Covisualization of full data and in situ data extracts from unstructured grid CFD at 160k cores

Michel Rasquin[1] michel.rasquin@colorado.edu
Patrick Marion[2] pat.marion@kitware.com
Venkatram Vishwanath[3] venkatv@mcs.anl.gov
Raymond M. Loy[4] rloy@alcf.anl.gov
Andrew C. Bauer[2] andy.bauer@kitware.com
Benjamin Matthews[5] matthb2@scorec.rpi.edu
Min Zhou[5] zhoum@scorec.rpi.edu
Onkar Sahni[5] osahni@scorec.rpi.edu
Jing Fu[6] fuj@cs.rpi.edu
Ning Liu[6] liun2@cs.rpi.edu
Christopher D. Carothers[6] chrisc@cs.rpi.edu
Mark S. Shephard[5] shephard@scorec.rpi.edu
Mark Hereld[3] hereld@mcs.anl.gov
Michael E. Papka[4] papka@anl.gov
Kalyan Kumaran[4] kumaran@alcf.anl.gov
Berk Geveci[2] berk.geveci@kitware.com
Kenneth E. Jansen[1] kenneth.jansen@colorado.edu

[1]University of Colorado at Boulder, 429 UCB, Boulder, Colorado 80309
[2]Kitware, Inc., 28 Corporate Drive, Clifton Park, NY 12065
[3]Math and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
[4]Argonne Leadership Computing Facility, Argonne, IL 60439
[5]Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180
[6]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180

ABSTRACT

Scalability and time-to-solution studies have historically been focused on the size of the problem and run time. We consider a more strict definition of “solution” whereby a live data analysis (covisualization of either the full data or in situ data extracts) provides continuous and reconfigurable insight into massively parallel simulations. Specifically, we used the Argonne Leadership Class Facility’s BlueGene/P machine using 163,840 cores tightly linked through a high-speed network to 100 visualization nodes that share 200 GPUs. Meshes ranging from 416M to 3.3B elements discretize the flow over a full swept wing with an unsteady synthetic jet to evaluate time-to-solution plus insight. On the full machine, the 416M element mesh takes 2 seconds per flow solve step including the extraction and rendering of a slice or a contour, slowing the simulation by only 3.4 and 6.6% respectively. The 3.3B element case proved scalable at 14.7 seconds per time step.

1. INTRODUCTION

Massively parallel computation provides enormous capacity to perform simulations on a time scale that can change the paradigm of how simulations are used by scientists, engineers and other practitioners to address discovery and design. Strong scalability solvers have demonstrated that a factor of 512 or 9 doublings of the number of cores has resulted in a factor of 424 compression of time [26]. That study, and many like it have been heavily focuses on the size of the problem that can be solved and the number of time steps per minute that can be completed. This loose definition of solution ignores the fact that it may take orders of magnitude longer time to perform any reasonable assessment of the insight gained due to the time it takes to write the data, load the data into post processing software, and to analyze and display insightful results. As these simulations have moved from O(10k) cores to O(100k) cores it has become clear that the classical paradigm of data creation, storage, and retrieval later for subsequent analysis must be reconsidered. For the machines of the near future and the exascale machines currently being co-designed, the amount of solution data that must be stored for later retrieval and post-processing can be prohibitive. Furthermore, the large time required to analyze the data does not effectively provide scientists and engineers with improved understanding of the problem they are simulating.

This situation strongly motivates coprocessing the simulation. Once that choice has been made several options become available. The opposite extreme from the classical run/store/read/analyze is to imbed the entire data analytics process into the solver. Here images of a pre-defined data analytics filter chain are processed within the primary simulation and exported either to files or directly via sockets to coprocessing resource whose only requirement/capability is to display the resource. While this approach has proven productive in some application areas [4], it typically limits the extent to which the data analytics can be reconfigured.

In many situations it is highly desirable to be able to set up an initial definition of the filter chain, view several live frames from an ongoing simulation, and then redefine the filter chain to provide a more insightful window into the ongoing simulation. Indeed, if these views can be provided at a live frame rate (display completed before the next data set is delivered to the visualization resource), computational steering becomes possible wherein not only can the data analytics be redefined in a way that maintains temporal continuity of the insight but also key parameters of the solve can be adjusted and their influence on the simulation observed. This visual feedback from parameter changes offers the potential to bring visual iteration of the design and discovery process to massively parallel simulation at unprecedented computational complexity and fidelity, experiential simulation.

The best coprocessing approach to realize this vision of experiential simulation and/or less aggressive visions of near time but not necessarily live covisualization will likely be best accomplished by something between the two extremes described above. In this paper we consider two covisualization models. The first might be considered classical covisualization (CCV) wherein the entire data set is exported from the ongoing simulation to a smaller computational resource with more appropriate resources for data analytics (visualization resource). No data reduction is performed on the solver resource which has the advantage of not burdening it with the computational load of filtering the data but it does burden it with the time to ship the data to the visualization compute resource which is typically blocking to the solution process to some extent. Once the full data is shipped to the covisualization resource, any desired filter chain can be executed there since the full data is resident. The second approach, which we will refer to as *in situ* data extracts (ISDE) performs the currently defined filter chain on the solver compute resource and then ships only the data extract to the visualization resource. While ISDE consumes time from the solver compute resource to do the data extraction, for many filter chains, it can dramatically reduce the amount of data that must be transported to the visualization resource. It is important to note that the filter chain performed by ISDE can be dynamically reconfigured without stopping the run and thus, both are suitable candidates for interactive monitoring of ongoing jobs and/or computational steering.

In this paper, we compare these two approaches, CCV and ISDE, on a challenging unsteady fluid flow problem at full machine scale (160k processors for solver and 100 nodes of coprocessing). To set the stage for this comparison, in Section 2 we provide background on the solver, the coprocessing library, the visualization tools, and the computational resource used. Since the data transport is key to performance, the two data transport mechanisms considered herein will be described in Section 3. The results of our studies are provided in Section 4 followed by related work in Section 5 and conclusions in Section 6.

2. BACKGROUND

2.1 Parallel Flow Solver: PHASTA

PHASTA is a parallel, hierarchic (2nd-5th order accurate), adaptive, stabilized (finite element) transient, incompressible and compressible flow solver. The discretization approach is representative of continuum partial differential equation solvers which have matured for a wide range of physical problems including ones in fluid mechanics, electromagnetics, biomechanics, to name a few. PHASTA (and it's predecessor ENSA) was the first massively parallel unstructured grid LES/DNS code [9, 10, 8] and has been applied to flows ranging from validation benchmarks to flows in complex geometries of practical interest. It is also the flow simulator for SimVascular [28] (supported by NSF and NIH).

