

Data-driven Codesign

Tom Peterka, ANL

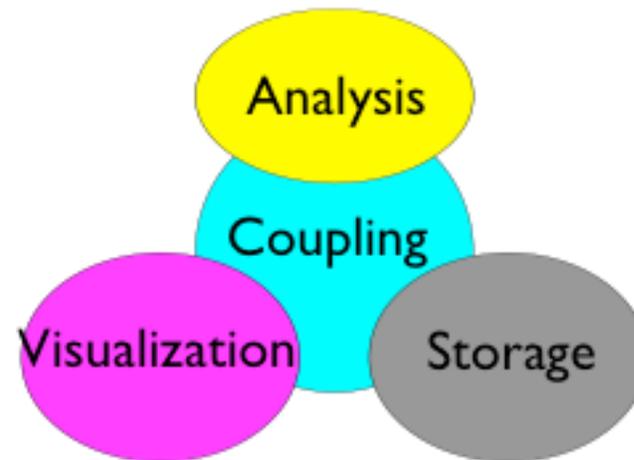
Tim Tautges, ANL

Venkat Vishwanath, ANL

Joe Insley, ANL

Vijay Mahadevan, ANL

Jon Woodring, LANL



Codesign and Data-driven Codesign

- The codesign process: apply the scientific method to the design of future computing hardware and software
 - Formulate driving questions
 - Find and implement canonical problems in the form of proxy apps
 - Design an experiment to using the proxy app to answer the questions
 - Form a hypothesis
 - Identify parameters, input and output
 - Identify instruments used to measure data
 - Run the tests, gather and analyze results
 - Conclude something with respect to original driving questions
- Data-driven codesign
 - Driving questions are related to data-intensive tasks: coupling, analysis, visualization, storage (as opposed to compute-intensive)
 - Proxy apps have data, not necessarily compute (as opposed to compute-intensive)



Driving Questions (i)

- Data-intensive algorithms and data structures
 - **Data models:** How do analysis data models differ from computational data models, and how does data access via a coupled data model compare to native data models?
 - **Data movement:** What is the impact of data movement on the system, and how can we design algorithms to minimize, optimize, or hide data movement? How can data-intensive tasks be located and scheduling ?
 - **Data operations:** What programming models are needed for data processing and (how) do they differ from computational programming models (for example, random access to remote data)? What are appropriate problem decompositions for such problems?
 - **Data access:** How does data movement and placement impact the design of new storage devices and systems, for example active memory cube or NVRAM?



Driving Questions (ii)

- Data-intensive systems and architectures
 - **System level:** What is the recommended configuration of compute, analysis, and storage in the entire leadership facility? Will data analytics clusters continue to exist and how should they be sized w.r.t. HPC machines, or will data analytics be folded into the design of HPC machines (for example, Blue Waters, Titan)?
 - **Component level:** What **compute node** specifications can be derived from data-intensive findings of the previous slide? What **network** parameters (for example bandwidth and latency) will support the needed data movement? What network topologies will provide these capabilities? What **storage hierarchy**, from registers to archives, is needed for data-intensive science?



Data-Intensive Proxy Apps Help Answer Questions

First, we have to rethink what a data-intensive proxy is. Unlike a compute proxy that keeps computation or communication and removes data, we need just the opposite: data products without the computation.

■ Goals

- Acquire representative raw data in the coupled application interface
 - Meshes and solutions are stored and later read back into proxies without rerunning simulation
- Make data available to analysis, visualization, storage
 - Write proxies that test canonical data problems: coupling, analysis, visualization, storage
- Test algorithms and benchmark performance
- Use results to influence co-design process
- Make proxies available to other subgroups (eg. GPGPUs, programming models, performance modeling)

