CAAM 520 · Computational Science II

Spring 2019 · Rice University

MAIN PAGE // HOMEWORK

Notes, Codes, and Supplementary Material

Lecture 33 (4/17):	OCCA lab: install OCCA on both Ubuntu and NOTS. OCCA 2-step installation instructions Installation on NOTS: • get a GPU using the interactive shell (--reservation=CAAM520_2) • Run "module load CUDA" • Follow the installation procedure here • Add "export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:/home/your_username/occa/lib"" to your "~/.bashrc" file. This will ensure OCCA remains in your path when you log back in • "cd examples/cpp/1_add_vectors/", type "make -j", and run ./main to est. Installation on Ubuntu: • Follow the installation procedure here • Add "export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:/home/your_username/occa/lib"" to your "~/.bashrc" file. This will ensure OCCA remains in your path when you log back in • "cd examples/cpp/1_add_vectors/", type "make -j", and run ./main to est. • You may observe that the examples run, but return a segmentation fault. This is likely due to a documented issue with Intel's OpenCL installation + the OpenCL command clReleaseProgram (which is run during clean-up by OCCA) okl_demo.zip: example showing the use of inner/outer loops, shared and exclusive memory in OCCA . Once you've verified the installation, you may work on the EC portion of HW 5 involving implementations of Jacobi + reduction in OCCA.
Lecture 32 (4/15):	Structure and syntax of OCCA. Implicit "outer" and "inner" for loops Implicit synchronization between loops New memory space: exclusive variables are thread-local but persist between loops.
Lecture	Finishing up OpenCL, introduction to OCCA.

31 (4/12): Differences between OCCA and OpenCL: translation into native languages for OCCA
OCCA website

Lecture 30 (4/10): OpenCL lab.
Goal: install OpenCL on your Ubuntu VirtualBox installation, test with an OpenCL implementation of reduce.cu

Main steps (based off of this askUbuntu answer):
- Download the OpenCL SDK for Intel CPUs (if you have another type of CPU, you may need a different SDK).
- Install OpenCL libraries and tools using

- sudo apt install ocl-icd-libopencl1
- sudo apt install opencl-headers
- sudo apt install clinfo
- sudo apt install ocl-icd-opencl-dev

- Install package to convert .rpm to .dev files:

- sudo apt-get install -y rpm alien libnuma1

- Untar downloaded OpenCL SDK files

- tar -xvf opencl_runtime_16.1.1_x64_ubuntu_6.4.0.25.tgz

- Turn rpm files to deb

- cd opencl_runtime_16.1.1_x64_ubuntu_6.4.0.25/rpm/
- fakeroot alien --to-deb opencl-1.2-base-6.4.0.25-1.x86_64.rpm
- fakeroot alien --to-deb opencl-1.2-intel-cpu-6.4.0.25-1.x86_64.rpm

- Install .deb packages

- sudo dpkg -i opencl-1.2-base_6.4.0.25-2_amd64.deb
- sudo dpkg -i opencl-1.2-intel-cpu_6.4.0.25-2_amd64.deb

- Create local config file

- sudo touch /etc/ld.so.conf.d/intelOpenCL.conf

- Open the file and add OpenCL config info

- sudo emacs -nw /etc/ld.so.conf.d/intelOpenCL.conf
- Type "/opt/intel/opencl-1.2-6.4.0.25/lib64/clinfo" in the file and close

- Create a "vendors" dir and create an icd link

- sudo mkdir -p /etc/OpenCL/vendors
- sudo ln /opt/intel/opencl-1.2-6.4.0.25/etc/intel64.icd /etc/OpenCL/vendors/intel64.icd
- sudo ldconfig

- Test the installation:

- Build "cldevices.cpp" using "g++ cldevices.cpp -o cldevices -lOpenCL"

- Run "./cldevices" to view available devices (should just be a CPU)

End goal of the lab: convert the optimized reduce.cu CUDA kernel to working OpenCL code. Test the code on Ubuntu (VirtualBox), and if possible, on NOTS.

Lecture 29 (4/8): Converting CUDA to OpenCL, timing OpenCL code
clmxm_timing.cpp: OpenCL matrix-matrix multiplication code with OpenCL event timing.
mxm.cl: kernel file for above example. Converting between CUDA and OpenCL: a non-comprehensive list of changes

- "__global__" keyword for kernels becomes "__kernel"
- Add "__global" to pointer arguments to global (DRAM) memory
- "__shared__" keyword for shared memory becomes "__local"
- "__syncthreads()" becomes "barrier(CLK_LOCAL_MEM_FENCE)"

To build and run OpenCL code on NOTS, use "g++ -I$EBROOTCUDA/include clmxm_timing.cpp -lOpenCL". This will use NVIDIA's OpenCL implementation

Lecture 28 (4/5): More on OpenCL Wrappers to simplify OpenCL host code (setup and kernel build functions)
cldevices.cpp: code to determine OpenCL devices available
cldemo.cpp: (un-simplified) code demonstrating building and running of an OpenCL kernel
foo.cl: kernel for above code. cldemo.cpp: simplified code demonstrating building and running of an OpenCL transpose kernel
transpose.cl: kernel for above code.

