual prefetching, however, since cache locality of Streamcluster is very low, the overall system performance is going to be strongly linked to cache and memory bandwidth. This also explains the low impact of SMT on performance, since both threads are competing for the same bandwidth. As it happens with Canneal, SIMD can help to reduce overall energy requirements. Reducing operating frequency we can save energy. Computational power is maintained by widening the vector registers/ALUs.

Fluidanimate. This application has divergent branches, small number of arithmetic computations for each loaded data, variable number of loop indexes

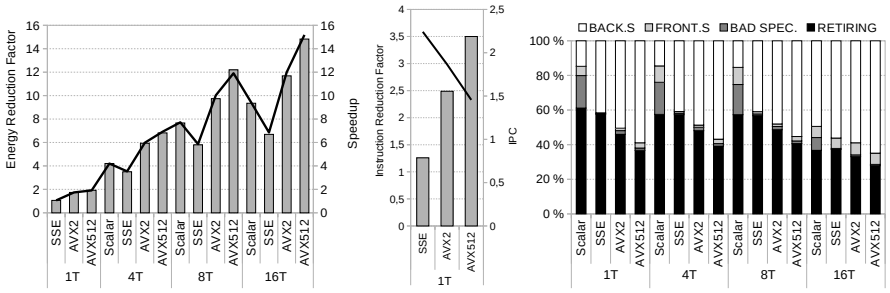

Fig. 8: Fluidanimate energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

(usually non-divisible by SIMD width), high output variance due to floating point rounding errors and small input size. The first step we took was to increase particle density of the default datasets. We will release the tools and dense datasets used in our experiments upon publication.

Figure 8-left shows the performance and energy scalability of Fluidanimate using a dense dataset. Thread performance scalability of this application is almost linear, while SMT only achieves 21% improvement. AVX-512 speedup is limited to around 2x for single thread runs, and peaks at 15x for 16 threads. Performance results are very similar to those obtained by AVX2. However, if we check the instruction reduction factor, we see how it increases from 2.5x with AVX2 to around 3.5x for AVX512 (Figure 8-centre). This means that while SIMD is scaling properly regarding the vectorized instructions, the overall performance remains unaffected. Energy improvements are close to 15x when running AVX-512 and 16 threads.

Figure 8-right points at backend stalls as the main performance limitation (around 60% for AVX-512). Breaking down the backend stalls reveals that almost 78% of backend stalls are core stalls (Figure 9-left). Further analyzing core stalls showed that 50% are due to busy divider/sqrt ports. The rest can be attributed to low ILP, since port utilization is low. This is reflected in the IPC drop from 2 to 1.5 when going from scalar to AVX-512 (Figure 8-centre).

Swaptions. This benchmark has a sub-linear improvement in performance an energy efficiency (Figure 10-left). In a single-thread scenario, performance improves by 2.1x for AVX-512, the same as energy. When running with 8 threads, AVX-512 performance jumps to 16.5x, and SMT adds additional 31% to that number. The best energy efficiency is achieved running AVX-512 with 16 threads. Instruction reduction factor (Figure 10-centre) shows improvements of 0.5 and 0.3 when moving from SSE to AVX2, and from AVX2 to AVX-512. This means that the fraction of vectorized code not only is small, but the number of vectorized instructions is shrinking rapidly. This will soon make the scalar part of the application the main performance limiting factor.

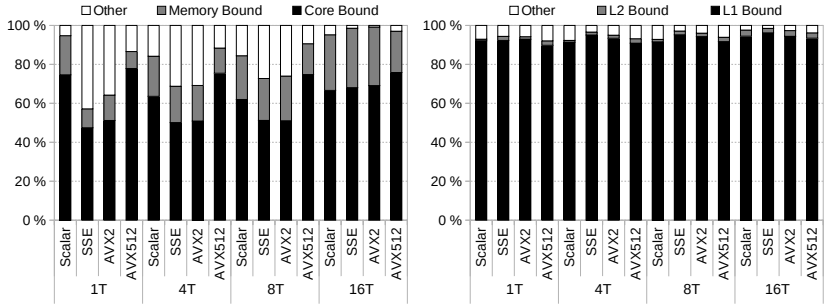

Fig. 9: Fluidanimate backend stalls breakdown (left) and memory stalls breakdown (right).

