AVX512: First Look

W. D. Cotton, April 8, 2020

***Abstract*—Some aspects of the 512 bit version of the "Advanced Vector eXtensions" (AVX512) are examined in the context of radio interferometric imaging. These are basically doubling the vector length to 16 floats and a strengthening of the vector instruction set. The 16 float vector length is an excellent match to the 7x7 complex gridding convolution function used in Obit imaging. The tests presented here show a 15-20% performance gain of AVX512 (512 bit) over AVX2 (256 bit) in major interferometry applications.**

***Index Terms*—vectors, interferometry, performance**

I. Introduction

Modern CPUs can use operands the width of the memory bus. When the memory bus is wider than the size of a given data type, this can be used to perform parallel operations or SIMD = "Single Instruction, multiple data" essentially forming a vector engine. There were several incarnations of the SSE instructions for 128 bit buses (4 floats or 2 doubles) and the initial SIMD operations for 256 bit buses (8 float, 4 double) was "AVX" ("Advanced Vector eXtensions") and AVX2. For 512 bit memory busses there is AVX512 with a further strengthening of the instruction set, this includes "fused multiply/add" instructions which save a vector store and load for operations like updating the imaging grid in interferometry. The functionality can be accessed via 1) an optimizing compiler, 2) assembly instructions or 3) "intrinsics" which are c function calls corresponding to an assembly instruction. The expensive, but buggy Intel compiler is good at implementing these features but the freebe gcc lags substantially. The testing reported here used the "intrinsics". A similar memo describing AVX2 is [1].

Vectors of sine/cosine pairs are frequently used in interferometry, especially in the "DFT" approximation of a Fourier Transform. Public domain libraries using SSE/AVX/AVX2/AVX512 for trigonometric functions are available (https://github.com/juj/MathGeoLib/blob/master/src/Math/sse_mathfun.h, https://github.com/reyoung/avx_mathfun) and https://github.com/aff3ct/MIPP/blob/master/src/src/math/avx512_mathfun.h (and .hxx)). This memo evaluates using these libraries and other tests of AVX512 in the Obit package [2] [1].

II. AVX512 Features

The enhancements of interest here are the increased vector length (to 16 floats/8 doubles) and the enhanced instruction set including "fused multiply/add". Testing used the sincos function in both a simple standalone mode and as implemented in the Obit interferometry software. Also tested was the use of the enhanced vector length in uv data gridding.

National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

[1]http://www.cv.nrao.edu/∼bcotton/Obit.html

TABLE I
Sin/cos timings

method	total Run time sec.	Rate
sinf+cosf	211.5	1.000
sincosf	142.2	1.5
SSE	33.8	6.2
AVX	33.8	6.2
AVX2	16.5	12.8
AVX512	10.4	20.4

III. Standalone Tests

These tests consist of simple c programs performing a function a large number of times and using the unix time utility to determine the run and cpu times. The tests were run on a workstation with twenty–four Intel Xeon Gold cores running @ 3.0 GHz and which supports SSE, AVX, AVX2, and AVX512 instructions. However, the standalone tests were performed single threaded. Compilation used gcc-8.3.1 with O3 optimization.

A. Sine/Cosine Tests

Usage of sines and cosines in interferometry is relatively simple and only accurate values are needed. The properties of continuity, differentiability etc. provided by the standard c libraries are overkill and the simplified version provided by (avx)avx512_mathfun.h is quite adequate. As described in [1], the instruction set in basic AVX is weak enough that it does not give a substantial improvement over SSE. The test program is given in Figures **??** and III-A.