■ Challenges

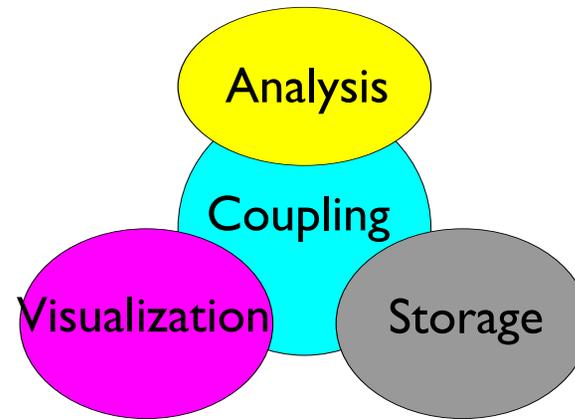
- How well do proxies represent actual problems? In our case, how well do they the represent real data?
 - Real features, eg., vortices, data distributions, outliers, correlations among variables
 - Stored solutions from actual computational runs



Cian

Cian (CESAR Integrated Analytics) is a suite of proxy apps covering our four main research areas:

- Coupling
- Analysis
- Visualization
- Storage

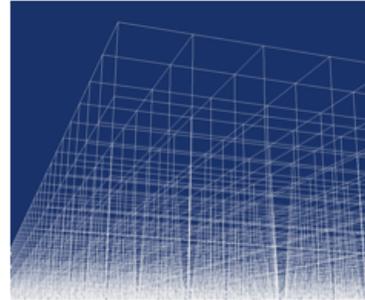


Software consists of four proxy apps for the above components in one package ([cian.tar.gz](#)), plus additional proxies to explore deeper research questions in more detail (MOAB2VTK, others TBD)

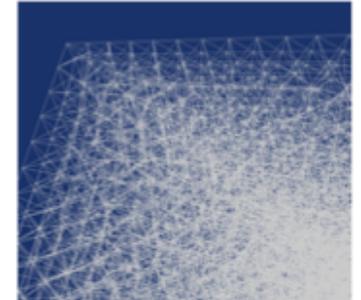
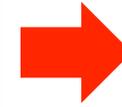
Cian plus additional proxies (will) all appear together on the same web page for download. Exact amount of integration between cian and additional coupling and datavis proxies is ongoing.



(a) Coupling Proxy App



Sample mesh of 1M
hexahedral cells.



Sample mesh of 1M
tetrahedral cells.

- Driving questions
 - What is the memory requirement to access data via the coupled interface, in terms of data and code?
 - How does it compare with access to the native interfaces of individual codes?
 - What is the performance and accuracy of coupling
- Method
 - Reads two different test meshes
 - Transfers solution from first mesh to second (ie, couples the meshes)
 - Computes error between transferred solution and original
- Relevance
 - Coupling essential to entire project and to multiphysics problems



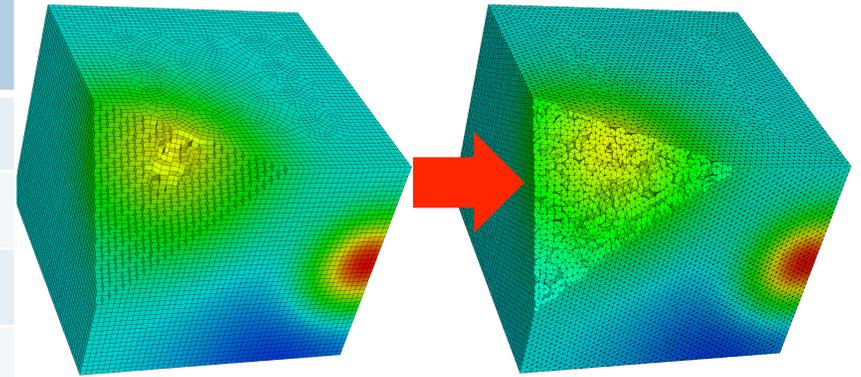
(a) Coupling Proxy App

- Parameters
 - Inputs: different mesh types (hex FEM, tet FEM, SEM), same or different communicators, same or different processes for source and target mesh.
 - Outputs: timing, memory footprint, accuracy
- Instrumentation
 - timing: `MPI_Wtime()`, memory footprint: time series of memory size, accuracy: max and RMS error between projected field and reference field.
- Hypothesis
 - Accuracy will vary with order of projection. Communication and time will be less with multiple meshes on same node, but memory cost will double. Node memory size will be determined by whether we choose to support 2 meshes on the same node. Unclear whether number of communicators will have an impact.