PHASTA has been shown [10, 13, 36] to be an effective tool using implicit techniques for bridging a broad range of time and length scales in various flows including turbulent ones (based on RANSS, DES, LES, DNS). It has also effectively applied recent anisotropic adaptive algorithms [18, 23, 24] along with advanced numerical models of flow physics [7, 11, 30, 31, 32, 33]. PHASTA has also extended its capability to simulate two-phase flows using the level set method [19, 20] to implicitly track the boundary between two immiscible fluids. Many of its application cases have been sufficiently complex that grid independent results could only be obtained through efficient use of anisotropically adapted unstructured grids or meshes capable of maintaining high quality boundary layer elements [23], and scalable performance on massively parallel computers (PHASTA has been shown to scale to 288k cores [26]).

The computational work in PHASTA, and other similar implicit methods, mainly consists of two components: (a) formation/assembly of algebraic system of equations and (b) computation of the solution to the formed system of equations. In the first component, element-wise integration based on numerical quadrature is performed to form the system of equations. The resulting system is highly sparse but involves a large number of unknowns and non-zero entries. Thus, the second work component of PHASTA finds solution to the formed system of equations by using pre-conditioned iterative solvers (e.g., GMRES [21, 27]) suitable for very large, sparse (distributed) systems. Both aspects have been carefully constructed for parallel performance and scaling on various systems (including Ranger-TACC, Kraken-NICS, Franklin-NERSC, BGL-CCNI and BGP-ALCF).

All computations are based on a decomposition or partition of the mesh into parts with equal work load. The term part is used to denote a set of mesh entities whereas term partition is used to indicate collection of all parts (i.e., together all the parts within a partition comprise the aggregate mesh). Graph or hypergraph [1, 14] based partitioning schemes are currently used as they are more suitable for unstructured meshes. Here, partitioning based on elements is applied as it is natural for equation-formation stage, making it highly scalable. Note that during the equation-formation stage the computational load (in any processing core) depends on the elements present in the local part whereas in the equation-solution stage it depends on the degrees-of-freedom (*dofs*), or unknowns in the system of equations, on that part. So long as the *dof* balance is preserved, such a partitioning also maintains the scalability in the equation-solution stage.

Further details of the parallelization strategy and the parallel performance are given in [22, 26]. The key aspect relative to this paper is that the same partitioning that achieves excellent scaling for the equation formation and the equation solution will also be used for the ISDE (when performed).

2.2 Visualization Tools: ParaView

Interactive simulation will likely be performed remotely so it is highly desirable to develop a visualization software chain that can link together visualization resources on various machines to achieve the best result. As the data on the massively parallel resource continues to grow it is critical that the data compression be carried out in stages on different hardware. *In situ* data extracts from a co-processing library can provide a way to substantially compress the data from full geometry to the viz geometry. The data can be further compressed from polygons to pixels on a parallel visualization server. This can in turn be linked to a GUI client which displays results and receives human input either running on the users laptop or over a VNC server on the visualization cluster front end with the display set to the laptop. While it is possible to create each of these codes ParaView is proving an efficient and reliable open-source alternative to meet these objectives.

ParaView [29] is an open-source, multi-platform data analysis and visualization framework. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. The ParaView framework can be used both interactively and for batch processing. In either case there is a server (pvserver) that is responsible for performing the computationally expensive visualization and analysis tasks. For interactive use, the client is typically the ParaView GUI or a Python interface. The server can be run on anything from laptops for smaller data to supercomputers to analyze datasets of petascale size.

2.2.1 The ParaView co-processing Library

The ParaView co-processing Library is a C++ library built atop the Visualization Toolkit (VTK) and ParaView. Through VTK the co-processing library can access a large number of algorithms including writers for I/O, rendering algorithms, and processing algorithms such as isosurface extraction, slicing, and streamline generation. Through ParaView the co-processing library controls the pipeline structure. The purpose of the library is to allow simulations to visualize and analyze their data with the same computational resources used by the simulation code. Additionally, through *in situ* analysis, the size of data outputted from a simulation run can be vastly reduced without losing the desired information from the run.

The co-processing library API was designed to be minimal and efficient. This is because we expected the library to be called at regular intervals and for time-dependent petascale computations this could potentially be very often. The only information the co-processing library expects from the simulation is the mesh, the field information defined over the mesh, and the current time and time step the library is getting called at. Since the co-processing library is a general tool, the interface will not be able to handle arbitrary information passed from the simulation code. Because of this, the co-processing library requires adaptor code to translate simulation data structures into VTK data structures.This allows the simulation code to pass as much information to the co-processing library as is supported by VTK data structures, thus not limiting it to the specific co-processing library API. The main cost tradeoff in the design of the adaptor is between processing resources and memory resources. The studies herein are focused on time compression and as such the meshes on each processor are small leaving sufficient memory available for using more memory resources to save on processing resources.

Beyond translating the mesh and field information, the adaptor is responsible for passing the simulation time and time step information as well. The reason for this is that the simulation does not know *a priori* what co-processing work needs to be performed or when it needs to be done. By having the simulation code call the co-processing library at regular intervals, the co-processing library can determine if any actual work needs to be done. If no co-processing needs to be performed then the library returns control to the simulation with negligible computational cost. If co-processing does need to be performed then the adaptor will provide the necessary information in order to complete the desired co-processing pipeline.

2.3 Argonne Leadership Computing Facility Architecture

The Argonne Leadership Computing Facility (ALCF) is a U.S. Department of Energy facility that provides leadership-class computing infrastructure to the scientific community. Figure 1 depicts the architecture of the primary ALCF resources consisting of the compute resource (Intrepid), the data analysis cluster (Eureka), and the file server nodes interconnected by a large Myrinet switch complex, which are the components of the ALCF that are of focus in this paper.