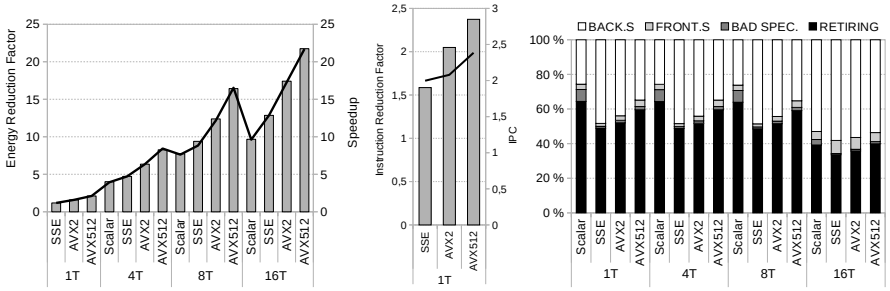

Fig. 10: Swaptions energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

The Top-Down analysis (Figure 10-right) reveals only a 40% processor stalls for AVX-512. 2.5% of the stalls are due to mis-speculation, around 5% are frontend stalls, while 32.5% of stalled cycles are due to backend limitations. The second level of the Top-Down model shows that 16% of stalls happen in pipeline stages different than execution, mostly issue stalls due to lack of resources (Figure 11-left). For the remaining backend stalls, 61% are core bound, and 23% memory bound. In this case, divider units blocking the pipeline happen rarely (around 7%). We can only conclude that the remaining stalls are due to lack of ILP. Figure 11-right breaks down memory stalls. Most stalls happen during L1D misses (around 88% for AVX-512), so manual prefetching could help to further reduce memory stalls.

Vips. Vips is an image editing tool where only some filters are vectorized (42% of total execution time). In addition, performance scalability with register size is minimal (1.24x for the single-thread environment). Energy improvements are also limited to around 20% for AVX-512 (Figure 12-left). IPC of this application is high, around 3, and is decreasing with register size (Figure 12-centre). Instruction reduction is minimal, given the small fraction of the total code being vectorized. However it shows a steady reduction across instruction

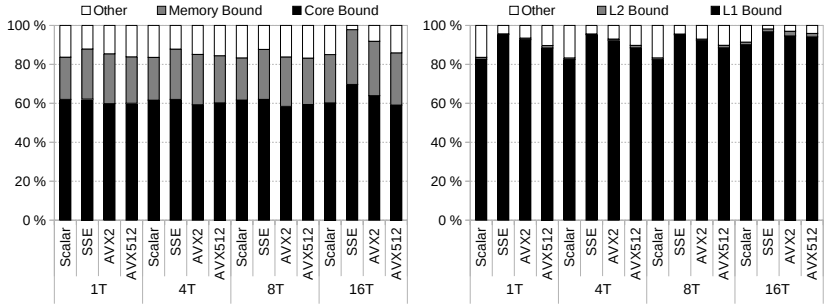

Fig. 11: Swaptions backend stalls breakdown (left) and memory stalls breakdown (right).

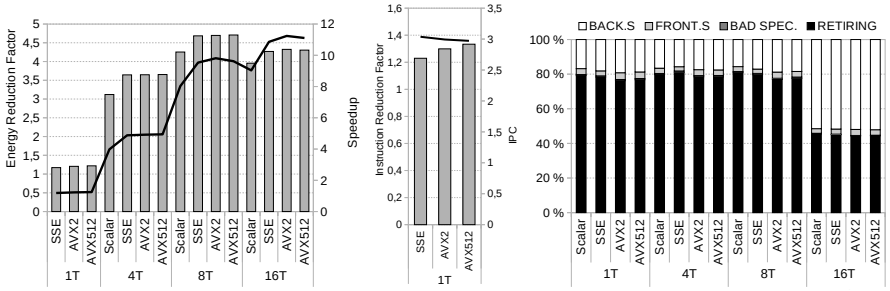

Fig. 12: Vips energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

sets. The best energy efficiency is achieved running on 8 threads with AVX-512. SMT marginally improves performance, but has a negative impact on energy efficiency.

