A Comparative Study of ISA Multimedia Extensions for HPC

Ayaz Akram, Sajjad Rahnama
Project Report ECS-201C, Spring 2019
{yazakram,sajjad.rahnama}@ucdavis.edu

ABSTRACT

Exploiting data parallelism with SIMD instructions can improve the performance of applications with data level parallelsim. In this work, we compare and analyze the effects of using different SIMD extensionsons on some HPC workloads. We try to figure out the reasons for the performance differences across different SIMD extensions.

1 INTRODUCTION

Instruction set architecture (ISA) defines the boundary between the hardware and software layers of a computer system [13]. Different processor vendors define their own instruction set architectures to build processors based on them e.g. Intel/AMD's x86, ARM's ARMv8/ARMv7, and DEC's Alpha. Different instruction set extensions are added to the ISAs over time considering the importance of target applications. A major addition to all the ISAs is the inclusion of multimedia extensions to accelerate execution of multimedia workloads. These extensions rely on the fact that the performance of most of the multimedia applications can be improved by processing distinct data elements in parallel i.e. the use of SIMD (Single Instruction, Multiple Data) instructions. Some recent examples of multimedia extensions are Intel's SSE, AVX extensions [1, 2] and ARM's NEON extension [3]. Such extensions can be used to accelerate applications from other domains like HPC (high performance computing) as well. This works performs a comparative study of these SIMD extensions and compare their performance on some high performance computing benchmarks.

The specific questions we try to answer in this study include:

- What performance differences do fundamentally exist across these multimedia extensions?
- How do these extensions compare in terms of programmability and usability?
- How does the compiler generated (auto-vectorization) SIMD code compare to manually vectorized code?

In the past decade, SIMD execution has seen a dramatic increase in the set of applications using it, which has motivated big improvements in hardware support in mainstream microprocessors. In this article, we investigate two famous architectures X86 and ARM. Intels X86 has SSE, FMA and AVX and variations of these three which has been developed over past years. For the ARM architecture, there are two extensions. First, NEON which is advanced SIMD extension for the ARM architecture and second, the Scalable Vector Extension(SVE) which allows implementation choices for vector lengths that scale from 128to 2048 bits.

2 RELATED WORK

There are a few old related works that survey different multimedia extensions [12, 17]. Lee [12] provides an overview of various multimedia extensions of different ISAs for general purpose processors used to accelerate media processing (e.g. MAX, MMX, VIS). Similarly, Slingerland and Smith [17] surveyed existing multimedia instruction sets of the time and examined the mapping of their functionality to a set of computationally important kernels. These studies are old and focus on the qualitative comparison of the multimedia extensions.

Mitra et al. compare the SSE and NEON SIMD performance on the Intel and the ARM processors within the context of the open CV image processing library[14]. They also showed that using hand-optimized SIMD intrinsic functions rather than GCC compiler auto-vectorization for ARM and Intel platforms respectively will cause speed-ups of 1.05-13.88 and 1.34-5.54. Their main argument is that with hand optimization the code for running with SIMD extensions we will get better performance and utilization of these extensions rather than using auto-vectorization by the compiler. Hoffman et al [11] investigated the efficiency of different SIMD extensions for medical imaging applications and found out that GPU implementations perform better than SIMD implementations for the given workload.

Recently, Patterson and Waterman performed a comparison of RISC-V vector code to SIMD extensions of ARM, x86 and MIPS [15]. They argued that vector architectures are more efficient to exploit data level parallelism in comparison to SIMD architectures

There also exist many studies [4–6, 9, 16] on general purpose ISAs which do not specifically focus on SIMD extensions.

3 METHODOLGY

Table 1: NAS Parallel Benchmarks

Benchmark	Description
BT	Block Tri Diagonal Solver
SP	Scalar Penta Diagonal Solver
LU	Lower-upper Gauss-Seidel Solver
IS	Integer sorting
EP	Embarrasingly Parallel
CG	Conjugate Gradient
MG	Multi-Grid Method
FT	Fourier Transform

Table 2: PARSEC Benchmarks

Benchmark	Application Domain
Blackscholes	Financial Analysis
Swaptions	Financial Analysis
Fluidanimate	Animation
Vips	Media Processing
Streamcluster	Data Mining

For this study, we made use of NAS parallel benchmarks [7] (NPB 3.3.1), PARSEC benchmark suite [8] and a vectorized version of PARSEC named ParVec [10]. NAS parallel benchmarks are used to evaluate the performance of parallel supercomputers. NAS parallel benchmarks contain five kernels and three pseudo applications. A brief description of these workloads is given in Table 1. Though, NAS parallel benchmarks have multiples classes (the workload size varies), in this work, we only use the class B workloads.

