Accelerating software radios by means of SIMD Instructions. A case for the AVX2 and AVX512 Extensions

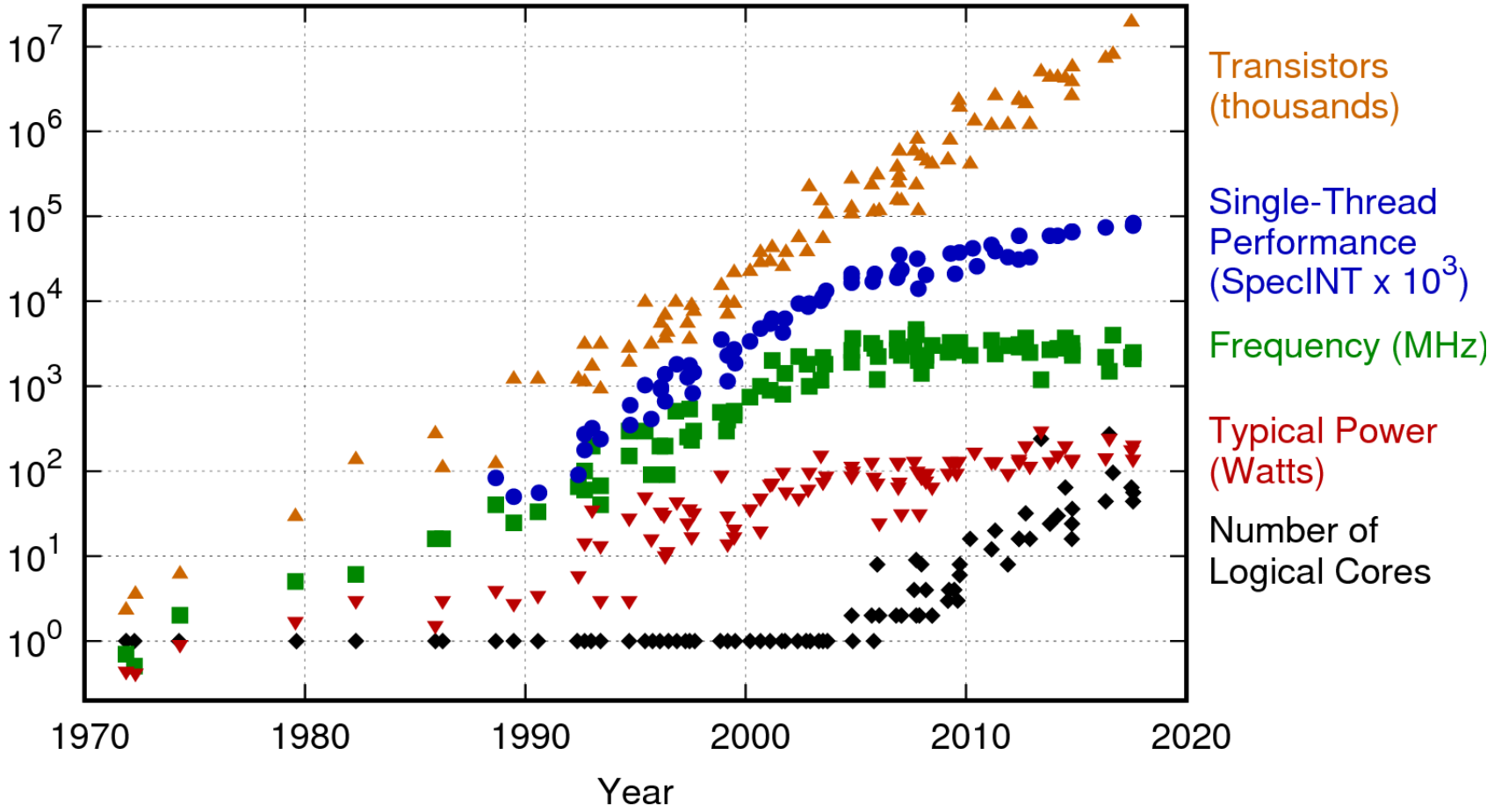
Damian Miralles, The Aerospace Corporation
Jessica Iwamoto, The Aerospace Corporation

GNU Radio Conference 2018
September 17 - 21, 2018

Approved for public release. OTR 2018-00932.

CPU Performance Over Time

CPU Frequency stalled, Number of Logical Cores advanced



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

How does this explain faster execution times for new computers?

