Using GPUs as CPUs for Engineering Applications: Challenges and Issues

Michael A. Heroux
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Outline of the Talk

1. A Brief Discussion of Some Sandia Applications.
2. Some Important Application Kernels.
3. Use of 32-bit arithmetic in Engineering Apps.
4. Simulating Double with Single.
5. How I hope to leverage GPUs.
6. Final Thoughts and Summary.

Sandia Applications

- Sandia is primarily an Engineering Lab:
  - Structures, Fluids, Electrical…
  - and Contributing Physics.
  - Coupling of Physics is primary focus of computing efforts.
- Some Apps use Explicit Methods:
  - Pronto: 3D Transient Solid Dynamics Code.
  - CTH:3D Eulerian Shock Code.
- Strong focus on Implicit codes:
  - SALINA (Structures)
  - ALEGRA (ALE Shock code)
  - SIERRA (Fluids and Interactions)
  - Xyce (Electrical)
  - …and more.
- I will focus mostly on the implicit codes.
- One basic assumption:
  - GPU will be viewed as a co-processor.
  - I will not emphasize related issues, but are critically important.

Common Implicit Code Characteristics

- Unstructured:
  - Grids are used, stitched together, but
  - Global irregularity is significant.
  - Most data structures assume arbitrary connectivity.
  - Most tools implicitly assume significant regularity.
- Finite Element/Volume/Difference:
  - Most apps discretize continuum via these methods.
  - We will discuss this issue later (as opportunity for 32-bit use).
  - Xyce circuit modeling is exception: Inherently discrete.
- Models are 1-3D, Nonlinear, both Steady/Transient.
- Solvers are a critical technology component:
  - Often considered "3rd-party".
  - Typically consume 50% or (much) more of run-time.
  - Direct (sparse) solvers often used.
  - Preconditioned iterative methods often and increasingly used.
  - The core is a sparse linear solver.

Problem Definition

- A frequent requirement for scientific and engineering computing is to solve:

$$Ax = b$$

where A is a known large (sparse) matrix,
b is a known vector,
x is an unknown vector.

- Goal: Find x.
- Methods: Preconditioned Krylov methods.
- The Conjugate Gradient (CG) method is simplest of these.
- CG is only appropriate for symmetric (Hermitian).
- Still serves as reasonable prototype for initial study.
- With some exceptions we will note.

Other Types of Solver Problems

- Nonlinear problems: $f(u) = 0$:
  - $u(x)u(x)' - \sin(x)\cos(x) = 0.$
- Eigenvalue problems: $Ax = \lambda x.$

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & -2 & 2 \\ 2 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- Many variations.
- Sparse matrix multiplication: Basic op for all above.
- Linear solver often basic component for all.
- Iterative linear solvers important on parallel machines.

- **Bottom line:**
  Study of Sparse Iterative solver makes sense.

Sandia National Laboratories

Iterative Methods

- Given an initial guess for x, called x^0, ($x^0 = 0$ is acceptable) compute a sequence x^i, $i = 1,2, \ldots$ such that each x^i is "closer" to x.
- Definition of "close":
  - Suppose $x^i = x$ exactly for some value of i.
  - Then $r^i = b - Ax^i = 0$ (the vector of all zeros).
  - And $norm(r^i) = sqrt(ddot(r^i, r^i)) = 0$ (a number).
  - For any x^i, let $r^i = b - Ax^i$
  - If $norm(r^i) = sqrt(ddot(r^i, r^i))$ is small (< 1.0E-6 say) then we say that x^i is close to x.
  - The vector r is called the residual vector.

The CG Method

$i = 0$; $x^{i-1} = 0$; $r^{i-1} = b$; A given by user;
while $norm(r^i) > tol$ {
    i ++;
    $rtr^{i-1} = ddot(r^{i-1}, r^{i-1})$;
    if (i=1) $p^i = r^{i-1}$;
    else {
        $b^i = rtr^{i-1}/rtr^{i-2}$;
        $p^i = r^{i-1} + b^i * p^{i-1}$;
    }
    $Ap^i = sparsemv(A,p^i)$;
    $a^i = rtr^{i-1} / ddot(p^i,Ap^i)$;
    $x^{i-1} = x^i$; $x^i = x^{i-1} + a^i * p^i$;
    $r^{i-1} = r^i$; $r^i = r^{i-1} - a^i * Ap^i$;
}
$x = x^i$; // When $norm(r^i) <= tol$, stop and set x to x^i

Three primary kernels:

- **Dot product (reduction).**
- **Vector update.**
- **Sparse matrix-vector multiply.**

Sandia National Laboratories

This might look familiar…

- Paper: Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid (J. Bolz, I. Farmer, E. Grinspun, P. Schröder), July 2003 ACM TOG, 22:3.
- Bolz, et. al. describe efficient implementations of all three kernels:
  - Vector updates: Trivial, very fast.
  - Dot Products: Use 2D layout, recursive 2-by-2 to 1 reduction.
  - Matrix-vector multiply:
    - Compressed-row-by-compressed-row.
    - Rows ordered by decreasing density.
    - Diagonal handled separately.
    - Fragment program handles a row.
    - Limitations on row density (up to 200):
      - Not a major practical limitation but annoying for bullet-proof implementations.