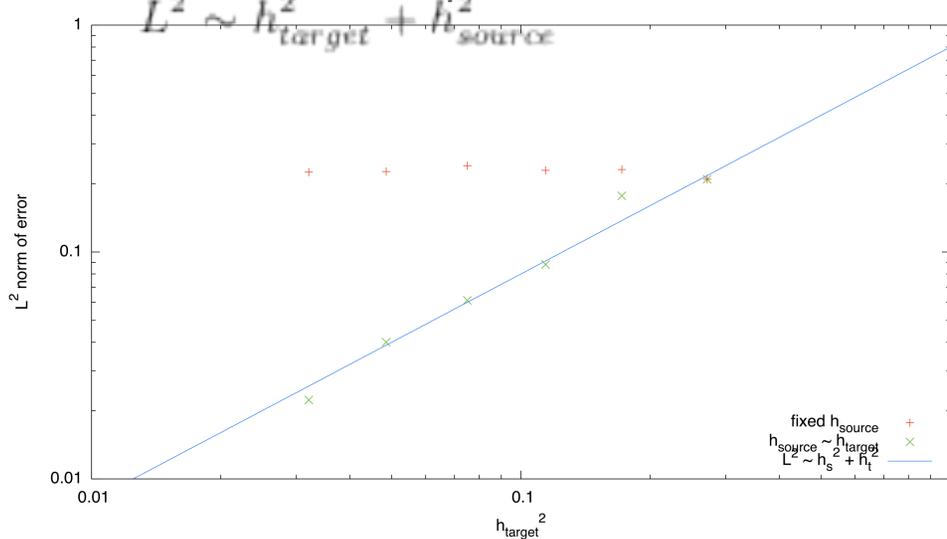
(a) Early Results

Source Mesh	Target Mesh	Norm. Vertex RMS Error	Norm. Element RMS Error
512 hex	1k hex	0.6%	4.6%
1k hex	512 hex	0.4%	2.6%
512 hex	1M hex	0.6%	4.4%
1M hex	512 hex	0.0%	0.3%
1M hex	1 M tet	0.0%	0.3%
1M tet	1 M hex	1.3%	0.8%