PARSEC [8] is another benchmark suite for parallel computers which targets differernt domains including HPC. A short description of the PARSEC workloads used in our study is shown in the Table 2. ParVec [10] includes modifications in PARSEC to make use of SIMD extensions of modern processors.

We used auto-vectorization feature of gcc to compile these benchmarks with different extensions (apart from ParVec). The x86 machine used in our experiments is an Intel Core i7-8700 (32GB) machine, while the ARM machine is based on AWS Graviton processor (8 GB Memory). To collect some performance statistics of the executed workloads, we make use of perf tool on both machines.

4 RESULTS

Next, we present results of our experiments. Figure 1 shows the execution cycles for class B of NAS parallel benchmarks, when the vectorization is enabled (with AVX and SSE extensions separately) and disabled on an x86 machine. The execution cycles are normalized to the AVX enabled case for each workload. As it is clear from the figure, the differences in execution cycles can vary from almost non-existent (as in is.x) to 1.8x (as in mg.x). Importantly, the performance improvements of vectorization are quite far from the theoretical speed-ups possible. Secondly, the performance difference between two vector extensions (AVX and SSE) with different length vector registers is not significant. No benchmark preferred smaller vector length.

mg.x has the highest improvement in performance as the benchmark contains a lot of contiguous memory operations in the innermost loop of the kernel. The benchmak has highly structured long distance communication. cg.x on the other hand is quite irregular with many indirect memory accesses, thus not exhibiting significant performance improvements. is.x has very small impact of auto-vectorization, the main reason being that the kernel function lack loops which can be vectorized and it has many indirect memory accesses. For bt.x, lu.x and sp.x, since the innermost loops are iterated for a limited number of time and they involve non-contiguous memory operations, the vectorization improvements are limited.

In order to make sure, that there are no DVFS effects involved in the cycle count comparison, we looked at the absolute time comparison shown in Figure 2, which shows the same trend with the same differences.

Figure 3 shows a comparison of the dynamic number of instructions for each benchmark among vectorized and scalar versions. The number of instructions are normalized to the instructions with AVX enabled. On average, AVX resulted into 27% less instructions compared to the non-vectorized code. It is expected that this decrease in the number of instructions will improve the energy efficiency of the system.

Total number of data cache loads is reduced to as much as 100% (for ft.x), when AVX is used compared to the scalar version of code. The reason behind this decrease in the number of data cache loads is that the a single vector load can load more data compared to a scalar one. However, this difference in the number of data cache loads does not affect the number of data cache misses as shown in the Figure 5, where the L1-Dcache misses stay same across all cases.

Figure 6 shows a comparison of last level cache misses across the workloads. In contrast to L1-dcache misses, the number of last level cache (LLC) misses vary across the vector and scalar version of the benchmarks. In almost all of the cases (except ep.x), the number of LLC misses are either same or high for vectorized code. One possible explanation of the high number of LLC misses is the increase in memory traffic that is caused by vector operations. It is possible that, when vector operations bring in data to LLC the amount of useful data that is kicked out is higher than the case of scalar versions of the benchmarks.

Figure 7 shows the number of branch mispredictions for the studied workloads. Like, LLC misses the impact of vectorization varies across the workloads.

Figure 8 shows a comparison of the execution time of class B workloads (NAS parallel) on the ARM machine, with vectorization enabled/disabled. Interestingly, we did not observe any significant difference here. We spent quite an amount of time to figure out if even the NEON instructions are used by the benchmarks. We were able to make sure that the machine supports SIMD extensions, and that the compiler generated few SIMD operations withe autovectorization enabled. But, the overall impact of vectorization on the performance is too little (mainly it seems like only small portion of the benchmarks is vectorized with auto-vectorization).

4.1 Impact of Increasing Number of Threads

Figure 9, 10 and 11 show the impact of increasing the number of execution threads on vectorization for threads=1,2 and 4 respectively. The performance improvement because of vectorization decreases with increase in the number of execution threads. One possiblity for such behavior could be that the opportunity of performance improvement because of data parallelism reduces because of thread level parallelism.

