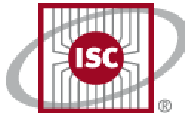
Hands-on with Intel Xeon Phi

Lab 2: Native Computing and Vector Reports

Bill Barth
Kent Milfeld
Dan Stanzione



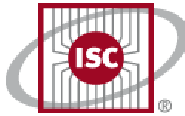
Lab 2

- What you will learn about
  - Evaluating and Analyzing vector performance.
  - Controlling Affinity, visualizing thread distribution.
- What you will do:
  - Determine the extent of total code vectorization:
    1st run with vectorize options,
    2nd run with vectorization turned off.
  - Use the vector reports to analyze loop dependencies (reasons for not vectorizing) and alignments.
  - Learn how to set affinity and observe location of threads.

Lab 2

Native and Vector Computing

Part 0 – Grab the Lab Files



- Login to Stampede with X forwarding turned on.
  If your laptop runs Windows OS you will need an X display manager (Xming).

```
$ ssh -Y <username>@stampede.tacc.utexas.edu
```

- Untar the file mic_native.tar file (in ~train00) into your directory:

```
$ tar xvf ~train00/mic_native.tar
```

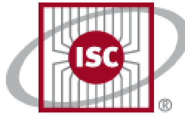
- Move into the newly created Lab 2 directory:

```
$ cd native
```

- Use idev to launch a 1-node interactive session in the development queue

```
$ idev
```

Part 1 – Total Vector Performance



Compile and run **vector.c** as a native vectorized MIC application. See vectorized loop with the –vec-report2 option.

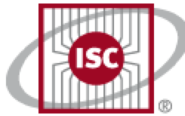
```
icc -openmp -O3 -mmic -vec-report1 ./vector.c -o vec.mic
```

Compile and run **vector.c** as a non-vectorized MIC application by disabling vectorization:

```
icc -openmp -O3 -mmic -no-vec ./vector.c -o novec.mic
                              #  you are on the host (compute node)
 ./vec.mic                    # mic binary will automatically
 ./novec.mic                  # run on the MIC
```

How much speedup comes from the vectorization? Does this make sense given what you have learned about the MIC architecture?

Part 2 – Vectorization Information



Let's get some information about the vectorization in this example code.

Compile the code again, but add the **`-vec-report5`** option to the compilation line.

There will be some lines in the code which are not vectorizable. Can you use a higher vector report level to find out why?

```
icc -O3 -openmp -vec-report5 -mmic ./vector.c -o vec.mic
```

./vector.c(33): (col. 33) remark: loop was not vectorized: statement cannot be vectorized.

./vector.c(34): (col. 33) remark: loop was not vectorized: statement cannot be vectorized.

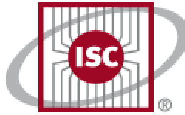
./vector.c(35): (col. 33) remark: loop was not vectorized: statement cannot be vectorized.

./vector.c(37): (col. 9) remark: loop was not vectorized: existence of vector dependence.

./vector.c(37): (col. 35) remark: vector dependence: assumed FLOW dependence \
between y line 37 and y line 37.

./vector.c(44): (col. 2) remark: loop was not vectorized: not inner loop.

Part 2 (cont.) – Vectorization Information



The `-vec-report6` has more information and includes `Alignment Information`.

There will be some lines in the code which are not vectorizable. Can you use a higher vector report level for additional info (transforms, unrolling, alignment, in-lining, etc.)

```
icc -O3 -openmp -vec-report6 -mmic ./vector.c -o vec.mic
```

./vector.c(33): (col. 2) remark: loop not vectorized: loop was transformed to memset or memcpy.

./vector.c(34): (col. 33) remark: loop was not vectorized: statement cannot be vectorized.

./vector.c(34): (col. 33) remark: vectorization support: call to function rand cannot be vectorized.

…

./vector.c(37): (col. 9) remark: loop was not vectorized: existence of vector dependence.

./vector.c(37): (col. 35) remark: vector dependence: assumed FLOW dependence between y line 37 and y line 37.

…

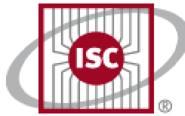
./vector.c(47): (col. 4) remark: vectorization support: reference z has aligned access.

./vector.c(46): (col. 3) remark: vectorization support: unroll factor set to 8.

./vector.c(46): (col. 3) remark: LOOP WAS VECTORIZED.

./vector.c(44): (col. 2) remark: loop was not vectorized: not inner loop.

Part 3 – Affinity



Run **vec_omp.mic** executable using 4 OpenMP threads and different affinity settings.

Use the KMP_AFFINITY variable and the "compact/scatter/balanced" and "verbose" settings as described in the lectures. First, set the number of threads.