Figure 12-right depicts the Top-Down breakdown for Vips. We can see it is well optimized, and the processor spends only 20% of the cycles stalled, being around 17% of them backend stalls. 55% of these backend stalls are core bound, 15% are memory bound, and 30% are resource limitations on issue or retire (Figure 13-left). Only 5.8% of core stalls are due to divider units being busy, leading to low ILP or other ports being busy as main sources of core stalls. Meanwhile, L1D cache stalls account for 63% of memory stalls, 13% for L2 stalls, and the remaining 24% to other levels of the memory hierarchy (Figure 13-right). We recommend to isolate the vectorized filters from the main application, since it is difficult to isolate the effects of vectorization from the main scalar code.

x264. This benchmark was vectorized by the original developers, and does not rely on ParVec's wrapper library. Thread scalability is sub-linear, reaching 3.12x performance improvement when running on 4 threads. SMT provides additional 30% performance improvement for AVX-512, so it is worth enabling

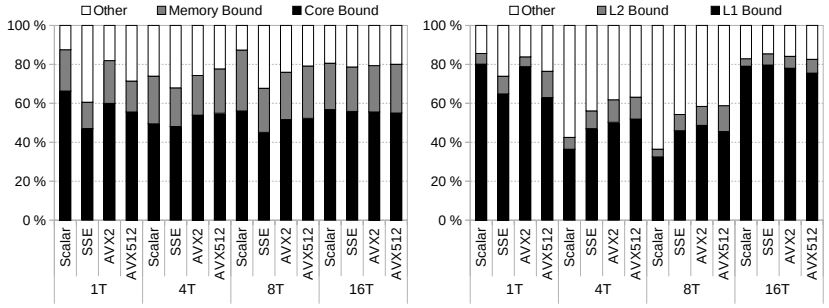

Fig. 13: Vips backend stalls breakdown (left) and memory stalls breakdown (right).

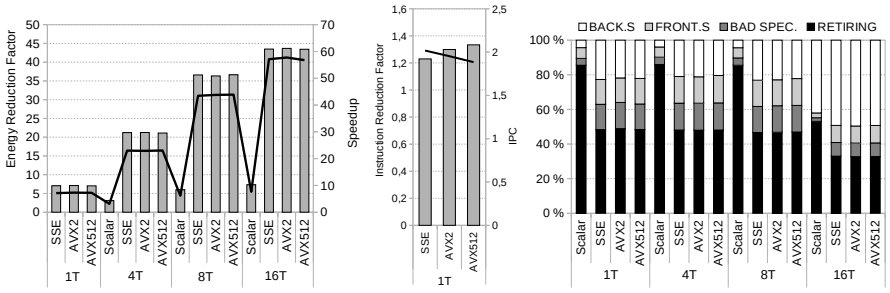

Fig. 14: x264 energy reduction factor (left, primary axis), speedup (left, secondary axis), instruction reduction factor (centre, primary axis), IPC (centre, secondary axis) and first Top-Down level (right) for different thread counts (1 to 16) and SIMD ISAs.

for this application. Figure 14-left shows no performance or energy scalability with register size. However, Figure 14-centre shows that there is a slight reduction on instruction count. From these results we can observe that the overall fraction of code that is being vectorized is very small (1.2x from the theoretical 4x for SSE).