Data Parallelization Techniques

An answer for faster execution given stalled CPU Frequency

- **Instruction-level parallelism:** A model in which compilers put considerable effort into the organization of programs such that functional units and paths to memory are busy with useful work.

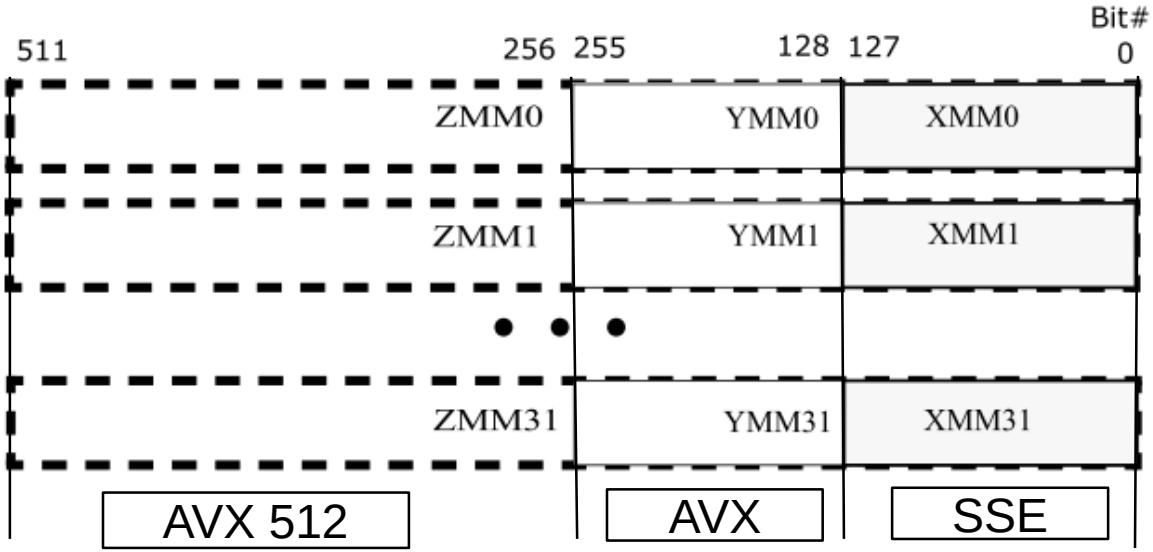
- **Task parallelism:** A fundamental model of architectural parallelism found in shared-memory parallel computers, which can work on several tasks at once by parceling tasks out to different processors by executing multiple instruction streams in an interleaved way in a single processor.

- **Data parallelism:** A model in which instructions can be applied to multiple data elements in parallel, thus exploiting data parallelism. This computer architecture extension is known as Single Instruction, Multiple Data (SIMD).