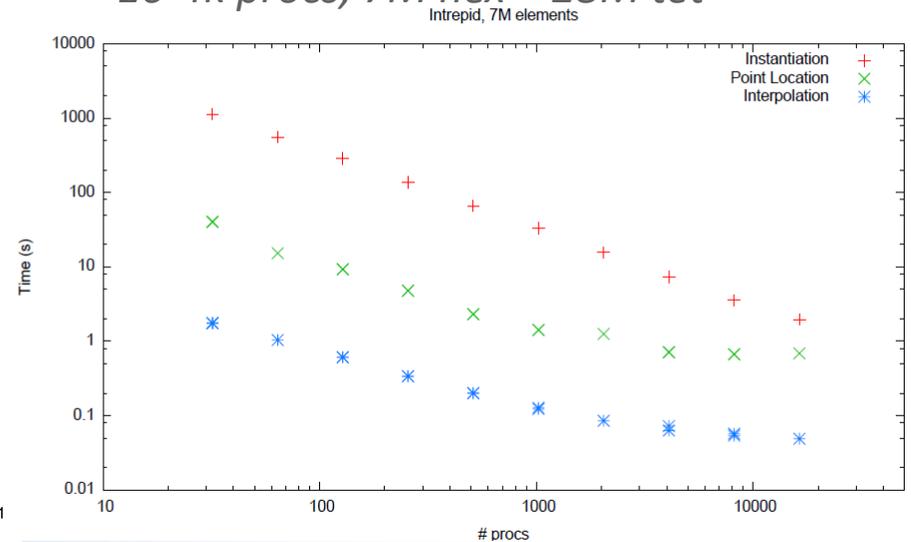


Coupling a 7M hex mesh to a 28M tet mesh

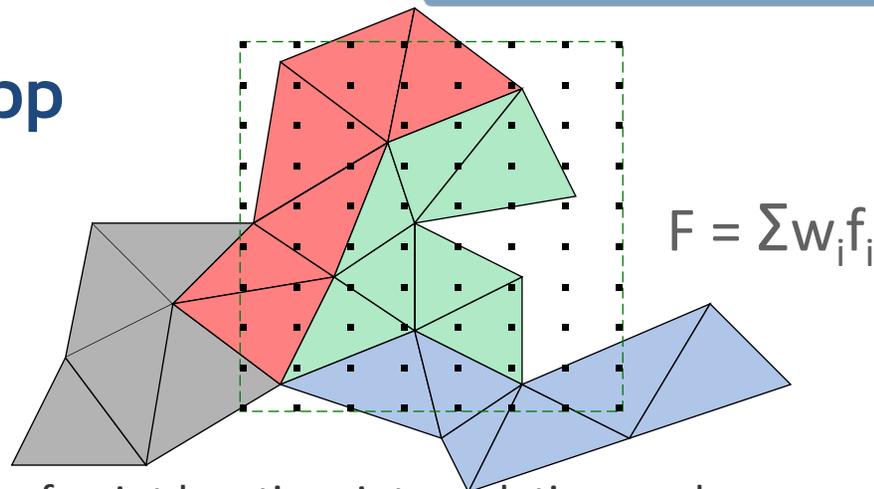
Error scales as expected with mesh size



16-4k procs, 7M hex – 28M tet



(b) Analysis Proxy App



- Driving questions
 - What is the relative cost of point location, interpolation, and communication?
 - How does the relative balance of these costs change with stencil size and mesh type?
 - What resources are needed?
- Method
 - Stencil computation accessing remote neighboring data to a specified distance in 3D space, requiring access to several topological neighbors. The analysis proxy in cyan reads a MOAB mesh and executes a stencil operator. It accesses data from neighboring blocks and samples it regularly in the stencil boundaries. The computation at each point in the stencil is multiply-add.
- Relevance
 - Stencils are common analysis operators used for convolution, filtering, and pattern recognition. The Lambda-2 vorticity finder is one example. The size of the stencil has a direct effect on data intensity and computational intensity of the operator.

(b) Analysis Proxy App

- Parameters
 - Inputs: cell type (spectral, finite), cell order, stencil size, stencil location from block boundary
 - Outputs: overall, computation, sampling, communication costs, where costs are defined below
- Instrumentation
 - Costs measured in time, power, memory accesses, communication rounds, communication volume, flops, instruction mix
- Hypothesis
 - Higher order elements require more sampling, local computation. Larger stencils require more communication. Look for opportunities to overlap computation and communication, ideally evenly matched. Block size shifts the balance of compute / communicate. RDMA, smart NICS, low latency interconnect, cache size proportional to stencil size.



(b) Early Results

Communicating enough neighboring cells to compute a stencil of additional width t

Load mesh from MOAB

Initialize DIY

For (all blocks)

 get block bounds and neighbors from MOAB

Replicate decomposition in DIY

while (!done) {

 for (cells) {

 for (neighbors) {

 if (cell intersects neighbor extents + t)

 post cell to neighbor;

