# Xeon Phi Native Computing and Optimization Lab

John D. McCalpin, PhD

mccalpin@tacc.utexas.edu

# Getting started

Connect to Stampede: ← you might want to repeat this to get 2-3 open windows on the login node

```
ssh username@stampede.tacc.utexas.edu
```

Extract the lab files to your account:

```
tar xvf ~train00/sc13_mic_native_lab.tar
```

Change to the lab directory:

```
cd ./sc13_mic_native_lab
```

Obtain an interactive session in Stampede:

```
idev
```

← only do this once – other windows can login to the compute node afterwards

(alternate if idev reservation is full):

```
srun -N 1 -n 16 –A 20131117MIC -p development -t 01:00:00 --pty /bin/bash -l
```

# Exercise 1: Vectorization

Compile **uni_test.c** as a native MIC application:

```
icc -mmic ./uni_test.c –o vector.uni.mic
```

And also as a MIC application with disabling vectorization:

```
icc -mmic -no-vec ./uni_test.c -o novec.uni.mic
```

And also as a MIC application with legacy x87 arithmetic:

```
icc -mmic –fp-model strict ./uni_test.c -o x87.uni.mic
```

Run the executables and take note of the timing differences.

Do the timings make sense given what you have learned about the MIC architecture?

# Exercise 2a: OpenMP overhead

Compare the source code for **omp_test_outer.c** and **omp_test_middle.c**

Compile **omp_test_outer.c** and **omp_test_middle.c** as native applications:

```
icc –openmp -mmic ./omp_test_outer.c \
          –o vector.outer.mic
icc –openmp -mmic ./omp_test_middle.c \
          –o vector.middle.mic
```

Set up the environment:

```
export KMP_AFFINITY=scatter  (or MIC_KMP_AFFINITY=scatter)
export OMP_NUM_THREADS=60    (or MIC_OMP_NUM_THREADS=60)
```

Run the executables and take note of the timing differences.

Can you use this information to estimate the overhead of an OpenMP parallel for loop?

# Exercise 2b: OpenMP Affinity

We will use the vector.out.mic code that you compiled previously

Set up the environment:

```
export OMP_NUM_THREADS=32    (or MIC_OMP_NUM_THREADS=32)
```

Run the executable with

```
export KMP_AFFINITY=scatter (or MIC_KMP_AFFINITY=scatter)
```

Run the executable again with

```
export KMP_AFFINITY=compact (or MIC_KMP_AFFINITY=compact)
```

Look at the “Sustained memory bandwidth” in the output.

Do the relative values make sense?

# Exercise 3a: Vectorization Reports

Some examples of vectorization reports

Compile the uniprocessor again, but add the **-vec-report5** option to the compilation lines:

```
icc -mmic –vec-report5 ./uni_test.c –o vector.uni.mic
```

There will be some lines in the code which are not vectorizable.

Use –vec-report6 to get more information on why loops did not vectorize.

The next two slides contain examples of this output.

# -vec-report5 Output

```
$ icc -mmic -vec-report5 uni_test.c -o foo
uni_test.c(49): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(50): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(51): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(56): (col. 4) remark: loop was not vectorized: loop was transformed to memset or memcpy.
remark: loop was not vectorized: operation cannot be vectorized.
remark: loop was not vectorized: operation cannot be vectorized.
uni_test.c(66): (col. 2) remark: loop was not vectorized: not inner loop.
uni_test.c(68): (col. 4) remark: loop was not vectorized: not inner loop.
uni_test.c(67): (col. 3) remark: loop was not vectorized: not inner loop.
```

# -vec-report6 Output

```
$ icc -mmic -vec-report6 uni_test.c -o foo
uni_test.c(49): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(49): (col. 42) remark: vectorization support: call to function rand cannot be vectorized.
uni_test.c(50): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
…
uni_test.c(56): (col. 4) remark: loop was not vectorized: loop was transformed to memset or memcpy.
…
uni_test.c(70): (col. 6) remark: vectorization support: reference M has aligned access.
uni_test.c(70): (col. 6) remark: vectorization support: reference z has aligned access.
…
uni_test.c(69): (col. 5) remark: LOOP WAS VECTORIZED.
uni_test.c(66): (col. 2) remark: loop was not vectorized: not inner loop.
uni_test.c(68): (col. 4) remark: loop was not vectorized: not inner loop.
uni_test.c(67): (col. 3) remark: loop was not vectorized: not inner loop.
```

# Ex 3b: vec-report serial vs OpenMP

Now we will compare vectorization reports for single-threaded and OpenMP compilation:

Compare the **-vec-report6** output for the serial and OpenMP codes:

```
icc -mmic –vec-report6 ./uni_test.c
icc –mmic –vec-report6 –openmp ./omp_test_outer.c
```

Pay special attention to the alignment messages!

Do you remember why there are differences?

The next two slides contain examples of this output.

# -vec-report6 Output: serial

```
$ icc -mmic -vec-report6 uni_test.c
uni_test.c(49): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(49): (col. 42) remark: vectorization support: call to function rand cannot be vectorized.
uni_test.c(50): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(50): (col. 42) remark: vectorization support: call to function rand cannot be vectorized.
uni_test.c(51): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
uni_test.c(51): (col. 42) remark: vectorization support: call to function rand cannot be vectorized.
uni_test.c(54): (col. 2) remark: loop was not vectorized: loop was transformed to memset or memcpy.
…
uni_test.c(70): (col. 6) remark: vectorization support: reference M has aligned access.
uni_test.c(70): (col. 6) remark: vectorization support: reference z has aligned access.
uni_test.c(69): (col. 5) remark: vectorization support: unroll factor set to 8.
uni_test.c(69): (col. 5) remark: LOOP WAS VECTORIZED.
uni_test.c(66): (col. 2) remark: loop was not vectorized: not inner loop.
uni_test.c(68): (col. 4) remark: loop was not vectorized: not inner loop.
uni_test.c(67): (col. 3) remark: loop was not vectorized: not inner loop.
```

# -vec-report6 Output: OpenMP

```
$ icc -mmic -openmp -vec-report6 omp_test_outer.c
omp_test_outer.c(48): (col. 42) remark: loop was not vectorized: statement cannot be vectorized.
omp_test_outer.c(48): (col. 42) remark: vectorization support: call to function rand cannot be vectorized.
…
omp_test_outer.c(66): (col. 1) remark: vectorization support: call to function __kmpc_ok_to_fork cannot be vectorized.
omp_test_outer.c(65): (col. 2) remark: loop was not vectorized: existence of vector dependence.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference M has aligned access.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference z has aligned access.
omp_test_outer.c(70): (col. 5) remark: vectorization support: unroll factor set to 4.
omp_test_outer.c(70): (col. 5) remark: LOOP WAS VECTORIZED.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference M has unaligned access.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference z has unaligned access.
omp_test_outer.c(71): (col. 6) remark: vectorization support: unaligned access used inside loop body.
omp_test_outer.c(70): (col. 5) remark: PEEL LOOP WAS VECTORIZED.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference M has aligned access.
omp_test_outer.c(71): (col. 6) remark: vectorization support: reference z has unaligned access.
omp_test_outer.c(71): (col. 6) remark: vectorization support: unaligned access used inside loop body.
omp_test_outer.c(70): (col. 5) remark: REMAINDER LOOP WAS VECTORIZED.
…
```