Work developed is around data parallelism techniques through SIMD

Intel SIMD Extensions

A brief description of releases and registers

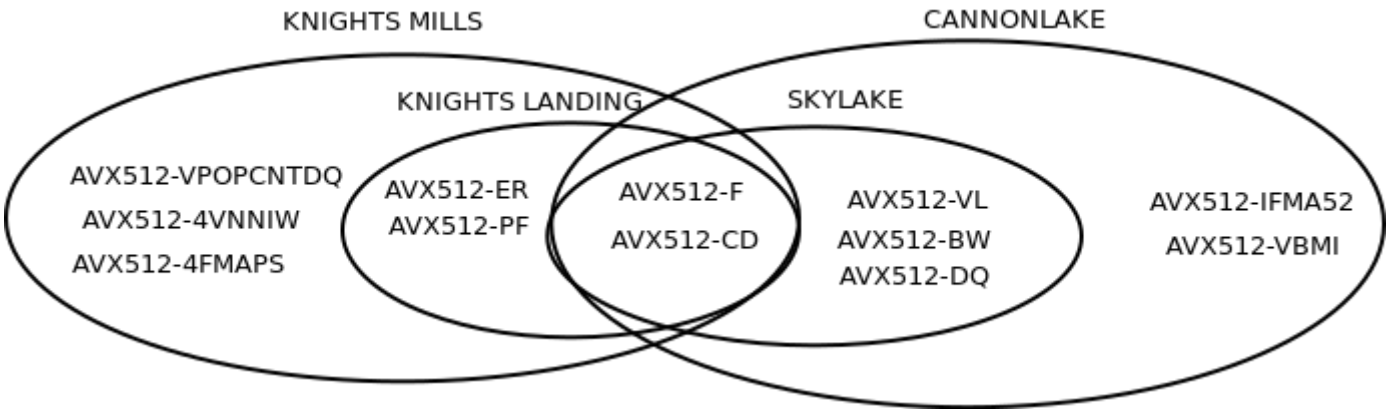


- **MMX:** First extension of Intel supporting SIMD. Initially released to support graphics and multimedia operations (Greene, 1997).
- **SSE:** Introduced eight 128-bit SIMD registers used for floating point and integer operations.
- **AVX/AVX2:** First of its kind in supporting 256-bit SIMD registers size (The Intel Corporation, 2017). Ample support for integer and floating point operations.
- **AVX512:** Extension family which comprises a collection of 128-256-512-bit SIMD registers.

VOLK lacking support for latest SIMD instructions (AVX2, FMA, AVX512)

Intel SIMD Instructions

AVX512 Instructions available on a range of Intel processors

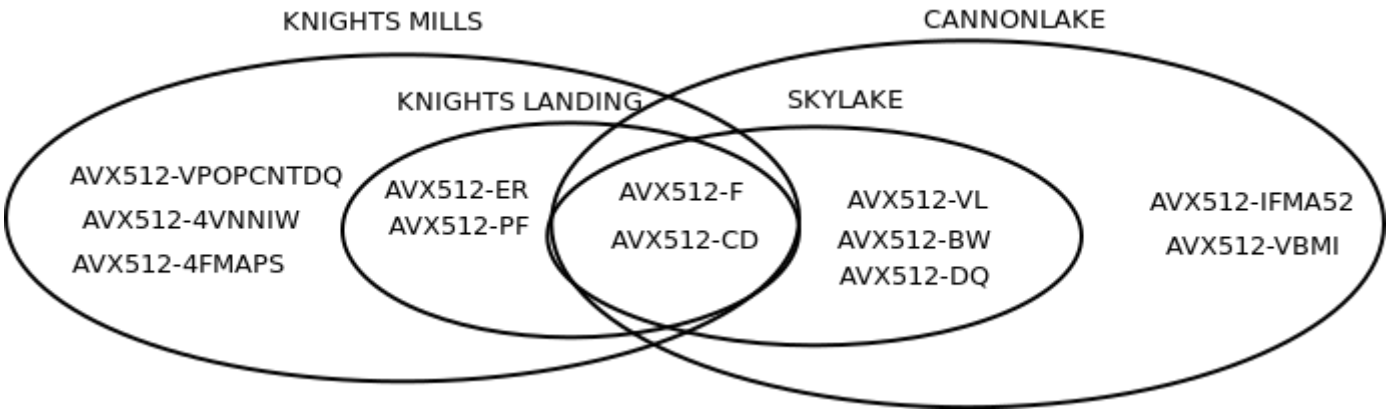


- **AVX512-F:** (“Foundation”) Contains vectorized arithmetic operations, comparisons, type conversions, data movement, data permutation, bitwise logical operations on vectors and masks, and miscellaneous math functions like min/max.
- **AVX512-CD:** (“Conflict Detection”) Contains vectorized operations for memory conflict detection, counting leading zeros count and performing broadcast operations.
- **AVX512-ER:** (“Exponential and Reciprocal”) contains instructions for base 2 exponential functions (i.e., 2x), reciprocals, and inverse square roots.

Work developed in Knights Landing (KNL) with access to AVX512-F, -CD, -ER and -PF

Intel SIMD Instructions

AVX512 Instructions available on a range of Intel processors



- **AVX512-PF:** (“Pre-fetch”) contains pre-fetch instructions for gather and scatter instructions. Even though these instructions provide software pre-fetch support, Knights Landing (KNL) processors have a much heavier emphasis on hardware pre-fetching.
- **AVX512-BW:** (“Byte and Word”) contain instructions that allow support to 8 and 16-bit arithmetic and memory handling operations for the 512-bit registers.
- **AVX512-DQ:** (“Double-word and Quad-word”) contain support for arithmetic operations on Intel 64 and IA-32 Architecture’s double-word (32-bits) and quad-word (64-bits) data types.
- **AVX512-VL:** (“Vector Length”) contain instructions that allow for the AVX-512 to operate on XMM (128-bits) and YMM (256-bits) registers.