 }

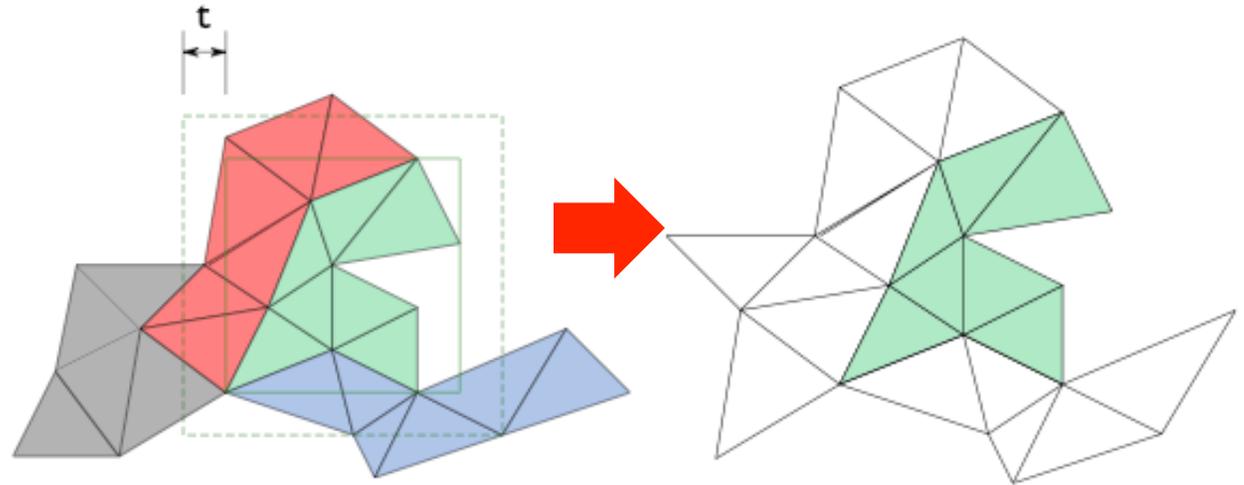
 }

 Exchange neighbors;

 Check whether done



}



Number of blocks = 512

12 ghost cells received by block 1

8 ghost cells received by block 2

18 ghost cells received by block 3

12 ghost cells received by block 4

18 ghost cells received by block 5

12 ghost cells received by block 6

26 ghost cells received by block 7

18 ghost cells received by block 8

18 ghost cells received by block 9

12 ghost cells received by block 10

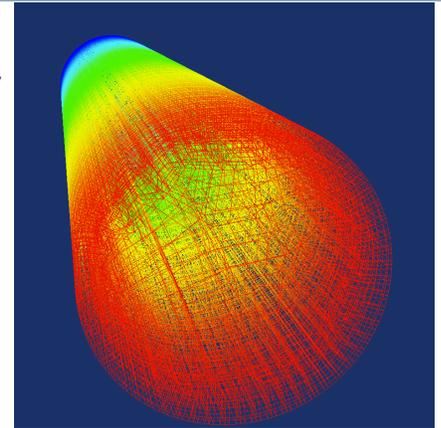
25 ghost cells received by block 11

18 ghost cells received by block 12

...

(c) In Situ Visualization and Analysis Study

5K 4th-order spectral element mesh with proxy CFD solution data, rendered in wireframe mode.



- Driving questions
 - How many additional resources are necessary for in situ analysis?
- Method
 - Analyze finite element and spectral element meshes with “in situ workflow”
 - Compare deep and shallow copy from simulation to analysis data structures
 - Read or generate mesh, perform deep or shallow copy from simulation to analysis data model, execute analysis filter, and write data product
- Relevance
 - Additional memory usage for in situ analysis can be significant if the mesh needs to be copied for an analysis data model required by visualization
 - HPC machines and science problems can be memory constrained
 - Study if run-time data model translation between simulation mesh and analysis mesh (shallow copy) is a viable technique to reduce memory pressure and what run-time costs, if any, are incurred by shallow copy vs. deep copy



(c) Experimental Proxy App - “MOAB2VTK”

- Parameters
 - Inputs
 - MOAB mesh: data file or generated mesh
 - System: number of processes, single node, distributed
 - VTK analysis: none, read all, isocontour, clip, slice, threshold
 - VTK render: none, surface render
 - VTK data model: base (interface = implementation), virtual (abstract interface with MOAB and VTK implementations)
 - Outputs
 - Time per execution stage per process, memory per execution stage per process
- Hypothesis
 - Existing VTK data model has a large footprint for unstructured meshes
 - Shallow copy will save memory to help simulation workloads that are tight on memory
 - Run-time translation (shallow copy) of data model should not be too expensive depending on number and type of visualization and analysis; no cost to doing virtual functions to implement the shallow copy



