Intel Xeon Phi Coprocessor

A guide to using it on the Cray XC40

Terminology Warning: may also be referred to as MIC or KNC in what follows!

What are Intel Xeon Phi Coprocessors?

- **Hardware designed to accelerate performance and scalability of parallel applications**
- **Each coprocessor (MIC) made up of Many (60) Integrated Cores**
  - Each core has slower clock speed than typical compute core
  - Each core can support up to 4 execution threads at once
  - Each MIC can support up to 240 concurrent execution threads (performance up to just over 1 Tflop)
- **On Cray XC40 node, a single x86 Intel Xeon processor (12-core 2.4ghz Ivybridge at SERC) is paired with a single Intel Xeon Phi coprocessor**
- **Usage at SERC:**

```
#PBS -l select=X:ncpus=Y:accelerator=True:accelerator_model="Xeon_Phi"
#PBS -l accelerator_type="Xeon_Phi"
```

Modes of Operation 1: Offload Mode

- **Main part of the code runs on the x86 (host part of the node), while sections of the code may be "offloaded" to the MIC using special Intel compiler directives**
  - Similar to accelerator mode used for GPUs, although offload mode does not use OpenACC directives.
- **User must modify code to specifically direct compiler to generate code to run on Xeon Phi**
  - For example, wrap blocks of OpenMP with offload directives
  - Can use up to 240 threads per Xeon Phi

Using Offload Mode

- **Load PrgEnv-intel module and set Intel compilation environment:**
  - module load PrgEnv-intel
  - source ${INTEL_PATH}/bin/compilervars.sh intel64
    OR
  - source ${INTEL_PATH}/bin/compilervars.csh intel64

- **Compile and run as usual:**
  - cc mycode.c
  - aprun –n2 –d4 ./a.out

- **NOTES:**
  - Do NOT load craype-intel-knc module
  - In offload mode, dynamic linking not enabled by default
  - Compiler flag "-no-offload" disables any offload directives
  - Main bottleneck is data transfers between host and accelerator – minimize them and keep data on MIC between computations using the same data (similar to GPU programming)

Offload Clauses

C/C++	Fortran	Description
#pragma offload target(mic)	!dir$ offload target(mic)	Tells the compiler to generate code for the MIC. Can be used to offload single statements or blocks of code.
in(*var*[:modifiers]) or out(*var* [:modifiers]) or inout(*var*[:modifiers])	in(*var* [:modifiers]) or out(*var* [:modifiers]) or inout(*var*[:modifiers])	Tells the compiler which data to move to (in) or from (out) or into and out of (into) the MIC and attributes about the data.
nocopy(*var*[:modifiers])	nocopy(*var*[:modifiers])	Tells the compiler to create persistent data on the MIC.
if(test) where test evaluates to 0 or 1	if(test) where test evaluates to .true. or .false.	Tests for a condition.
signal(&*var*)	signal(*var*)	Allows for asynchronous execution of offload code; give a signal.
wait(&*var*)	wait(*var*)	Allows for asynchronous execution of offload code; wait for a signal.

Modifiers for in, out and nocopy

C/C++	Fortran	Description
length(num_elements)	length(num_elements)	The length of a data element.
alloc_if(test) where test evaluates to 0 or 1	alloc_if(test) where test evaluates to .true. or .false.	Allocates space for data based on condition.
free_if(test) where test evaluates to 0 or 1	free_if(test) where test evaluates to .true. or .false.	Free space used by data based on condition.
align(val)	align(val)	Aligns data on boundaries based on val.
alloc([first:last])	alloc([first:last])	Allocate space on the Intel Xeon Phi.
into(*var*)	into(*var*)	Copy data into specified location.

Modes of Operation 2: Native (or Autonomous) Mode

- **Entire application runs on the MIC**
  - No part runs on the x86
- **Can run MPI-based or threaded OpenMP code**
- **Code must first be built to run directly on MIC (KNC)**
- **Environment: load PrgEnv-intel module, then unload any modules that might conflict with KNC, then load KNC module**
    - module swap PrgEnv-cray PrgEnv-intel
    - module unload cray-libsci atp craype-sandybridge craype-ivybridge
    - module load craype-intel-knc
      - Note: cray-libsci incompatible with KNC – must use Intel MKL
- **Compilation:**
  - Use –mmic compiler flag and –openmp if using OpenMP
- **At runtime add "-k" option to aprun to run on MIC**
  - aprun –k -d4 ./a.out

Code Optimizations for Intel Phi

- **Opts designed for x86 processors should also improve performance on MIC (which is also x86)**

- **Try to vectorize code as much as possible**
  - Each MIC core has multiple threads
  - Use –vec_report2 compiler option to show how Intel compiler has vectorized code (look for Message "*MIC* Loop was vectorized" etc.).