Work developed in Knights Landing (KNL) with access to AVX512-F, -CD, -ER and -PF

VOLK Implementation

Intel Intrinsic Instructions and VOLK

- The Intel Intrinsic Instructions are C-style functions that provide access to the SIMD instructions that were previously only accessible through assembly code.
- SIMD extensions not available in all architectures.
- VOLK as a library is an abstraction designed to fix portability problems.
  - *At its base, VOLK has a set of proto-kernels designed for particular platforms, SIMD architecture versions, or run-time conditions.*
  - *The VOLK library compiles all possible proto-kernels supported by the compiler toolchain. At run-time, during the first call to an abstract kernel, VOLK resolves the kernel to a specific proto-kernel.*
  - *VOLK tests the run-time platform for its capabilities to ensure the resolved proto-kernel will run correctly and employs a dynamic rank-ordering to select the best possible proto-kernel (Rondeau et al., 2013).*

VOLK's users call a kernel for performing the operation that is platform/architecture agnostic

VOLK Programming Model

Pseudocode study

- Line 1 is a safeguarded to ensure that only supported architectures are processed.
- Line 6 is reserved for the function name.
  - *Volk: All VOLK proto-kernels will start with the word volk.*
  - *Input tags: refer to the bit count and data types the function will receive, e.g. 32fc defines a 32 bit floating point complex number.*
  - *Kernel descriptor field names the kernel being implemented with the architecture.*
  - *Output tag: describes the bit count and data types of the results of the operation.*
  - *Alignment: holds the type of alignment used for the data loaded from memory.*
  - *Proto-kernel descriptor: identifies the specific architecture being targeted by the kernel, i.e. sse, avx, avx2, avx512f, etc.*
- Line 14 starts the process of data parallelization with SIMD registers. Initially the incoming data is loaded into the register type targeted by the proto-kernel.
- Line 18 will start the chain of operations needed to complete the proto-kernel functionality.
- Line 21 will store the results in the output variables dictated in the function call.
- Line 27 Code in this section will follow the generic implementation with standard library function calls.

```
1 #ifdef LV_HAVE_ARCH
2
3  #include <extension include files>
4
5  static inline void
6  volk_in-tag_kernel-desc_out-tag_align_proto-kernel-desc
7  (input_params){
8
9    // Variable declarations
10    .....
11
12    // Loop unrolling
13    // Data parallelization over SIMD register
14    for() {
15      // Load data from memory to register
16      .....
17
18      // Perform kernel operations
19      .....
20
21      // Store data from register to memory
22      .....
23    }
24
25    // Serial loop
26    // Items remaining after parallelization
27    for () {
28      // Perform kernel operations
29      .....
30
31      // Store data to memory
32      .....
33    }
34  }
35  #endif
```

Intrinsic allows for easy code development, just follow the model!

VOLK Programming Model

Sample code study

- Line 1 shows the targeted architecture of the protokernel, AVX512-F in this case.
- Line 5 shows the functions where:
  - *Volk: required key word volk in use to start each proto-kernel.*
  - *Input tags: defined as 32f x2, indicating that it receives two (x2) input vectors of 32 bits floats values (32f).*
  - *Kernel descriptor: defined as multiply, indicates that a multiply operation is to be performed*
  - *Output tags: defined as 32f, indicating that results of multiplication is to be saved in a single 32 bits float value (32f)*
  - *Alignment tag: input data is to be handle as unaligned (u)*
  - *Proto-kernel descriptor: defined as avx512f, indicates that proto-kernel targets AVX512-F extensions*
- Line 10, all variable declarations are performed, including the special types defined by the compiler to handle the intrinsics operation.
- Line 19 starts the loop unrolling process followed by the loading, multiplication and storage operation.
- Line 32 For vector sizes not multiple of the register size targeted, a serial operation is performed in the remaining data elements.

```
1  #ifdef LV_HAVE_AVX512F
2  #include <immintrin.h>
3
4  static inline void
5  volk_32f_x2_multiply_32f_u_avx512f(
6  float* cVector, const float* aVector,
7  const float* bVector, unsigned int num_points)
8  {
9    // Variable declarations
10   unsigned int number = 0;
11   const unsigned int sixteenthPoints = num_points / 16;
12   float* cPtr = cVector;
13   const float* aPtr = aVector;
14   const float* bPtr=  bVector;
15   __m512 aVal, bVal, cVal;
16
17   // Loop unrolling
18   // Data parallelization over SIMD register
19   for(;number < sixteenthPoints; number++){
20     // Load data from memory to register
21     aVal = _mm512_loadu_ps(aPtr);
22     bVal = _mm512_loadu_ps(bPtr);
23
24     // Perform kernel operations
25     cVal = _mm512_mul_ps(aVal, bVal);
26
27     // Store the results back into the C container
28     _mm512_storeu_ps(cPtr,cVal);
29     aPtr += 16; bPtr += 16; cPtr += 16;
30   }
31
32   // Serial loop. Items remaining after parallelization
33   number = sixteenthPoints * 16;
34   for(;number < num_points; number++){
35     *cPtr++ = (*aPtr++) * (*bPtr++);
36   }
37 }
38 #endif /* LV_HAVE_AVX512F */
```

