CAAM 520 Spring 2015 Homework 06

T. Warburton

April 15, 2015

You must complete the following tasks by 5pm on Friday 04/24/15.

Your solutions to the homework must be committed to your Subversion repository in a sub-directory HW06 with subdirectories HW06/Q1 and HW06/Q2.

You may work in one group of one or two students for this homework. You can adapt your CUDA code from HW05 as the base implementation and convert to OpenCL and OCCA as appropriate.

All source code, header files, \LaTeX files, Makefiles etc should be committed to your own repository in a directory named HW06. You are required to include a Makefile that makes an executable called main in each of the two sub-directories. You may submit the same report as the partner from your team.

Use \LaTeX to write and typeset your report (saved as HW06/report.pdf in your Subversion repository).

You may only collaborate with your team partner and/or consult the instructor or grader for verbal assistance. You are encouraged to use textbooks and internet resources. You must cite all resources used via footnotes or a bibliography.

[Hint: If you work in a team your report may be strengthened if you assign one question to each team member and construct your report around the difference in experiences of implementing in OpenCL and OCCA]

150 points each will be awarded for the successful completion of Q1 and Q2.
Q1 Translate your CUDA accelerated sparse matrix solver for a structured resistor network to OpenCL.

Problem description: port your HW05 code from CUDA to OpenCL and evaluate the performance on a range of compute devices.

Tasks: Your task is to translate your CUDA HW05 code to OpenCL. You are expected to maintain the two-dimensional thread-array and use local memory within each work-group to reduce the amount of memory transfers.

Kernel translation: Describe the translation process you use to convert your CUDA compute kernel to OpenCL. For instance mention syntax changes.

Host code translation: Describe the OpenCL API calls you used in the translation of your CUDA host code.

OpenCL implementation: Implement your shared-memory algorithm using features from OpenCL that we have discussed so far including:

- Local memory shared between threads in a work-group.
- Barriers to make sure that loads to local memory have completed.
- OpenCL event based timing.
- Coalesced memory accesses.
- Avoiding bank conflicts (if relevant).

Test case: Compare the output of your HW06/Q1 OpenCL code with your HW05 CUDA code. Explain any differences.

Performance:

- Compare the performance of your OpenCL code with your CUDA code on the NVIDIA GTX 980 GPU in ketchup. Experiment with a range of $N$ to investigate the relative overhead of launching an OpenCL kernel compared to a CUDA kernel.
- Compare the performance of your OpenCL code across all the different GPU and CPU platforms and devices in ketchup.

Extra credit: the Intel OpenCL platform may be able to use SIMD vectorization for the work-group items on the Intel CPU in ketchup if the size of the work-group is a multiple of 8. Experiment to see if this is the case for your code.
Q2 Translate your CUDA or OpenCL accelerated sparse matrix solver for a structured resistor network to OCCA using the OKL kernel language.

Problem description: port your HW05 or HW06/Q1 codes to OCCA and evaluate the performance on a range of compute devices using a range of thread models. You are expected to maintain the two-dimensional thread-array and use occaShared memory within each work-group to reduce the amount of memory transfers.

Kernel translation: Describe the translation process you use to convert your CUDA or OpenCL compute kernel to the OCCA OKL kernel language. For instance mention syntax changes.

Host code translation: Describe the OCCA host library calls you used in the translation of your original code. Contrast the OCCA coding progress with coding in OpenMP, OpenCL, and CUDA.

OCCA implementation: Implement your shared-memory algorithm using features from OCCA that we have discussed so far including:

- occa:device object.
- occa:kernel objects.
- occa:memory objects.
- Explicit outer# and inner# loops.
- shared memory scratch pads used by all threads in a work-group.
- barrier to make sure that loads to local memory have completed.
- Coalesced memory accesses.
- Avoiding bank conflicts (if relevant).

Test case: Compare the output of your HW06/Q2 OCCA code with your HW04 OpenMP, HW05 CUDA code, and HW06/Q1 OpenCL code. Explain any differences.

Performance:

- Compare the performance of your OCCA code with your previous homework codes on the NVIDIA GTX 980 GPU, and Intel CPU in ketchup using OCCA to run in OpenCL, CUDA, and OpenMP mode where possible. Experiment with a range of N to investigate the relative performance of the OCCA code with the hand written native codes from previous homeworks. Present your results in table format.