Blue Gene/P (BG/P) is the second in a series of supercomputers designed by IBM to provide extreme-scale performance together with high reliability and low power consumption. Intrepid is a 160K core BG/P system with a peak performance of 557 TF, 80 TB local memory, and 640 I/O nodes, connected to the switching interconnect with an aggregate 6.4 Tbps. BG/P systems are composed of individual racks that can be connected together; each rack contains 1024 four-core compute nodes, for a total of 4096 cores per rack. Blue Gene systems have a hierarchical structure; Intrepid has 64 compute nodes grouped into a "pset", and 8 psets together form a midplane that contains 512 nodes. Each rack contains two such midplanes. Large Blue Gene systems are constructed in multiple rows of racks.

Both compute and I/O nodes on the BG/P use a quad-core, 32-bit, 850 MHz IBM Power PC 450; Intrepid nodes each have 2GB of memory. Each node is connected to multiple networks. The I/O and interprocess communication of BG/P travel on separate internal networks. A three-

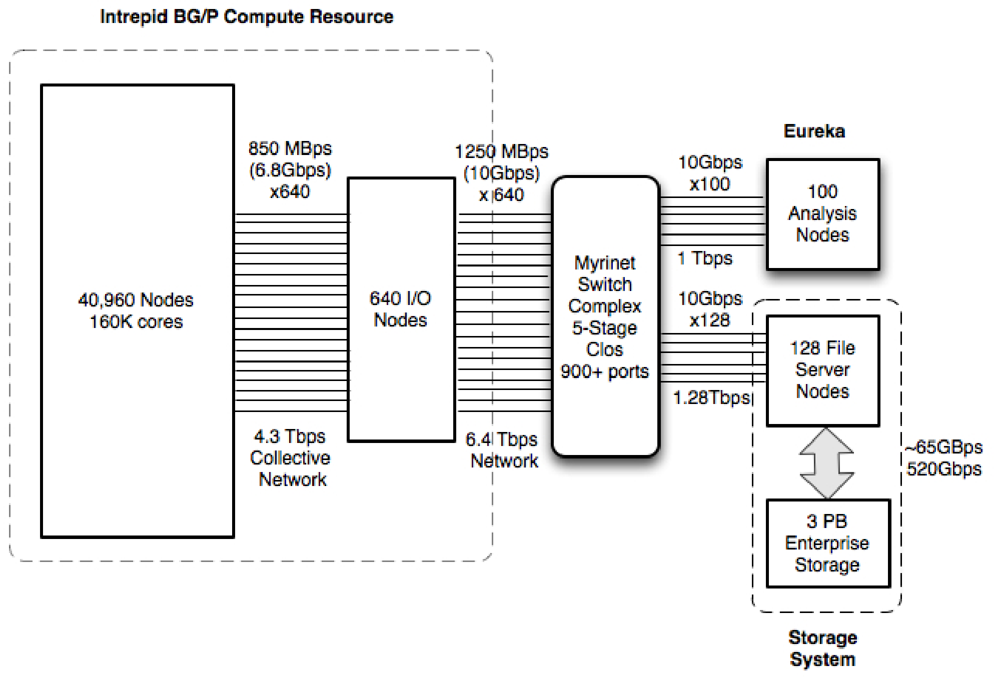


Figure 1: The Argonne Leadership Computing Facility maintains a 160K core Blue Gene/P (Intrepid), data analysis cluster (Eureka), and the file server nodes all interconnected by a 5-stage Myrinet switch complex, as well as other compute infrastructure including several test systems and a large computing cloud resource

dimensional torus network is used for point-to-point communicating among compute nodes (CNs), while a tree network allows CNs to forward their I/O to dedicated I/O nodes (IONs). For each pset the ION receives I/O requests from the CNs in that group and satisfies those requests via its 10 gigabit Ethernet port to the external I/O network. The tree network can also be used for optimized MPI collective operations among the CNs.

The external I/O network provides I/O connectivity to file servers nodes (FSNs) of a clusterwide file system as well as connectivity to the data analysis (DA) cluster nodes. Eureka, the data analysis cluster, contains 100 servers with 800 Xeon cores, 3.2 TB memory, and 200 nVidia Quadro FX 5600 GPUs. Eureka is connected to the switch with 100 links at 10 Gbps each. There are 128 file server nodes (FSNs), each node of which is a dual-core dual-processor AMD Opteron with 8 GB RAM per core. Each FSN is connected to the Myrinet switch complex over 10 Gbps. The FSNs are connected via InfiniBand 4X DDR to 16 Data Direct Network 9900 storage devices.

3. DATA TRANSPORT

As noted earlier we consider two forms of covisualization, CCV and ISDE. CCV exports the full data while ISDE utilizes the ParaView coprocessing library to create data extraction. Both approaches require an efficient way to transport their data from the massively parallel computer (hereafter referred to as Intrepid) running the solver to the parallel visualization resource (hereafter referred to as Eureka see 2.3). In this study we consider two ways to transport the data, VTK sockets and GLEAN, a library developed for fast IO forwarding.

3.1 VTK sockets

The geometry and solution data are sent from the Intrepid to Eureka via a two stage process: aggregation and socket sending. During the aggregation stage, data is moved from M to N processes, where M is the total number of simulation processes on Intrepid and N is a subset of these processes. The size of N is dependent on the number of sockets opened by the pvserver on the Eureka analysis cluster. Each process in the subset N sends its aggregated data through a tcp socket to a pvserver process. For example, if there are 200 pvserver processes, and each one opens 8 sockets, then the number of aggregators, N, will be 1600.

To aggregate the data, the Intrepid processes are split into groups and each process within a group uses a sub communicator to send its data arrays to the group root. The arrays are sent via point to point communication using the blocking MPI_Send. The group roots append the coordinates, connectivity, and solution arrays as they are received. Since the connectivity arrays use local point indexing starting at index 0, offsets are added to each element of the received connectivity arrays before they are appended. The process groups are organized using the 3D torus topology, and chosen so that the group root processes are those that have socket connections to the pvserver.

During the socket send stage, each Intrepid process with aggregated data sends its arrays to a pvserver process on the analysis cluster by writing to a TCP socket. Before the socket sends begin, the root process on Intrepid sends a message to the root pvserver process over a separate control socket. The root pvserver process broadcasts the message to its satellites, telling them to begin reading from their sockets. An individual pvserver process may open more than one socket, so this means that multiple Intrepid aggregators may be connected to the same pvserver process. Each pvserver process is single threaded and reads from one socket at a time. It receives all the data from an Intrepid aggregator before reading from the aggregator on the next socket. After each pvserver process has finished reading from its socket, the root pvserver process notifies the ParaView client that new data is available by sending a signal over the client-server communication channel. When the ParaView client receives a notification that a new data has arrived from Intrepid, the client sends a request back to the pvserver to begin processing the new time step.

