Notes on icc and AVX

W. D. Cotton, May 24, 2019

***Abstract*—Timing tests are described comparing various combinations of optimizing compilers and hand coded AVX intrinsics in a multi–threaded program. For the radio interferometric imaging test case, the hand coded routines reduced the run time by 10% for both the gcc and icc compiler runs. The Intel compiler, icc, reduced the run time by 20% with and without the AVX intrinsics enabled. Both using a good optimizing compiler and hand-coding critical routines make a substantial improvement in performanence.**

***Index Terms*—compilers, vectorizing**

I. Introduction

MANY operations in imaging radio interferometric data are quite compute intensive but, if properly implemented, allow a fair degree of prallelization. Both vectorication and multi–threading are extensively used for performance gains. This memo discusses vectorization as implemented in the program MFImage in the Obit package [1] [1]. Both use of the Intel optimizing compiler, icc, and hand-coded AVX "intrinsics" routines are examined.

II. Shared Memory Parallelism

There are two simple (shared memory) means of parallelism in modern CPUs, multi–threading to spread work over multiple cores and vectorization to perform multiple arithmetic operations in the same cycle on a given core. This memo only considers the latter and the influence of compilers in utilizing the vector hardware. Multiple thread operations also can make heavy use of vectorized functions.

The vector units operate on blocks of data the size of the memory bus. For the older models, this is 128 bits corresponding to 4 floats or 2 doubles. Since most radio interferometry operations are on floats, this vectorication gets up to a factor of 4 performance improvement. More recent computers have a 256 bit wide bus and vector oparations are on 8 floats. 128 bit vector operations are implemented in various versions of SSE and the 256 bit vector operations in a version of AVX.

The two main ways of implementating vector operations in software are 1) letting the compiler figure it out or 2) using the "intrinsics", c function calls that get mapped to vector assembly-level instructions. Different compilers have different levels of knowledge of the vector units hence, produce code with different levels of performance. The expensive Intel compiler, icc, generally gives the best performance as it was developed by the CPU chip maker but the freebee gcc lags behind in vectorization. Software using the intrinsics can be hand coded and run on any compiler and can, in principle, even out perform a good compiler.

This note compares Obit software which uses extensive hand–optimized routines employing the intrinsics compiled with various options using icc and gcc. Timings of various software options and compilers are given.

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

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

III. Obit Intrinsics Implementation

Obit software is intended to be run on machines using 128 or 256 bit (or more) memory buses hence requires either SSE or AVX intrinsics. Which set is controlled by #ifdefs using command line options to control. SSE is always assumed to be available but AVX and AVX2 are optional.

Various compute intensive functions in Obit have optimized SSE/AVX implementations. Heavy use is made of the sincos function from the opensource avx_mathfun.h and sse_mathfun.h libraries and many float array (ObitFArray class) functions have avx implementations. A vectorized routine critical to radio interferometry is the uv data gridding routine which has a hand-coded avx version [2].

IV. Compilers

The tests described below used either the Intel compiler icc (version 16.0.2) or the GNU gcc compiler (version 4.4.7). Use of AVX is enabled/disabled using compiler options set in the Makefiles; -DHAVE_SSE=1, -DHAVEAVX=1 (-DHAVEAVX2=1 for AVX2) control the Obit #ifdefs and -msse -mavx -mavx2 enable SSE, AVX and AVX2 in the compiler. Compilation used the -O3 optimization compiler switch.

V. Test Case

The test case is a run of Obit wideband/widefield imaging task MFImage run on a small EVLA dataset. This data had 19,147 visibilities in S Band with 16 IFs averaged to 54 channels. Imaging was to a radius of 0.25 degrees producing an image of 1062×1062 pixels using 7 facets; CLEANing proceeded to 1500 components. This test case contains a good mix of I/O and CPU bound components, scalar, vector and multi-threading.

The tests were run on zuul05 running Redhat 6, this has 16 × 2 GHz cores with AVX, 64 GByte RAM, RAID and SSD disks and a GPU. The GPU was not used in these tests. Output and scratch files were on the SSD and up to 16 threads were allowed. No other users were on this machine at the time of the tests.

TABLE I
TIMING TESTS

compiler	Obit AVX	Real sec	CPU sec	rel. Real	rel CPU
gcc	N	135	656.2	1	1
gcc	Y	123	535.0	0.91	0.82
icc	N	110	471.3	0.81	0.72
icc	Y	94	346.3	0.70	0.53

Notes: “rel Real” and “rel CPU” are the ratio of the time to that from the gcc/no AVX test.

VI. Timings

Timings are the average of multiple runs using the Obit real and CPU time values. All test runs allowed up to 16 parallel threads. Results are given in Table I where “compiler” is the compiler used, “Obit AVX” is “Y” or “N” indicating if the Obit AVX intrinsics were enabled, “Real” is the elapsed time, “CPU” is the CPU time used.

VII. Discussion

The results shown in Table I show that vectorization from use of both hand–optimized AVX intrinsics and the Intel (icc) compiler in a multi–threaded program result in increased performance. For both the icc and gcc tests, use of AVX intrinsics reduced the run time by about 10% and using both icc and the AVX intrinsics reduced the run time by 30% relative to using neither. Either with or without AVX enabled, icc produces a runtime 20% less than gcc.

Both using the Intel compiled and hand–coded routines make a substantial performance improvement; the use of the Intel compiled does not appear to reduce the utility of hand–optimization of critical routines. For larger, more realistic problems, the optimized portions of the program take a larger fraction of the computation, probably resulting in larger performance gains.

References

[1] W. D. Cotton, “Obit: A Development Environment for Astronomical Algorithms,” *PASP*, vol. 120, pp. 439–448, 2008.
[2] W. D. Cotton, “Comparison of GPU, Single- and Multi-threading for Interferometric Gridding,” *Obit Development Memo Series*, vol. 36, pp. 1–14, 2014.