4.2 Comparison of Auto-vectorization and Manual Vectorization

To study the difference in auto-vectorized code and the manually vectorized code, we compared the performance improvements observed for vector extensions in ParVec benchmark suite and the improvement that is seen if the auto-vectorization is enabled in ParSec benchmarks. This is shown in Figure 12 nad 13. The performance improvements in case of manually vectorized code are significantly higher than the case of auto-vectorization. This makes sense, since manual vectorization in ParVec also involves othe modifications in the code as well, e.g. they replaced data structures Array of Structures (AoS) to Structure of Arrays (SoA) for Fluidanimate benchmark.

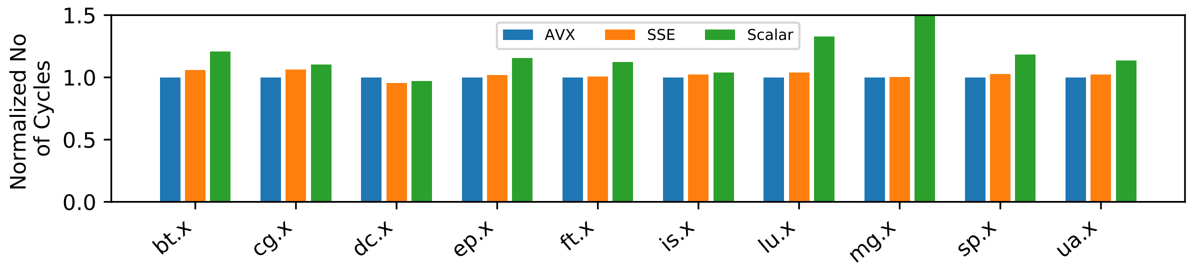

Figure 1: Normalized Execution Cycles with and without vectorization (x86: AVX, SSE)

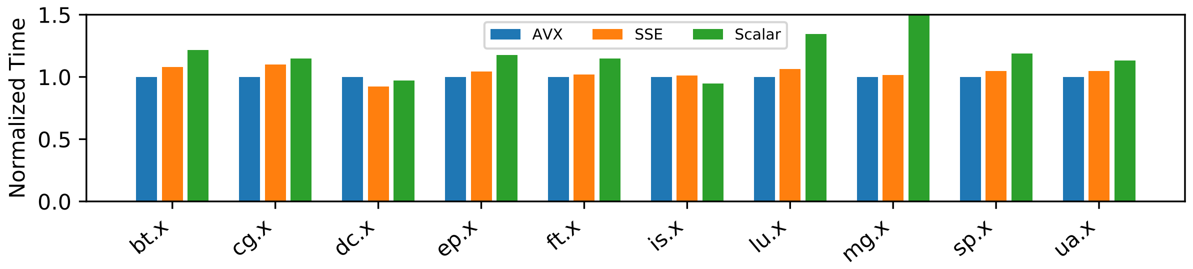

Figure 2: Normalized Execution Time with and without vectorization (x86: AVX, SSE)

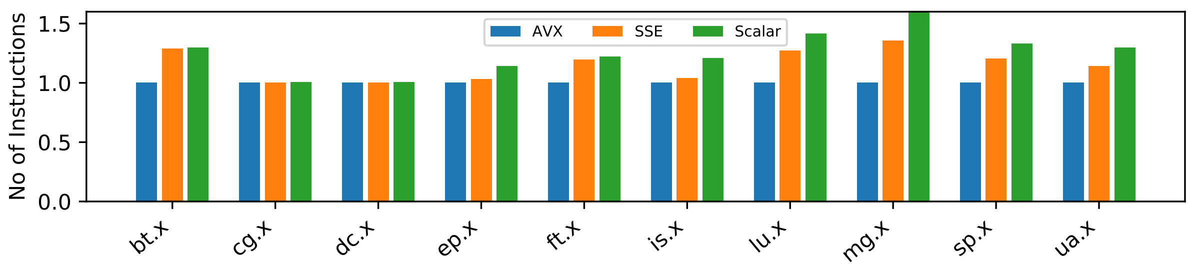

Figure 3: Normalized Instruction Counts with and without vectorization (x86: AVX, SSE)

