L14: Design Review, Projects, Performance Cliffs and Optimization Benefits

Administrative

- Class cancelled, Wednesday, April 1 (no fooling!)
- Makeup class Friday, April 3 during normal class time
- Seminar today immediately following class:
  - "A Healthy Skepticism about the Future of Multi-Core"
  - LCR, 12:15PM
- Bill Dally (Chief Scientist, NVIDIA and Stanford)
  - Monday, April 6, 11-12, WEB 3760
  - "Stream Programming: Parallel Processing Made Simple"
- Design Reviews, starting April 8 and 10
- Final Reports on projects
  - Poster session the week of April 27 with dry run the previous week
  - Also, submit written document and software
  - Invite your friends! I'll invite faculty, NVIDIA, graduate students, application owners, ..

Design Reviews

- Goal is to see a solid plan for each project and make sure projects are on track
  - Plan to evolve project so that results guaranteed
  - Show at least one thing is working
  - How work is being divided among team members
- Major suggestions from proposals
  - Project complexity – break it down into smaller chunks with evolutionary strategy
  - Add references – what has been done before? Known algorithm? GPU implementation?
  - In some cases, claim no communication but it seems needed to me

Design Reviews

- Oral, 10-minute Q&A session
  - Each team member presents one part
  - Team should identify "lead" to present plan
- Three major parts:
  - I. Overview
  - Define computation and high-level mapping to GPU
  - II. Project Plan
  - The pieces and who is doing what.
  - What is done so far? (Make sure something is working by the design review)
  - III. Related Work
  - Prior sequential or parallel algorithms/implementations
  - Prior GPU implementations (or similar computations)
- Submit slides and written document revising proposal that covers these and cleans up anything missing from proposal.

Publishing your projects?

- I would like to see a few projects from this class be published, perhaps in workshops
  - I am willing to help with writing and positioning
- Publishing the work may require additional effort beyond course requirements or timetable of semester
  - So not appropriate for everyone, and certainly not part of your grade in course
- Let's look at some examples (also consider for related work)

Places to look for examples

- NVIDIA CUDA Zone
  - Huge list of research projects using CUDA with speedups ranging from 1.3x to 420x
  - Many of your projects are related to projects listed there
  - http://www.nvidia.com/cuda
- GPGPU
  - http://www.gpgpu.org
  - Links to workshops, research groups, and news from industry
- Some recent workshops
  - SIAM CSE'09: Scientific Computing on Emerging Many-Core Architectures, http://people.maths.ox.ac.uk/~gilesm/SIAM_CSE/index.html
  - WORKSHOP on GPU Supercomputing 2009, National Taiwan University, http://cqse.ntu.edu.tw/cqse/gpu2009.html
  - Workshop on General-Purpose Computation on Graphics Processing Units, http://www.ece.neu.edu/groups/nucar/GPGPU/

Places to look for examples, cont.

- Upcoming calls
  - PPAM (Parallel Processing and Applied Mathematics): due 4/10, also in Poland…
  - Symposium on Application Accelerators in High Performance Computing (SAAHPC'09), http://www.saahpc.org/, 2-3 page abstracts due 4/20
  - Probably, some new calls over the summer
  - Also, application workshops and conferences

Homework Assignment #3

Problem 2: Select one of the following questions below. Write a CUDA program that illustrates the "optimization benefit" (OB) or "performance cliff" (PC) in the example. These codes will be shared with the rest of the class. Also provide a brief (a few sentences) description of what is happening as a comment inside the code.

a. [PC] Show an example code where you fill up the register file due to too many threads. You should have two versions of the code, one where the number of threads is within the range of registers, and one where the register capacity is exceeded.

b. [OB] Show the performance impact of unrolling an innermost loop in a nest. See how far you can push it before you run into the problems of a. above.

c. [OB/PC] Explore when the compiler decides to put array variables that are local to the device function in registers. What access patterns lead to the compiler using a register vs. using local memory.

d. [OB/PC] Show the performance advantage of constant memory when the data is cached, and what happens to performance when the data exceeds the cache capacity and locality is not realized.

