Scientific Programming 17 (2009) 261–274
DOI 10.3233/SPR-2009-0287
IOS Press

Strong scaling analysis of a parallel, unstructured, implicit solver and the influence of the operating system interference

Onkar Sahni [a,*], Christopher D. Carothers [b], Mark S. Shephard [a] and Kenneth E. Jansen [a]
[a] *SCOREC, Rensselaer Polytechnic Institute, 110 8th St., Troy, NY 12180, USA*
E-mails: {osahni, shephard, kjansen}@scorec.rpi.edu
[b] *Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th St., Troy, NY 12180, USA*
E-mail: chrisc@cs.rpi.edu

Abstract. PHASTA falls under the category of high-performance scientific computation codes designed for solving partial differential equations (PDEs). Its a massively parallel unstructured, implicit solver with particular emphasis on fluid dynamics (CFD) applications. More specifically, PHASTA is a **p**arallel, **h**ierarchic, **a**daptive, **s**tabilized, **t**ransient **a**nalysis code that effectively employs advanced anisotropic adaptive algorithms and numerical models of flow physics. In this paper, we first describe the parallelization of PHASTA's core algorithms for an implicit solve, where one of our key assumptions is that on a *properly balanced* supercomputer with appropriate attributes, PHASTA should continue to strongly scale on high core counts until the computational workload per core becomes insufficient and inter-processor communications start to dominate. We then present and analyze PHASTA's parallel performance across a variety of current near petascale systems, including IBM BG/L, IBM BG/P, Cray XT3, and custom Opteron based supercluster; this selection of systems with inherently different attributes covers a majority of potential candidates for upcoming petascale systems. On one hand, we achieve near perfect (linear) strong scaling out to 32,768 cores of IBM BG/L; showing that a system with desirable attributes will allow implicit solvers to strongly scale on high core counts (including petascale systems). On the contrary, we find that the relative tipping point for strong scaling fundamentally differs among current supercomputer systems. To understand the loss of scaling observed on a particular system (Opteron based supercluster) we analyze the performance and demonstrate that such a loss can be associated to an unbalance in a system attribute; specifically compute-node operating system (OS). In particular, PHASTA scales well to high core counts (up to 32,768 cores) during an implicit solve on systems with compute nodes using lightweight kernels (for example, IBM BG/L); however, we show that on a system where the compute node OS is more heavy weight (e.g., one with background processes) a loss in strong scaling is observed relatively at much fewer number of cores (4,096 cores).

Keywords: Strong scaling, massively parallel processing, unstructured and implicit methods, OS jitter

1. Introduction and contributions

PHASTA is a parallel, hierarchic (2nd to 5th order accurate), adaptive, stabilized (finite-element) transient analysis tool for the solution of compressible or incompressible flows. It falls under the realm of computational/numerical methods for solving partial differential equations 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 [8,9,12] and has been applied to flows ranging from validation benchmarks to cases of practical interest. The practical cases of interest not only involve complicated geometries (such as detailed aerospace configurations or human arterial system) but also complex physics (such as fluid turbulence or multi-phase interactions) resulting in discretizations so large that only massively parallel processing (MPP) systems offer the resources required for obtaining desirable solutions in a relevant time frame.

PHASTA has been shown [9,14,38,39] 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 URANSS,

*Corresponding author.

DES, LES, DNS). It has also effectively applied recent anisotropic adaptive algorithms [19,25,26] along with advanced numerical models of flow physics [7,10, 33–36]. Many of its application cases have been sufficiently complex that grid independent results could only be obtained by efficient use of anisotropically adapted unstructured grids or meshes capable of maintaining high quality boundary layer elements [25] and through scalable performance on massively parallel computers [30].

In this paper we do not provide a detailed description of the physical models and mathematical formulations used in PHASTA (which are discussed in detail in the above references) rather we focus our attention on the parallelization of PHASTA's core algorithms for massively parallel processing and present how they scale across a variety of current near petascale systems, including IBM BG/L, IBM BG/P, Cray XT3, and custom Opteron based supercluster.