Lecture 27 (3/29): Introduction to OpenCL (Chapter 2 of OpenCL book)
OpenCL book pdf.

Steps to an OpenCL program:

Setup
-choose platform and device (clGet*Info, clGetPlatformIDs, clGetDeviceIDs)
-combine platform and device to create compute environment (clCreateContext)
-create “stream” (clCreateCommandQueue).
- Options for sequential or out-of-order scheduling (CL_QUEUE_PROFILING_ENABLE / CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE).

Build program
-create program object from one or more kernel sources (clCreateProgramWithSource)
-program object gets built into executables (clBuildProgram).
-build executable kernel from program (clCreateKernel)

Allocate memory
-allocate memory (clCreateBuffer). Specify type of memory
-CL_MEM_COPY_HOST_PTR = memcpy from specified host array
-CL_MEM_USE_HOST_PTR = specify host pointer. In some implementations, uses pinned memory.
-CL_MEM_ALLOC_HOST_PTR = pinned memory (host-accessible)

Queue kernels, set arguments
-set kernel arguments (clSetKernelArg)
-queue kernel to run (clEnqueueNDRangeKernel).
-get result back (clEnqueueReadBuffer)
-not used in demo: write buffer (clEnqueueWriteBuffer)

Synchronize and clean up
-clFlush - blocks until commands in queue have executed (not completed!)
-clFinish - blocks until commands in queue have finished
-Context creates queue, program, memory buffers. Program creates kernels. Kernels + memory feed into queue.

Lecture 26 (3/27):	CPU/GPU data transfers pinnedMemoryExample.cu: example of faster transfers using pinned memory and cudaMallocHost Can overlap CPU computation with CUDA kernels trivially Multiple CUDA streams + pinned memory allow overlap of GPU compute/memory transfer async.cu: example from the CUDA samples on overlapping GPU computation with data transfer using 4 CUDA streams.
Lecture 25 (3/25):	Optimizing matrix multiplication with shared memory Matrix multiplication with shared memory tiling. mxm.cu: code with three versions of matrix transposition. Discussion of full CUBLAS strategy for optimizing matrix-matrix multiplication
Lecture 24 (3/22):	Optimizing matrix transposes with shared memory Bank conflicts for 32-by-32 arrays transpose.cu: code with three versions of matrix transposition (global memory, shared memory, shared memory with bank conflict treatment)
Lecture 23 (3/20):	Even more on profiling GPU kernels for efficiency Roofline model: compute bound vs memory bound kernels Peak vs realistic performance Comparison of matrix-matrix multiplication with CUBLAS matmult_cublas.cu: example code for CUBLAS (compilation instructions in header) ilp.cu: optimizing code for adding two vectors by increasing work per thread. roofline.m: Matlab code for computing roofline plots.
Lecture 22 (3/18):	More on profiling GPU kernels for efficiency Roofline model: compute bound vs memory bound kernels Peak vs realistic performance
Lecture 21 (3/8):	Profiling GPU kernels for efficiency GPU occupancy: online occupancy calculator nvprof (Nvidia profiler) for computing timings, number of floating point operations, bandwidth.
Lecture 20 (3/6):	Optimizing a reduction kernel Concepts: warp divergence, shared memory bank conflicts. Example optimized reduction code: reduce.cu

Reduction code based off of Mark Harris' optimized reduction kernel talk.

Lecture 19 (3/4): More on shared memory
Matvec example code: matvec.cu
Stencil example code: stencil.cu

Lecture 18 (2/21): Introduction to shared memory
matvec.cu

Lecture 17 (2/19): More on GPU computing
Matrix-matrix multiplication: matmult.cu
Global GPU memory and coalesced memory access
Introduction to nvprof (Nvidia profiler) for timing
Kirk and Hwu Chapter 3.

Lecture 16 (2/18): Introduction to GPU computing
Host/device setup
Nvidia, AMD GPUs: CUDA vs OpenCL
add.cu: CUDA code to add two vectors. Compile using "nvcc add.cu -o add", run as usual ("./add").
To run on NOTS in interactive mode: "srun --pty --partition=commons --gres=gpu:1 --time=1:00:00 $SHELL" will request a single GPU for 1 hour.

