* Introduction
  + Finish related works on
    - GPU
    - Distributed GPU
    - GPU RBF
    - Distributed RBF
* Parallelizing RBF-FD solutions
  + Put these in the RBF-FD introduction.
    - Explicit solutions require SpMV, SAXPY for update
    - Implicit solutions require GMRES or another iterative solver; which in turns requires multiple SpMV, SAXPY internally.
    - Reduce as a means to calculate norms and monitor progress of application.
    - Parallelization is achieved at two levels
      * A domain decomposition allows us to distribute the SpMV and SAXPY operations
      * We target the GPU with OpenCL and ViennaCL
  + Multi-CPU/Multi-GPU Implementation is first in the RBF-FD community
    - Related work for RBF methods on GPU is limited to Schmidt
    - Distributed RBF methods limited to Knepley and a few others\*
* Leveraging GPU
  + GPU features
    - Memory layout
    - Multi-Processors
    - Bandwidth on GPU (refer to Bell for significance)
    - Table comparing hardware of M2070, M2090, Phi
    - Trends in hardware since 2006
      * Cheap to purchase, superior performance
      * Trending technology that nearly all supercomputing centers are buying into; predominantly CUDA hardware, until 2012 when Intel released the Phi cards
      * Initially research focused on porting codes and determining the limits of the almost black-box hardware. Today it seems as though the buzz/hype over GPU computing is winding down as more and more research leverages existing code that was previously optimized for the GPU. That is understandable; focus on getting the science done, rather than the computer science. Newcomers are more interested in leveraging the GPUs rather than optimizing for them.
  + MIC is new on scene
    - Describe hardware
    - State that we are just starting investigations but results are not included here.
      * Too soon to tell what benefits
    - MIC has pragmas and OpenCL support (beta 2012/2013).
  + Choice to work with OpenCL and language summary
    - Functional portability vs performance portability
    - Promise for open standard in parallel languages
    - Work-Items, Work-Groups, NDRange, etc.
    - Local memory
    - Asynchronous kernel execution, Queues
    - Memory transfers
    - How does it compare to CUDA? OpenACC?
      * OpenCL attempts to harness the power of all hardware. Supports ATI, Nvidia, etc. Even has support for Cell BE and Phi.
      * No support for direct MPI communication (e.g., CUDA-MPI)
    - Language notation is heavy. Simplification is a must.
      * OpenCL C++ headers are one option, but this does not simplify the task our problem
      * Approach our problem from a higher level of abstraction with a sparse matrix library and primitives like SpMV. Answer question: why use libraries like ViennaCL?
        + Faster development
        + Public project with shared interests
        + Simple API
    - Why avoid the language?
      * Changes are occurring too fast
        + Hardware is great for performance (>1TFLOP) compared to price to purchase (<$1K)
        + But hardware is arriving at an incredible rate.
        + Not to mention adoption is successful in existing codes where optimization was possible on day 1. For codes developing from scratch it is still easier to debug without using the hardware. Most problems that people would put on GPU can be reduced to some BLAS or LAPACK primitive, so it makes sense to use a library like ViennaCL
      * ViennaCL already adopted back-end switch for OpenMP, CUDA and OpenCL to allow more versatility.
        + Whatever vendor or language available libraries like ViennaCL will adapt and our code will change little.
    - MIC limitations in OpenCL
      * Images
      * Device Fission
      * No such limitations in GPU, but language is in beta
  + GPU hardware features
    - Custom kernels with one thread vs one warp
  + SpMV will be bandwidth limited
    - 2:1 operations; or 1:1 if a fused multiply and add is used
  + Custom kernels vs libraries like ViennaCL
    - Simple interface where transfer to/from GPU happens behind the scenes; incl. Kernels
    - Fast prototyping
    - Compatibility with BOOST, Eigen, MKL
      * BOOST in turn has compatibility with
    - SpMV formats
      * Focus on ViennaCL and clSpMV options
        + We can implement test PDEs in ViennaCL, or consider GPU optimizations external to PDEs. clSpMV test weights (read as mtx format) and benchmark for a better view of optimization potential on the GPU.
      * COO, CSR
        + Only enough description to state that COO is the storage format and CSR is most common format in literature. Most results compare on CSR format.
      * Focus on ELL
        + Ideal for RBF-FD given assumption all stencils are size n
      * SELL, BELL, SBELL
        + Differences in kernels
        + What optimizations can we make?

