OLCF-6 Benchmark M-PSDNS Code Run Rules:

See Section 3.4 of the Technical Requirements Document for OLCF-6 for a general discussion of the requirements. The following discussion will explain how these requirements apply to the M-PSDNS benchmark. Offerors are asked to run the benchmarks with "as is" source code and, if desired, with "optimized" source code. For the "as is" set of benchmarks, Offerors are asked to limit the modifications to the source code to only those changes which are required to run the benchmarks. An example of this situation is if an Offeror wants or needs to use a FFT implementation other than hipFFT such as cuFFT or OneMKL FFT. For the "optimized" benchmarks, Offerors are allowed to make additional modifications to the source code. However, even these changes should be limited to minor restructuring instead of large wholesale changes. The intent here is to demonstrate the potential improvements between the "as is" and the "optimized" source codes with a minimal cost of time and development. Examples for this situation include changes to data management and movement between the CPU and GPU, changes to the OpenMP Target directives, minor restructuring of the nested computing and working loops and the MPI communications.

For both the "as is" and "optimized" benchmarks, Offerors are asked to run the cases using the input file provided with each case if possible. This will provide a consistent comparison of performance across systems. However, Offerors can modify the input files according to their needs:

1. If you need a smaller problem due to memory limitations:
    a. Smaller problem sizes can be attempted by changing the values for nx, ny, and nz in the input file (*the first three values on the 2nd line of the input file*). The values for nx, ny, and nz should be identical and evenly divisible by the number of MPI ranks being used. Further, values that are pure multiples of 2 are best.
    b. Alternatively, you can spread the current cases across more MPI ranks by increasing the values in the dims file. This process was done for the double precision computations in this repo. The values for nx, ny, and nz should be evenly divisible by the number of MPI ranks being used.

2. If you can't perform MPI communications on the GPU or you simply want to investigate MPI on the CPU:
    a. Set gpumpi=0 in the input file (*last entry on the 2nd line of the input file*).

3. If you want detailed information about the MPI communications:
    a. Uncomment the line for DETAIL_MPI in the makefile_initialization file and rebuild the code. Collecting this data will increase the runtime and should not be used during formal performance testing.

Run rules for "as is" source code:

Without hipfort:
If the user chooses not to use hipfort, then the user must supply their own fortran interfaces for the library functions. The Offeror must modify the source code and makefile to remove the dependence upon hipfort and to provide instructions for the new fortran interfaces.

Without hipfort and hipFFT:
If the user chooses not to use hipFFT, then the user must modify the source code to use their desired package. Using CUDA cuFFT instead of hipFFT is a relatively straightforward change, but moving to another FFT implementation will require more work. The Offeror must provide the Fortran interfaces for their chosen FFT implementation.

Regardless of the FFT implementation being used, code modifications for the "as is" benchmarks should be limited only to those changes that are required for correct function.

For the "as is" benchmarks, changes to the source code should be limited to the following non-critical and critical files in the src directory:

1. Non-critical to code:
    a. GPUmeminfo.F90 (this checks GPU memory usage)
    b. hipcheck.F90 (checks for hipfft success)
    c. procinfo.F90 (uses a hip function to get device info for reporting purposes)
2. Critical to code:
    a. module.F90 (where we declare variables)
    b. epfftw.F90 (fft plans and workbuffer space is set here)
    c. itransform_vel.F90 (performs hipfftExec*** function)
    d. transform.F90 (performs hipfftExec*** function)
    e. kxtran_gpu.F90 (performs hipfftExec*** function)
    f. kxcomm1_gpu.F90 (performs hipfftExec*** function)
    g. kxcomm2_gpu.F90 (performs hipfftExec*** function)
    h. xktran_gpu.F90 (performs hipfftExec*** function)
    i. xkcomm1_gpu.F90 (performs hipfftExec*** function)
    j. xkcomm2_gpu.F90 (performs hipfftExec*** function)

The files in the non-critical list contain uses of hip functionality that can be removed and/or altered without any impact on the performance of the M-PSDNS algorithm. However, files in the critical list must be altered with care to maintain correct function of the algorithm.

For CUDA cuFFT: Since hipFFT uses interfaces similar to cuFFT's interfaces, switching to cuFFT is straightforward and requires the user to supply the fortran interfaces for the CUDA libraries.

For another FFT implementation: Moving to another FFT implementation will likely require substantial changes to the code. The majority of changes will revolve around creating the plans and setting with the work buffers in epfftw.F90. Once the FFT plans are created, modifying the

rest of the critical files will involve changes required to perform a forward or inverse transform. The user will also need to provide the fortran interfaces to their FFT implementation.

Run rules for "optimized" source code:

Although not as restrictive as the "as is" rules, Offerors should avoid large, wholesale changes to the algorithm. Instead, the purpose here is to demonstrate potential improvements that can be obtained with a minor investment of time, effort, and expense.