Due to the single threaded nature of pvserver, it is possible that the pvserver processes will be occupied with analysis work when the Intrepid aggregators become ready to send their data. In this case, Intrepid will be blocked until the pvserver processes complete their current tasks and respond to the Intrepid request to read from their sockets. During covisualization, the pvserver may be tasked with running several analysis filters in addition to the rendering work. The execution of a pipeline of analysis filters and rendering algorithms in parallel across pvserver processes is composed of numerous tasks separated by interprocess communication breaks for information gathering and remote method invocations. Each time a pvserver process completes a subtask and returns control to the interprocess communication loop, it is an opportunity for the request from Intrepid to be picked up and handled. When the workload on pvserver is high, and the compute time per time step on Intrepid is low, it is possible for Intrepid to complete multiple timesteps with data aggregation and socket sends while pvserver continues to work on a single filter execution cycle.

A user observing the display on the ParaView client may see the covisualization output for time step 1 followed by time step 3, without seeing time step 2. This means that while pvserver worked to filter and render the data for time step 1, it received data from Intrepid for time steps 2 and 3. The pvserver is guaranteed to receive each time step from intrepid, but is not required to process each one. For our experiment we asked the pvserver to begin processing whatever was the most recent time step immediately after it finished processing the current timestep. The requests were automated using a client python script to remove all human intervention from the experiment. It is also possible to set a maximum number of time steps to keep in memory. For the 3 billion element case we set the maximum to 1. This means the pvserver would delete the previous time step data before receiving a new one. The VTK reference counting model ensures that any portions of the data still working their way through the filter pipeline will not be freed until the processing for that time step has completed.

3.2 GLEAN

GLEAN is a flexible and extensible framework taking application, analysis and system characteristics into account to facilitate simulation-time data analysis and I/O acceleration. The GLEAN infrastructure hides significant details from the end user, while at the same time providing them with a flexible interface to the fastest path to their data and in the end scientific insight. In designing GLEAN we are motivated to improve the performance applications that are impeded by their own demanding I/O. We strive to move the data out of the simulation application to dedicated staging nodes with as little overhead as possible to the system. GLEAN is implemented in C++ leveraging MPI, threads and providing interfaces for Fortran and C-based parallel applications. It provides a flexible and extensible API that can be customized to meet the needs of the application.

A goal in GLEAN is to leverage the topologies to move the data out of Intrepid as soon as possible enabling the simulation to continue on with it's computation. To achieve this, GLEAN fully exploits the underlying network topology by utilizing both the 3D torus as well as the tree network for data movement. In GLEAN, we restrict the aggregation traffic to be strictly within the *pset* boundary. The goal being that we move the data out of the system as fast as possible to the staging nodes and use the staging nodes to perform any data shuffling. In each aggregator group, the node where the aggregation is performed is chosen such that they aggregator nodes are distributed across the tree network.

To create the aggregators, For each pset, we create a MPI sub-communicator giving us only the MPI processes belonging to the pset. We split the sub-communicator into aggregation sub-groups and create communicators for each sub-group. We would like to note that these communicators are create once at initialization and reused during an entire simulation run. To create the aggregation groups, we leverage the "personality" information to get the X,Y,Z rank in the torus of the each process and groups the nearest neighbors in all 3 dimensions. The data aggregation includes the associated data semantics too. Once the data is aggregated, the aggregator nodes send the data out to the staging nodes.

A distinguishing characteristic of GLEAN's data staging is that it leverages the data models and semantics of applications for staging instead of viewing data simple as files and/or buffers. On the staging nodes, typically the Eureka analysis cluster nodes, GLEAN runs an MPI job. It communicates with the GLEAN aggregator nodes over sockets, as sockets are the only way to communicate between BG/P CNs and the external I/O network. A key requirement is to scale to the large number of connections from BG/P - for 8 GLEAN aggregators per *pset*, an entire machine run (160K cores) will have 5,120 connections. These connections are distributed among the various GLEAN staging nodes. Further, each staging node is designed with a thread-pool wherein each thread handles multiple connections via a poll-based event multiplexing mechanism. Asynchronous data staging blocks the computation only for the duration of copying data from the CN to the staging nodes. The data staging serves as burst buffer for the simulation I/O that can be written out asynchronously while the simulation's computation proceeds ahead. The data semantics enables GLEAN to transform the data on-the-fly to various I/O formats as well as to facilitate covisualization.

4. RESULTS

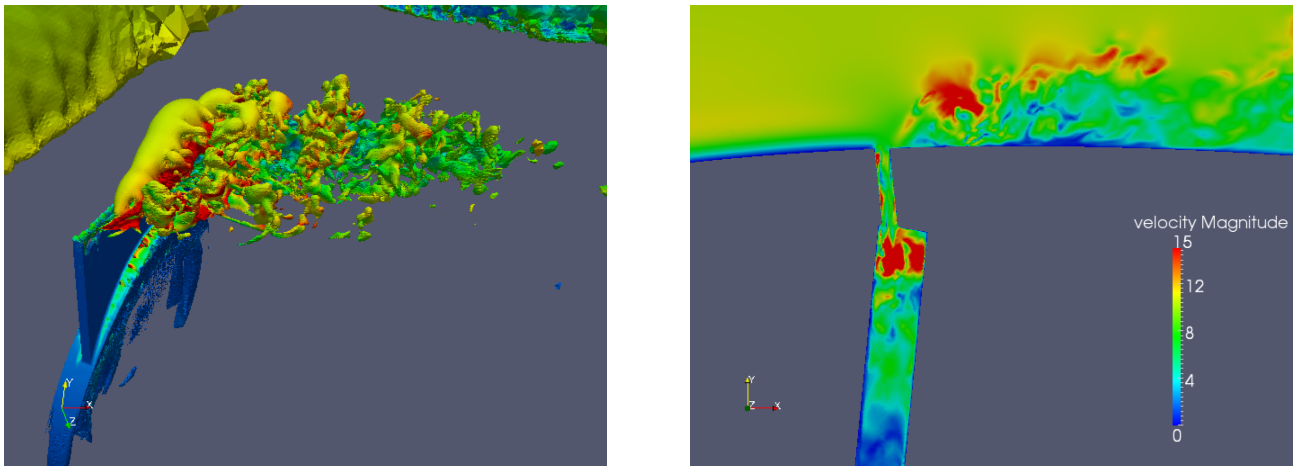


Figure 2: Isosurface of vertical velocity colored by velocity and cut plane through the synthetic jet (both on 3.3 Billion element mesh)

Weak scaling applications can double the mesh size with each processor doubling to bring ever more detailed resolution in finite time. While those simulations are also in need of covisualization, that is not the focus of this study. As noted earlier, PHASTA's strong scaling can compress the time-to-solution to O(1) second per unsteady flow time step. If visual images from that solution can be rendered in a comparable time, live visualization and even computational steering becomes viable. The goal of this paper to evaluate current hardware and to evaluate and extend current software's ability to meet this objective.

