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

**Unit 7: CUDA**

**Lesson 11: Branching and GPGPU Efficiency (profiling and debugging)**

**Exercise Instructions for Students**

*Developed by David A. Joiner 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)

Compile and run the “branching\_student.cu” code. (“nvcc -o branching\_student branching\_student.cu”)

This code will take a large loop of operations and run it inside of a CUDA kernel. Each thread will be assigned a task, and depending on the thread ID, either all threads will be assigned the same task, or some threads will be assigned different tasks. This will be done in sets of threads of size nBreak, and work will be assigned vie a modulus statement, so that threads 0 to nBreak-1 will get one set of instructions, nBreak to 2 nBreak-1 will get another set of instructions, and so on.

The code is designed to run for nBreak ranging from 8 to 32, with most GPUs having a warp size of 32.

1. Run the code “./branching\_student 32” with nBreak=32. Run it repeatedly for decreasing nBreak, and make a plot of runtime as a function of the work set size in the GPU kernel.
2. Do you see evidence for branching having an impact on efficiency of a GPU kernel? If so, how does the warp size affect this? Explain in detail, referring back to the plot you made in example 1.