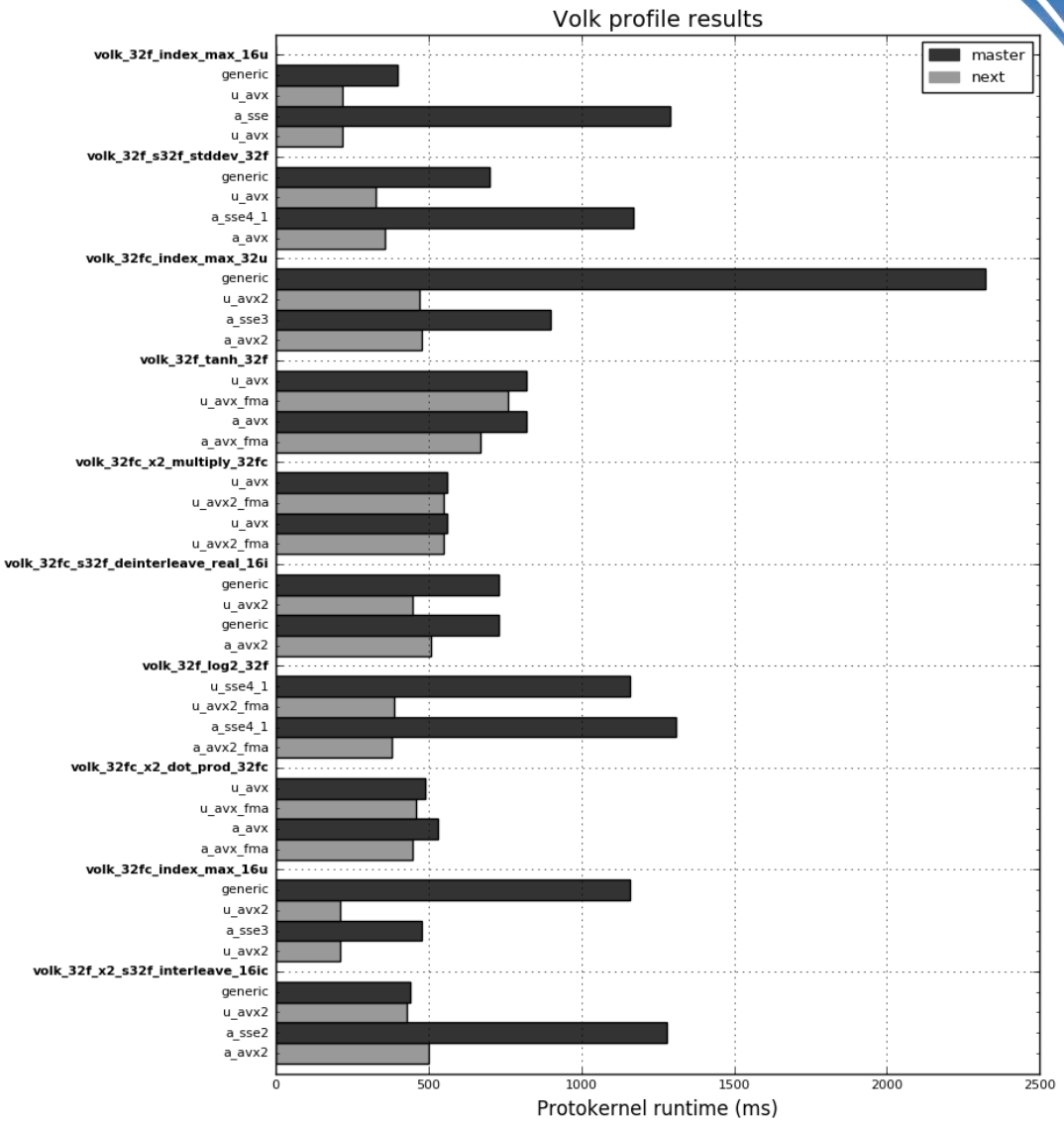
Intrinsic allows for easy code development, just follow the model!