Many applications have looked into similar issues pertaining to weak or strong scaling using structured or unstructured grids with explicit or implicit solves, for example, see [2,13,16,18,21,23,31,37] and references cited therein. Our contributions are two fold:

1. We demonstrate for the first time to the best of our knowledge that an *unstructured, implicit solver* is able to achieve *strong scaling* out to 32,768 cores on a balanced system like IBM BG/L. This result was achieved by employing a set of distributed data structures that enabled proper use of mesh partitioning schemes and in turn allows for: (a) balancing both of the major work components of an implicit solver, i.e., forming the linear system of equations and finding solution to the formed linear system, without involving any re-distribution of data and (b) balancing communications per core despite the irregular mesh structures that are integral to unstructured, implicit solvers.
2. We observe that systems on which PHASTA does not scale well (such as Opteron based supercluster) there is interference between OS jitter and the amount of *real compute work* that exists between subsequent global allreduce operations in an implicit solve. In contrast, previous studies on OS jitter have largely examined its effects from a pure time delay perspective [1,20,22]. We observe when modest amounts of real work occur (such as 1 million multiply-add operations (MADDS)) between subsequent global allreduce operations, the time spent in allreduce increases significantly due to OS interference and in turn leads to the loss of strong scaling at relatively fewer core counts.

2. Parallel flow solver

2.1. Basics of flow solver

The computational work involved in PHASTA, and other similar implicit methods, mainly consists of two components: (a) formation/assembly of the linearized algebraic system of equations and (b) computation of solution to the linear system of equations. In the first component, entity-level evaluations over the mesh, specifically element-wise integration based on numerical quadrature, are performed to form system of equations, $\mathbf{Ax} = \mathbf{b}$ (where $\mathbf{b}$ is the right-hand side or residual-vector and $\mathbf{A}$ is the left-hand side or linearized tangent matrix of $\mathbf{b}$ with respect to the unknown solution coefficients $\mathbf{x}$ that need to be computed at any given non-linear iteration step). The resulting system is highly sparse but involves large number of unknowns and non-zero entries in an implicit solve. Thus, the second work component of PHASTA finds solution to the formed system of equations by using pre-conditioned iterative solvers suitable for large, sparse systems (e.g., GMRES [24,29]).

More specifically, in PHASTA the Navier–Stokes equations (conservation of mass, momentum and energy) plus any auxiliary equations (as needed for turbulence models or level sets in two-phase flows) are discretized in space and time. Discretization in space is carried out with a stabilized finite element method which interpolates using hierarchic, piecewise polynomials [38,39] that are integrated using Gauss quadrature whereas implicit integration in time is performed using a generalized-α method [11]. The resulting nonlinear algebraic equations are linearized to yield a system of equations which are solved using iterative solvers, e.g., GMRES is applied to the linear system of equations $\mathbf{Ax} = \mathbf{b}$. Note that under the explicit solve (e.g., when generalized-α time integrator is replaced by an explicit scheme such as explicit Runge–Kutta scheme) there is no need for iterative solvers and thus the steps required for parallelization of explicit methods are a subset of the ones required in implicit methods. However, implicit methods are highly desirable for stiff cases with multiple time scales and thus, employed by PHASTA. This brief description of the numerics allows us to focus our attention on current parallel-paradigm for an implicit solver.

2.2. Parallel paradigm

In this section, we discuss the parallelization of the two main work components of PHASTA described in the previous section. Element-level integrals involved in forming the system of equations for finite element methods are well suited for parallel computers as the underlying grid or mesh can be partitioned into balanced parts which can be distributed among processors. Similarly, the resulting system of algebraic equations are distributed among processors and are solved in parallel using iterative methods (which employ sparse matrix-vector **Ap** products). For a mesh with fixed element topology and order, balanced parts within a partition implies that each part contains as close to the average number of elements as possible (see Fig. 1).