The test program was compiled with a number of options using gcc-8.3.1. All options were tested comparing with the system library sincosf function and then a number of test timing runs were made without this comparison; the setup time is negligible compared to the run time of each test. The AVX512 comparisons gave an average difference of 1.646e-16, and RMS difference of 1.366e-11 a maximum error of 5.96e-08 and sum of the sine residuals of -1.646e-16. Timings are given in Table I including the ratio of the speed compared to using the library sinf + cosf routines ("rate"). The poor performance of AVX appears to be the result of having to use SSE integer functions with more shuffling data around between 256 and 128 bit variables. Using AVX512 gives a 60% improvement over AVX2 and a factor of 20 over sinf+cosF. Using the library sincosf rather than sinf + cosf gives a 50% improvement.

```
/* Test AVX2/512 improvements */
#include "ObitSinCos.h"
void sincosf(float x, float *sin, float *cos);
/* use library sinf & cosf */
void sincos1(ollong n, ofloat *x, ofloat *s, ofloat *c);
/* use library sincosf */
void sincos2(ollong n, ofloat *x, ofloat *s, ofloat *c);
int main ( int argc, char **argv )
{
  ofloat sss,ccc, d, dmax;
  ofloat phase, randnorm, fazRange;
  ofloat a[50000], c[50000], s[50000];
  odouble sum, sum2, count, rms;
  ollong i, j, m, n;
  gboolean doCompare = FALSE;  /* Do comparison with standard library */

  n = 9000000000;  m = 5000;
  sum = sum2 = count = 0.0;
  dmax = 0.0;
  fazRange = 100.0;
  randnorm = fazRange / RAND_MAX;
  /* Fill array */
  for (j=0; j<m; j++) {
    /* Random phase between fazMin, fazMax */
    phase = (rand()-RAND_MAX/2)*randnorm;
    a[j] = phase;
  }
  /*n=0; *//*debug*/
  sum = 0.0; j=0;
  for (i=0; i<n; i+=m) {
    /* Compute vector  */
    ObitSinCosVec (m, a, s, c);
    /*sincos1 (m, a, s, c);  sinf + cosf */
    /*sincos2 (m, a, s, c);   sincosf */
    /*for (j=0; j<m; j++) sum += s[j];*/
    sum += a[j]; /* Don't optimize away */

    /* Compare  */
    if (doCompare) {
      for (j=0; j<m; j++) {
sincosf(a[j], &sss, &ccc);
d = (sss-s[j]);
dmax = MAX (dmax, fabs(d));
sum += d;
sum2 += d*d;
count++;
d = (ccc-c[j]);
dmax = MAX (dmax, fabs(d));
sum += d;
sum2 += d*d;
count++;
      }
      sum  /= count;
      sum2 /= count;
      rms = sqrt (sum2 - sum*sum);
    }
  }
```

Fig. 1. Sine/Cosine Test Program, part 1

```
 if (doCompare)
    fprintf (stdout,"Avg difference %lg rms %lg max err=%g\n", sum, rms,dmax);

   fprintf (stdout,"sum=%g\n", sum);
  return 0;
} /* end main */
/* use library sinf & cosf */
void sincos1(ollong n, ofloat *x, ofloat *s, ofloat *c) {
  ollong i;
  for (i=0; i<n; i++) {s[i]=sinf(x[i]); c[i]=cosf(x[i]); }
} /* end sincos1 */
/* use library sincosf */
void sincos2(ollong n, ofloat *x, ofloat *s, ofloat *c) {
  ollong i;
  for (i=0; i<n; i++) {sincosf(x[i], &s[i], &c[i]);}
} /* end sincos1 */
```

Fig. 2. Sine/Cosine Test Program cont'd

TABLE II
GATHER TIMINGS

method	total Run time sec.	CPU time sec.
manual	38.9	38.9
gather	27.1	27.1

TABLE III
UVSUB TIMINGS

method	REAL time sec.	CPU time sec.
AVX2	972	21238
AVX512	819	17007
GPU	101	not measured

B. Gather Tests

The tests of the AVX512 "gather" function used the program in Figure III-B. This was compiled with either the "Manual" vector load or the "Gather" vector load sections commented out and the execution timed. The loaded vector was "stored" to keep the operation from being optimized away. The AVX512 gather function is substantially faster (40%) than the "manual" version.

C. Interferometry Tests

Interferometry tests were performed in Obit using various combination of compiler and AVX options on the same workstation as used for the standalone tests. This machine has an SSD disk which was used for all input files and 256 GByte of memory 150 GByte of which was used as RAM disk for output and scratch files. This workstation also has a GPU (NVIDIA GeForce RTX 2040 TI with 13,056 cores).

All programs were allowed to use tthe twenty-four threads for multi-threaded operations. Many routines which had been adapted to use AVX/AVX2 were also modified to use AVX512, expecially the gridding and SkyModelMF (degridding) routines. More functionally was vectorized in the gridding and degridding routines as the residual scalar operations were becoming a larger fraction of the total run time.