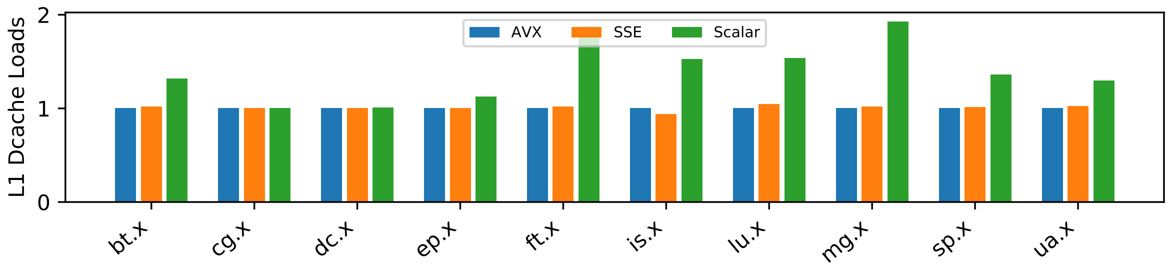

Figure 4: Normalized L1-Dcache Loads with and without vectorization (x86: AVX, SSE)

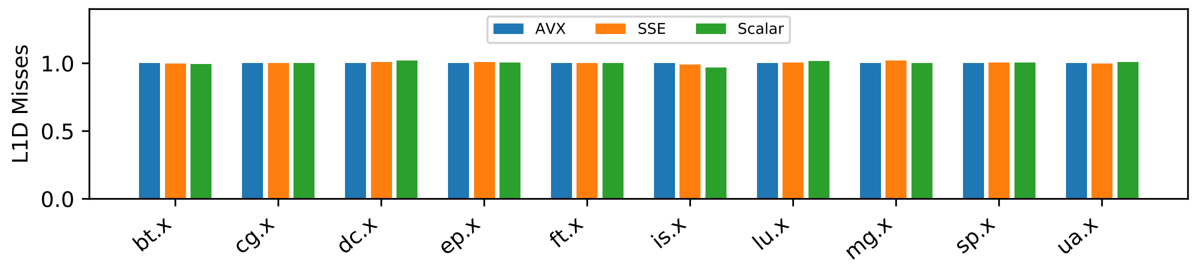

Figure 5: Normalized L1-Dcache Misses with and without vectorization (x86: AVX, SSE)

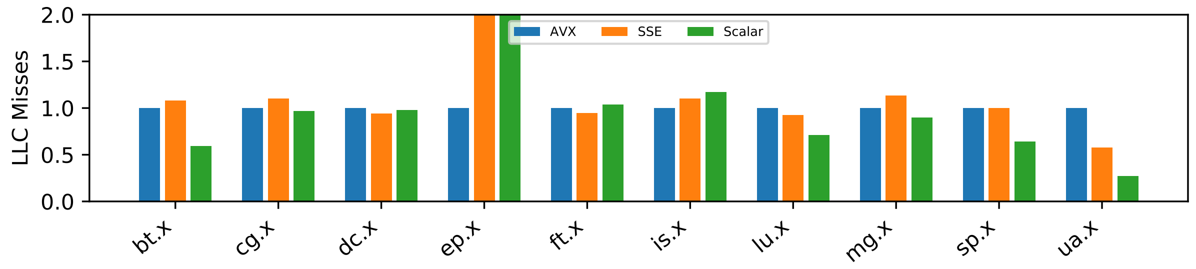

Figure 6: Normalized LLC Misses with and without vectorization (x86: AVX, SSE)

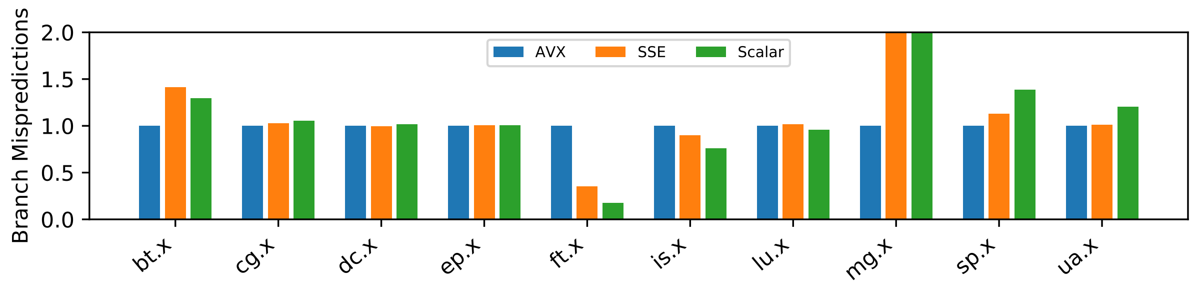

Figure 7: Normalized BP Misses with and without vectorization (x86: AVX, SSE)