Lecture 15 (2/18): Divide and conquer using OpenMP tasks
Recursive blocked matrix-matrix multiplication code

Lecture 14 (2/15): More advanced OpenMP: task-based parallelism
OpenMP tutorial with several examples
An extensive list of more OpenMP resources

Lecture 13 (2/13): Code optimization
matmat_omp.c: inefficient matrix-matrix multiplication code
. dot_omp.c: optimized matrix-matrix multiplication code
Cache memory, data layouts, compiler optimization flags
Assorted OpenMP options: nested parallelism, loop collapsing (e.g. collapse(2)), conditional statements in OMP pragmas.

Lecture 12 (2/11): Introduction to shared memory parallelism and OpenMP
helloworld_omp.c: hello world using OpenMP
dot_omp.c: computing a dot product
Enabling OpenMP: "gcc -fopenmp ..."
Pragmas, parallel for loops, shared and private variables, race conditions.

Lecture 11 (2/6): Domain decomposition using graph partitioning
Link to METIS by the Karypis lab.
Installation instructions: install Cmake (sudo apt-get install cmake), cd into metis-5.1.0/, type "make config" and "make".
metisDriver.c: example code using Metis to partition a structured grid.
Other methods for load-balancing and domain decomposition: space-filling curves (see Figures 1,2 for illustrations)

Lecture 10: Debugging MPI
Debug option 1: launch multiple gdb processes using "mpiexec -n 2 xterm -e gdb

(2/4):	./a.out" Debug option 2: automatically attach gdb (see debug_mpi.c, run using "mpiexec -n 2 ./a.out") Open-MPI FAQ for debugging MPI.
Lecture 9 (1/31):	MPI_Probe, MPI_Getcount, timing probe.c: demo of MPI_Probe, MPI_Getcount matvec2.c for timing and matvec_timing.slurm for running on NOTS (must compile matvec2.c first).
Lecture 8 (1/25):	Parallel even-odd sort Even-odd sort code based on Pachecho 3.7.2: parsort.c.
Lecture 7 (1/23):	Lab on matrix-vector products, NOTS Description of the lab. Matrix-vector product code matvec.c using row-based storage.
Lecture 6 (1/18):	More on collective communication in MPI Butterfly communication: MPI_Allreduce, MPI_Allgather, MPI_Alltoall all_collectives.c: demo of above routines. Parallel matrix-vector products Using NOTS
Lecture 5 (1/16):	Collective communication in MPI MPI_Reduce, MPI_Bcast, MPI_Barrier, MPI_Gather, MPI_Scatter collectives.c: demo of above routines. Butterfly communication: MPI_Allreduce, MPI_Allgather, MPI_Alltoall
Lecture 4 (1/14):	Hw 1, reductions in MPI. MPI_REDUCE reduce.c: computes a reduction using tree parallelism and compares the result to MPI_REDUCE.
Lecture 3 (1/11):	Interactive lab. Set up VirtualBox (https://www.virtualbox.org/). - Set up Ubuntu 16.04 on VirtualBox (64 bit at http://releases.ubuntu.com/16.04/ubuntu-16.04.5-desktop-amd64.iso, 32 bit at http://releases.ubuntu.com/16.04/ubuntu-16.04.5-desktop-i386.iso) - Information for running 64 bit on a 32 bit machine from Seth Brown (optional): Link - I recommend setting up VM with at least half system memory for performance. - Install emacs: sudo apt install emacs (or VIM). - Install MPICH2: sudo apt-get install mpich (should be MPICH2) - Install Git: sudo apt-get install git Set up Github repository at https://github.com/ - Create git repo on Github - Check out Git repo in VirtualBox (cd ~, git clone https://github.com/*github_id*/*repo_address*) - Edit your README file to add your name, then commit (git add README.md, git commit -m “committing update to README file”, git push) - Try running MPI demo programs. Given time, write "ping-pong" example. - add "jlchan" and "cthl" as Collaborators to your Github repository. Lab: construct a "ping-pong" MPI program with 2 ranks which sends a ping_pong

variable between each rank, incrementing it until ping_pong is larger than 10.

Lecture 2 (1/9):	More message passing with MPI. MPI_Send/Recv behavior, MPI_Sendrecv Issues with message passing (deadlock) Nonblocking MPI communication: MPI_Isend, MPI_Irecv, MPI_Wait even_odd.c Pacheco 3.3
Lecture 1 (1/7):	Introduction to MPI (Chapter 3 in Pacheco) Simple MPI programs, myhelloworld.c Initializing an MPI program, MPI_Send, MPI_Recv. Piazza forum signup. Pacheco 3.1