Vectorization Lab
Parallel Computing on Stampede

Aaron Birkland
Cornell Center for Advanced Computing

Oct 30, 2013

This lab serves as an introduction to using a vectorizing compiler. We will work with code containing a tight loop that should be easily vectorizable by the compiler. Our goal is to try out various compiler options and compare vectorized with non-vectorized code.

1. Unpack the lab materials into your home directory, and change into the `vector` directory.

   ```
   $ cd
   $ tar xvf ~tg459572/LABS/vector.tar
   $ cd vector
   ```

2. We noted that the Intel compiler starts applying vectorization with `-O3`. Let's see if we can view a vectorization report to see what it did.

   ```
   $ icc simple.c -vec-report=2 -O3 -o simple
   simple.c(19): (col. 2) remark: LOOP WAS VECTORIZED.
   simple.c(26): (col. 3) remark: LOOP WAS VECTORIZED.
   simple.c(25): (col. 5) remark: loop was not vectorized: not inner loop.
   ```

   This shows that *two* loops were vectorized: The initial value loading loop, and our computation loop.

3. Now that the compiler has told us that it vectorized our loops, let's verify this by compiling with vectorization disabled.

   ```
   $ icc simple.c -no-vec -vec-report=2 -O3 -o simple_no_vec
   ```

   Notice that all the vectorization reports disappeared, even though we specified reporting as a compile option. When vectorization is disabled, the reports disappear.

4. As mentioned in the talk, the Intel compiler will use SSE (128-bit) instructions by default. Compile the code with vectorization enabled, but add the argument `-xAVX` to the compilation flags to use 256-bit AVX. Name your executable `simple_avx`.

5. Now compile vectorized and non-vectorized variants of the code to run natively on the MIC coprocessor. Use the compile flag `-mmic` to compile for the MIC architecture.

   ```
   $ icc simple.c -mmic -O3 -o simple.mic
   $ icc simple.c -no-vec -mmic -O3 -o simple_no_vec.mic
   ```

6. The `simple.sh` batch file will record the execution time each of our vectorized and non-vectorized applications. Take a look at the batch script, then run it and examine the output.

   ```
     $ sbatch simple.sh
     $ cat slurm-951653.out
   simple_no_vec: 0.67
   simple 0.37
   simple_avx 0.25
   simple_no_vec.mic 13.22
   simple.mic 2.78
   ```

7. Lastly, the intel compiler flag `-xhost` can be used to automatically detect all the advanced features of the hardware (like AVX). The downside is that the resulting binaries may only be run on machines with an architecture similar to Stampede (e.g. the binaries would not be able to be run on Lonestar or Longhorn). Try compiling with `-xhost` and see if the runtime is similar to the `-axAVX` example from before.

As we have seen, vectorization on the Intel compiler can be simple and straightforward. Correlating vectorization reports with the source code can be a little bit tricky, especially if the compiler implements optimizations such as loop reordering. However, as long as we have some sense of what the compiler ought to be doing, this can usually be figured out with a little effort.