There are several important steps in this process. Before presenting our study we consider each of them. The first decision is whether to allow the coprocessing library to deep copy the solver data or whether it should be shared. The CPU costs are typically negligible but the memory costs could be significant and fatal if the solver is operating near the memory limit of the machine. For studies where time compression is pushed to the extreme, parallel flow solver's like PHASTA will be using a small fraction of the available memory making this decision easy. The second decision is to follow ISDE and do filtering on Intrepid or not. The resulting data, whether full (CCV) or a data extract (ISDE) must then be transported to the Eureka for covisualization but this step has two sub choices: i) to what extent should the data be aggregated on Intrepid (thereby reducing the number of "sockets" that must be opened with Eureka) and ii) the choice of socket (see Section 3). When we use VTK sockets we select the total number of sockets which then sets the level of aggregation required. When we use GLEAN, internal processes make these selections. The last remaining decision is the pvserver configuration. Again this breaks into two parts, the number of nodes and the number of processes per node. Since there are two GPU's per node there may be performance benefit from using 2 processes per node. To be clear, if ISDE was engaged on Eureka, then the pvserver only has to render the polyData results while if CCV was employed, a filter pipeline would have to be executed. The user controls the filter being applied and the view from a ParaView client that is being run on the front-end node of Eureka. Finally that front-end node's display is exported via VNC to wherever the user is sitting (e.g, at UCB), thereby allowing completely remote monitoring of a full machine simulation.

From this description it is clear that there are many steps and many things to vary within this study. The first rank in our study was the covisualization approach: CCV or ISDE. The second rank was the data transport approach: GLEAN or sockets (yielding CCVG, CCVS, ISDES, ISDEG but ISDEG remains under development). The third rank was the pvserver configuration $N_E - N_p$ which stands for the number of Eureka nodes and the number of pvservers ($N_p = \{1, 2\} * N_E$). The 4th rank, available only to the socket approach, was the number of sockets per N_p). The 5th rank was the filter pipeline chosen: slice or contour. The sixth rank was the size of the mesh: 416M elements and 3.32B elements. It will not be possible to show all the results but in the sections that follow we will provide an illustrative sample of the performance.

The performance will be evaluated in the context of a simulation of application involves flow over a full wing where a synthetic jet [25] issues an unsteady crossflow jet at 1750 Hz. One frame of this live simulation is shown in Figure 2 where two different filter pipelines have been evaluated in two separate runs of PHASTA. This particular application is in great need of this capability as a live visualization would enable computational steering. For example, the frequency of the jet or the amplitude of the blowing can be iterated with live visual feedback of the new flow structures guiding the iteration.

4.1 416 million element case

In the studies below we consider the scenario with the highest stress on the coprocessing library, where the filter pipeline is evaluated on every flow step. In this high time compression mode. The first filter pipeline that was evaluated was a slice that cuts through the synthetic jet cavity and the jet cross flow. As this plane is easily defined in advance of the simulation, it did not need to be altered while the simulation was ongoing so there was no live updating of the filter pipeline making every step very close to the same computational effort. The computational time was measured in several stages of the covisualization process. The time spent in the flow solve was verified to be independent of the filter and averaged 1.84 s (seconds) per step (two non-linear iterations per step).

The first interesting case is ISDES. The best pvserver configuration for this case in terms of total blocked time was 0.175 s resulting in a 9.5% tax on the simulation. Further breaking down this tax, the key contributions are: the data copy (0.00109 s), filter execution (0.0183 s), aggregation (0.0319 s), transport via VTK sockets (0.0104 s), initialize pipeline (0.0754) and cleanup (0.0359 s). Note that the last two account for 64% of the time. Future developments that allow these two costs to be done only on the first and last step are straightforward and worthwhile for runs where the filter pipeline is not being altered throughout the simulation. With these two removed the tax of a slice visualization would be reduced to 3.4% which is very small for such a worst cases scenario (visualize every step of a very fast simulation). Clearly if only every n_{viz} steps were visualized this tax would be amortized across n_{viz} steps.

The aggregation time and the transport time showed a significant dependance upon the number of sockets since this directly sets the number of levels that the data must be reduced and communicated on the BGP. By varying the number of sockets per pvserver, the aggregation time was able to be reduced substantially. While this does result in more, smaller messages being sent over more sockets to a finite number of pvservers and ultimately an appending of the data onto the fixed number of pvservers, the penalty to these phases was seen to slowly grow, yielding the best performance for pvservers that used 1600 sockets (e.g, 50-100 with 16 sockets per pvserver). Other combinations that produced 1600 sockets (e.g, 100-100 with 16 sockets per pvserver) yielded almost identical results (see later figures) suggesting that the number of Eureka nodes did not play a significant role in the performance. The number of sockets however played a very large role as will be discussed later. A similar analysis on the contour filter showed similar trends on this, somewhat heavier filter (e.g., the data extract of the contour averaged 1000 MB while the slice was 64.8 MB or 15.5 times as large). Here the current covis tax was 14.6% which could be reduced to 6.6% with the elimination of setup and close costs.