(c) “MOAB2VTK” Early Results

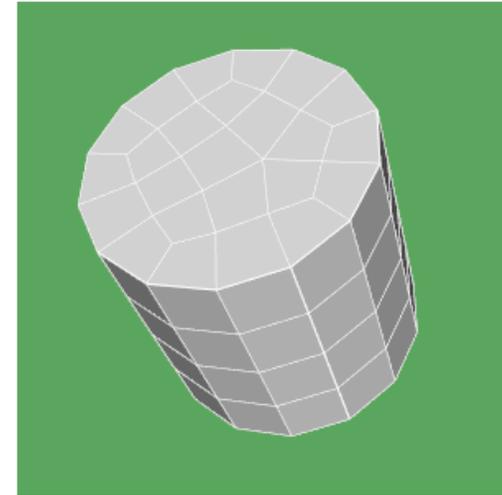
- Running 1-16 processes on HP SL230
 - 1-16 million tets
- Running 1-16 processes on HP DL980
 - 1-16 million tets
- Running 16-512 processes on LANL cluster
 - 1-512 million quads
- Preparing the results for a Supercomputing 2013 paper

```

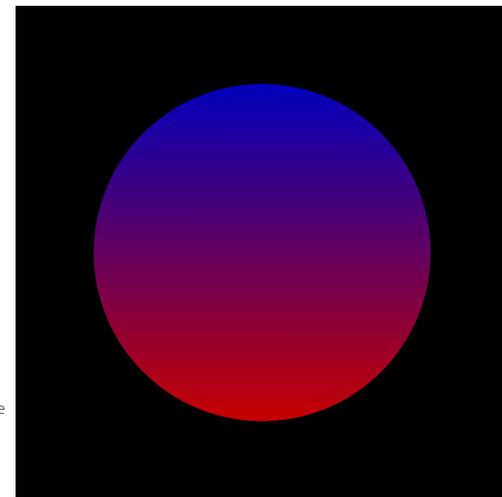
timer (s), start, end, elapsed
read MOAB, 71777.464122988, 71778.000071778, 1.090433001
copy + benchmark total, 71778.600565748, 71798.000071798, 20.264674139
deep copy total, 71778.600620503, 71780.000071780, 2.007776966
create id map, 71778.600630560, 71778.000071778, 0.000137727
create cells, 71778.600773037, 71778.000071778, 0.281072035
create points, 71778.881932793, 71779.000071779, 0.118359177
create data arrays, 71779.000509037, 71780.000071780, 1.540699166
create grid, 71780.541217143, 71780.000071780, 0.066829164
clip total, 71780.672058353, 71795.000071795, 14.935525324
render total, 71795.657863810, 71798.000071798, 3.207372445
rendering, 71796.450663454, 71798.000071798, 2.174128275

name (kB), size, rss, pss, shared clean, shared dirty, private clean, private dirty, referenced, anonymous, anon huge
before reading mesh, 988380, 80004, 28797, 67704, 52, 0, 12248, 80004, 12168, 0, 0, 0, 80004
after reading mesh, 1070372, 150308, 98426, 68584, 68, 0, 81656, 150308, 81580, 40960, 0, 0, 70304
after copying to VTK, 1159020, 238960, 186872, 68872, 68, 0, 170020, 238960, 169944, 81920, 0, 0, 88652
after clipping, 1230824, 310564, 258103, 69356, 68, 0, 241140, 310564, 241064, 120832, 0, 0, 71604
after running the filter, 1230824, 310812, 258351, 69356, 68, 0, 241388, 310812, 241312, 120832, 0, 0, 248
after rendering, 1508988, 581492, 524754, 71836, 4740, 344, 504572, 581492, 504524, 198656, 0, 0, 270680
after running everything, 1398964, 482784, 428067, 71872, 644, 344, 409924, 482784, 409876, 143360, 0, 0, -98708

```



Very first example of a simple MOAB mesh with shallow copy direct data access in VTK



One of the tests in MOAB2VTK – clip 4 million quads, surface render colored by element id, parallelized by 4 processors



