LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T2100460-v1	Advanced LIGO	11/5/2021

A Biquad Implementation Using Advanced Vector Extensions

Daniel Sigg

Distribution of this document:
LIGO Scientific Collaboration

This is an internal working note
of the LIGO Laboratory.

California Institute of Technology
LIGO Project – MS 18-34
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 159
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

http://www.ligo.caltech.edu/

1   Introduction

Intel based CPU have implemented advanced vector extensions since a while. We look at AVX2 and AVX-512[1], AVX2 has become common, whereas AVX-512F is newer and less widely available. Our newest set of front-end computers based on Cascade Lake processors, like the Intel® Xeon® W-2245 Processor, support both.

Advanced vector extensions allow to process multiple floating-point operations in parallel: 4 in case of AVX2 and 8 in case of AVX-512F. These operations also use a separate CPU register file. There are 16 registers for AVX2 and 32 for AVX-512.

A single IIR filter and its implementation as a cascaded set of biquad sections cannot easily be parallelized. Instead, we investigate how to implement multiple IIR filters in parallel consisting of a fixed set of 3 biquad sections. The main use case for this parallelization would be the decimation filters that reduce the data rate from a fast ADC running 2^{19} Hz.

2   Thermal Throttling

Vector extensions produce a lot of heat and potentially lead to thermal throttling of the CPU. Figure 1 shows the potential impact on CPU clock for a CPU that is from the same family as the W-2245 with the same core count and similar frequencies (3.9 vs 3.6 GHz). For reference see en.wikichip.org[2], and online articles[3,4].

For our front-end computers we disable turbo mode in the BIOS, so they never run faster than their normal base frequency of 3.9 GHz. From the table below, we can conclude that it is unlikely that thermal throttling will be an issue for AVX2 and maybe effect performance if AVX-512 is run on all cores. In our targeted use case, these operations would only run on the IOP and hence one core. So, **we don't envision any issues due to thermal throttling.**

Mode	Base	Turbo Frequency/Active Cores							
		1	2	3	4	5	6	7	8
Normal	3,600MHz	4,400MHz	4,400MHz	4,300MHz	4,300MHz	4,300MHz	4,300MHz	4,300MHz	4,300MHz
AVX2	3,000MHz	4,000MHz	4,000MHz	3,900MHz	3,900MHz	3,900MHz	3,900MHz	3,900MHz	3,900MHz
AVX512	2,600MHz	3,800MHz	3,800MHz	3,600MHz	3,600MHz	3,500MHz	3,500MHz	3,500MHz	3,500MHz

Figure 1: Thermal Throttling on a Xeon Gold 6244 CPU 3.6GHz with 4.4GHz turbo.

---

[1] https://en.wikipedia.org/wiki/Advanced_Vector_Extensions

[2] https://en.wikichip.org/wiki/intel/xeon_gold/6244

[3] https://extensa.tech/blog/avx-throttling-part1/

[4] https://lemire.me/blog/2018/09/07/avx-512-when-and-how-to-use-these-new-instructions/

2.1 Implementation

We use the LIGO biquad implementation as a starting point. See the zip file in T2100460 for the C/C++ code that was used. We first slightly rewrite it by using array indices instead of pointer arithmetic. Next, we then implement both AVX2 and AVX-512 versions. The filter coefficients don't need to be identical, but the number of sections needs to be the same. Finally, we write versions that use identical coefficients and exactly 3 sections, implement decimation and work on longer strides of data. The later versions are targeted a ADC data that is arranged in vectors of fixed length corresponding to the number of ADC channels and is stacked up 8 samples deep that need to be filtered and decimated.

Table 1: Tested Biquad Implementations.

iir_filter_biquad	Copied from fm10Gen.c.
biquad	Same as above but using array indices instead of pointer arithmetic.
biquadAVX2	AVX2 implementation of biquad, working on 4 input samples in parallel.
biquadAVX2double	AVX2 implementation of biquad, working on 8 input samples in parallel, this function uses the same parameters as biquadAVX512 and can be interchanged.
biquadAVX512	AVX512 implementation of biquad, working on 8 input samples in parallel
biquad_stride1	Biquad implementation that works on a longer stride of data and includes a decimation. All filter coefficients are identical.
biquad_stride2	Poor man's way to parallelize the above biquad_stride1 working on 2 samples intermixed.
biquad_stride4	Working on 4 samples intermixed.
biquad_avx2_stride4_section3	AVX implementation of biquad_stride1, working on 4 samples in parallel.
biquad_avx2_stride8_section3	Poor man's way of parallelization, working on 2 vectors intermixed.
biquad_avx2_stride16_section3	Working on 4 vectors intermixed.
biquad_avx2_stride32_section3	Working on 8 vectors intermixed.
biquad_avx512_stride8_section3	AVX-512 implementation of biquad_stride1, working on 8 samples in parallel.
biquad_avx512_stride16_section3	Poor man's way of parallelization, working on 2 vectors intermixed.
biquad_avx512_stride32_section3	Working on 4 vectors intermixed.

3   Results

We run two version on the Xeon W-2245: once compiled using the RGC compiler flags, "-O -ffast-math -m80387 -msse2 -fno-builtin-sincos -march=native", and once using a higher optimization level, "-O5 -march=native". The native architecture flag has been added to both, otherwise AVX2 and AVX-512 wouldn't be available. The biquad functions are run many times to get a stable execution time.

Table 2: Test Results Xeon W-2245

Function	RCG flags		Performance	
	Time(s)	Speedup	Time(s)	Speedup
iir_filter_biquad	13.8	1.0	8.4	1.6
biquad	10.7	1.3	8.6	1.6
biquadAVX2	2.42	5.7	2.32	5.9
biquadAVX2double	2.62	5.3	2.39	5.8
biquadAVX512	1.86	7.4	1.53	9.0
biquad_stride1	14.8	0.9	8.0	1.7
biquad_stride2	9.9	1.4	6.5	2.1
biquad_stride4	9.6	1.4	6.4	2.2
biquad_avx2_stride4_section3	2.92	4.7	1.76	7.8
biquad_avx2_stride8_section3	2.41	5.7	1.93	7.2
biquad_avx2_stride16_section3	1.85	7.5	1.75	7.9
biquad_avx2_stride32_section3	1.73	8.0	2.32	5.9
biquad_avx512_stride8_section3	1.91	7.2	1.35	10.2
biquad_avx512_stride16_section3	1.33	10.4	1.05	13.1
biquad_avx512_stride32_section3	1.22	11.3	1.18	11.7

4   Conclusions

AVX2 and AVX-512 can provide a good speedup over the current implementation of the internal decimation filters.

We would profit from being able to compile the front-end code with a higher optimization level.

Both AVX2 and AVX-512 implementations are less dependent on the selected optimization level.

At high optimization level the speedup by AVX2 is between 3-5, whereas for AVX-512 it is between 5 and 8. At the RGC optimization level, the AVX2 speed up is between 2.5 and 8, whereas the AVX-512 speedup is between 7 and 11. The real world speedup of an IOP is of course less, since it has to perform other operations, but with both a fast and a low noise ADC running at 2^{19} Hz the burden of the decimation filters is significant.