For other cases such as ones with mixed element topology or order, weights reflecting the work for every individual element are assigned to enable creation of parts with balanced work load. Good partitioning schemes (such as graph-based ones) not only balance the work load but also minimize the amount of communication required between parts (software libraries such as ParMETIS [15] and Zoltan [40] are commonly used). In case of PHASTA, mesh elements are used as the basis for partitioning and in turn the amount of communication is proportional to the number of degrees-of-freedom (*dofs*), or unknowns in the system of equations, that are shared between parts, i.e., ones that appear on inter-part boundaries as shown in Fig. 2. Note that the computational effort of the equation formation stage involves load proportional to the number of volume (interior or n-dimensional) elements in a part whereas the communication effort is peer-to-peer and depends on surface (boundary or $(n-1)$-dimensional) elements at inter-part boundaries of a part that is shared in segments with multiple neighboring parts [5].

The second work component involves on-part **Ap** products, which is not proportional to the number of elements but to the number of *dofs* on a part (both shared and non-shared, where a non-shared *dof* resides solely on one part and do not appear on inter-part boundaries). Partitioning schemes can be used with *dofs* as the basis for load balance but typically this is not necessary as element balance, with sufficient load per part, and minimization of amount of communications typically results in a reasonable *dof* balance as well.

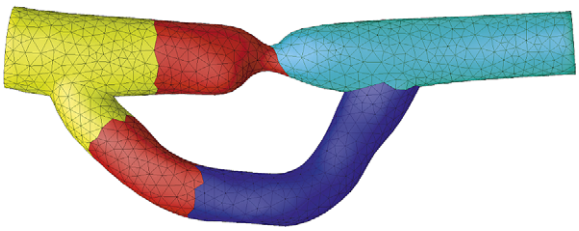


Fig. 1. Partition of coarse mesh of arterial bypass.

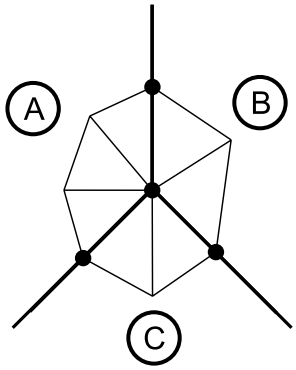


Fig. 2. Solid dots indicate shared *dofs*.

A concept of a *partition-graph* describing the interactions between parts within a partition is used as the kernel for parallelization under PHASTA. Each partition-graph vertex represents a part whereas each partition-graph edge represents interaction between a pair of parts sharing *dofs* that is required to recover complete values for entries associated with shared *dofs*. Since partitioning of a mesh leads to sharing of *dofs* between two or more parts, as shown in Fig. 2, every shared *dof* resides as an image on each part sharing it. Only one image among all images of a shared *dof* is assigned to be the owner thereby making all other images explicitly declared to be non-owners, see Fig. 3. This process insures that the sum total of *dofs* based on owner images over all the parts within a partition is independent of the partitioning and is equal to the number of (unique) *dofs* in the aggregate mesh.

Such a control relationship among images of shared *dofs* allows owner image of each shared *dof* to be in-charge for data accumulation and update to obtain complete values, and in turn bounds communication tasks between only those pairs of parts that involve owner image(s) on one side, i.e., there is no communication task between two parts that contain only non-owner images of *dofs* shared among them (as shown in Fig. 4). Thus, under PHASTA any partition-graph edge connects only those pair of parts that involves

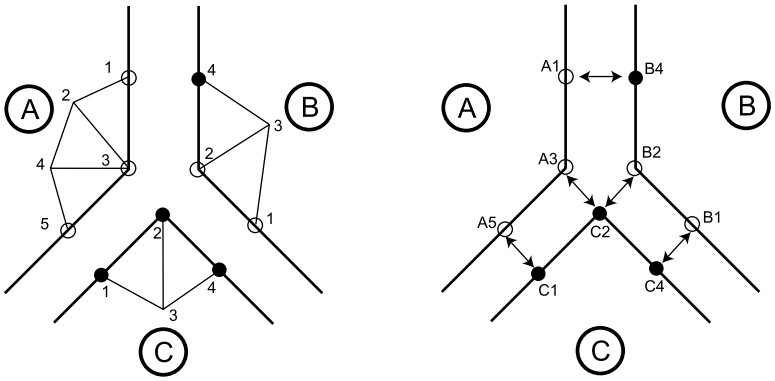


Fig. 3. Solid dot denotes an owner image whereas hollow ones indicate non-owners.