The Top-Down breakdown for this benchmark is depicted in Figure 14-right. It shows that 23% of stalls happen on the backend, 16% are due to mis-speculation, and the remaining 16% stall at the frontend when using the AVX-512 extensions. From that 23% of backend stalls, around 42% are core bound, 23% are issue and retire stalls, and 35% are memory bound (Figure 15-left). Core bound stalls are mainly due to port utilization and low ILP stalls (in fact IPC is going down with wider ISAs). Memory bound can be broken down into L1D (around 42%), around 8% for L2, and 50% for other levels of the memory hierarchy (Figure 15-right). Prefetching may be helpful to further push scalability.

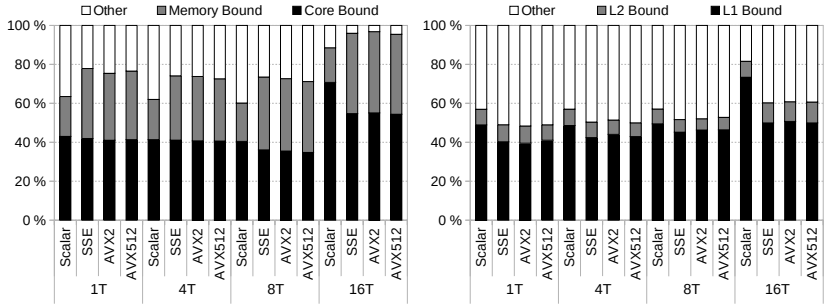

Fig. 15: x264 backend stalls breakdown (left) and memory stalls breakdown (right).

5 Conclusions

Modern computer architectures focus nowadays on parallelism to improve performance under a specific TDP. Instruction-level (ILP) and thread-level parallelism (TLP) and data-level parallelism (DLP) are at the core of these architectures. However, DLP is usually underutilized in CPUs, despite its huge potential. In this paper, we evaluate the performance and energy scalability of Intel's latest SIMD extensions, AVX-512. We have extended and updated the ParVec benchmark suite to test our Skylake-X evaluation processor with the latest versions of GCC. This includes 7 vectorized benchmarks from the PARSEC benchmark suite.

The evaluation shows that not all applications experience a linear speedup when vectorized (based on SIMD width). Speedup is limited by several factors, including algorithmic limitations, data structure organization, application input or the underlying memory hierarchy. We have identified applications that are limited by ILP or port utilization, but still benefit linearly from SIMD extensions (Blackscholes). Other applications, such as Canneal or Streamcluster, completely lose any core boundedness and become strongly memory bound. Applications like Fluidanimate are limited by long latencies on divider/sqrt units. Finally, Vips and x264 are well balanced applications, and have a variety of limitations at different levels, from issue resources (e.g., reservation stations, reorder buffer, store buffer, etc), memory hierarchy and ILP.

It is important to note that, although some benchmarks obtain similar performance using SIMD and threading, there is a huge difference in energy efficiency since SIMD cost much less power. SIMD implementations reduce the total number of memory accesses but maintain similar number of cache misses. The usage of manual or hardware prefetching is critical to make good use of vector capabilities.

An intelligent scheduler may also be helpful. This scheduler can be used to reduce core frequency in memory bound applications to a point that it minimizes the pressure on the memory subsystem, improving energy efficiency. An application that has a scalar memory-bound implementation will barely benefit from vectorization for high processor frequencies (when the memory

hierarchy saturates). This means that SIMD implementations can outperform scalar implementations at lower frequencies. Another alternative would be to implement compression techniques. This way we could use the exceeding computational capabilities of SIMD to reduce the pressure on the memory subsystem.

Finally, improving ILP or using alternative instructions that approximate results, whenever possible, would be of interest in CPU bound applications.