CG Performance

Kernel\Processor	GeForce FX 500MHz	Pentium-M 1.6GHz (This Laptop)
Dot Product	172 MFLOPS	159 MFLOPS
Vector Update	718 (Implied)	68 MFLOPS
Sparse MV	62 MFLOPS	116 MFLOPS
Total CG Performance	98 MFLOPS	109 MFLOPS

- GeForce FX results computed from Bolz, et. al. (Single precision)
- Pentium-M results from Cygwin/GCC 3.3.3 (Double precision).
- Encouraging results! (I think).
- From Pat Hanrahan's talk:
  - ATI Radeon 9800XT 1.5x faster than GeForce FX for vector update.
  - X800 2x faster than 9800XT
  - NV40?

CG Solver Performance Observations

- GPU results appear to be "generalizable" …
- But are also "Wind at our back" results:
  - Problem size, layout tightly constrained.
  - How do we write general-purpose code that works for all sizes?
  - Seems like writing assembler.
- Choice of details avoid recursive preconditioners.
  - Bolz, et. al. also discuss Multigrid, but use Jacobi smoother.
  - No clear path to implementing recursive preconditioners: Gauss-Seidel, ILU, etc.
- Memory size (up to 256MB) allows healthy problem size:
  - Unpreconditioned CG requires 72n words storage.
    - 4n words – vectors,
    - 7n words – matrix values, 7n words – matrix indices/pntrs
  - Max problem dimension: 3.5M equations.
  - However, CG is simplest algorithm. ILU-GMRES more common, much more involved, much more memory.
- Then there's the issue of FP precision…

Use of Single Precision: Background

- 20-30 years ago, single precision was commonly used in scientific/engineering apps.
- Single precision persists in pockets:
  - LS-DYNA still has SP capability, accumulates into DP.
  - SP FFTs are still very common, e.g., seismic calculations.
- Most other apps have migrated to DP:
  - Literature is fairly silent about which apps phases need DP.
  - Lots of anecdotal information. Tough problems really need DP or higher.
  - General attitude has been "use the highest precision that has reasonable cost."
  - Going back to SP would be difficult.
- Mixed precision has been and is being used:
  - Construct preconditioner in DP, store and apply in SP.
  - Course grid solves (smaller condition number).
  - These approaches rely on ability to have SP, DP co-mingled.

Double Precision is Required

- In my opinion, going back to SP is not attractive.
- DP has allowed us to advance modeling capabilities.
  - We do not want to take a step back.
  - In fact, we want more precision in many cases.
- Solvers need DP (and higher) regularly:
  - Ill-conditioned problems need the mantissa bits to avoid (or at least delay) severe cancellation.
  - SP exponent bits are too few (DP are more than needed) to handle range of values for many problems.
- It seems like native DP on GPUs is not near-term.
- So how about simulated DP?

Simulated DP

- Two approaches:
  - Single-single.
  - True simulated double.
- Single-single:
  - Uses two SP numbers to store the upper/lower mantissa bits.
  - Exponent is not split: Same exponent range as SP.
- True simulated double:
  - Double the size of SP.
  - Has larger exponent range.

Lessons from Simulated Double-Double, Quad

- Software techniques are used frequently to provide double-double and quad precision on modern CPUs.
- Number of packages to facilitate use.
- Some lessons from simulated double-double on CPUs:
  - Portable simulated double-double is about an order of magnitude slower than double.
    - Add takes 19 DP ops, Mult takes 25+ DP ops.
    - Temporal locality keeps cost down.
  - A Fused Mult-add (FMAD) HW instruction can cut this in half.
- True simulated quad:
  - Is significantly more costly, especially if FMAD available.
  - Has better round off properties, larger exponent.

• Reference: Design, Implementation and Testing of Extended and Mixed Precision BLAS, X. Li, J. Demmel, D. Bailey, G. Henry, et. al., ACM TOMS, 28:2, 2002.

True Simulated Double is Needed

- Some articles suggest adding FMAD to GPU.
- My concern:
  - Range of single is 1E+/-38.
  - We often see numbers near the limit of this range in our computations.
- Assertion: True Simulated Double needed:
  - More bits are needed for exponent.
  - But we don't need as many as IEEE DP:
    - 1E+/-308 is overkill.

GPUs for other parts of Eng Apps

- Many FEM/FVM applications "strip mine" the loading of local element stiffness matrices into *working sets.*
- This approach seems a natural fit for GPUs.
- Difficulty: At the end of load phase, nodal values must be scattered into global sparse matrix.
- It appears that scatter ops are hard to perform on GPUs.
- If so, there are two approaches to address scatter problem:
  - Add scatter to GPUs.
  - Reorganize apps to be node-oriented (instead of element oriented). <= This will never happen.