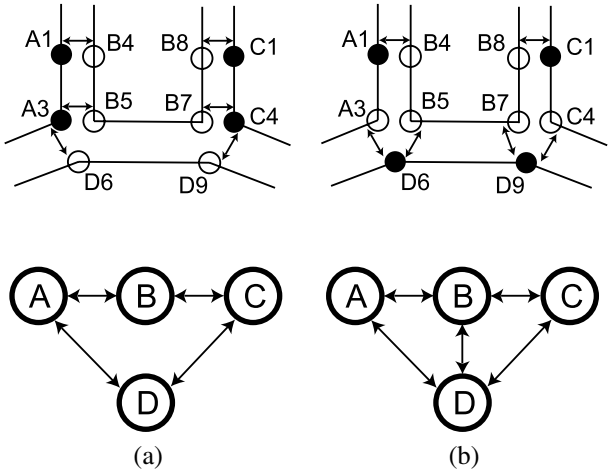


Fig. 4. Schematic of partition-graph: arrows indicate communication task. (a) Parts B and D do not interact (as no shared owner). (b) Pair of parts sharing *dofs* interacts (as owner image(s) involved).

communication task, where a communication task (defined on both parts in a pair) is based on peer-to-peer or point-to-point communication. A communication task involving two connected parts is comprised of all the shared *dofs* among them such that owner images reside on a part at one side (referred to as owner side within a given task) and the corresponding non-owner images on the other part (referred to as non-owner side). Typically for three-dimensional unstructured meshes each part contains on the order of 40 partition-graph edges connecting it with other neighboring parts (where a connected neighboring part is referred as peer). Moreover, the structures specifying the communication tasks are pre-computed and stored for each part while the partition or mesh remains fixed during the analysis phase. The control relationship among images based on ownership is established and maintained by the underlying mesh database library (for more details see [28]).

Typically, with one part per processor (or process), each processor executes a copy of the analysis code to handle the mesh elements and communication tasks associated with its part. In each non-linear iteration step, every processor first performs interpolation and numerical integration of the linearized finite element equations over the elements on its local part to form the associated portion of the residual vector (**b**) and tangent matrix (**A**). Collectively, all processors have the same information as in the serial case but no one processor holds the entire tangent matrix, **A**, nor the residual vector **b**. This bounds the amount of memory required on any processor as the number of rows of **A** and **b** on a given processor depends on the number of (non-shared and shared) *dofs* residing on its local part (which is a fraction of the aggregate number of *dofs*). To understand our progress towards the solution of equations, we introduce the notion of a *com-*

plete value. We will consider a matrix- or vector-entry to be complete when it has exactly the same value in a parallel (or partitioned) case as it will have in a serial one (i.e., unpartitioned case). After numerical integration on local parts, values only in those rows of **b** (in each processor) are complete that are associated with non-shared *dofs* since shared *dofs* residing at inter-part boundaries are individually incomplete (referred as on-processor value) because their contributions are distributed among their images (due to the compact support of basis or shape functions used in finite element methods). Similarly, rows of (sparse) **A** that are associate with non-shared *dofs* contain complete values. On the other hand, rows of **A** that are related to shared *dof* contain incomplete values in those columns (with non-zero entries) that are linked with shared *dofs*. In other words, any non-zero entry in (sparse) **A** is incomplete when both its row and column are associated with a shared *dof*, conversely an entry is complete when either its row or column is associated with a non-shared *dof*.