(d) Storage Proxy App

- Driving questions
 - How does actual checkpoint writing compare to benchmarks
 - How does writing analysis data differ from checkpoint data
- Method
 - Read mesh interface handle
 - Writes mesh to storage (checkpoint)
 - Writes results of analysis or visualization proxy to storage
 - Compare using different formats and I/O libraries
 - Compare performance with I/O benchmarks and published performance for similar I/O workloads
 - Test performance of parallel readers for visualization tools
- Relevance
 - Storage rates lag computational rates by greater factors with each new machine, making storage a primary bottleneck. Storage isn't an issue at the moment, but planning for the future
 - Storage data models: building relationship with MOAB team as part of this project (and Damsel) and analysis work helping to understand data models present in applications
 - Future storage resources: CODES work will help in assessing access patterns WRT future storage (in coming years)



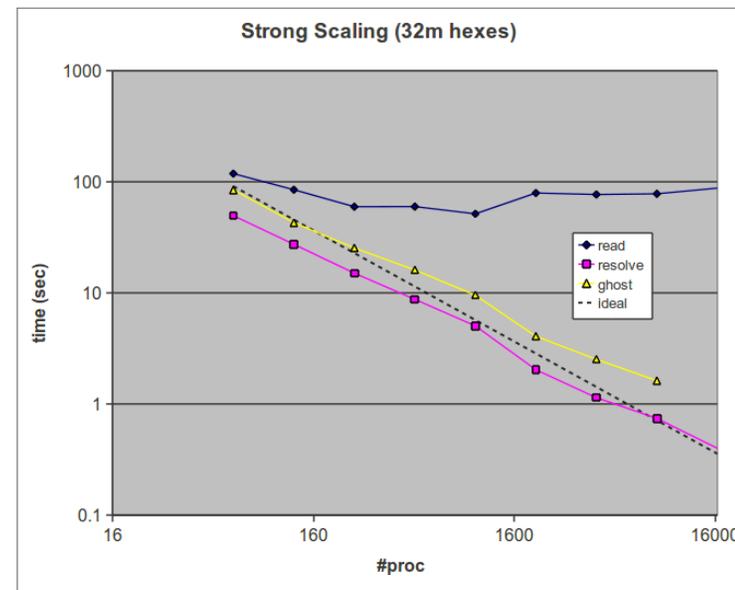
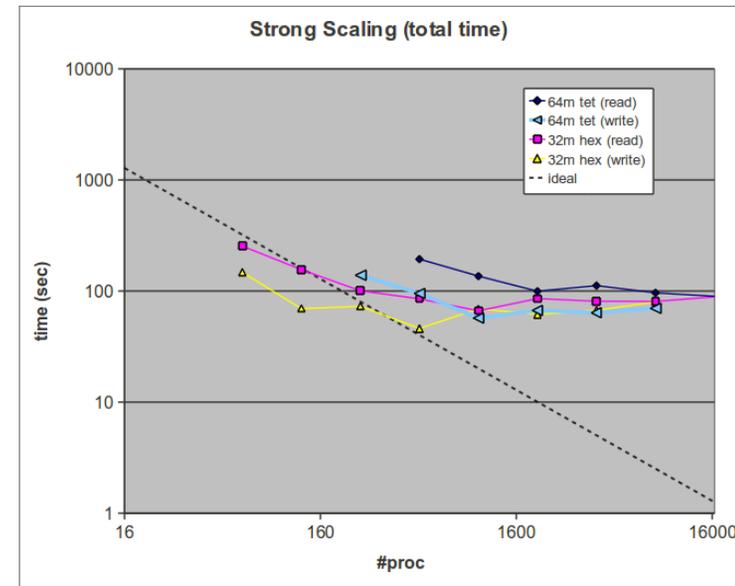
(d) Storage Proxy App

- Parameters
 - Inputs: mesh type, I/O library type (MOAB, MPI-IO), number of output files, collective / independent I/O
 - Outputs: time, percent peak storage bandwidth, strong and weak scaling
- Instrumentation
 - MPI_Wtime()
 - Darshan
- Hypothesis
 - Performance of shared file storage at high process counts degrades due to lock contention; implication for exascale storage systems is to reduce or eliminate locks in storage systems; Large numbers of files resulting from subfiling will also be a problem for subsequent use. Scalability of high-level libraries also will be problematic at high process counts.