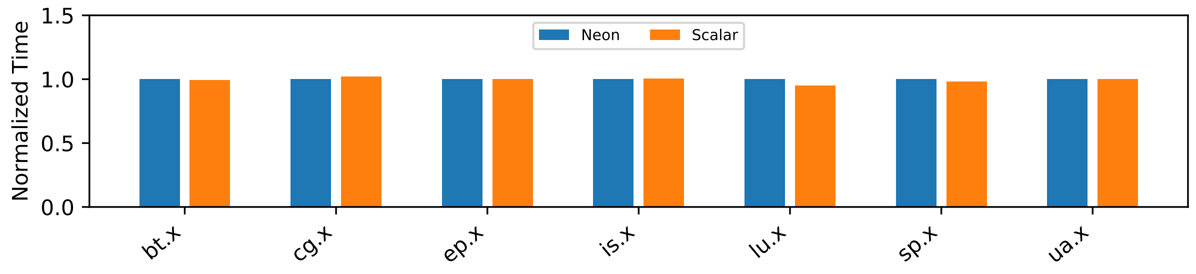

Figure 8: Execution Time Comparison for ARM with and without vectorization (Class B workloads)

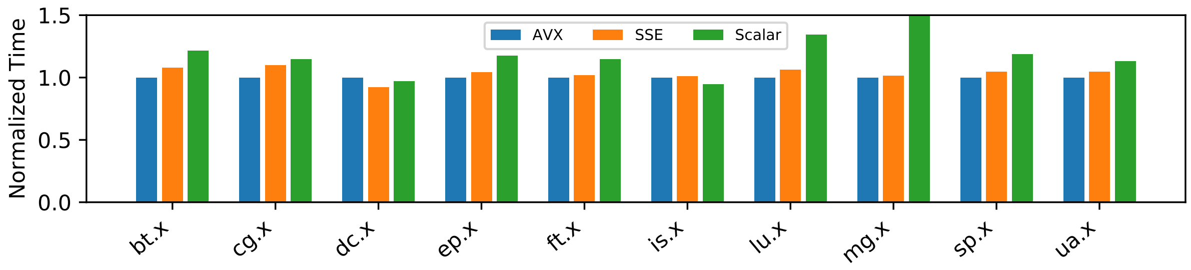

Figure 9: Impact of Vectorization with Threads = 1

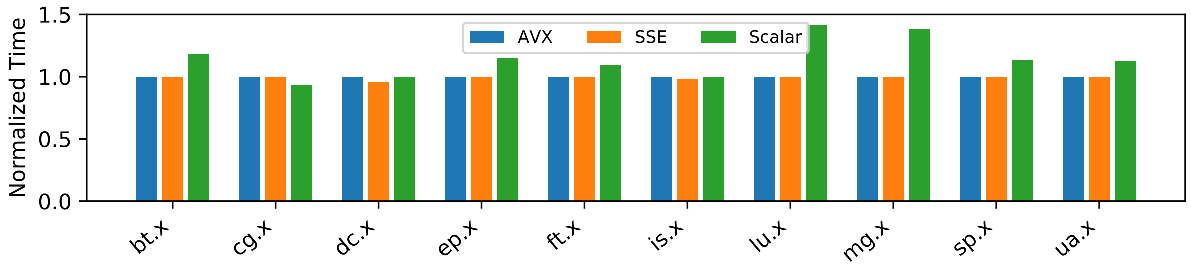

Figure 10: Impact of Vectorization with Threads = 2

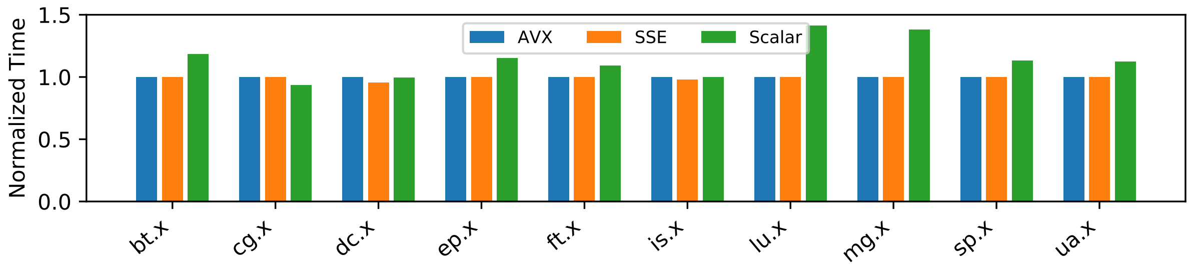

