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

**Unit 6: Hybrid MPI + OpenMP**

**Lesson 3: Pebble in Pond Wave Equation**

**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 random matrix addition serial code in C and Fortran.

* Add memory allocation methods to the code that defines the variables for size of the matrices
* Define arrays for each matrix and allocate required memory
* Write **do/for** loop(s) to initialize the matrix arrays with some numbers, you could use random number generators.
* Write loops to find the min and max array values.

Distribute the arrays across a one-dimensional processor network with MPI.

* Assume a three deep ghost region between processors.
* Compute the min – max array values in the individual memory spaces and gather them into the global min-max values.
* Add OpenMP to the MPI code and document runtime improvements and results accuracy.
* Use the Linux time command to document entire job runtime.
* Use the Linux timer command to document performance as the array size scales to large numbers – gigabytes of memory.
* Be careful in using reduction operators like: min and max.
* Compare performance of your parallel program with a single processor program.
* Do a scaling and algorithm complexity study of these programs.