VOLK Kernel Profiling

- Profiled using VOLK library's profiler (volk_profile), which runs a randomized test case on each architecture and checks and times the result.
- Ran on an Intel Xeon Phi KNL processor.
- The following results will compare the 'master' branch of VOLK to our changes, denoted as the 'next' branch.
- We are only interested in the relative time differences between the results of the two branches, not the absolute run time, as this is only a product of the processor clock rate.

Profiling Results

- In the plots, the top two bars show the best unaligned kernels and the bottom two bars show the best aligned kernels.
- 82 kernels were improved by adding AVX, AVX2, FMA, and AVX512-F intrinsics.
- AVX, AVX2 and FMA were added to most kernels, while AVX512-F was only added to a subset.
- Previously, most kernels used generic or SSE as their fastest architecture.
- Generic is especially slow because it does not take advantage of any parallel processing.

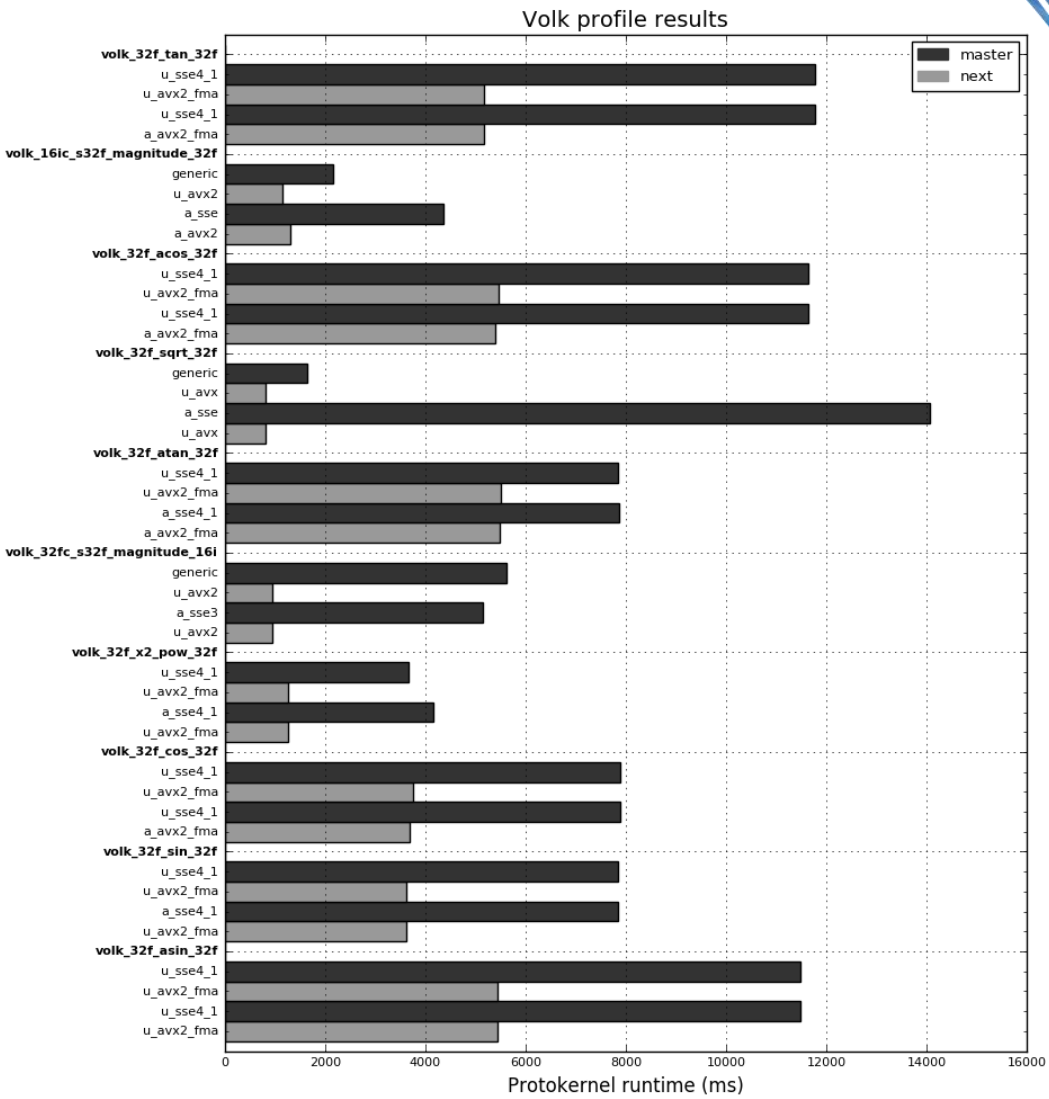


AVX, AVX2, FMA and AVX512-F were added to the majority of the VOLK kernels to speed them up significantly

Profiling Results

AVX, AVX2, and FMA Improvements

- One of the greatest speed improvements was 94% from generic to unaligned AVX in the volk_32f_sqrt_32f kernel.
- Many kernels show an improvement of 50% or more.
- Greater performance gains were achieved in kernels with more mathematical processing.
- Most speedup obtained from generic or SSE architecture to AVX/AVX2, generally smaller speedups between AVX and AVX2 and FMA.