Obit implements segments of code using AVX512/AVX2/AVX/SSE using #ifdef statements giving preference in that order. Compiling Obit software enabling AVX needs compiler options "-DHAVE_AVX=1 -mavx", for AVX2, "-DHAVE_AVX2=1 -mavx2" and for AVX512, "-DHAVE_AVX512=1 -mavx512f"

D. UVSub

Task UVSub subtracts the Fourier transform of a CLEAN model from a visibility data set and is used heavily for "degridding". If this is done using the Cmethod='DFT" method, the processing for a large CLEAN model is dominated by the calculation of sin/cos pairs and is a good test of the avx/avx512_mathfun libraries. The data set used for this test was 9 hours of MeerKAT (57 antenna) data in continuum mode. The data set used had 1,369,266 visibilities each with 952 channels and dual polarization although roughly half the data are flagged. The CLEAN model subtracted had 1934 wideband CLEAN components. AVX2 and AVX512 were used as was the GPU and the results shown in Table III.

As expected from Section III-A, the AVX512 results are better than the AVX2 results by $\sim$20% but by less than the difference between the pure sincos tests indicating that the residual scalar operations are taking significant time. The GPU based run was far faster than either AVX2 or AVX512 as almost all of the calculations had been offloaded onto the GPU (see [3]).

E. MFImage

Imaging is a more general test with I/O dominated, scalar and a variety of vector operations. For problems with large datasets (e.g. MeerKAT) the run time is dominated by the

```
#include <immintrin.h>
#include <stdio.h>
typedef __m512  v16sf; // vector of 16 float (avx512)
typedef __m512i v16si; // vector of 16 int   (avx512)
int main ( int argc, char **argv )
{
  float a[16000], dump[1600], sum=0;
  long  i, j, m, n;
   /* reordering - Multiply x 10 to keep in separate 256 bit mem reads*/
  long re[16] = {5*10,2*10,1*10,3*10,6*10,7*10,0*10,4*10,
13*10,10*10,9*10,11*10,14*10,15*10,8*10,12*10};
  v16si order;
  v16sf vector, vector2;
  n = 90000000;  m = 1600;
  order = _mm512_set_epi32(re[0], re[1], re[2],re[3], re[4], re[5], re[6], re[7],
re[8], re[9], re[10],re[11], re[12], re[13], re[14], re[15]);
  /* Fill array with random numbers */
  for (j=0; j<m*10; j++) {
    /* Random values */
    a[j] = (rand()-RAND_MAX/2)*100. / RAND_MAX;
  }
  /* loop loading in reverse order */
  for (i=0; i<n; i++) {
    for (j=0; j<m; j+=16) {
      /* Manual*/
      vector = _mm512_set_ps(a[j+re[0]], a[j+re[1]], a[j+re[2]], a[j+re[3]],
                             a[j+re[4]], a[j+re[5]], a[j+re[6]], a[j+re[7]],
     a[j+re[8]], a[j+re[9]], a[j+re[10]], a[j+re[11]],
                             a[j+re[12]], a[j+re[13]], a[j+re[14]], a[j+re[15]]);
      _mm512_storeu_ps(&dump[j], vector);
      /* Gather
      vector2 = _mm512_i32gather_ps(order, (float*)&a[j], 4);
      _mm512_storeu_ps(&dump[j], vector2);*/
      sum+= dump[j];
    } /* end inner loop */
  } /* end outer loop */
  fprintf (stderr,"sum=%f\n",sum);
  return 0;
} /* end main */
```

Fig. 3. Gather Test Program

gridding and degridding operations. The wide-band imager MFImage was used for these tests with different AVX/GPU and compiler options to do a shallow CLEAN on a MeerKAT dataset.

The initial AVX implementation was described in [4]. The gridding convolution kernal primarily used in Obit imaging is a 7×7, separable, prolate spheroid and the segments updated in the uv grid are 8 (for end effects) real/imaginary pairs of floats. The AVX2 implementation uses a pair of 8 float vectors for the grid update; for AVX512, this becomes a single 16 float vector and the gather/scatter functions were used. Thus, the inner gridding loop should be twice as fact in AVX512 as AVX2. However, with the increased speed of the inner loop, the previously scalar operations became the dominant cost (Amdahl's Law). More of the previously scalar operations were vectorized for both AVX2 and AVX512 which reduced the cost of the scalar parts but they are still a major contributor to the run time.

This test used the visibility data set used in the UVSub test and created a 4822x4822x16 image using 109 facets. The MFImage test makes extensive use of both gridding and degridding, gridding is only implemented in the CPU (AVX/AVX512) but the degridding is also implemented in the GPU, see section III-D.