If the same two filters were applied on Eureka, rather on Intrepid we examine the CCVS and CCVG case depending on the data transport choice. The time spent transporting the data for CCVS (with 6400 sockets) is 8.9% while CCVG is 18.9% irrespective of the filter applied. This is not the whole tax though because the pvserver was not able to complete either filter and render before the next solver flow step. If one insists on live visualization with no lag or missed frames there was a significant additional lag when we blocked the flow solver until the pvserver completed. Alternatively, if the solver was not blocked frames were missed. When using ISDES and the slice filter, the pvserver was able to render the frame before the next time step for all cases, even with only 5 nodes. The contour filter had more data to render and was not able to complete the render before another step of the flow solver with less than 25 nodes. Consequently, a preliminary conclusion is that ISDES is much more suitable for live simulations with these relatively simple filters that parallelize well on Intrepid and substantially reduce the data that must be transported. Clearly, much more complex analysis of the data could be performed on the pvserver with the full data resident and in those cases, the lack of interactivity could be a fair trade off.

To better understand the dependence on the pvserver-socket configuration with ISDES, we plot the normalized time of all the pvserver configurations considered vs the number of sockets. The time is normalized by the best performer (1600 sockets). The plot for ISDES is shown in Figure 3 for the contour filter (slice filter looks similar). The minimum at 1600 sockets is the result of the tradeoff between reduced aggregation time at large socket counts and higher transport time for more, smaller transport sockets as shown in Figure 3.

4.2 3.32 billion element case

Next we consider the same flow but on a more refined mesh. Since the number of processors has not changed the load on the solver is significantly higher (8 times) and the amount of data that must be sent for CCV is also roughly 8 times larger. While this case is not as interactive at 14.27 seconds per step (two non-linear iterations per step) this also demonstrates that the flow solver is still scaling well (only 7.76 fold increase for a factor of 8 increase in mesh).

The first interesting case is ISDES slice. The best pvserver configuration for this case in terms of total blocked time was 0.1996 s resulting in a 1.40% tax on the simulation. Further breaking down this tax, the key contributions are: the data copy (0.00166 s), filter execution (0.0344 s), aggregation (0.0335 s), transport via VTK sockets (0.0185 s), initialize pipeline (0.0751) and cleanup (0.0347 s). Note that the last two account for 55% of the time. With these two removed the tax of a slice visualization would be reduced to 0.63%.

As before, the variation in the number of sockets dominated the performance with almost no dependence on the number of nodes or pvservers. In this case, the best performance for pvservers that used 3200 sockets (e.g, 100-100 with 32 sockets per pvserver). A similar analysis on the contour filter showed similar trends on this, somewhat heavier filter (e.g., the data extract of the contour averaged 3796MB while the slice was 240MB or 15.8 times as large). Here the delay time to the solver was 0.581 s yielding a current covis tax was 4.07% which could be reduced to 3.02% with the elimination of setup and close costs.

If the same two filters were applied on Eureka, rather on Intrepid we examine the CCVS and CCVG case depending on the data transport choice. The time spent transporting the data for CCVS (with 6400 sockets) is 99% while CCVG is 14% irrespective of the filter applied. It is clear that this on larger case (98.6 GB of data) the accelerated data transport algorithms in GLEAN are beneficial. Considering the time spent in GLEAN (aggregation plus transfer), a rate of 48.8GB per second was observed which is close to the theoretical rate that GLEAN is capable of. For CCVS the

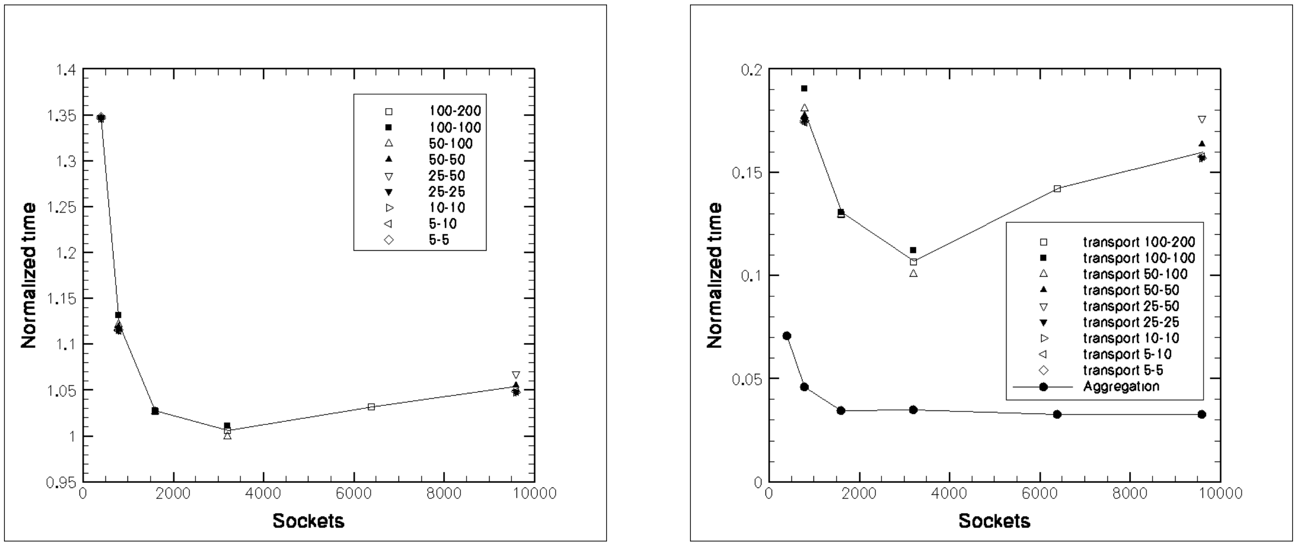


Figure 3: ISDES total (left) and key, varying component costs (right) for contour filter (slice similar) with 416M elements

data is larger due to additional VTK requirements (114 GB), the 6400 sockets achieved 8.23GB/s without aggregation or 8.06GB/s including aggregation. It is important to reiterate that this is not the whole tax though because again, the pvserver was not able to complete either filter and render before the next solver flow step. For ISDES on this large case we did not have sufficient time to explore as low of Eureka node counts as was done for the 416M case but all cases studied were able to render the extract before the subsequent time step was delivered.

When using ISDES and the slice filter, the pvserver was able to render the frame before the next time step for all cases, even with only 5 nodes. The contour filter had more data to render and was not able to complete the render before another step of the flow solver with less than 25 nodes.

Revisiting the dependence on the pvserver-socket configuration with ISDES on this much larger case shows similar trends. The time is normalized by the best performer (3200 sockets). The plot for ISDES is shown in Figure 4 for the slice filter (contour filter looks similar but the minimum shifts to 6400). The minimum at 1600 sockets is the result of the tradeoff between reduced aggregation time at large socket counts and higher transport time for more, smaller transport sockets as shown in Figure 4.

