Quick Start Guide for Native Mode Execution

Arturo Argueta, Roberto Camacho Barranco, Esthela Gallardo, and Pat Teller
UTEP Stampede Technology Insertion Project
Leonardo Fialho and Jim Browne
UT-Austin Stampede Technology Insertion Project

A Xeon Phi can execute a highly-parallel OpenMP program (see sample code below) independently of the host processor of a Stampede node. Using native mode execution, the entire program can be executed on the co-processor without interacting with the host processor (e.g., no memory copies are required).

To use native mode execution make sure that you use the -mmic flag when you compile your program on the host processor. For example, to compile the program Example.c execute the following command:

```bash
icc -openmp -mmic -o Example Example.c
```

The -mmic flag causes the Intel compiler to cross-compile the source code and build a Xeon Phi executable on the host CPU. Since the current Xeon Phi and the Xeon CPU are not binary compatible, it is essential to use the -mmic flag, which performs the cross compilation.

To execute the program on a Phi in native mode, first access the Phi by executing the command `ssh mic0`. This can be done through an interactive session on Stampede. To create an interactive session execute the following command:

```bash
srun --pty -A acct -p queue -t hh:mm:ss -n tasks -N nodes /bin/bash -l
```

For more details on creating an interactive session, including information about the job queues for nodes with Xeon Phis, go to [https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide#overview-phi](https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide#overview-phi).

To launch the execution of Example on a Phi execute the command `.\Example` on the Phi.

Note that this process also can be performed using a script. See the [Stampede User Guide](https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide) for more details.

Sample Code

The following OpenMP vector addition code is designed for execution in native mode on the cores of a Phi (or on the cores of a Sandy Bridge processor). In native mode, it executes solely on the cores of a Phi (or a Sandy Bridge) using the OpenMP parallel pragma to distribute the computation among the threads of the cores. Included in the code are instructions that time the execution of the array initialization and the vector addition loops.
Add two vectors in parallel using OpenMP.

```c
#include <stdio.h>
#include <omp.h>
#include <time.h>
#include <sys/time.h>

/* Define size of vectors; change this value to study how problem size, along with number of cores and threads per core, affect execution time. */
const int size = 500000000;

// Declare vectors.
float a[size];
float b[size];
float c[size];

int main()
{
    /* Timeval structures store start and end times of vector initialization and computational loops. */
    struct timeval start, end;
    // Start timing vector initialization loop.
    gettimeofday(&start, NULL);
    // Initialize vectors.
    for (int i = 0; i < size; ++i) {
        a[i] = (float)i;
        b[i] = (float)i;
        c[i] = 0.0f;
    }
    // End timing vector initialization loop.
    gettimeofday(&end, NULL);
    // Print execution time of vector initialization loop.
    printf("%f\n",
        (double) (end.tv_usec - start.tv_usec) / 1000000 +
        (double) (end.tv_sec - start.tv_sec));

    // Start timing computational loop.
    gettimeofday(&start, NULL);
    // Compute vector addition in parallel using OpenMP.
    #pragma omp parallel for default(none) shared(a,b,c)
    for (int i = 0; i < size; ++i) {
        c[i] += a[i] + b[i];
    }
```
// End timing computational loop.
gettimeofday(&end, NULL);
// Print execution time of computational loop.
printf("%f
",
      (double) (end.tv_usec - start.tv_usec) / 1000000 +
      (double) (end.tv_sec - start.tv_sec));
return 0;
}