High Performance Computing Modernization Program (HPCMP) Summer 2011 Puerto Rico Workshop on
Intermediate Parallel Programming & Cluster Computing
in conjunction with the National Computational Science Institute (NCSI)/ SC11 Conference

Jointly hosted at Polytechnic U of Puerto Rico and U Oklahoma

and available live via videoconferencing (streaming video recordings coming soon)

Sponsored by DOD HPCMP, SC11/ACM, NCSI and OK EPSCoR

Intermediate Parallel Programming & Cluster Computing

Hybrid CUDA/MPI

Josh Alexander, University of Oklahoma
Ivan Babic, Earlham College
Ken Gamradt, South Dakota State University
Andrew Fitz Gibbon, Amazon.com
Mobeen Ludin, Earlham College
Tom Murphy, Contra Costa College
Henry Neeman, University of Oklahoma
Charlie Peck, Earlham College
Stephen Providence, Hampton University
Jeff Rufinus, Widener University
Luis Vicente, Polytechnic University of Puerto Rico
Aaron Weeden, Earlham College

Sunday July 31 – Saturday August 6 2011

Sponsored by
DOD HPCMP,
SC11/ACM,
NCSI and
OK EPSCoR

Goal: Effectively use hardware

- Algorithm should lay nicely on the hardware.
- CUDA deal killers
  - Recursion
  - Dependence between threads
  - High data motion / computation ratio
- CUDA/MPI Division of labor
  - MPI moves data to/from nodes with CUDA devices
  - CUDA is used for the compute it does best
  - Design implementation around compute, hence CUDA

Development Strategy

- Create Serial , CUDA versions of code, then add MPI
- Move data to/from CPU memory to CUDA Global memory
- Core computation in threads of a block
  - Using Shared memory
  - Synchronization of threads is possible
- Blocks execute independently
- CUDA orchestra can have only one conductor
  - If multiple MPI processes on processor
    then only one can successfully drive a CUDA device YMMV
  - MPI_Get_processor_name helps debug
    (remember Charlie and Aaron's talk)

Mojo allowing Hybrid

- Both CUDA and MPI compilation necessary
  - So we do both-> CUDA first then MPI
  - nvcc -arch sm_13 --compiler-bindir mpicc driver.c kernel.cu
  - driver.c does serial and MPI parts
  - Kernel.cu does CUDA care and feeding

CUDA/MPI on sooner

- You have example makefiles and bsub files for each
- Key things to navigate
  - Job queue is "cuda"
  - CUDA cards can be reserved by using:
    #BSUB –R "select[cuda > 0]"
    #BSUB –R "rusage[cuda=2]"
  - Ken's CUDA code reveals two CUDA devices per node, so no more than two MPI processes per node
- My code is in progress
  - Bsub files need to be moved to standard form
  - Makefile designed to show all mojo in one place
  - tmurphy/NCSIPARII2011_exercises/PI_Hybrid

Possible Lab Assignments

- My code is in progress
  - Bsub files need to be moved to standard form
  - Makefile designed to show all mojo in one place
  - tmurphy/NCSIPARII2011_exercises/PI_Hybrid
  - pi_cuda.cu
    - Doesn't produce correct results
    - Needs to alter and be altered by other pi codes for as much similarity as possible
- Assignments
  - Get pi_cuda.cu working
  - Design and code pi_cudampi.cu and pi_cudampi.c

Coming to a Classroom near You

- I have 50 line area under a semicircle code to calculate pi
- Goal is Rosetta stone of simple codes
  - To see similarities and differences
  - To speed learning since algorithm very familial
  - Error management implicit as number of segments rises
- Have serial and MPI versions
- CUDA might be in place even as we speak
- CUDA/MPI is your (and my) lab assignment
- CUDA/OpenMP/MPI is my dream
  since it will lay down nice on LittleFe

The Code (first comment block)

```
/*  calculating pi via area under the curve
 *  This code uses an algorithm fairly easily ported to all
    parallel methods.
 *  Since it calculates pi, it is easy to verify that results are
    correct.
 *  It can also be used to explore accuracy of results and
    techniques for managing error.
*/
```

The Code (second comment block)

```
/*  students learn in grammar school that the area of a circle is
    pi*radius*radius.
 *  They learn in high school that the formula of a circle is x^2
    + y^2 = radius^2.
 *  Using these facts allows students to calculate pi by
    estimating area by constructing trapezoids
 *  Area of unit circle is pi, y = sqrt(1-x^2) is formula for
    semicircle from -1 to 1
 *  Because of symmetry we only need to consider the area
    under the curve from 0 to 1
 */
```

The Code (setup)

```
numSeg = atoi(argv[1]);
    /* get number of segments from command line */

segWidth = 1.0 / numSeg;
    /* calculate width of each segment*/
areas = (double *) malloc(numSeg*sizeof(double));
    /* allocate dynamic array to hold areas of trapezoids*/
```

The Code (heavy lifting)

```
/* calculate area of trapezoid for each segment*/
for (i=0; i<numSeg; ++i) {
        new_x  = (1.0 + i) / numSeg;
        new_y  = sqrt(1.0 - new_x * new_x);
        areas[i] = segWidth * 0.5 * (old_y + new_y);
        old_y   = new_y;
  }
/* calculate pi/4, with room for better error mgmt     */
for (i=0; i<numSeg; ++i) quarterPI += areas[i];
printf ("pi = %15.10f\n", 4.0 *quarterPI);
```

Thanks for your attention!

Questions?