5. RELATED WORK

The concept of running a visualization while the solver is running is not new. It is mentioned in the 1987 National Science Foundation Visualization in Scientific Computing workshop report [16], which is often attributed to launching the field of scientific visualization. Over the years, there have been many visualization systems built to run in tandem with simulation, often on supercomputing resources. Recent examples include a visualization and delivery system for hurricane prediction simulations [4] and a completely integrated meshing-to-visualization system for earthquake simulation [35]. These systems are typically lightweight and specialized to run a specific type of visualization under the given simulation framework. A general coupling system exists [5] which uses a framework called EPSN to connect M simulation nodes to N visualization nodes through a network layer. Our approach differs in that we link the codes and run on the simulation nodes, directly accessing the simulation data structures in memory.

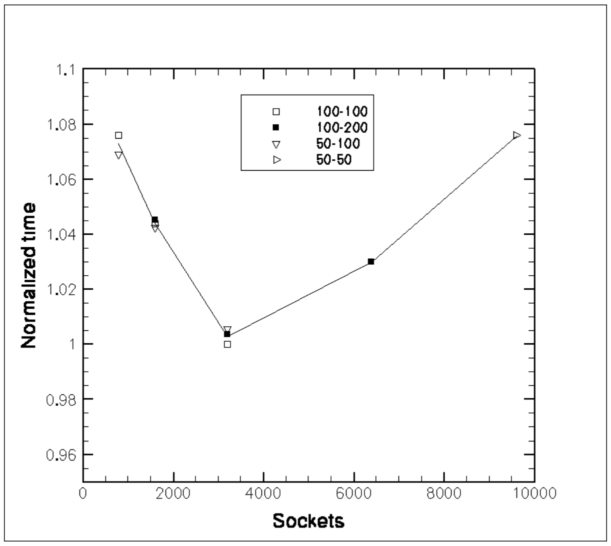
SCIRun [12] provides a general problem solving environment that contains general purpose visualization tools that are easily integrated with several solvers so long as they are also part of the SCIRun problem solving environment. Other more general purpose libraries exist that are designed to be integrated into a variety of solver frameworks such as pV3 [6] and RVSLIB [3]. However, these tools are focused on providing imagery results whereas in our experience it is often most useful to provide intermediate geometry or statistics during coprocessing rather than final imagery.

Recent efforts are utilizing the largest supercomputing platforms to run visualization post-processing. These tools include ParaView [17], which provides the framework for our coprocessing, and VisIt [34].

Ultimately, the integration of coprocessing libraries into solvers gets around the issues involved with file I/O. There are also some related efforts in making the I/O interfaces abstract to allow loose coupling through file I/O to be directly coupled instead. Examples include the Interoperable Technologies for Advanced Petascale Simulations (ITAPS) mesh interface [2] and the Adaptable I/O System (ADIOS) [15].

6. CONCLUSIONS

A live data analysis (covisualization of either the full data or *in situ* data extracts) was demonstrated to provide continuous and reconfigurable insight into massively parallel simulations. Specifically, the full Argonne Leadership Class Facility's BlueGene/P machine using 163,840 cores tightly linked through a high-speed network to 100 visualization



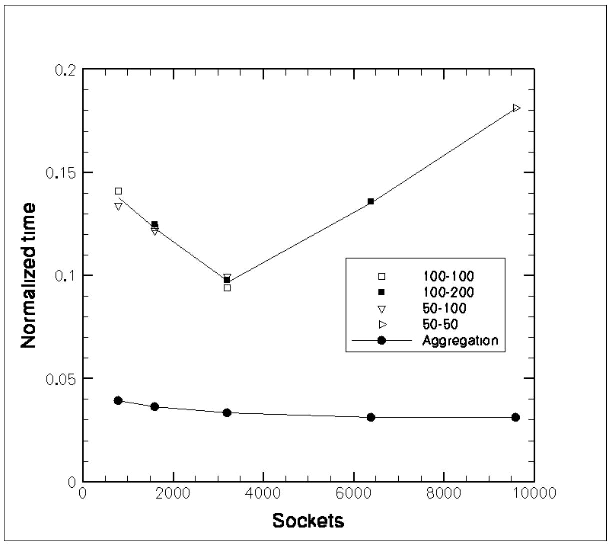


Figure 4: ISDES total (left) and key, varying component (right) costs for slice filter (contour similar) with 3.3B elements

nodes that share 200 GPUs was engaged to evaluate the current software and hardware's ability to deliver visualizations from an on going simulation. Meshes ranging from 416M to 3.3B elements were used to discretize the flow over a full swept wing with an unsteady synthetic jet to evaluate time-to-solution plus insight. On the full machine, the 416M element mesh takes 2 seconds per flow solve step including the extraction and rendering of a slice or a contour, slowing the simulation by only 3.4 and 6.6% respectively. The 3.3B element case proved scalable at 14.7 seconds per time step. Classical covisualization was also explored using two data transport mechanisms. While both data transport approaches were able to deliver data from the compute nodes to the visualization nodes at a very high rate, the visualization cluster was not able to filter and render the data at a rate that kept up with the solver on a step by step basis. Consequently, these results suggest that *in situ* data extracts that are coprocessed on the compute resource are more suitable for live simulations that use these relatively simple filters that parallelize well and substantially reduce the data that must be transported. Clearly, much more complex analysis of the data could be performed on the visualization server with the full data resident and in cases where that is needed, the lack of live interactivity could be a fair trade. A similar conclusion could be made for simulations running at a small enough time step that a significant number of steps could be skipped with an acceptable loss of interactivity. It was also noted that *in situ* data extracts where successful at 25% of the full ALCF visualization facility which bodes well for exascale hardware which is expected to not be as data analytics rich as current generation machines .

7. ACKNOWLEDGMENTS

We gratefully acknowledge the use of the resources of the Argonne Leadership Computing Facility at Argonne National Laboratory. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, an Argonne National Laboratory Director's Fellowship, U.S. Department of Energy under grant DE-FC02-06ER25769 and in part by the National Science Foundation under grant 0749152. The solutions presented herein made use of the linear algebra solution library provided by Acusim Software Inc. and meshing and geometric modeling libraries by Simmetrix Inc.

8. REFERENCES

[1] U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proc. of IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Long Beach, CA, USA, March, 2007. IEEE.