Once on-processor values in both **A** and **b** are assembled from element-level contributions on a part, pre-computed communication tasks are used to obtain complete values (only) in the residual vector (**b**) within each processor. Though many codes elect to communicate the (incomplete) entries of matrix (**A**) to make them complete and then (re-)distribute the matrix based on rows, PHASTA limits its communication to entries of vectors (such as **b**) and do not perform any re-distribution of data, as the former approach has more significant scaling challenges at high core counts. Complete values in any vector (such as **b** and similar ones) are obtained through two-passes over all the communication tasks. In the first pass the non-owner side of each task sends whereas owner side receives to accumulate into complete values. In the second pass the owner side with complete values sends (to all non-owner sides) whereas each non-owner side receives to update its values with complete ones. To be clear, at this point of the algorithm the right-hand side or residual vector (**b**) is distributed across parts but each entry in it is complete while the left-hand side or tangent matrix (**A**) is distributed but with incomplete values in entries associated with shared *dofs* (as described above).

The second work component (followed by the first step of formation of system of equations) involves finding the solution update vector (**x**) based on iterative solvers that employ $\mathbf{q} = \mathbf{Ap}$ products. Note in case of PHASTA, on-processor **Ap** products results in vector **q** that is partitioned and distributed similar to **b**. And it contains incomplete on-processor values due to its formation using incomplete on-processor values in the entries of **A** (provided vector **p** contains complete values as in vector **b**). Complete values in **q** are then assembled through two-pass communication stage that exploits the distributive property in **Ap** (or any) product (i.e., $(x+y)z = xz+yz$, where x and y are incomplete values in distributed **A** associated to two images of a shared *dof* on different processors and z is a complete value in vector **p** for the same shared *dof*; such a logic can be applied similarly to cases with more than two images of a shared *dof*).

It is important to mention that obtaining complete values in **q** is not the end of the step. Iterative solvers also require computation of global norms of vector **q**, and its dot-product with vectors obtained from prior **Ap** products. Since any vector such as **q** or other similar ones are partitioned and distributed among processors, first an on-processor dot-product is computed (requiring no communication) but then, to obtain a complete dot-product, a sum across all processors is performed through global summation using collective communication (that is of allreduce type). It is important to notice that such a collective communication involves reduction of data globally based on arithmetic operations. Also note that in computing an on-processor dot-product value, only the owner image of each shared *dof* takes active part to correctly account for its contribution in the complete (or global) dot-product. Successive **Ap** products, along with obtaining complete values in resulting vector and its orthonormalization, lead to an orthonormal basis of vectors which are used to find an approximate solution to update vector **x** (e.g., GMRES [24,29]) and mark the end of a non-linear iteration step. See [27] for further details on parallel aspects of PHASTA.

3. Near petascale systems

In this section, we describe each of the three types of supercomputer systems used in this performance study (summarized in Table 1). We begin with IBM Blue Gene architecture, followed by the Cray XT3 and finish with custom Opteron based supercluster (Ranger, TACC).

IBM Blue Gene/L is an ultra large-scale supercomputer system that has grown to 212,992 processors in one specific instance at Lawrence Livermore National Laboratory. The Blue Gene ar-

Table 1
Summary of near petascale systems considered in this study

System type	Processor	Network	RAM	CPCN	CPION	OS
IBM BG/L @ RPI CCNI	PPC 700 MHz	Torus+	12 TB	2	64	IBM BLRTS
IBM BG/P @ ALCF ANL	PPC 850 MHz	Torus+	80 TB	4	256	IBM CNK
Cray XT3 @ PSC	AMD 2.6 GHz	Torus	4 TB	2	188	Cray Catamount
Ranger @ TACC	AMD 2.3 GHz	Full CLOS	123 TB	16	218	Linux CentOS

Notes: CPCN refers to the number of cores per compute-node. CPION represents the number of compute cores per I/O node. + denotes (only in the case of IBM Blue Gene systems) that there are additional networks beside the torus network (see discussion for details). CCNI stands for Computational Center for Nanotechnology Innovations at Rensselaer Polytechnic Institute (RPI), ALCF stands for Argonne Leadership Computing Facility at Argonne National Laboratory (ANL), PSC stands for Pittsburgh Supercomputing Center and TACC for Texas Advanced Computing Center.

chitecture balances the computing power of the processor against the data delivery speed and capacity of the interconnect, which is a 3D torus along with auxiliary networks for global communications, I/O and management. This led designers to create slower, lower-power/energy-efficient compute nodes (only 27.5 kW per 1,024-nodes) consisting of two IBM 32-bit PowerPCs running at only 700 MHz with a peak memory of 1 GB per node. A rack in BG/L system is composed of 1,024 nodes consisting 32 drawers with 32 nodes in each draw. Additionally, there are specialized I/O nodes that perform all file I/O and higher-level OS functionality. Nominally there is one dedicated I/O node for every 16 compute nodes. The BG/L compute node do not support virtual memory, sockets or many of the other standard Unix system interfaces.

