Quick Start Guide for Using Intel's Xeon Phis on Stampede

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

Incorporation of Phis in Application Execution

Each of Stampede's compute nodes is comprised of 2 Sandy Bridge (SB) CPUs and 1 or 2 Xeon Phi processors (Phis), alternatively called MICs because they employ Intel's Many Integrated Core (MIC) architecture [1]. While the instruction sets of the SB CPU (E5 Xeon processor) and Phi are both X86, their execution characteristics are quite different. The potential execution-time performance of a 61-core Phi is about that of 2 8-core SB CPUs (which, on Stampede, have 1 thread per core). Additionally, Intel compilers and libraries make it straightforward to compile and execute applications on a Phi with little or no code modification. Thus, *there is the potential for at least doubling the execution speed of an application by utilizing both Stampede's SB CPUs and Phis*. However, it should be noted that the *Phis will only perform favorably if an application both vectorizes well and scales to many threads per Phi (up to 244 threads, i.e., 4 threads per core)*. In addition, *the cost of data movement between the SB CPUs and Phis must be factored into the potential performance gain*.

An application can be executed on a system that incorporates both SB CPUs and Phis in the following four ways. Methods 1 and 4 make use of only one type of processor, while Methods 2 and 3 concurrently execute on both.

1. **Native Mode** – Execute solely on Phis; make no use of CPUs except to `ssh` into Phi(s).
2. **Offload Mode** – Execute on both CPUs and Phis splitting *code segments* across them, i.e., some segments are executed on CPUs, while others are executed on Phis ("offloaded" to Phis).
3. **Symmetric Mode** – Execute on both CPUs and Phis splitting *data* across them, i.e., the same code segments can be executed on both the CPUs and Phis. Communication between the CPUs and Phis is supported by a compatible MPI library (e.g., MVAPHIC2 or IMPI), while data can be shared by cores of a CPU or cores of a Phi through the use of OpenMP.
4. **CPU Mode** – Execute solely on CPUs; make no use of Phis.

The question that you will want to answer in order to determine how to use Stampede nodes is: *Which of these four possible modes of execution will yield best performance for my application?* The answer can be found by conducting the four simple experiments specified below.

How to Select Best Execution Mode

The following process, which is depicted in Figure 1 beginning with Step 2b, specifies a set of four experiments that determine the best performing execution mode for your application when executed on Stampede nodes. To follow this process you will need to generate an OpenMP version of your code for efficient execution on a CPU or Phi. An example of a simple application with appropriate OpenMP pragmas is in the *Quick Start Guide for Native Mode Execution* [URL].

