**Blue Waters Petascale Semester Curriculum v1.0**

**Unit 8: OpenACC**

**Lesson 1: Accelarating Scientific Applications**

**Exercise Instructions for Students**

*Developed by R. Phillip Bording for the Shodor Education Foundation, Inc.*



*Except where otherwise noted, this work by The Shodor Education Foundation, Inc. is licensed under CC BY-NC 4.0. To view a copy of this license, visit*[*https://creativecommons.org/licenses/by-nc/4.0*](https://creativecommons.org/licenses/by-nc/4.0)

*Browse and search the full curriculum at*[*http://shodor.org/petascale/materials/semester-curriculum*](http://shodor.org/petascale/materials/semester-curriculum)

*We welcome your improvements! You can submit your proposed changes to this material and the rest of the curriculum in our GitHub repository at*[*https://github.com/shodor-education/petascale-semester-curriculum*](https://github.com/shodor-education/petascale-semester-curriculum)

*We want to hear from you! Please let us know your experiences using this material by sending email to* [*petascale@shodor.org*](mailto:petascale@shodor.org)

Write a simple matrix multiplication serial code in C and Fortran:

* Add directive to the code that defines the variable for size of the matrices
* Define arrays for each matrix and allocate required memory
* Write for loop(s) to initialize the matrix arrays with some numbers. You could use random number generators.
* Write **do/for** loops to carry out the matrix multiplication

Parallelize the ***for*** loops with OpenACC:

* Find the compute intensive loops to parallelize.
* Use the Linux time command to document entire job runtime.
* Use OpenACC directives to copy data from CPU host to GPU
* Use timers’ routines available either in OpenACC, C, OpenMP libraries to measure performance.
* Be careful in using reduction operators like: min and max.
* Compare performance of your program with using PARALLEL LOOP vs. KERNELS.
* Using the matrix multiply triple loops – test all possible combinations of the **do/for** loops and compare run times.
* Do a scaling and algorithmic complexity analysis of these programs.