Native Computing and Optimization Lab

Getting started

Connect to Stampede:

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

Extract the lab to your account:

```
tar xvf ~train00/mic_native_lab.tar
```

Change to the lab directory:

```
cd ./mic_native_lab
```

Obtain an interactive session in Stampede:

```
srun -N 1 -n 16 –A 20130425MIC -p development -t 01:00:00 --pty /bin/bash -l
```

Exercise 1

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

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

And also as a MIC application with disabling vectorization:

```
icc -openmp -O3 -mmic -no-vec ./vector.c -o novec.mic
```

Run both executables and take note of the timing difference. Does this make sense given what you have learned about the MIC architecture?

Exercise 2

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?

The next two slides contain examples of this output.

-vec-report5 Output

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

./vector_contig.c(36): (col. 34) remark: loop was not vectorized: statement cannot be vectorized.
./vector_contig.c(37): (col. 34) remark: loop was not vectorized: statement cannot be vectorized.
./vector_contig.c(38): (col. 34) remark: loop was not vectorized: statement cannot be vectorized.
./vector_contig.c(44): (col. 3) remark: loop was not vectorized: not inner loop.
./vector_contig.c(43): (col. 2) remark: loop was not vectorized: not inner loop.
./vector_contig.c(44): (col. 3) remark: loop was not vectorized: not inner loop.
./vector_contig.c(43): (col. 2) remark: loop was not vectorized: not inner loop.

-vec-report6 Output

Lots of additional information, including alignment:

```
icc -O3 -openmp -vec-report6 -mmic ./vector_contig.c -o vec.mic
./vector_contig.c(36): (col. 34) remark: loop was not vectorized: statement cannot be vectorized.
./vector_contig.c(36): (col. 34) remark: vectorization support: call to function rand cannot be vectorized.
...
./vector_contig.c(46): (col. 5) remark: vectorization support: reference M has aligned access.
./vector_contig.c(46): (col. 5) remark: vectorization support: reference z has aligned access.
...
./vector_contig.c(45): (col. 4) remark: LOOP WAS VECTORIZED.
./vector_contig.c(46): (col. 5) remark: vectorization support: reference M has unaligned access.
./vector_contig.c(46): (col. 5) remark: vectorization support: reference z has unaligned access.
...
./vector_contig.c(45): (col. 4) remark: PEEL LOOP WAS VECTORIZED.
```

Exercise 3

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

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

Use the KMP_AFFINITY variable and the "compact/scatter/balanced" and "verbose" settings as described in the lectures.

Do the results make sense given what you have learned?

Affinity Results

c557-404$ export MIC_OMP_NUM_THREADS=4
c557-404$ export MIC_KMP_AFFINITY=compact,granularity=fine,verbose
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}
Vectorization exercise completed in 1.282020e-01 seconds.

c557-404$ export MIC_KMP_AFFINITY=balanced,granularity=fine,verbose
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}
Vectorization exercise completed in 6.789207e-02 seconds.

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