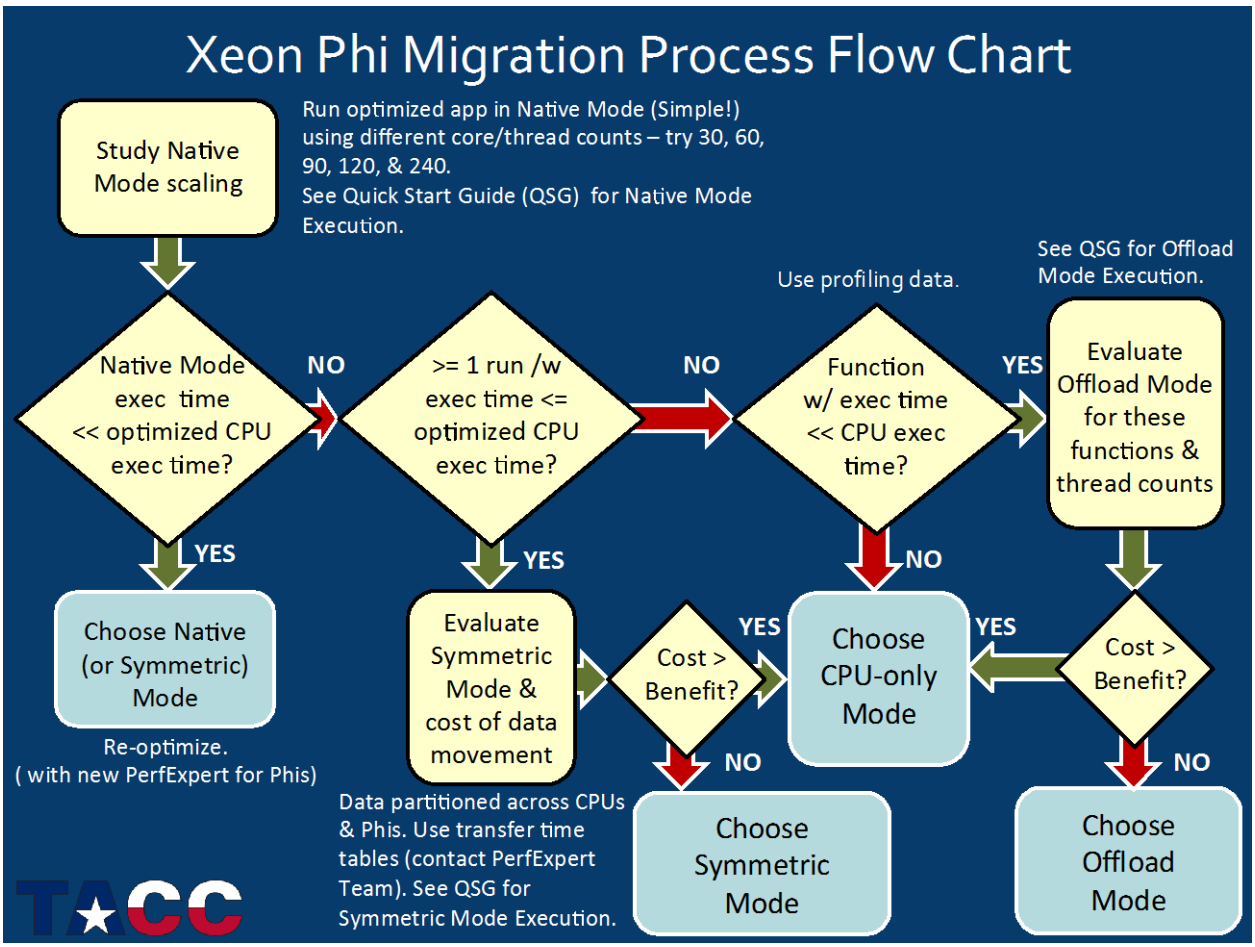

Figure 1: Flow chart of process to determine the best performing execution mode for your application when executed on Stampede nodes.

1. **Optimize Application Code** (This step is beneficial under any and all circumstances, whether or not you intend to use Phis. It has been established that code that is well optimized for SB CPUs is often well optimized for Phis and vice versa.)
   a. **Analyze Original Code using PerfExpert**: Use *PerfExpert* to analyze your code, paying particular attention to: (i) vectorization, (ii) scaling, and (iii) the *optimal scaling*, i.e., the optimal number of tasks/threads to execute per node. (*PerfExpert* is simple to use and does not require you to modify your code.)
   b. **Optimize Code**: Using the feedback from PerfExpert, optimize your code if necessary and use the optimized code to determine the best execution time on the CPUs.

c. **Profile Optimized Code using PerfExpert**: Using *PerfExpert*, profile the execution of the optimized code to determine the execution time of important functions/procedures, i.e., those using a significant fraction of the total execution time at the optimal scale.

2. **Generate and Compile OpenMP Code for Phi**
   a. **Generate Code for Phi**: If necessary, add OpenMP pragmas for each loop nest in each important function as determined by the PerfExpert profiling of Step 1c. An example of a simple application with appropriate OpenMP pragmas is in the *Quick Start Guide for Native Mode Execution* (attached).
   b. **Compile Phi Code**: Compile the code for the Phi; e.g., compiling a C program looks something like this:

```
login1$ icc -openmp -mmic –O3 -o myprog.exe.mic myprog.c
```

   Consult the *Compiling* section, under Application Development, of the *Stampede User Guide* for detailed instructions on the various compilation targets. (Since Phis will only perform favorably if an application both vectorizes well and scales to many threads per Phi, to see what loops were vectorized for the Phi and which may benefit from vectorization, you should examine the vectorization reports, which are generated by the Intel compiler when the compiler option `–vec-report2` is used. Information about Intel's Vectorization Toolkit is at *https://software.intel.com/en-us/articles/vectorization-toolkit*.)
   c. **Run Phi Executable**: Run the Phi executable in Native Mode (on Phi only) using 60, 120 and 240 threads.
   d. **Profile Phi Code using PerfExpert**: Using *PerfExpert*, for each important function/procedure measure the optimal number of tasks/threads to execute on a Phi.

