L9: Next Assignment, Project and Floating Point Issues

Administrative Issues

- CLASS CANCELLED ON WEDNESDAY!
  – I'll be at SIAM Parallel Processing Symposium
- Next assignment, triangular solve
  – Due 5PM, Friday, March 5
  – handin cs6963 lab 3 <probfile>"
- Project proposals (discussed today)
  – Due 5PM, Wednesday, March 17 (hard deadline)

Outline

- Triangular solve assignment
- Project
  – Ideas on how to approach
  – Construct list of questions
- Floating point
  – Mostly single precision
  – Accuracy
  – What's fast and what's not
  – Reading:

  Ch 6 in Kirk and Hwu, http://courses.ece.illinois.edu/ece498/al/textbook/Chapter6-FloatingPoint.pdf

  NVIDA CUDA Programmer's Guide, Appendix B

Triangular Solve (STRSM)

```
for (j = 0; j < n; j++)
  for (k = 0; k < n; k++)
     if (B[j*n+k] != 0.0f) {
          for (i = k+1; i < n; i++)
            B[j*n+i] -= A[k * n + i] * B[j * n + k];
     }
```

Equivalent to:

```
cublasStrsm('l' /* left operator */, 'l' /* lower triangular */,
            'N' /* not transposed */, 'u' /* unit triangular */,
             N, N, alpha, d_A, N, d_B, N);
```

See: http://www.netlib.org/blas/strsm.f

Assignment

- Details:
  - Integrated with simpleCUBLAS test in SDK
  - Reference sequential version provided
1. Rewrite in CUDA
2. Compare performance with CUBLAS 2.0 library

Performance Issues?

- + Abundant data reuse
- - Difficult edge cases
- - Different amounts of work for different <j,k> values
- - Complex mapping or load imbalance

Reminder: Outcomes from Last Year's Course

- Paper and poster at Symposium on Application Accelerators for High-Performance Computing http://saahpc.ncsa.illinois.edu/09/ (May 4, 2010 submission deadline)
  - Poster: Assembling Large Mosaics of Electron Microscope Images using GPU - Kannan Venkataraju, Mark Kim, Dan Gerszewski, James R. Anderson, and Mary Hall
  - Paper: GPU Acceleration of the Generalized Interpolation Material Point Method Wei-Fan Chiang, Michael DeLisi, Todd Hummel, Tyler Prete, Kevin Tew, Mary Hall, Phil Wallstedt, and James Guilkey
- Poster at NVIDIA Research Summit http://www.nvidia.com/object/gpu_tech_conf_research_summit.html
  Poster #47 - Fu, Zhisong, *University of Utah (United States)*
  Solving Eikonal Equations on Triangulated Surface Mesh with CUDA
- Posters at Industrial Advisory Board meeting
- Integrated into Masters theses and PhD dissertations
- Jobs and internships

Projects

- 2-3 person teams
- Select project, or I will guide you
  - From your research
  - From previous classes
  - Suggested ideas from faculty, Nvidia (ask me)
- Example (published):
  - http://saahpc.ncsa.illinois.edu/09/papers/Chiang_paper.pdf (see prev slide)
- Steps
  1. Proposal (due Wednesday, March 17)
  2. Design Review (in class, April 5 and 7)
  3. Poster Presentation (last week of classes)
  4. Final Report (due before finals)

1. Project Proposal (due 3/17)

- Proposal Logistics:
  - Significant implementation, worth 55% of grade
  - Each person turns in the proposal (should be same as other team members)
- Proposal:
  - 3-4 page document (11pt, single-spaced)
  - Submit with handin program:
    "handin cs6963 prop <pdf-file>"

Content of Proposal

I. Team members: Name and a sentence on expertise for each member

II. Problem description
- What is the computation and why is it important?
- Abstraction of computation: equations, graphic or pseudo-code, no more than 1 page

III. Suitability for GPU acceleration
- Amdahl's Law: describe the inherent parallelism. Argue that it is close to 100% of computation. Use measurements from CPU execution of computation if possible.
- Synchronization and Communication: Discuss what data structures may need to be protected by synchronization, or communication through host.
- Copy Overhead: Discuss the data footprint and anticipated cost of copying to/from host memory.

IV. Intellectual Challenges
- Generally, what makes this computation worthy of a project?
- Point to any difficulties you anticipate at present in achieving high speedup

Content of Proposal, cont.