We can test padding to nearest 32

* + - A range of other formats exist, but we do not concern ourselves with them here.
      * Would be appropriate for cases where stencils have variable number of nodes. Our assumption is that we have a uniform number.
    - Performance comparison of ViennaCL formats reveals expected 27x speedup over CPU (boost SpMV, not as optimal as MKL; only optimized for one core).
      * What is the fastest SpMV for RBF-FD?
      * Need: MKL SpMV for comparison
      * Need: table of GFLOPs
    - Other known formats
      * CUSPARSE offers some (limited to CUDA)
      * MKL offers some
        + Block options in clSpMV; clSpMV (OSKI) is also best competitor
      * Performance of Real vs Fake system
        + What is GFLOPs for 1D with n=32? This is the max
        + What is GFLOPs for fake system when stddev from bandwidth is 0.2?
  + Distributed Multi-GPU
    - PETSc, Hypre, Trilinos are all libraries/frameworks that we could have developed in, but none of them had GPU support. We continued with custom built code, but that decision required additional decisions from us.
    - Achieved GFLOPs, GBytes/s
    - Scaling
      * Multiple kernels, one host (1, 2, 4, 8, 16)
        + Fill in with 3, 4, 6, 7, 9, 10, 11, 12, 13, 14, 15
* Explicit and Implicit Solutions
  + Performance of CosineCL vs CosineVCL
  + Implicit
    - Don’t invert large sparse matrices. Use iterative solvers to get the solution easier
    - GMRES algorithm
      * Cite Saad
      * Cite H\*\*\*\* for multi-GPU CUDA GMRES
      * Givens vs Householder implementation
      * ViennaCL had Householder, Givens is easier to parallelize so implemented to include parallel options
      * Introduced ILU0 preconditioner
      * Need: convergence plots and tables for GMRES to demonstrate correctness
        + Regular grid, simple equation; what is VCL test problem for convergence?
    - Stokes problem
      * Equations
      * Discretization
        + Projection operators to get on sphere
      * DM form
      * Manufacturing solution
        + Spherical harmonics on Divergence-free field
      * Node interleaving condenses the weights for better memory load, easier partitioning
      * Need: convergence plots
      * Regular noise in the solution may point to need for Hyperviscosity or stable method for weight calculation
        + Need: RBF-GA test for convergence
  + Conclusion: we have RBF-FD implementations of building blocks necessary for a complex geophysics simulation.
    - Now all we need to do is make sure it is efficient.
* Performance and Optimization
  + Define throughput
    - Measure performance as GFLOPs
    - Compare to CPU (even if limited to 1 core) GFLOPs
    - Need: What are the peaks for each hardware type?
      * Given the achieved and the peak we can normalize the throughput as % peak and consider speedup comparison.
  + Optimizations for CL kernels
    - How custom kernels compare to ViennaCL
    - Recall that operation is limited by bandwidth
      * What is the minimum N and n to justify the GPU?
    - Some hope exists in using clSpMV approach to optimize
* Node Ordering and Preconditioning
  + Consider space filling curves to reorder nodes.
    - Results in different sparsity patterns.
  + Node ordering condenses bandwidth
    - clSpMV heuristics applied to our modified sparsity patterns may prove enlightening.
    - Benchmarks to compare orderings, RCM and SpMV times for each matrix (ELL, SBELL, BELL, SELL)
      * What is the best consideration for performance? Max, Min, Mean or Stddev of bandwidth?
      * Better to have all rows consistent BW or a few very wide and the rest tiny?
      * What is the gain as the BW grows?
    - Compare benchmarks for clSpMV and ViennaCL for different curves and formats.
  + Other preconditioners
    - Most preconditioners are based on node or stencil information
    - Preconditioners:
      * ILU0
      * ILUP
      * AMG
      * MG
    - What is the impact of the preconditioners with and without RBF-GA?
    - What is the best preconditioner for RBF-FD?
    - Does masking blocks when preconditioning help?
* Conclusion
  + First implementation of RBF-FD to span multiple CPUs. Went above and beyond to extend to multiple GPUs.
  + Considered both explicit and implicit PDEs to ensure we have all building blocks necessary to tackle large-scale Geophysical simulations
  + Looked at preconditioning and node orderings to ensure the fastest possible time to accurate solutions
  + Etc.
* Appendices

Benchmarks

* ApplyWeights CL vs VCL (GFLOPs)
* Cosine Bell CL vs VCL (GFLOPs)
* Multi-GPU weak scaling
* Multi-GPU strong scaling
* GMRES 1 GPU vs 1 CPU
* GMRES Multi-GPU vs Multi-CPU

Convergence Studies

* GMRES regular grid
* GMRES Stokes
* ILU GMRES regular grid, stokes (table)
* LSH and RCM GMRES

New content requested by Gordon:

Evidence of clSpMV algorithms for even more throughput

Optimization of SpMV

RCM, LSH preconditioning

Bandwidth analysis

Additional:

Parallel ILU

RBF-GA

Questions:

Does GMRES converge quickly for Poisson on Regular grid?

If not, what preconditioner (ILU, LSH, etc) can we use to accelerate?

What GFLOPs do we get on CPU (UBLAS, VCL, Nested Loop)

Codes:

* GMRES on square test convergence
* Impacts on conditioning from Matrix reorder (X,U,Z)

1. Analysis: Get data in GFLOPs for Keeneland and Spear
   1. Single GPU in appendix a (reference from GPU SpMV chapter)
   2. Multi-GPU in appendix b (reference from Distributed GPU chapter)
2. Benchmark: no overlap and overlap distributed GPU on cascade Fermi
   1. Analysis: data directly into chapter as evidence of overlap scaling
   2. Analysis: data for K20 scaling
3. Benchmark: optimized ViennaCL on Cascade Fermi (n=17,31,50,101)
4. Benchmark: optimized ViennaCL on Cascade K20 (n=17,31,50,101)
   1. Analysis: data comparison of ViennaCL achieved for each hardware.
   2. Analysis: comparison of GFLOPs achieved for Cosine and Vortex (Keeneland and Spear)
5. Distributed GPU
   1. related work
   2. No overlap (Alltoallv)
   3. Overlap with tuned MPI
   4. Overlap K20 vs Fermi
   5. Overlap K20 multi-per card

b) can compute condest of DM with Original ordering and X,U,Z,4-node Z,RCM orders