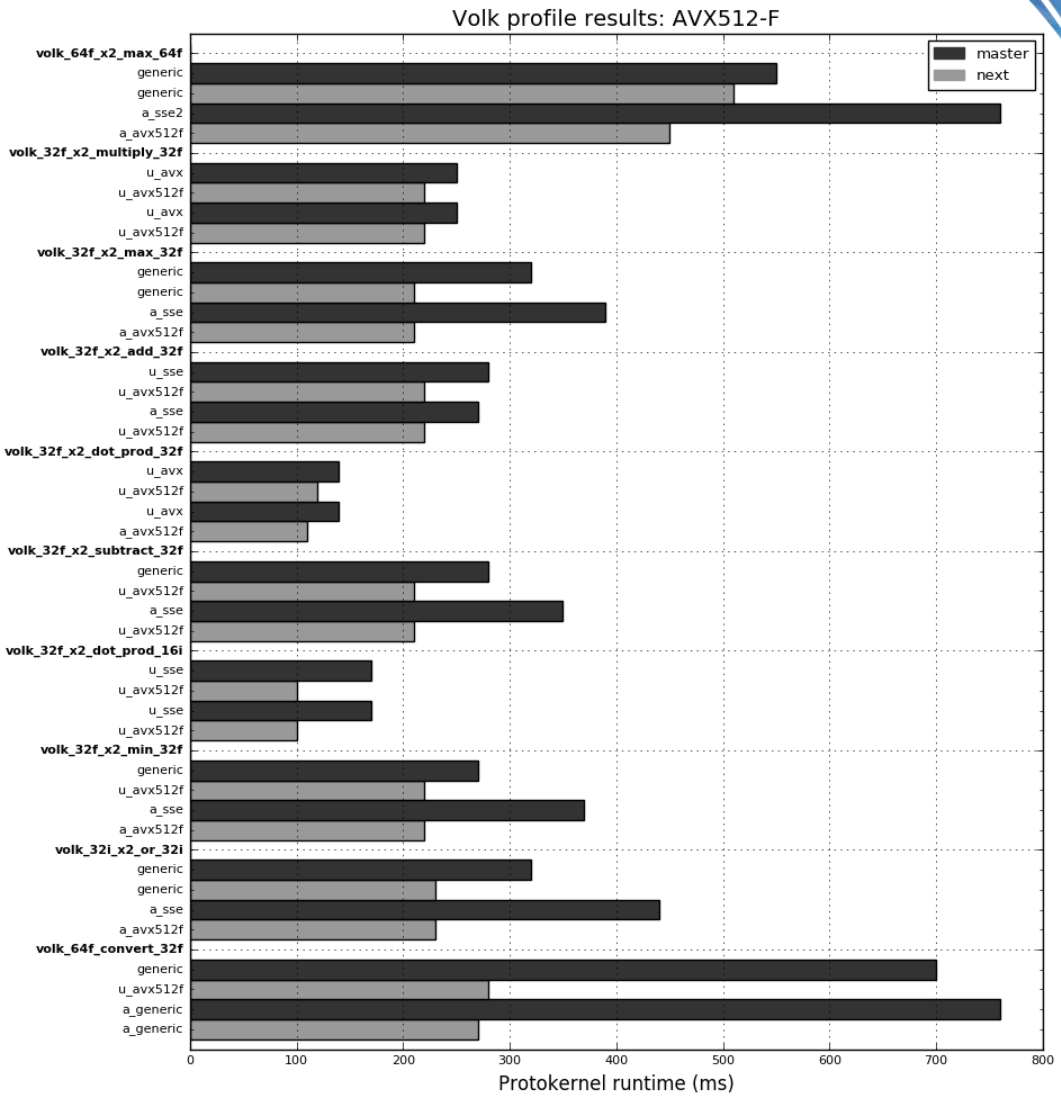


AVX and AVX2 significantly improved VOLK kernels, often by >50%

Profiling Results

AVX512-F Improvements

- Lack of permute, xor, and shuffle operations limited the improvements that could be implemented using AVX512-F.
- Only able to improve simpler operations with AVX512-F.
- AVX512-F shows some improvement over SSE and AVX, but not as much as the AVX2 and FMA improvements on more complex kernels.
- Would expect to see greater improvements with the expanded instructions of other versions of AVX512.



AVX512-F improvements were smaller but limited by fewer intrinsics available in the AVX512-F set

Conclusions

- Addition of AVX2 and AVX512 to VOLK will improve the performance of applications such as GNU Radio and GNSS-SDR.
- Improvements only tested on Xeon Phi x86 processors, but cross-compatibility with similar architectures should allow them to work on architectures similar to Xeon processors.
- AVX512 capabilities only present in x86 processors, but this work could possibly be extended to similar technologies in embedded architectures, particularly ARM processors.
- Further work will focus on supporting the remaining AVX512 flavors, including AVX512-BW, AVX512-DQ, etc, with the intention of further improving DSP speed in SDR applications.

Keeping VOLK updated with latest extensions will increase software radio applications running in real time

Acknowledgments

- The authors would like to thank Eric K. Lai and the Digital Communications Implementation Department (DCID) at The Aerospace Corporation for their contributions to this work.
- They would also like to thank the GPS Directorate Advanced Technology Branch for sponsoring this investigation of software radios.

References

