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

**Unit 4: OpenMP**

**Lesson 4: OpenMP Target Offload**

**Sample Assessment**

*Developed by Maria Pantoja for the Shodor Education Foundation, Inc.*



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

•**APPLY** basic OpenMP pragmas to solve a common scientific parallel pattern

Questions:

1. Ask Students to download and run the sample code
2. Ask students to change number of threads executing the OpenMP pragmas
3. Ask Students to comment all OpenMP pragma lines and measure the difference in execution time

•**IDENTIFY** which loops in a program are parallelizable

Questions:

1. Does the next loop contain a loop dependency?

for( i = 1; i<100; i++){

A(i) = B(i)+1

C(i) = A(i-1)\*2 <- yes here, to calculate this iteration we need the prev. one

}

1. Does the next loop contain a loop dependency?

for( i = 1; i<100; i++){

C(i) = A(i-1)\*2 <- yes here, to calculate this iteration we need the prev. one

}

1. Does the next loop contain a loop dependency?

for( i = 1; i<100; i++){

A(i) = B(i-1)\*2 <- no

}

•**COMPARE** sequential to parallel execution times

1. Use openMP timing functions to measure execution times of different loops and/or functions on the code. Example

**double start;**

**double end;**

**start = omp\_get\_wtime();**

... work to be timed ...

**end = omp\_get\_wtime();**

**printf(~Work took %f seconds\n~, end - start);**

•**USE** Profiler tools to evaluate bottlenecks in code

Use gprof: <https://users.cs.duke.edu/~ola/courses/programming/gprof.html>

And evaluate the provided code

Some possible advanced questions to students:

1. Ask Students to implement the complete Laplace Solver

A.Using OpenMP parallel for pragma

B.Using OpenMP target teams distribute for pragma

C.Compare the results