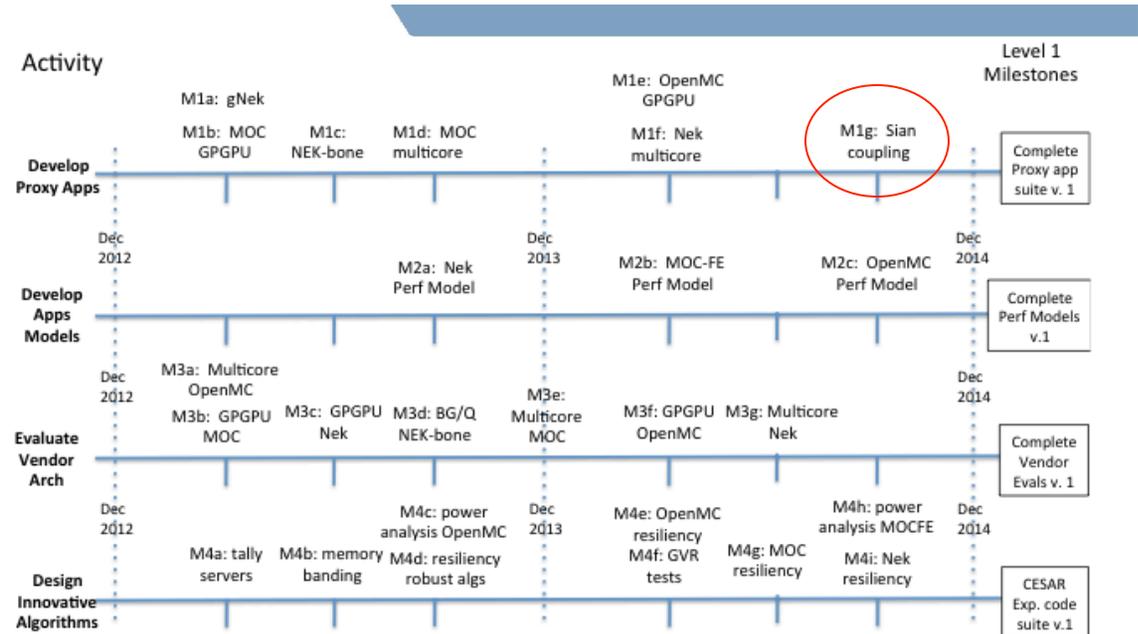


(d) Early Results

- Data taken on Intrepid (IBM BG/P)
- Read/write for 32m hex, 64m tet elems
 - Nowhere near ideal speedup
 - Absolute time tolerable in most cases
 - Drastic tet time improvement after reordering by partition
 - Fewer small fragments of HDF5 datasets
- Read/resolve/ghost times
 - Read times about constant
 - Resolving shared entities, exchanging ghost entities close to linear speedup
- 7/8 time reduction for models w/o edges/faces (geometry sets)
 - Could also change partitioning tool so this data didn't have to be inferred



Milestones Progress



■ L1 Milestones

- 1. L1 Milestone: Completion of the next phase of mini-apps development, including a quantification of coverage with respect to the corresponding full application, careful documentation and testing, etc. for third-party usage and made available on website.
 - L2 Milestone: Completion of v.1 of coupling mini-app• deliverable: new stand-alone mini app that explores both numerical and data movement issues for neutronics/CFD coupling for both particle-based or PDE-based methods. • metric for progress: completion of sub-components of coupler • components involved: Data/Vis, Apps, Programming Models
 - L3 Milestone: Prototype of Cian coupling mini-app for deterministic neutronics/CFD coupling • deliverable: technical report demonstrating key findings. • metric for progress: lines of code written, early results • components involved: Data/Vis, Apps, Programming Models • June 1, 2013

