**Offload Exercises: Option B**
**(based on code examples used in presentation)**

**Contents of README in offload_demos.tar:**

```
        offload_demos.tar -- Introduction to Xeon Phi Explicit Offload
        ************************************************************

djames (at) tacc.utexas.edu
Initial release: 01 Oct 2013

Revision History
****************

  04 Oct 2013    Removed references to alloc_if and free_if in
                 off06stack.  These two keywords apparently have
                 no effect on stack-based arrays.
  06 Jul 2014    Added 4 items to 1st offload exercise. KM

 ---------------------------------------------------------------------

  Preparation
  ***********

  -- Log into Stampede

  -- From an appropriate directory (e.g. $WORK), copy and extract
     the files by executing:

       tar xvf ~train00/offload_demos.tar

  -- Enter the directory containing the files:

       cd offload_demos

  -- Start a 1-node srun or idev session; e.g.

       srun --pty -t 01:00:00 -n 16 -p development /bin/bash -l

          ...or...

       idev


  Explicit Offload
  ****************

        Run experiments and exercises as desired from the off* source code
        files used in the tutorial.  The list below includes suggestions
        and possibilities.

        All C files should compile with "icc -openmp off0Xxxxxx.c",
        where off0Xxxxxx.c represents the name of the file.
        The resulting executable will have the name a.out.

        All Fortran files should compile with "ifort -openmp off0Xxxxxx.f90",
        where off0Xxxxxx.f90 represents the name of the file.
```

```
The resulting executable will have the name a.out.

Suggested experiments/exercises:

off00host through off04proc
**************************

Exercise: modify off00host to offload a single line of code,
  a block of code, an OpenMP region, or a procedure that
  you write.  See off01 through off04 respectively for
  appropriate ways to do so.
```

1.) Offload a single line:
totalProcs = omp_get_num_procs();

2.) Offload a block (2 lines) of code
totalProcs = omp_get_num_procs();

maxThreads = omp_get_max_threads();

3.) Create a loop, execute it in parallel by using an OpenMP parallel do/for construct and offload it onto the MIC. (Loop: Assign values in an array.)

for ( i=0; i<500000; i++ ) { a[i] = (double)i; } //C

do i=1,500000; a(i) = real(i); end do //F90

4.) Create 2 simple routines , and offload them

i = successor( 123 ); // adds 1 to constant argument, return in value in i

increment( &i ); // increments i

i = successor( 123 ) // function adds 1 to constant argument, return in value in i

call increment( i ) // subroutine increments i

```
Exercise: export OFFLOAD_REPORT=1, then 2, then 3, and observe
  the results on your offloaded code(s).  Experiment as well
  with the compiler flag "-opt-report-phase=offload".

Exercise: remove all declspec/attribute decorations, then
  compile with "-offload-attribute-target=mic" and observe
  the results.

off05global
***********

Exercise: experiment with decorated and undecorated variables
  of various types (global, static, automatic) and observe the
```

```
  effect on compilation/execution.  Observe as well the effect
  on the output results when OFFLOAD_REPORT=1,2 or 3.

Exercise: remove all declspec/attribute decorations, then
  compile with "-offload-attribute-target=mic" and observe
  the results. “unset OFFLOAD_REPORT” will return to no report.

off06stack
**********

Exercise: experiment with appropriate and inappropriate
  choices of in, out, in/out, and nocopy and observe the
  results.

Exercise: experiment with MIC_STACKSIZE and the size of the
  arrays.  For example, increase array size until the code
  exceeds stack limits, then modify the stack limits to
  fix the problem.  Try to predict in advance the array size
  that will cause problems.  How does the code behave when
  you exceed stack limits?  How helpful is the error message?

Exercise: in the C code, define "const int N=100000", then
  replace the hard-wired literal "100000" with N.  Use icc
  to compile the program as C code, then use icpc to compile
  the program as C++ code (icpc interprets the .c suffix
  as a C++ source file).  Expect a difference in behavior.

off07heap
*********

Exercise: experiment with the size of the arrays.
  How large a heap are you able to offload?  Attempt
  to predict in advance the problem size at which
  you might begin to expect trouble.  How does the
  code behave if you attempt to use too much memory?

Exercise: export OFFLOAD_REPORT=1, then 2, then 3, and observe
  the results on your offloaded code(s).  Experiment as well
  with the compiler flag "-opt-report-phase=offload".
  How do the reports for heap data differ from those for
  stack data?  How do reported times vary with problem
  size?  What do you think the reported times are
  measuring?

off08asynch
***********

Exercise: experiment with signal and wait settings to
  generate conditions under which the asynchronous
  offload does and does not finish before the code
  prints the results.

Exercise: experiment with offload (with wait) vs offload_wait;
  compare the functionality.

off09transfer
*************

Exercise: experiment with signal and wait settings to
```

```
          generate conditions under which the code does and
          does not execute properly.  Conduct similar experiments
          with in/in/inout/nocopy as well as alloc_if/alloc_free.

        Exercise: given invalid offload conditions (for example,
          no MIC allocation before data is needed), compile
          the code with "-no-offload" and observe its behavior.

--------------------------------------------------------------------------------

  MKL Automatic Offload
  *********************

        Run experiments and exercises on MKL Automatic Offload using the
        Intel demonstration codes used in the tutorial.  The list below
        includes suggestions and possibilities.

        To compile the C demo:   icc -openmp -mkl ao_intel.c
        The resulting executable will have the name a.out.

        To compile the C demo:   icc -openmp -mkl ao_intel.f
        The resulting executable will have the name a.out.

        Remember to set appropriate environment variables:

          export MKL_MIC_ENABLE=1
          export OMP_NUM_THREADS=16
          export MIC_OMP_NUM_THREADS=240
          export OFFLOAD_REPORT=2

        Suggested experiments/exercises:

        Exercise: experiment with the values of the environment
          variables (e.g. thread counts, AO enabled/disabled) and
          compare the results.

        Exercise: experiment with problem size and observe the results.
          At what threshold(s) does MKL begin to use the MIC?
          At what threshold(s) is it faster to use Automatic Offload?

        Exercise: experiment with work division (see
          http://software.intel.com/sites/products/documentation/
          doclib/mkl_sa/11/mkl_userguide_lnx/
          GUID-3DC4FC7D-A1E4-423D-9C0C-06AB265FFA86.htm).  What
          settings are optimal?

        Experiment with two MICs in the normal-2mic queue.  How much
          better can you do?  With what size problems?
```