IBM Blue Gene/P is an successor of BG/L system where the core count per node is increased from 2 to 4 and the CPU frequency is increased from 700 MHz to 850 MHz. The 3D torus, global and barrier networks of both the systems are fairly similar. Like BG/L, the operating system on BG/P is divided into two parts, one consists of compute node kernel (CNK) which is a minimal OS with simplified memory management and no direct I/O. All I/O is handled through dedicated I/O nodes which run a full OS kernel. Further, its energy efficiency is better than a BG/L system. The two Blue Gene systems used in this study are BGL-CCNI, which is a 32,768 core BG/L system (16 racks) with 12 TB of aggregate memory located at RPI's CCNI [4], and Intrepid which is a 163,840 core (40 racks) BG/P system with 80 TB of aggregate memory located at ANL [3].

Cray XT3 is similar in design to the IBM Blue Gene systems in that it uses a custom-designed interconnect based on a 3D torus topology and a custom reduced OS kernel called Catamount that executes on all compute nodes. All file I/O is routed through 22 dedicated IO processors for the 4,136 cores of the Bigben system (at PSC) used in this performance study. A key design difference as compared to IBM Blue Gene systems is the use of CPU cores with much higher clock rate. In the case of the XT3, 2.6 GHz AMD Opteron processors are used. The relative power usage based on Flops is higher on Cray XT3 systems as compared to IBM Blue Gene systems (see Table 2 in [32]).

Sun Constellation Linux Cluster located at the University of Texas at Austin is a fully custom tightly-coupled cluster based supercomputer system (Ranger, TACC) built using 2.3 GHz AMD quad-core Barcelona processors with 16 cores per node (using 4 quad-cores) with 32 GB of RAM per node. The core count per node is significantly increased as compared to either IBM Blue Gene or Cray XT3 systems. The total system has 62,976 processors with a combined total RAM of 123 TB. Each node connects to a fully non-blocking CLOS InfiniBand switch as opposed to a 3D torus interconnect used by both IBM Blue Gene and Cray XT3 systems. Additionally, there is a significant departure from the use of a customized, vendor specific operating system. On each compute node the CentOS Linux distribution is installed with its own local storage as well as connection to a 1.73 PB global file system managed through 72 I/O 4-way Sun servers (where an I/O core will handle traffic for roughly 218 CPU cores).

A contrast of these systems leads into two distinct categories (such a distinction will become more clear