I. Team members: Name and a sentence on expertise for each member

Obvious

II. Problem description
- What is the computation and why is it important?
- Abstraction of computation: equations, graphic or pseudo-code, no more than 1 page

Straightforward adaptation from sequential algorithm and/or code

III. Suitability for GPU acceleration
- Amdahl's Law: describe the inherent parallelism. Argue that it is close to 100% of computation. Use measurements from CPU execution of computation if possible

Can measure sequential code

Content of Proposal, cont.

III. Suitability for GPU acceleration, cont.
- Synchronization and Communication: Discuss what data structures may need to be protected by synchronization, or communication through host.

Avoid global synchronization

- Copy Overhead: Discuss the data footprint and anticipated cost of copying to/from host memory.

Measure input and output data size to discover data footprint. Consider ways to combine computations to reduce copying overhead.

IV. Intellectual Challenges
- Generally, what makes this computation worthy of a project?

Importance of computation, and challenges in partitioning computation, dealing with scope, managing copying overhead

- Point to any difficulties you anticipate at present in achieving high speedup

Projects – How to Approach

- Some questions:
  1. Amdahl's Law: target bulk of computation
     and can profile to obtain key computations…
  2. Strategy for gradually adding GPU execution to CPU code while maintaining correctness
  3. How to partition data & computation to avoid synchronization?
  4. What types of floating point operations and accuracy requirements?
  5. How to manage copy overhead?

1. Amdahl's Law

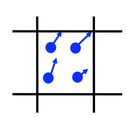
- Significant fraction of overall computation?
  - Simple test:
    - Time execution of computation to be executed on GPU in sequential program.
    - What is its percentage of program's total execution time?
- Where is sequential code spending most of its time?
  - Use profiling (gprof, pixie, VTUNE, …)

2. Strategy for Gradual GPU…

- Looking at MPM/GIMP from last year
  - Several core functions used repeatedly (integrate, interpolate, gradient, divergence)
  - Can we parallelize these individually as a first step?
  - Consider computations and data structures

3. Synchronization in MPM

Blue dots corresponding to particles (pu).
Grid structure corresponds to nodes (gu).



How to parallelize without incurring synchronization overhead?

4. Floating Point

- Most scientific apps are double precision codes!
- In general
  - Double precision needed for convergence on fine meshes
  - Single precision ok for coarse meshes
- Conclusion:
  - Converting to single precision (float) ok for this assignment, but hybrid single/double more desirable in the future

5. Copy overhead?

- Some example code in MPM/GIMP

```
sh.integrate (pch,pch.pm,pch.gm);
sh.integrate (pch,pch.pfe,pch.gfe);
sh.divergence(pch,pch.pVS,pch.gfi);
for(int i=0;i<pch.Nnode();++i)pch.gm[i]+=machTol;
for(int i=0;i<pch.Nnode();++i)pch.ga[i]=(pch.gfe[i]+pch.gfi[i])/
    pch.gm[i];
…
```

Exploit reuse of gm, gfe, gfi
Defer copy back to host.

Other Project Questions

- Want to use Tesla System?
- 32 Tesla S1070 boxes
  - Each with 4 GPUs
  - 16GB memory
  - 120 SMs, or 960 cores!
- Communication across GPUs?
  - MPI between hosts

Brief Discussion of Floating Point

- To understand the fundamentals of floating-point representation (IEEE-754)
- GeForce 8800 CUDA Floating-point speed, accuracy and precision
  - Deviations from IEEE-754
  - Accuracy of device runtime functions
  - -fastmath compiler option
  - Future performance considerations

GPU Floating Point Features

	G80	SSE	IBM Altivec	Cell SPE
Precision	IEEE 754	IEEE 754	IEEE 754	IEEE 754
Rounding modes for FADD and FMUL	Round to nearest and round to zero	All 4 IEEE, round to nearest, zero, inf, -inf	Round to nearest only	Round to zero/ truncate only
Denormal handling	Flush to zero	Supported, 1000's of cycles	Supported, 1000's of cycles	Flush to zero
NaN support	Yes	Yes	Yes	No
Overflow and Infinity support	Yes, only clamps to max norm	Yes	Yes	No, infinity
Flags	No	Yes	Yes	Some
Square root	Software only	Hardware	Software only	Software only
Division	Software only	Hardware	Software only	Software only
Reciprocal estimate accuracy	24 bit	12 bit	12 bit	12 bit
Reciprocal sqrt estimate accuracy	23 bit	12 bit	12 bit	12 bit
log2(x) and 2^x estimates accuracy	23 bit	No	12 bit	No

What is IEEE floating-point format?