Figure 11: Impact of Vectorization with Threads = 4

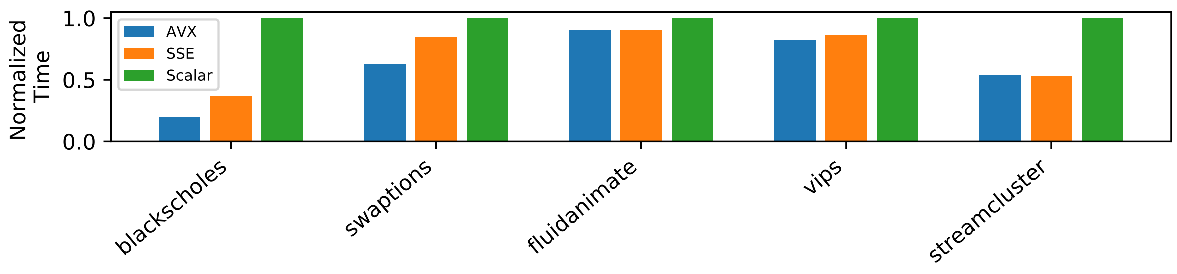

Figure 12: ParVec Time with and without vectorization (x86: AVX, SSE)

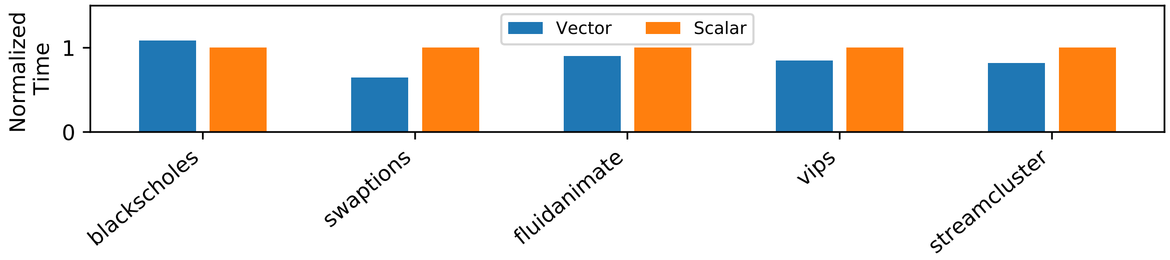

Figure 13: ParSec Time with and without vectorization (x86: AVX)

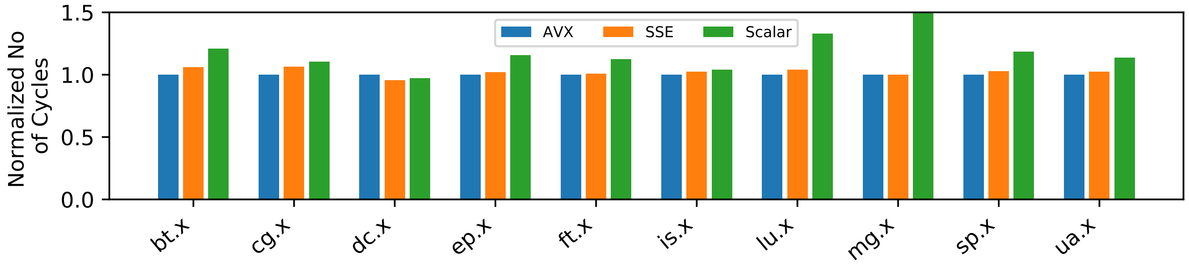

Figure 14: Execution Cycles with and without vectorization (x86: AVX, SSE), Fortran Version