CLEANing proceeded for 5000 CLEAN components; results for the various tests were essentially identical; however, CLEAN is nonlinear and minor variations in numerical noise caused different numbers of major cycles which affects the runtime. To evaluate the relative speeds of the gridding, the real time for the first set of beams and images ("1st image")

TABLE IV
MFIMAGE TIMINGS

method	REAL time min	CPU time min	Maj. cycles
AVX2	126.9	1756.0	13
AVX512	104.3	1430.9	13
AVX512 icc	80.7	1130.5	11
AVX2+GPU	106.5	1257.0	13
AVX512+GPU	69.1	799.5	11
AVX2 1st image	27.4		
AVX512 1st image	23.9		
AVX512 icc 1st image	23.1		

are also compared.

Some testing was done using the Intel compiler (icc 19.1.1.217) in addition to AVX as this compiler should be more aware of the hardware capabilities than gcc. Unfortunately, this version of icc is also buggy and full optimization causes MFImage to produce very bad results. This behavior was localized to module ObitUVGrid which produces good results with the "-O1" optimization option. This module is relatively high level and mostly calls other modules to do the heavy lifting so a lower level of optimization is less significant.

Execution times are given in Table IV. All tests used gcc except "AVX512 icc" and "AVX512 icc 1st image" which used icc. The "method" column gives the version of AVX and whether or not the GPU was used.

IV. DISCUSSION

Tests were presented comparing the use of the 512 bit (16 float) AVX512 with 256 bit (8 float) AVX2 vector operations. Many of the cpu intensive modules with relatively simple loops were upgraded to use AVX512 as a compile time option when available.

Stand alone tests were performed timing sin+cos performance using the mathfun vector libraries and of the gather operation in AVX512. The AVX512 version of the sincos test ran 1.6 $\times$ faster than the AVX2 version and 20 $\times$ faster than calling the library sinf and cosf routines. The AVX512 "gather" test ran 1.4 times faster than the older "manual" method of loading arbitrary words from memory.

Testing of a sky model calculation (AKA "degridding") using the DFT method used Obit task UVSub on a largish MeerKAT dataset. AVX2, AVX512 and GPU based versions were compared. The AVX512 version ran 1.19 $\times$ faster than the AVX2 version but still took 8 $\times$ longer than the GPU based version. This test is an illustration of Amdahl's Law that as you speed the vector part of your problem, the scalar parts become more dominant. Prior to these tests, much of the previously scalar parts of the model calculations were vectorized. As essentially all of the calculation was done in the GPU when it is used, it is effectively totally parallel.

A comparison of the imaging tests are somewhat muddled by the nonlinear nature of CLEAN causing a range of numbers of major cycles which significantly affected the run times. GPU enhanced runs clearly were faster in the cases which allow a simple comparison. To avoid this problem, the runtimes of the first set of images and dirty beams were compared. This step is linear and should be the same for all implementations and is dominated by the gridding process. The AVX512 implementation was 1.15 $\times$ faster than AVX2 which increased to 1.19 $\times$ using the Intel compiler. An interesting comparison is between the AVX512+GPU and AVX512 icc tests which had the same number of major cycles; the AVX512 icc implementation was 86% the speed of the GPU based version which is quite different from the factor of 8 in the UVSub test using the same dataset. This is another case of Amdahl's Law as in this shallow CLEAN the ratio of gridding to degridding was rather large and the greater speed of the GPU less important.

On the basis of tests presented here, the use of AVX512 gives a 15-20% performanence enhancement in interferometry applications over AVX2.

ACKNOWLEDGMENT

I would like to thank Tracy Halstead for help with various compiler issues.

REFERENCES

[1] W. D. Cotton, "AVX2: First Look," *Obit Development Memo Series*, vol. 49, pp. 1–4, 2017. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/AVX2.pdf

[2] W. D. Cotton, "Obit: A Development Environment for Astronomical Algorithms," *PASP*, vol. 120, pp. 439–448, 2008.

[3] W. D. Cotton, "Comparison of GPU and Multithreading for Interferometric DFT Model Calculation," *Obit Development Memo Series*, vol. 35, pp. 1–5, 2014. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/GPUDFTv2.pdf

[4] ——, "Comparison of GPU, Single- and Multi-threading for Interferometric Gridding," *Obit Development Memo Series*, vol. 36, pp. 1–14, 2014. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/GPUGrid.pdf