3. **Analyze Performance (and answer that question!)**
   a. Native Mode
      i. If the Phi-only runs of Step 2b give significantly better performance than the CPU-only runs of Step 1b, then either Native or Symmetric Mode will be the best choice.
      ii. For Native Mode runs on Phis, re-optimize your code for the Phis using the *PerfExpert* option for the Phis.
   b. Symmetric Mode
      i. If none of the Phi-only runs, with different numbers of threads (Step 2b), is significantly faster than the CPU-only runs (Step 1b), but at least one Phi-only run is about the same speed or faster than one CPU-only run, consider Symmetric Mode. You will need to split the *work and data* across the CPUs and Phis by creating MPI tasks and assigning subsets of MPI tasks to CPUs and Phis. (Note that non-blocking MPI communication can be used to overlap communication with computation.) A simple example of an application prepared for Symmetric Mode execution is in the *Quick Start Guide for Symmetric Mode Execution* (attached), which also provides more complete information on compiling and annotating applications to execute on Phis in Symmetric Mode.
      ii. Symmetric Mode is beneficial only if the speedup in execution time from the additional parallelism gained by using Phis exceeds the cost of moving data between CPU and Phi memories. See the sample code in the *Quick Start Guide for Offload*

Mode Execution (attached) for a way to measure code execution time and refer to the example in the *Quick Start Guide for Symmetric Mode Execution* for a way to estimate the cost of data movement. If the cost of data movement is less than the performance gain from additional parallelism, then use Symmetric Mode. Note: As mentioned in the *Quick Start Guide for Symmetric Mode Execution*, even if it has been determined that Symmetric Mode may be the best option for the code at hand, load balancing to attain good performance scaling can still be challenging.

c. Offload Mode

If none of the Phi-only runs (Step 2b) is faster than any CPU-only run (Step 1b), (i) identify each function that consumes a significant fraction of the total execution time and executes considerably faster on the Phi, as compared to the CPUs (you already have this data from Steps 1c and 2c); and (ii) identify the optimal number of tasks/threads to employ for each function. If there are such functions, evaluate Offload Mode as follows. Further information on compiling and annotating applications to execute on Phis in Offload Mode is given in the *Quick Start Guide for Offload Mode Execution* (attached).

i. Following the example in the *Quick Start Guide for Offload Mode Execution* (attached), for each of these functions, determine the execution time saved on a Phi, in comparison to 2 CPUs. If there is no savings, execute the function on 2 CPUs.

ii. Otherwise, determine the cost of data movement by following the example in *Offload Mode: Estimation of Execution Time and Cost of Data Movement* (attached). If the time saved in execution time exceeds the cost of data movement, then consider using Offload Mode. Note: As mentioned in the *Quick Start Guide for Offload Mode Execution*, to achieve good performance in Offload Mode, it may be necessary to overlap computation with data movement, thus, hiding some of the cost of data transfers between the host and the Phi across the PCIe lane. In addition, there may be costs associated with setting up transferred data so that the Phi can utilize it.

Regardless of the mode of execution utilized, the performance of the Phi can be hindered by oversubscribing the Xeon Phi cores or threads. Although there are more cores or threads on the Phi than there are on a Xeon SB, those on the Phi are not as fast in terms of servicing interrupts or context switching, thus, they will be slower in terms of getting back to servicing the application workload. Finally, do not ignore thread affinity – it can have a significant impact on performance.

References and Additional Information

1. *Intel Xeon Phi Coprocessor High Performance Programming* by Jim Jeffers and James Reinders, Morgan Kaufmann, ISBN 9780124104945, February 2013.
2. slide presentations - link to ppt
3. PerfExpert and MACPO: Which code segments should (not) be ported to MIC?
4. Advanced Offloading document