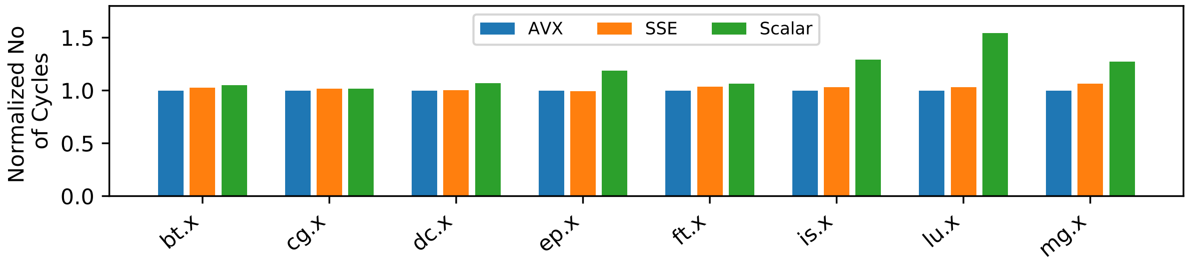

Figure 15: Execution Cycles with and without vectorization (x86: AVX, SSE), C Version

4.3 Difference Between Fortan and C NAS Parallel Benchmarks

Finally, we also looked at how the Fortran and C versions of NAS parallel benchmarks will differ with respect to vectorization improvements, when auto-vectorization is enabled in gcc compilers. Interestingly, Fortran version showed more improvement in performance with vectorization.

5 CONCLUSION

In this work, we studied the performance differences across SIMD extensions for HPC workloads and tried to figure out where do the performance differences come from. We also looked at few other important characteristics to compare studied SIMD extensions.

REFERENCES

[1] [n. d.]. *Intel Streaming SIMD Extensions Technology Defined*. https://www.intel.com/content/www/us/en/support/processors/000005779.html Available: https://www.intel.com/content/www/us/en/support-/processors/000005779.html [Online; accessed 20-June-2017].

[2] [n. d.]. *Introduction to IntelÂő Advanced Vector Extensions*. https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions Available: https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions [Online; accessed 20-June-2017].

[3] [n. d.]. *NEON*. https://developer.arm.com/technologies/neon Available: https://developer.arm.com/technologies/neon [Online; accessed 20-June-2017].

[4] Ayaz Akram. 2017. A study on the impact of instruction set architectures on processor's performance. (2017).

[5] Ayaz Akram and Lina Sawalha. 2017. The impact of isas on performance. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD) co-located with 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada*.

[6] Ayaz Akram and Lina Sawalha. 2019. A Study of Performance and Power Consumption Differences Among Different ISAs. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 628–632.

[7] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. 1991. The NAS parallel benchmarks. *The International Journal of Supercomputing Applications* 5, 3 (1991), 63–73.

[8] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 72–81.

[9] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1–12.

[10] Juan M Cebrian, Magnus Jahre, and Lasse Natvig. 2015. ParVec: vectorizing the PARSEC benchmark suite. *Computing* 97, 11 (2015), 1077–1100.

[11] Johannes Hofmann, Jan Treibig, Georg Hager, and Gerhard Wellein. 2014. Comparing the performance of different x86 SIMD instruction sets for a medical imaging application on modern multi-and manycore chips. In *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*. ACM, 57–64.

[12] Ruby B Lee. November 1997. MULTIMEDIA EXTENSIONS FOR GENERAL-PURPOSE PROCESSORS. In *IEEE Workshop on Signal Processing Systems*. 9–23.

[13] Milo Martin and Aaron Roth. [n. d.]. *Instruction Set Architecture*. https://www.cis.upenn.edu/~milom/cis501-Fall05/lectures/02_isa.pdf Available: https://www.cis.upenn.edu/~milom/cis501-Fall05/lectures/02_isa.pdf [Online; accessed 1-June-2017].

[14] Gaurav Mitra, Beau Johnston, Alistair P Rendell, Eric McCreath, and Jun Zhou. 2013. Use of SIMD vector operations to accelerate application code performance on low-powered ARM and Intel platforms. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 1107–1116.

[15] David Patterson and Andrew Waterman. 2017. *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon.

[16] Rafael Rico, Juan-Ignacio Pérez, and José Antonio Frutos. 2005. The impact of x86 instruction set architecture on superscalar processing. *Journal of Systems Architecture* 51, 1 (2005), 63–77.

[17] Nathan T Slingerland and Alan Jay Smith. June 2005. Multimedia Extensions for General Purpose Microprocessors: A Survey. *Elsevier Microprocessors and Microsystems* 29, 5 (June 2005), 225–246.