[2] K. Chand, B. Fix, T. Dahlgren, L. F. Diachin, X. Li, C. Ollivier-Gooch, E. S. Seol, M. S. Shephard, T. Tautges, and H. Trease. The ITAPS iMesh interface. Technical Report Version 0.7, U. S. Department of Energy: Science Discovery through Advanced Computing (SciDAC), 2007.

[3] S. Doi, T. Takei, and H. Matsumoto. Experiences in large-scale volume data visualization with RVSLIB. *Computer Graphics*, 35(2), May 2001.

[4] D. Ellsworth, B. Green, C. Henze, P. Moran, and T. Sandstrom. Concurrent visualization in a production supercomputing environment. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), September/October 2006.

[5] A. Esnard, N. Richart, and O. Coulaud. A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006)*, pages 7–14, Torremolinos, Malaga, Spain, October 2006. IEEE Press.

[6] R. Haimes and D. E. Edwards. Visualization in a parallel processing environment. In *Proceedings of the 35th AIAA Aerospace Sciences Meeting*, number

AIAA Paper 97-0348, January 1997.

[7] T. J. R. Hughes, L. Mazzei, and K. E. Jansen. Large-eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, 3:47–59, 2000.

[8] K. Jansen. Unstructured grid large eddy simulation of wall bounded flow. In *Annual Research Briefs*, pages 151–156, NASA Ames / Stanford University, 1993. Center for Turbulence Research.

[9] K. E. Jansen. Unstructured grid large eddy simulation of flow over an airfoil. In *Annual Research Briefs*, pages 161–173, NASA Ames / Stanford University, 1994. Center for Turbulence Research.

[10] K. E. Jansen. A stabilized finite element method for computing turbulence. *Comp. Meth. Appl. Mech. Engng.*, 174:299–317, 1999.

[11] K. E. Jansen and A. E. Tejada-Martínez. An evaluation of the variational multiscale model for large-eddy simulation while using a hierarchical basis. Number 2002-0283, Reno, NV, Jan. 2002. 40th AIAA Annual Meeting and Exhibit.

[12] C. Johnson, S. Parker, C. Hansen, G. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, December 1999.

[13] A. K. Karanam, K. E. Jansen, and C. H. Whiting. Geometry based pre-processor for parallel fluid dynamic simulations using a hierarchical basis. *Engineering with Computers*, 24(1):17–26, 2008.

[14] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irr. graphs. *SIAM Review*, 41:278–300, 1999.

[15] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, pages 15–24, 2008.

[16] B. H. McCormick, T. A. DeFanti, and M. D. Brown, editors. *Visualization in Scientific Computing (special issue of Computer Graphics)*, volume 21. ACM, 1987.

[17] K. Moreland, D. Rogers, J. Greenfield, B. Geveci, P. Marion, A. Neundorf, and K. Eschenberg. Large scale visualization on the Cray XT3 using ParaView. In *Cray User Group*, May 2008.

[18] J. Mueller, O. Sahni, X. Li, K. Jansen, M. Shephard, and C. Taylor. Anisotropic adaptive finite element method for modeling blood flow. *Computer Methods in Biomechanics and Biomedical Engineering*, 8(5):295–305, 2005.

[19] S. Nagrath, K. E. Jansen, and R. T. Lahey. Three dimensional simulation of incompressible two phase flows using a stabilized finite element method and the level set approach. *Comp. Meth. Appl. Mech. Engng.*, 194(42-44):4565–4587, 2005.

[20] S. Nagrath, K. E. Jansen, R. T. Lahey, and I. Akhatov. Hydrodynamic simulation of air bubble implosion using a fem based level set approach. *Journal of Computational Physics*, 215:98–132, 2006.

[21] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[22] O. Sahni, C. Carothers, M. Shephard, and K. Jansen. Strong scaling analysis of an unstructured, implicit solver on massively parallel systems. *Scientific Programming*, 17:261–274, 2009.

[23] O. Sahni, K. Jansen, M. Shephard, C. Taylor, and M. Beall. Adaptive boundary layer meshing for viscous flow simulations. *Engng. with Comp.*, 24(3):267–285, 2008.

[24] O. Sahni, J. Mueller, K. Jansen, M. Shephard, and C. Taylor. Efficient anisotropic adaptive discretization of cardiovascular system. *Comp. Meth. Appl. Mech. Engng.*, 195(41-43):5634–5655, 2006.

[25] O. Sahni, J. Wood, K. Jansen, and M. Amitay. Three-dimensional interactions between a finite-span synthetic jet and a crossflow. *Journal of Fluid Mechanics*, 671:254–287, 2011.

[26] O. Sahni, M. Zhou, M. Shephard, and K. Jansen. Scalable implicit finite element solver for massively parallel processing with demonstration to 160k cores. In *Proceedings of the SC09*, Springer, Berlin, 2009.

[27] F. Shakib, T. J. R. Hughes, and Z. Johan. A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis. *Comp. Meth. Appl. Mech. Engng.*, 75:415–456, 1989.

[28] SimTK. Simtk website. Hosted by Stanford University under the NIH Simbios project, http://simtk.org/xml/index.xml, 2007.

[29] A. H. Squillacote. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2007. ISBN 1-930934-21-1.

[30] A. E. Tejada-Martínez and K. E. Jansen. Spatial test filters for dynamic model large-eddy simulation on finite elements. *Communications in Numerical Methods in Engineering*, 19:205–213, 2003.

[31] A. E. Tejada-Martínez and K. E. Jansen. A dynamic Smagorinsky model with dynamic determination of the filter width ratio. *Physics of Fluids*, 16:2514–2528, 2004.

[32] A. E. Tejada-Martínez and K. E. Jansen. On the interaction between dynamic model dissipation and numerical dissipation due to streamline upwind/Petrov-Galerkin stabilization. *Comp. Meth. Appl. Mech. Engng.*, 194(9-11):1225–1248, 2005.

[33] A. E. Tejada-Martínez and K. E. Jansen. A parameter-free dynamic subgrid-scale model for large-eddy simulation. *Comp. Meth. Appl. Mech. Engng.*, 195:2919–2938, 2006.

[34] K. Thomas. Porting of VisIt parallel visualization tool to the Cray XT3 system. In *Cray User Group*, May 2007.

[35] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

[36] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *International Journal of Numerical Methods in Fluids*, 35:93–116, 2001.