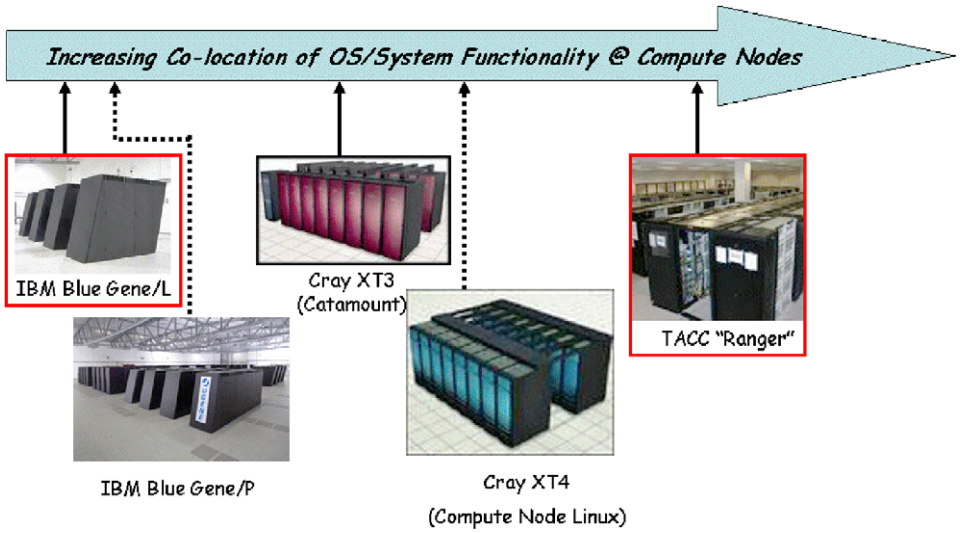


Fig. 5. Spectrum of co-location of OS and system software functionality on compute-nodes. This study focuses on the two extremes which is IBM BG/L (near complete separation) and Ranger, TACC (full OS services and systems software available on each compute node). The dotted lines denote estimations of these systems based on available system literature.

as we present the results on parallel performance of PHASTA). One category includes IBM Blue Gene and Cray XT3 systems which will be shown to demonstrate near perfect scaling, while other includes custom Opteron based supercluster (Ranger, TACC) where parallel performance is shown to degrade (under strong scaling of an implicit solve). The spectrum of these systems is depicted in Fig. 5. Figure 5 coupled with data from Table 1, shows that as the amount of OS functionality placed on compute nodes is increased, the number of CPU cores being served by a single I/O core increases from as few as 64 compute cores per I/O node in the case of IBM BG/L system up to 218 compute cores as in the case of Ranger at TACC. Additionally, we see a marked shift from the use of custom, reduced OS kernels, to stock Linux distributions. This suggests that more and more of the OS heavy lifting, such as access to local and remote file systems, memory management services and even system health services, are executed on compute nodes. The key advantages of such integrated systems approach are both the flexibility and its costs. The increased flexibility allows system administrators the ability to tune OS services at a finer granularity such as controlling the number of file system processes which impacts file system performance. Additionally, users are able to directly access any compute node allocated under their run-job and directly monitor and debug their high-performance codes. On the other hand, in the case of current IBM Blue Gene systems the number and capacity of I/O nodes are fixed for each system (as are other hardware aspects of the system) and debugging complexity is increased due to a lack of a full Unix socket interface along with support for a multi-programmed compute-node environment (e.g., a debugger process cannot simply be started on each compute node and attach itself to a current collection of running MPI tasks). Additionally, in the case of BG/L, simple alignment exceptions are only reported to the RAS logs and so users are required to contact system support staff to get their program exception data as part of an overall debugging and performance tuning process. On the cost side of integrated systems, the compute nodes and I/O nodes are able to leverage readily available hardware and open source software resulting in potentially more computational power per unit dollar spent. However, a relevant question to ask is; does this translate to programs that can scale to a significant fraction or all of the system's available computing power? Our results suggest at least for strong scaling performance of an implicit solver the answer to this question is no. We therefore discuss the primary concerns related to those system attributes that are relevant to strong scaling performance of PHASTA, or other scientific computation codes in general, using implicit techniques:

- *How operating system design (on compute nodes) limits strong scaling?*: Compute nodes on both IBM Blue Gene and Cray XT3 systems run microkernel or lightweight OS for minimum overheads (such as BLRTS or CNK in case of IBM

Blue Gene along with Linux on I/O nodes and Catamount in case of Cray XT3), whereas on Ranger at TACC compute nodes run x86_64 Linux 2.6 kernel from the stock CentOS Linux distribution. When coupled with many cores per node, the theory says that the OS load can be evenly spread over all the CPUs using services like `irq-balance` [1] and no one single task is overly penalized. However, we observe that the amount of OS functionality co-located on compute nodes becomes important for strong scaling in an implicit solve due to fine grain synchronizations dictated by necessary global collective communications (of allreduce type). This attribute is specifically critical for strong scaling on high core counts as for a fixed size problem each core takes a relatively small fraction of aggregate computational load and the effect of system overheads *at any instance in time* becomes more and more significant on higher and higher core counts.

