* RBF-FD is new method
* Scaling RBF research to large problems
  + Global RBF problems have some large problems
    - Cite Knepley for interpolation
    - Other papers
  + Limited work on RBF-FD for large problems
* Leveraging modern architecture
  + Future of HPC is in accelerators
  + RBF research on GPU is limited
    - Cite Schmidt
* Challenges in RBF-FD
  + Conditioning
  + Stencil size
  + Support
* Grand Applications
  + Toward Mantle Convection
  + Need building blocks for applications
* All RBF Literature
  + Global Methods
* Define RBF (Introduction)
* Define RBF interpolation
* Related Methods
  + Global
  + Compact
  + Other
* Define the RBF-FD method
  + History
  + Computing weights
    - Direct solve
    - Stable algorithms that are known
  + Choosing epsilon
    - Contours
    - Related work: what functions did they use?
* Customized Weights
  + Direct compute
  + Linear combinations
  + Projections
* Neighbor queries
* Solving PDEs with RBF-FD
  + Derivative as stencil evaluation
  + All Derivatives as SpMV
  + Assemble system
* Hardware architecture
  + Memory layout
  + Processing cores
  + Trends in hardware since 2006 (additions and benfits)
* Optimization
  + SpMV memory layout
  + Scheduling threads
  + Reductions
* OpenCL
  + Why?
    - Cross platform support
    - Asynchronous Queuing with Dependencies
  + Implementations details
    - Kernel
    - Work-Item
    - Work-Group
    - NDRange
    - Queue
    - Etc.
  + How does it compare to CUDA? Phi
  + Latest trends
    - Phi: bind against MKL for optimized CPU and MIC
    - CUDA-MPI
    - CUDA Sub-Kernel calls
    - CUDA uptake
      * E.g., Matlab (MEX compiled kernel wrappers)
* Conclusions on GPGPU
  + Benefits are good
    - Cheap to purchase < $1K
    - superior performance 1.2 TFLOPs possible in one card
    - was a trending technology (major uptake in supercomputing and national labs)
  + Downsides were varied
  + Overall Impression is that
    - Uptake was wide-spread for research projects
    - Focus was on determining limits of the hardware
      * Many studies focused on optimization of primitives which allow general use in applications such as RBF-FD without recreating the wheel when it comes to optimal algorithms. Allows researchers to concentrate on other investigations into application, preconditioning, data analysis, etc.
* Newcomers to the field are interested USING gpgpu applications, rather than writing them
* Peak throughput
  + CPU
  + GPU
* Achieved throughput
  + CPU
  + GPU
* Scalability
  + Weak
  + Strong
* Time reduction
  + Neighbor
* Conditioning Impacts
  + Interleaving?
  + Node order for GPU?
  + Node order LSH?
* Preconditioners Tested
  + ILU0
  + ILUP
  + MG
  + AMG
  + Etc.
* Mask for Stokes
  + All blocks of non-interleaved
  + Diagonal blocks of non-interleaved
  + Etc.
* Don’t forget
  + Annulus test case
  + 3D sphere shell test cases
  + lid driven cavity
  + cvt description if doing annulus
  + overlap
  + metis
  + MKL
  + Conditioning due to reordering
  + Preconditioners ILU0, ILUP, AMG, etc etc.
  + Distributed Preconditioners
  + GPU kernels:
    - Weights
    - Neighbors
  + clSpMV vs VCL vs MKL (vs Phi)
  + petsc impl?
  + Hypre impl? ß
  + System is not banded!
* Sparse linear systems occur in many many applications
* Non-linear is sparse linear solved many times over
* A long time ago, routines were put together to avoid need to implement ourselves. So we can reuse a library rather than spedn time on it
* Better performance and better numeric for free

See slides for SiPE

* + Don’t reinvent the wheel, unless its unavailable (Distributed GMRES was not avail in VCL; could use another package)ß
  + Don’t invert the matrix
* Note the types of BLAS
* Will have a month on applications:
  + PETsc implementation
    - Does PETsc overlap?
  + METIS decomposition with overlap
  + Lid driven cavity
  + Annulus
  + Nested sphere implicit solver
    - Expand to time evolution of stokes
* Will have a month on preconditioning:
  + Node reordering
    - Z
    - X
    - U
    - Hilbert (?)
    - RCM
    - AMD
  + Impacts on condition
  + Conditioners like ILU, ILUP, block preconditioners, etc.
    - PETsc offers a few that might be useful
* Will have a month to test performance:
  + Compare MKL, VCL and CUSPARSE, PETsc(?) on assembled systems SpMV
  + Include test on phi compare MKL on westmere, sandybridge
  + Test scaling
    - Weak
    - Strong
  + Compare to CUDA-MPI(?)

Additional:

* + Add portable binary: NetCDF and HDF
* Daily:
  + 6-12 pm on code
  + 8-4pm writing
* Iterate existing and polish it
  + Add figures:
    - Large dense matrix, compact sparse support, local dense + sparse DM.
    - Algorithm for time-stepping
  + Remember to add:
    - Clarify difference between continuous derivative and DM
    - Approximate Nearest Neighbors are sufficient, nothing in RBF literature requires true Nearest Neighbors
      * Ball query vs KNN
      * Typical algorithms used in other papers (Bengt has one mentioning KDTREE and Fasshauer’s slides/book mention faster brute)
  + Avoid pole singularities
* Go through references:
  + Use evernote to queue books with tags
* Finish comparison
  + Need to add psuedospectral
* Go back through references
  + Every single reference should be classified
    - Old methods (put in Preliminaries)
    - RBF-FD (put in RBF-FD chapter)
      * Parallel/Performance
      * Application
      * Numerics
    - GPGPU (GPGPU chapter)
      * Multi-GPU
      * SpMV
      * PDEs
      * Hardware/Optimization
    - Parallel PDEs
      * Decomposition
      * SpMV optimizations
      * GMRES
* The RBF-FD chapter needs:
  + Related papers on RBF-FD specifically (i.e., the complete history; follow Flyer Fornberg book).
    - Clearly state what every paper in the RBF-FD group accomplished
  + Weight method
  + Similarly to spectral and pseudospectral (only collocation points allow optimizations) modes, RBF-FD and FD are related and share much of the same approach. Use RBF-FD for general node placement and high order accuracy. Use FD for optimized solution, faster solvers (Fourier decomposition, Band solver, etc).
  + Apply weights to single node (figure from Paper 1) or form a DM
  + Solve PDEs with explicit/implicit form (solvers)
    - Mention GMRES
  + List of weight types
    - Projection operators
  + Choosing epsilon