- Fernandez, Carles, Arribas, Javier, and Closas, Pau. Accelerating GNSS Software Receivers. In Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+2016), pp. 44–61, Portland, OR, 2016.
- Fernandez–Prades, C., Arribas, J., Closas, P., Aviles, C., and Esteve, L. GNSS-SDR: An open source tool for researchers and developers. In Proc. of the ION GNSS 2011 Conference, Portland, Oregon, Sept. 2011.
- Greene, M. A. Pentium(r) processor with mmx/sup tm/ technology performance. Proceedings IEEE COMPCON 97. Digest of Papers, pp. 263–267, 1997.
- Karl Rupp. 42 Years of Microprocessor Trend Data, 2018. URL https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.
- Naishlos, Dorit. Autovectorization in GCC. Proceedings of the 2004 GCC Developers Summit, pp. 105–118, 2004.
- Philip E. Ross. Why CPU Frequency Stalled -, 2008. URL https://spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled.
- Rondeau, T, McCarthy, N, and O'Shea, T. SIMD Programming in GNU Radio: Maintainable and User-Friendly Algorithm Optimization with VOLK. WinnForum's SDR conference, 86:101–110, 2013. URL http://50.19.239.22/redmine/attachments/422/volk.pdf.
- The Intel Corporation. Intel Intrinsics Guide, 2012.URL https://software.intel.com/sites/landingpage/IntrinsicsGuide/{#}techs=AVX,AVX2{&}expand=3924,3894,2758.
- The Intel Corporation. IntelR 64 and IA-32 Architectures Software Developer ' s Manual Volume 1 : Basic Architecture. Technical Report 253665, The Intel Corporation, 2017.
- Wikipedia contributors. Avx-512 — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=AVX-512&oldid=837564204 ,2018. [Online; accessed 23-April-2018].
- Zhang, Bonan. Guide to Automatic Vectorization with Intel AVX-512 Instructions in Knights Landing Processors. Technical report, Colfax International, 2016. URL https://colfaxresearch.com/knl-avx512


THANK YOU
Q/A