- **Align data targeted for MIC on 64-byte boundaries**
  - MIC has 512-bit SIMD width
  - Use –align array64byte compiler option

- **Intel MKL**
  - Can offload segments of computational work by default
    - Users should not have to modify Makefiles or compile options to take advantage of this

Selected environment Variables

Variable	Default Value	Description
MIC_ENV_PREFIX	Not defined by default	Sets the prefix for MIC environment variables so can have different values for env vars on host and MIC. Set it to “MIC” in what follows.
MIC_OMP_NUM_THREADS	Not defined by default	Sets the number of threads to utilize per MIC
MIC_KMP_AFFINITY	Not defined by default	Sets the thread layout on MIC e.g. balanced, compact, scatter. See next slide for details.
MIC_LD_LIBRARY_PATH	Not defined by default	Sets the $LD_LIBRARY_PATH value for MIC environment.
OFFLOAD_REPORT	0	0: No report. 1: Name of function using Automatic Offload; Effective work division; Time spent on host during call; Time spent on MIC during call. 2: Reports all of Level 1 plus the amount of data transferred to and from the coprocessor.
MKL_MIC_ENABLE	1	Enables automatic offload within MKL routines

MIC_KMP_AFFINITY

- **None**
  - This is the default setting on the compute hosts.
- **Compact**
  - Binds threads as close to each other as possible; this is the default setting on the MIC cards
- **Scatter**
  - Binds threads as far apart to each other as possible
- **Balanced**
  - This option applies only to MIC cards. It first scatters threads to each core, so that each core has at least one thread, and it sets thread numbers such that the different hardware threads of the same core are close to each other
- **Explicit**
  - Use explicit bindings

MIC_KMP_AFFINITY examples (6 OpenMP threads on one MIC card)

For simplicity, assume each MIC card has only 3 cores instead of 60 cores.

- **MIC_KMP_AFFINITY=compact**

Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	1	2	3	4	5						

- **MIC_KMP_AFFINITY=scatter**

Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	3			1	4			2	5		

- **MIC_KMP_AFFINITY=balanced**

Node	Core 1				Core 2				Core 3			
	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4	HT1	HT2	HT3	HT4
Thread	0	1			2	3			4	5		

MIC_KMP_PLACE_THREADS

- New environment variable available only for the MIC cards.
  - Does not replace KMP_AFFINITY, but works with it to set exact but still generic thread placement.
- Format: <n>Cx<m>T,<o>O
  - Use <n> Cores times <m> Threads with <o> of cores Offset.

Example:

- setenv KMP_PLACE_THREADS 40Cx3T,1O,
  - Use 40 physical cores, and 3 threads (HT2,3,4) per core.

Final Note on Placement

- **Although Intel Xeon Phis may support up to 240 threads (4 threads per core x 60 cores), it is strongly advised to leave at least 1 core available for the Intel Xeon Phi operating system to use.**
  - When setting environment variables such as $MIC_OMP_NUM_THREADS or $MIC_KMP_PLACE_THREADS, it is better to use 236 total threads and 59 cores.

- **For example (2-node MPI with offload, MKL and report):**

```
# Set up the environment
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=236
export MIC_KMP_AFFINITY="granularity=fine,compact"
export MIC_KMP_PLACE_THREADS="59cx4t"
export MKL_MIC_ENABLE=1
export OFFLOAD_REPORT=2

# Launch executable
aprun –n48 -cc none ./myexe.off
```

Coding example of offloaded matrix-matrix computation

```
main(){

  double *a, *b, *c;
  int i,j,k, ok, n=100;

  // allocated memory on the heap aligned to 64 byte boundary
  ok = posix_memalign((void**)&a, 64, n*n*sizeof(double));
  ok = posix_memalign((void**)&b, 64, n*n*sizeof(double));
  ok = posix_memalign((void**)&c, 64, n*n*sizeof(double));

  // initialize matrices
  ...
  //offload code.  Specify sizes of a, b and c since were dynamically allocated
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
  {
  //parallelize via OpenMP on MIC
#pragma omp parallel for
    for( i = 0; i < n; i++ ) {
      for( k = 0; k < n; k++ ) {
#pragma vector alignedN
#pragma ivdep
            for( j = 0; j < n; j++ ) {
              //c[i][j] = c[i][j] + a[i][k]*b[k][j];
              c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];

        }
      }
    }
  }
}
```

Functions but performs badly!

Obtain information about offloading

```
> cc  -vec-report2   -openmp offload.c
offload.c(57): (col. 2) remark: loop was not vectorized: vectorization
                                  possible but seems inefficient.

offload.c(57): (col. 2) remark: *MIC* LOOP WAS VECTORIZED.
offload.c(54): (col. 7) remark: *MIC* loop was not vectorized: not inner
loop.
offload.c(53): (col. 5) remark: *MIC* loop was not vectorized: not inner
loop.
```

- export OFFLOAD_REPORT=2
- aprun –n1 –d4 ./a.out

```
[Offload] [MIC 0] [File]            offload2.c
[Offload] [MIC 0] [Line]            50
[Offload] [MIC 0] [CPU Time]        12.853562 (seconds)
[Offload] [MIC 0] [CPU->MIC Data]   9830416 (bytes)
[Offload] [MIC 0] [MIC Time]        12.208636 (seconds)
[Offload] [MIC 0] [MIC->CPU Data]   3276816 (bytes)
```