References

1. SLEEF Vectorized Math Library. https://sleef.org/
2. AMD: 3DNow! Technology Manual. Motorola (2000)
3. ARM NEON Technology
4. Asanovìc, K.: Vector Microprocessors. Ph.D. thesis (1998)
5. Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L., Stokes, R.A.: The ILLIAC IV Computer. IEEE Transactions on Computers **C-17**(8), 746–757 (1968)
6. Bienia, C.: Benchmarking modern multiprocessors. Ph.D. thesis, Princeton University (2011)
7. Borkar, S., Chien, A.A.: The future of microprocessors. ACM, New York, NY, USA (2011). DOI 10.1145/1941487.1941507. URL `http://doi.acm.org/10.1145/1941487.1941507`
8. Cebrian, J.M., Jahre, M., Natvig, L.: Parvec: Vectorizing the parsec benchmark suite. Computing pp. 1077–1100 (2015)
9. Cebrian J. M., J.M., Natvig, L.: ParVec Git Repository. `https://github.com/magnusjahre/parvec`
10. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Proc. of the 2009 IEEE Int. Symp. on Workload Characterizatio, pp. 44–54. IEEE (2009). DOI http://doi.ieeecomputersociety.org/10.1109/IISWC.2009.5306797
11. Cray Research, I.: Cray X-MP Series Model 48 Mainframe Reference Manual (1984)
12. Dennard, R., Gaensslen, F., Rideout, V., Bassous, E., LeBlanc, A.: Design of ion-implanted mosfet's with very small physical dimensions (1974). DOI 10.1109/JSSC.1974.1050511
13. Espasa, R., Valero, M., Smith, J.E.: Vector Architectures : Past , Present and Future. Proceeding ICS '98 Proceedings of the 12th international conference on Supercomputing pp. 425–432 (1998)
14. Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A.D., Ailamaki, A., Falsafi, B.: Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In: 17th Int. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2012)
15. Fuller, S.: Motorola AltiVec Technology. Motorola (1998)
16. Hennessy, J.L., Patterson, D.A.: Computer Architecture, Fourth Edition: A Quantitative Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2006)
17. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. (2012)
18. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference. (2015)
19. ITRS: Int. Technology Roadmap for Semiconductors report (2012). URL `http://www.itrs.net/Links/2012ITRS/Home2012.htm`
20. Kim, C., Satish, N., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M., Dubey, P.: Technical report: Closing the ninja performance gap through traditional programming and compiler technology (2012)
21. Li, M., Sasanka, R., Adve, S.V., kuang Chen, Y., Debes, E.: The alpbench benchmark suite. In: In Proc. of the IEEE Int. Symp. on Workload Characterization (2005)

22. Molka, D., Hackenberg, D., Schöne, R., Minartz, T., Nagel, W.: Flexible workload generation for HPC cluster efficiency benchmarking. Springer Berlin / Heidelberg (2011). URL http://dx.doi.org/10.1007/s00450-011-0194-9
23. Mucci, P.J., Browne, S., Deane, C., Ho, G.: PAPI: A portable interface to hardware performance counters. In: Proc. of the Department of Defense HPCMP Users Group Conference (1999)
24. NEC: Vector Supercomputer SX Series: SX-Aurora TSUBASA (2017)
25. Ren, B., Jo, Y., Krishnamoorthy, S., Agrawal, K., Kulkarni, M.: Efficient execution of recursive programs on commodity vector hardware. In: ACM SIGPLAN Notices, vol. 50, pp. 509–520. ACM (2015)
26. Russell, R.M.: The CRAY-1 computer system. Commun. ACM **21**(1), 63–72 (1978)
27. Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M., Dubey, P.: Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications? In: Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA), pp. 440–451 (2012)
28. Sodani, A.: Knights landing (KNL): 2nd Generation Intel Xeon Phi processor. In: Hot Chips (2015)
29. Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., Reid, A., Rico, A., Walker, P.: The ARM Scalable Vector Extension. IEEE Micro **37**(2), 26–39 (2017)
30. Watson, W.J.: The TI ASC: A Highly Modular and Flexible Super Computer Architecture. In: Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I (AFIPS), pp. 221–228 (1972)
31. Yasin, A.: A Top-Down method for performance analysis and counters architecture. ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software pp. 35–44 (2014). DOI 10.1109/ISPASS.2014.6844459
32. Yoshida, T.: Introduction of Fujitsu's HPC processor for the Post-K computer. In: Hot Chips (2016)