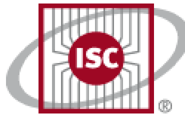
```
$ export MIC_ENV_PREFIX=MIC
$ export MIC_OMP_NUM_THREADS=4
```

See examples on next page- set KMP_AFFINITY

- `export MIC_KMP_AFFINITY=compact,…  #use different values`
- `./vec_omp.mic                      #and rerun executable`

Write down the timings, and the processor number to which each thread is bound.

Part 3(cont.) – Affinity



First, compile vec_omp.c, the OMP version of the vec.c code:

```
$ icc -openmp -O3 -mmic ./vector_omp.c -o vec_omp.mic
```

Now execute it on the MIC.

```
$ export MIC_KMP_AFFINITY=compact,granularity=fine,verbose
$ ./vec_omp.mic
...                                                    # at end of listing
```

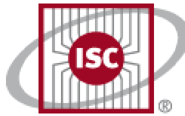
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {3}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {4}

```
$ export MIC_KMP_AFFINITY=balanced,granularity=fine,verbose
$ ./vec_omp.mic
...
```

OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {5}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {9}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {13}

KMP_AFFINITY=scatter should give (approximately) the same timing as balanced since it pins to the same threads for this small example.

Part 4 – Thread Location (`top`)



- You can observe the location of threads on the MIC with the standard Unix **top** utility or the Intel **micsmc** utility.
- For **top**, you will need to open up another window directly on the MIC:

```
$ ssh <username>@stampede.tacc.utexas.edu
$ ssh <compute_node>
$ ssh mic0
$ extend window, reduce font size & execute:  resize
$ ssh mic0
$ top        # press the 1 key to see the threads
```

In your original idev window create a MIC *load* binary, ssh over to the MIC and execute with different affinity options. Watch top in the other window.

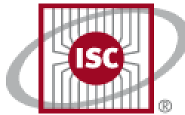
```
$ icc -O3 -openmp -mmic load.c -o load.mic
$ ssh mic0
$ cd native
$ export OMP_NUM_THREADS=60
$ export KMP_AFFINITY=compact
$ ./load.mic                    #watch top, execute ^C to quit

$ export KMP_AFFINITY=scatter

$ ./load.mic  #watch top, execute ^C to quit
```


10

Part 4 – Thread Location (`micsmc`)



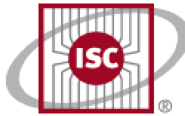
- Unlike top, **`micsmc`** is run on the compute node (it is not available on the MIC). It has a command line (text) and GUI interfaces.
- Linux/Mac laptops will use their default X11 display manager; windows systems use an X display manager (e.g. Xming) and launch it.
- You will need to use X forwarding through all of your connections, so you should logout and login using the –Y option.

```
$ ssh -Y <username>@stampede.tacc.utexas.edu
$ idev      #the session will "X forward" to the node
            # when connected, run micsmc in the background.
c559-100$ /opt/intel/mic/bin/micsmc
```

If the display is too unresponsive (through wireless), use the command line:

```
c559-100$ watch -n 3 /opt/intel/mic/bin/micsmc -c
```

This is the compute node IP, ssh to this node in another window, see next page. It is different for each idev session.



Part 4 – Thread Location (micsmc)

- When the smc display appears only 1 panel will be visible. To observe the core usage histogram, select "show all" under the Cards tab (Cards→Show all) and a 2nd panel will appear; next click on the the "core histogram" icon in the new panel if it isn't already displayed.

Cards Advanced Cards: 1

Utilization View (All Devices)

System [%]
User [%]
100 80 60 40 20 0
Average Core Utilization

59
Average Core Temperature °C

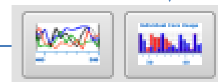
0.301 GB
Total Memory Usage

187 Watts
Total Power Usage

mic0: Core Histogram View

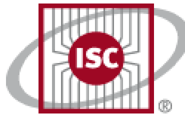
Individual Core Usage

System [%]
User [%]
100 80 60 40 20 0
0 10 20 30 40 50 60 70



Core Histogram Icon

Part 4 – Thread Location (`micsmc`)



- In another window login to Stampede , ssh to the compute node, and then ssh into the MIC. Execute the *`load`* binary with different Affinities, and watch the behavior in the the **`micsmc`** display.
-

```
$ ssh <username>@stampede.tacc.utexas.edu
$ ssh <compute_node>        #access to the compute node
$ ssh mic0                  #access to MIC coprocessor
$ cd native

$ export OMP_NUM_THREADS=60
$ export KMP_AFFINITY=compact
$ ./load.mic       #watch micsmc, execute ^C to quit

$ export KMP_AFFINITY=scatter
$ ./load.mic       #watch micsmc, execute ^C to quit
                    # Why is the load only 25%/core?
```