Austin Herring

CS 540 – Homework 8

December 3, 2015

1. The compiler options do the following things:

- `mtune` "tunes" the performance for the specific processor to get more performance, using the supplied processor type as the target. In this case, "opteron-sse3" is used, because the processor on tux is an Opteron and it has capatabilities for SSE3 instructions.
- `g` is the compiler flag that ensures debug symbols are included in the compiled program
- `c` stops the compilation after the assembler runs (i.e., the linker does not run), so the result is assembly object code
- `fverbose-asm` puts extra comments and information into the assembly code output
- `Wa` passes the arguments after the comma in the argument to the assembler. The assembler options are as follows:
  - `a` means turn listings on. Then `d` means turn off debugging directives; `h` means include the high-level source; `l` means include the assembly; and `n` means leave out forms processing
- `msse3` allows the use of SSE3 instructions in the compiled program
- `march=x86-64` is similar to `mtune` but generates code that might only run on x86-64, beyond just tuning for the architecture
- `ftree-vectorize` will allow GCC to automatically vectorize loops where possible
- `ftree-vectorizer-verbose` will output information about diagnostics and decisions made while trying to vectorize loops to `stderr`. The option will put out more or less information depending on how high the number is set; at 7, the most information will be output.
- `mfpmath=sse` tells the compiler to use the SSE instruction set for scalars for floating point arithmetic

- `O2` is the “second level”of optimization, after `O1`. It uses all the optimizations of level as well as many more. According to the man pages, this means “performing nearly all supported optimizations that do not involve a space-speed tradeoff.”
- `ffast-math` sets the following compiler flags: `-fno-math-errno, -funsafe-math-optimizations, -ffinite-math-only, -fno-rounding-math, -fno-signaling-nans and -fcx-limited-range`. The names of each of these options is fairly straightforward in regards to what they do, but each of them introduces optimizations that might break IEEE floating point or ANSI standards.

2. The source code files were compiled with the options given. The assembly language listings are given in the `Code/` directory, under the appropriate directories, as the `*.s` files. The comments on the use of the SSE instructions in these assembly programs are given as comments in the files. All comments start with “`#####CS540 COMMENTS`,” Making them easily identifiable.

The following files showed different listings when using the `-ffast-math` option, for the following reasons. These listings are also included (sans comments) as `*_fastmath.s` files.

- test4f.c and test4h.c: these files are different because the fast math option can disable certain hardware optimizations, including SSE. That is, in the compiled programs, no SSE instructions, at all, appear. This is perhaps because things like SSE expect conformance to IEEE standards, and if those cannot be guaranteed (as when fast math is enabled), then the instructions cannot be used. Alternatively, it could be the fact that the statically sized arrays (of length 1024) are not long enough to justify setting up the SSE mechanisms.
- ti.c: For one, enabling fast math causes SSE instructions to be inserted into the function inner. This is perhaps because the fast math option will assume that no pointer aliasing will happen between a and b, which would allow SSE instructions to be valid, though disallowing memory aliasing is violating C standards, which is inconsequential when using some of the fast math

options. The other functions are mostly similar with some minor differences. For example, using the fast math options, the assembly listing shows the use of the `haddps` instruction. This instruction adds values in the same packed register, so it likely requires less instructions to perform some similar things as shuffling, meaning it should likely be quicker. However, it adds the floating point values in a different order than doing things the way the program was originally specified, thus breaking some of the guarantees the original program had, which is only possible when using the fast math options. Other minor optimizations like these, though not many other large ones, are spread throughout the fast math listing.

- test21.c: There do not seem to be many differences at all between these two listings. The only differences seem to be the use of the haddps instruction for doing some of the reductions at the end in the fast math version, likely for the same reasons as above.

Finally, one note about why test9.c is not different when using fast math. This is because in test9.c, there are not actually any floating point operations in the program, so no fast math operations need to be performed and nothing in the program will be non-IEEE compliant. Therefore, it is still safe to use the SSE instructions.