- *How interconnect design limits strong scaling?*: IBM Blue Gene and Cray XT3 systems are based on custom three-dimensional torus interconnect with auxiliary networks on IBM Blue Gene systems dedicated to global communications, I/O and management, which results in a logarithmic growth in message delay as the core count grows. On the other hand, Ranger uses InfiniBand interconnect with a full-CLOS fat-tree topology (managed by 2 core switches). The message delay (as reported) between any two nodes on the (roughly) 62K cores Ranger system is held constant at 2.1 μs across the 7-stage switch fabric vs. a minimum delay of sub-1 μs and a maximum delay of 6 to 7 μs on a BG/L system with 64K cores. Features of interconnect (including bandwidth, latency, capacity and topology) play an important role in both point-to-point and collective communications. However, with our current study, we have not hit these potential (interconnect related) barriers to strong scaling. PHASTA scales near perfectly on 32K cores of IBM BG/L at the full system scale we have available at RPI's CCNI facility (note, we are still conducting our larger scaling studies on BG/P at ANL). Thus, for the reminder of this study we focus on the OS induced scaling limitation, but clearly acknowledge the need to re-examine any interconnect induced scaling limitations that arise in the future.

4. Strong scaling results and analysis

In this section, we present strong scaling performance results and analysis for PHASTA when using an implicit solve on three types of supercomputer systems including IBM Blue Gene, Cray XT3 and Sun Constellation Linux Cluster. The physical problem case considered under this study is a real application and involves blood flow (incompressible) simulation in the cardiovascular system as shown in Fig. 6, specifically it is a subject-specific case of abdominal aortic aneurysm (AAA), which develops complex flows involving transitional/turbulent features (that commonly arise in diseased arteries like ones with aneurysms [17]). The mesh used in this case consists of approximately 105M elements (where M denotes million), which was created using parallel adaptive meshing techniques, lead-

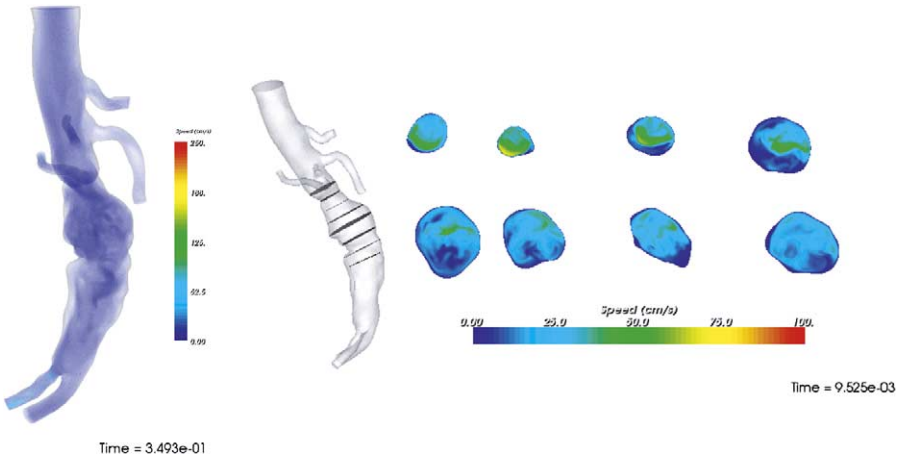


Fig. 6. Flow speed at an instant in a subject-specific aneurysm (left: volume rendered, right: cross-sections).

ing to distributed system of equations with approximately 4×18.5M unknowns in aggregate (note that there are 4 field variables at every one of the 18.5M mesh points comprising pressure and three components of velocity). In aggregate, there are around 275M non-zero blocks in the tangent matrix (**A**) with 13 ($= 4 \times 4 - 3$) entries in each block. 3 entries less from 4×4 is due to the fact that in every block, terms related to coupling of pressure with each of the three velocity components are symmetric. Note that the execution times included are those of the analysis portion where fixed number of time steps and non-linear iterations per step are solved to obtain a converged solution at every step; in this study we considered 5 time steps with 4 non-linear iterations per step. Thus, this study excludes the time spent in pre-processing and post-processing steps (e.g., does not include the time spent in initialization of simulation or checkpoint of solution data).

4.1. