Serial Optimization and Vectorization Lab

Todd Evans
Texas Advanced Computing Center
Summer School in Advanced Scientific Computing

June 20, 2016

Setup

Open up a terminal

Run these commands in the terminal

- `ssh username@stampede.tacc.utexas.edu`
- idev -m 60 (this puts you on a compute node for 60 mn)
- `source ~train00/SSASC_sourceme`
- `tar xvf ~train00/SS_ASC2016_opt_lab.tar`
- `cd SS_ASC2016_opt_lab`

Exercise 1: Optimization Levels and Performance

We are going to compare the run times of a program compiled with gcc and icc at different levels of optimization.

Setup - Perform these steps from the command line

1. Inspect `auto.c`. It times the execution of loops of different floating point operations
2. module swap intel gcc
3.
```
gcc -O0 auto.c -o gauto_O0 -lm
gcc -O1 auto.c -o gauto_O1 -lm
gcc -O2 auto.c -o gauto_O2 -lm
gcc -O3 auto.c -o gauto_O3 -lm
```
4. module swap gcc intel
5.
```
icc -O0 auto.c -o iauto_O0 -lm
icc -O1 auto.c -o iauto_O1 -lm
icc -O2 auto.c -o iauto_O2 -lm
icc -O3 auto.c -o iauto_O3 -lm
```

Exercise 1: Optimization Levels and Performance

We are going to compare the run times of a program compiled with gcc and icc at different levels of optimization.

Run these 8 binaries and record execution times for each operation (multiplication/division/sqrt)

1. module swap intel gcc
2. `./gauto_O0`
   `./gauto_O1`
   `./gauto_O2`
   `./gauto_O3`
3. module swap gcc intel
4. `./iauto_O0`
   `./iauto_O1`
   `./iauto_O2`
   `./iauto_O3`
5. Which combination is the fastest? Does gcc or icc perform best?

Exercise 2: To Block or Not to Block

-Compare the run times of two matrix transposition programs
-One version will have manual loop-blocking and one will not
-`-O1` optimization flag has NO compiler performed blocking

Setup - Perform these steps from the command line

1. Inspect `cache_blocking.c`. It performs transposition w/ manual blocking and w/o
2. `icc -O1 cache_blocking.c -o nonblocked` (This produces an unblocked executable)
3. `icc -O1 cache_blocking.c -DBLOCK -o blocked` (This produces a manually blocked executable)
4. Run the following and note the "Time for transposition" reported

```
./nonblocked
./blocked 32
./blocked 256
./blocked 512
```

Exercise 2: To Block or Not to Block

-Compare the run times of two matrix transposition programs
-One version will have manual loop-blocking and one will not
-O2 optimization flag HAS compiler performed blocking

Setup - Perform these steps from the command line

1. `icc -O2 cache_blocking.c -o nonblocked_opt` (This produces a compiler blocked executable)
2. `icc -O2 cache_blocking.c -DBLOCK -o blocked_opt` (This produces a manual and compiler blocked executable)
3. Run the following and note the "Time for transposition" reported
   ```
   ./nonblocked_opt
   ./blocked_opt 32
   ./blocked_opt 256
   ./blocked_opt 512
   ```
4. Who is better at loop-blocking: the compiler or us?

Exercise 3: Use -xhost: Why Vectorization Matters

Setup - Perform these steps

1. Inspect `vector.c`. It initializes a matrix `B` and vector `c` and performs a matrix-vector multiply
2. `icc vector.c -no-vec -o no-vectorization` (generates un-vectorized code)
3. `icc vector.c -o sse2-vectorization` (generates SSE2 vectorized code (128-bit width))
4. `icc vector.c -xhost -o avx-vectorization` (generates AVX vectorized code (256-bit width))
5. Compare the run times and flops/cycle for each executable
6. How many doubles fit in 256-bits? Is this even possible? (Hint: SNB cores have 2 vector units)
7. feel free to repeat above steps w/ `-opt-report` flag and inspect generated optimization report