- A floating point binary number consists of three parts:
  - sign (S), exponent (E), and mantissa (M).
  - Each (S, E, M) pattern uniquely identifies a floating point number.
- For each bit pattern, its IEEE floating-point value is derived as:
  - value = $(-1)^S * M * \{2^E\}$, where $1.0 \leq M < 10.0_B$
- The interpretation of S is simple: S=0 results in a positive number and S=1 a negative number.

Single Precision vs. Double Precision

- Platforms of compute capability 1.2 and below only support single precision floating point
- New systems (GTX, 200 series, Tesla) include double precision, but much slower than single precision
  - A single dp arithmetic unit shared by all SPs in an SM
  - Similarly, a single fused multiply-add unit
- Suggested strategy:
  - Maximize single precision, use double precision only where needed

Summary: Accuracy vs. Performance

- A few operators are IEEE 754-compliant
  - Addition and Multiplication
- … but some give up precision, presumably in favor of speed or hardware simplicity
  - Particularly, division
- Many built in intrinsics perform common complex operations very fast
- Some intrinsics have multiple implementations, to trade off speed and accuracy
  - e.g., intrinsic __sin() (fast but imprecise) versus sin() (much slower)

Deviations from IEEE-754

- Addition and Multiplication are IEEE 754 compliant
  - Maximum 0.5 ulp (units in the least place) error
- However, often combined into multiply-add (FMAD)
  - Intermediate result is truncated
- Division is non-compliant (2 ulp)
- Not all rounding modes are supported
- Denormalized numbers are not supported
- No mechanism to detect floating-point exceptions

Arithmetic Instruction Throughput

- int and float add, shift, min, max and float mul, mad: 4 cycles per warp
  - int multiply (*) is by default 32-bit
    - requires multiple cycles / warp
  - Use __mul24() / __umul24() intrinsics for 4-cycle 24-bit int multiply
- Integer divide and modulo are expensive
  - Compiler will convert literal power-of-2 divides to shifts
  - Be explicit in cases where compiler can't tell that divisor is a power of 2!
  - Useful trick: foo % n == foo & (n-1) if n is a power of 2

Arithmetic Instruction Throughput

- Reciprocal, reciprocal square root, sin/cos, log, exp: 16 cycles per warp
  - These are the versions prefixed with "__"
  - Examples:__rcp(), __sin(), __exp()
- Other functions are combinations of the above
  - y / x == rcp(x) * y == 20 cycles per warp
  - sqrt(x) == rcp(rsqrt(x)) == 32 cycles per warp

Runtime Math Library

- There are two types of runtime math operations
  - __func(): direct mapping to hardware ISA
    - Fast but low accuracy (see prog. guide for details)
    - Examples: __sin(x), __exp(x), __pow(x,y)
  - func() : compile to multiple instructions
    - Slower but higher accuracy (5 ulp, units in the least place, or less)
    - Examples: sin(x), exp(x), pow(x,y)
- The -use_fast_math compiler option forces every func() to compile to __func()

Make your program float-safe!

- Future hardware will have double precision support
  - G80 is single-precision only
  - Double precision will have additional performance cost
  - Careless use of double or undeclared types may run more slowly on G80+
- Important to be float-safe (be explicit whenever you want single precision) to avoid using double precision where it is not needed
  - Add 'f' specifier on float literals:
    - `foo = bar * 0.123;    // double assumed`
    - `foo = bar * 0.123f;   // float explicit`
  - Use float version of standard library functions
    - `foo = sin(bar);    // double assumed`
    - `foo = sinf(bar);   // single precision explicit`

Next Class

- Reminder: class is cancelled on Wednesday, Feb. 24
- Next class is Monday, March 1
  - Discuss CUBLAS 2 implementation of matrix multiply and sample projects
- Remainder of the semester:
  - Focus on applications
  - Advanced topics (CUDA->OpenGL, overlapping computation/communication, Open CL, Other GPU architectures)