Homework Assignment #3

Problem 2, cont.:

e. [OB] Show the performance impact of control flow versus no control flow. For example, use the trick from slide #13 of Lecture 9 and compare against testing for divide by 0.

f. [PC] Demonstrate the performance impact of parallel memory access (no bank conflicts) in shared memory. For example, implement a reduction computation like in Lecture 9 in shared memory, with one version demonstrating bank conflicts and the other without.

g. [OB] Show the performance impact of global memory coalescing by experimenting with different data and computation partitions in the matrix addition example from lab1.

General

- Timing accuracy
  - Event vs. timer
  - Duration of run as compared to timer granularity
  - What is standard deviation?
- Consider other overheads that may mask the thing you are measuring
  - For example, global memory access versus control flow
- Errors encountered
  - Erroneous results if max number of threads exceeded (512), but apparently no warning…

a. Exceeding register capacity

- Compile fails if code exceeds number of available registers. (supposed to spill to "local" memory?)
- Simple array assignment with slightly more variables
  - Compare 7680 registers vs. 8192 registers
  - 1.5x performance difference!

b. Impact of Loop Unrolling

- Unroll inner loop of mmul_cu from the tiled code
- Compute 16 elements with fully unrolled loop
- Performance difference negligible
  - EITHER, too much unrolling so performance harmed
  - OR, timing problem

d. Constant cache

```
// d_b in constant memory and small enough to fit in cache
__global__ void cache_compute(float *a) :
  for(int j=0; j<100000; j++) a[(j+threadIdx.x) % n] += d_b[(j+threadIdx.x) % n];
// d_b2 in constant memory
__global__ void bad_cache_compute(float *a):
  for(int j=0; j<100000; j++) a[(j+threadIdx.x) % BadCacheSize] += d_b2[(j +threadIdx.x) % BadCacheSize];

// b in global memory
__global__ void no_cache_compute(float *a, float *b) :
  for(int j=0; j<100000; j++) a[(j+threadIdx.x) % n] += b[(j+threadIdx.x) % n];
```

1.2x and 1.4x performance improvements, respectively, when input fits in cache vs. not as compared to global memory.

Similar example showed 1.5X improvement.

e. Control flow versus no control flow

```
float val2 = arr[index];
// has control flow to check for divide by zero
if(val1 != 0)
  arr[index] = val1/val2;
else
  arr[index] = 0.0;
```

```
float val2 = arr[index];
// approximation to avoid to control flow

val1 += 0.000000000000001;
arr[index] = val1/val2;
```

2.7X performance difference!
(similar examples showed 1.9X and 4X difference!)

Another example,
check for divide by 0 in reciprocal
1.75X performance difference!

e. Control flow vs. no control flow (switch)

```
for(int i=0; i < ARRAYLOOP; i++)
   switch(z)
     case 0:  a_array[threadIdx.x] += 18;
        break;
     case 1:  a_array[threadIdx.x] += 9;
        break;
     …
     case 7: a_array[threadIdx.x] += 15;
        break;
}
```

```
efficientArray[0] = 18;
efficientArray[1] = 9;
   …
efficientArray[7] = 15;
__syncthreads();
        for(int j=0; j < ARRAYLOOP; j++)
                for(int i=0; i < ARRAYLOOP; i++)

        a_array[threadIdx.x] +=
                efficientArray[z];
```

Eliminating the switch statement makes a 6X performance difference!

f. Impact of bank conflicts

```
if ( cause_bank_conflicts ) {
   min = id * num_banks ;
   stride = 1;
   max = (id + 1) * num_banks;
}
else {
   min = id;
   stride = num_banks ;
   max = ( stride * ( num_banks - 1))
     + min + 1;
}
```

```
for (j = min; j < max; j+= stride )
  mem[j] = 0;
for (i = 0; i < iters ; i++)
  for (j = min; j < max; j += stride )
    mem[j]++;
for (j = min; j < max; j+= stride )
  out[j] = mem[j];
```

5X difference in performance!
Another example showed 11.3X difference!

g. Global memory coalescing

- Experiment with different computation and data partitions for matrix addition code
- Column major and row major, with different data types
- Row major?
- Column major results
  - Exec time for
    - Double 77 ms
    - Float 76ms
    - Int 57 ms
    - Char 31 ms

Coming soon

- Reminder
  - Class cancelled on Wednesday, April 1
  - Makeup class on Friday, April 3