Compilation Tip (aka lessons learned with Amber)

- **When using configure/autoconf scripts, sometimes they want to run a test program**
- **Since build is cross-compile from login node, binary is built to run on MIC but will be tested at configure time on login node and will fail**
- **Possible workarounds:**
  - First, try --host=x86_64-unknown-linux-gnu or maybe --host=x86_64-k1om-linux for configure, which tends to skip many test programs
  - Or, define –DMIC for compiler options such as CC, CXX, FC, etc., used in configure (e.g., export CC="cc –DMIC" then replace all "-DMIC" in generated Makefile with "-mmic" before compiling and building
    - find . -name Makefile | xargs sed -i 's/-DMIC/-mmic/g'
  - Or, hack configure script to add "aprun –k" (native) or "aprun –n1" (offload) before test programs and build within batch session launched on the MIC nodes

Should I use Native Mode?

- **Important considerations:**
  - Overheads associated data movement
  - How to avoid unnecessary data movement
  - Peak transfer rates

Is my code suitable for native mode?

- **Building a native app is a fast way to get existing software running with minimal code changes.**
- **First, ensure that the application is suitable for native execution.**
  - Data parallelism, usage of parallel algorithms, and application scalability are criteria for targeting the MIC, but not for distinguishing between the usage of offload or native mode.
- **An application likely to benefit from the large number of cores available with native execution tends to have the following characteristics.**
  - A modest memory footprint, less than the available physical memory on the device
  - Very few serial segments
  - Does not perform extensive I/O
  - A complex code structure with no well-identified hot kernels that could be offloaded without substantial data transfer overhead
  - Just as for offload mode, additional software optimizations, especially vectorization, are likely to be needed to achieve good performance.

Offload mode (heterogeneous)

- **Offload model uses simple pragmas/directives to specify code sections and data to offload to the MIC. No further special programming API is needed.**
- **All setup/teardown, data transfer, and synchronization is managed by the compiler and runtime.**
  - Application starts on host platform. When a offload region is encountered, either:
    - If a supported MIC is found and is available, the offload region and data is transferred (if needed) and run on the target device.
    - If for some reason there is no available MIC target, the code region is run on the CPU.
- **Host application and MIC target device do not share memory. Two techniques are used to deal with the independent memory spaces:**
  - App uses offload pragmas/directives to help guide moving data to and from the host. This model is ideal for apps with large blocks of contiguous data.
  - For complex data structures with pointers and noncontiguous blocks of data, the Intel C++ compiler supports language extensions to emulate virtual shared memory to keep the data structures synchronized between host and target MIC device(s).

Selecting Code Sections to Offload to MIC

- **Based on Parallelism**
  - Choose highly-parallel sections of code. Serial code offloaded to MIC will run much slower than on the CPU.
- **Changing Scope of Offloaded Sections Based on Data Flow**
  - Selecting code regions based on parallelism to offload may yield many small sections to offload. Must be balanced with the need for transferring data back and forth between CPU and MIC.
  - Data exchange can be slow (subject to PCI-E speeds) and difficult.
  - If two parallel sections do some serial processing between them, choose between
    - moving the output data of the first parallel section back to the CPU, running the serial code on the CPU and then moving the input data of the second parallel region from CPU to coprocessor, or
    - keeping the data on the MIC and running the serial code there (making the entire parallel-serial-parallel section of code an offload unit).

Some interesting offload notes

- **Initialization Overhead**
  - When app performs first #pragma offload, all MIC devices assigned to app are initialized – this can take time. So...
    - Don’t put first offload within timer measurement
    - Perform dummy offload to device at start of app
    - Or, use OFFLOAD_INIT=on_start env var to pre-initialize MIC devices
- **Minimize input data**
  - Compute locally if possible
- **Keep data persistent across offloads**
  - If data vals at end of offload are needed for later offload, keep on MIC
- **Statically allocated data persistence**
  - Static data (“static” in C, Fortran common blocks, save data, etc.) retains values across offloads unless overwritten
    - Use nocopy clause to reuse values
- **Function inlining into offload constructs**
  - Functions called within #pragma offload region are not inlined by the compiler
    - Either manually inline functions or place entire offload construct into its own function

Resources – lots of information out there!

Intel Xeon Phi Main Page: http://software.intel.com/mic-developer
Programming and Compiling for Intel MIC Architecture: http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
Native and Offload Programming Models: https://software.intel.com/en-us/articles/native-and-offload-programming-models

Great guide:http://www.prace-ri.eu/best-practice-guide-intel-xeon-phi-html

Cray Programming Users Guidehttp://docs.cray.com/books/S-2529-116/ (or more recent vs)

Section “**Targeting for Intel Xeon Phi**”

Performance Measurement and Analysis Tools http://docs.cray.com/books/S-2376-63 (or possibly more recent version)

Section “**Using CrayPat on Intel Xeon Phi**”

Real world example using WRF (note no symmetric mode available on XC40) https://www.nccs.nasa.gov/images/MIC-Talk3.pdf