Other Issues

- In papers I have read, nobody reports time to load/unload graphics memory. Why not?
- Has anyone considered a segmented sum algorithm for sparse matrix-vector multiplication?
  - Reference: Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors, G. Blelloch, M. Heroux, M. Zagha, CMU Tech report, CMU-CS-93-173, 1993.
- To those who are considering "novel" algorithms for GPUs:
  - We have been in similar situations in the past (only 10 years ago).
  - General observation: "The best parallel algorithm is your best parallel implementation of your best serial algorithm."
  - Example: Domain Decomposition methods.

Segmented Sum MV

```
VAL =   .5  .2  .3  .1  .4  .2  .1  .2  .1  .5  .1  .3  .2  .2  .4  .2  .4  .5  .3  .4  .4  .1  .6  .2  .5  .4  .3  .6  .3  .5
FLAG =  T   F   T   F   F   F   T   F   F   F   T   F   F   F   T   F   F   T   F   F   F   F   F   F   F   T   F   T   T   F
        ----- ---------- ---------- ---------- ------- ---------------------- ----- -- -----
```

VAL

0.5	0.1	0.2	0.3	0.5
0.2	0.2	0.2	0.4	0.4
0.3	0.1	0.4	0.4	0.3
0.1	0.5	0.2	0.1	0.6
0.4	0.1	0.4	0.6	0.3
0.2	0.3	0.5	0.2	0.5

FLAG

T	T	F	F	F
F	F	F	F	T
T	F	T	F	F
F	F	F	F	T
F	T	F	F	T
F	F	T	F	F

LAST	3	5	6	***	5
PRESENT	T	T	T	F	T

Sandia National Laboratories

How I hope to Leverage GPUs

- Trilinos Project: Solvers.
  - Linear, Eigen, Nonlinear, Time-dependent, …
  - Most C++ class libraries.
- Significant investment in templated classes:
  - Vector (VectorSpace< OrdinalType, ScalarType > const &VectorSpace)
  - OrdinalType: Indexing (int)
  - ScalarType: Floating point values (double, float, …)
- Next generation of apps will use these templated class libraries.
- Templates allows use of any ADT that has "+", "-", "*" and sometimes "/".
- Hope: Use this templating mechanism to utilize GPU data types.
- One key feature of our abstract model:
  - Ops can migrate to data.
  - Details in **Vector Reduction/Transformation Operators,** R. Bartlett, B. van Bloemen Waanders and M. Heroux,. *ACM Trans. Math. Softw.*, Vol 30, Issue 1, 2004.

ARPREC

- The ARPREC library uses arrays of 64-bit floating-point numbers to represent high-precision floating-point numbers.
- ARPREC values behave just like any other floating-point datatype, except the maximum working precision (in decimal digits) must be specified before any calculations are done
  - `mp::mp_init(200);`
- Illustrate the use of ARPREC with an example using Hilbert matrices.
- Lately also incorporated GMP library.

Hilbert Matrices

- A Hilbert matrix H_N is a square N–by–N matrix such that:

- For Example: $$H_{N_{ij}} = \frac{1}{i + j - 1}$$

$$H_3 = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

Sandia National Laboratories

Hilbert Matrices

- Notoriously ill-conditioned
  - $\kappa(H_3) \approx 524$
  - $\kappa(H_5) \approx 476610$
  - $\kappa(H_{10}) \approx 1.6025 \times 10^{13}$
  - $\kappa(H_{20}) \approx 7.8413 \times 10^{17}$
  - $\kappa(H_{100}) \approx 1.7232 \times 10^{20}$
- Hilbert matrices introduce large amounts of error

Hilbert Matrices and Cholesky Factorization

- With double-precision arithmetic, Cholesky factorization will fail for H_N for all $N > 13$.
- Can we improve on this using arbitrary-precision floating-point numbers?

Precision	Largest N for which Cholesky Factorization is successful
Single Precision	8
Double Precision	13
Arbitrary Precision (20)	29
Arbitrary Precision (40)	40
Arbitrary Precision (200)	145
Arbitrary Precision (400)	233

Summary

- There is low-hanging fruit:
  - Seismic processing almost certainly.
  - Some explicit calculations.
- Proof-of-concept (via CG solver) works for some implicit calculations :
  - Timing results are competitive for SP.
  - Limitations on preconditioning are problematic.
  - Some kind of “compiler” seems necessary to equal generality of CPUs. Brook is a good start.
- Double precision is necessary for broad acceptance of GPUs.
  - I don’t see the simulation community taking a step back (some need to go to 128-bit!).

Summary (cont).

- True double precision is necessary:
  - Single-single is not sufficient (and FMAD is not needed).
  - HW double precision would be great (and GPUs would have the attention of many more people).
  - True Simulated Double is a start, but then performance is set back.
- Scatter capabilities are needed in GPUs in order to broaden impact to physics portion of engineering apps, or maybe I am missing something.
- What about load/unload time between CPU/GPU memories?
- What about segment sum MV?
- I hope to (easily) used GPUs via existing software libraries.

Summary (cont.)

- Please report times for all phases of CPU/GPU use.
- Beware novel parallel algorithms.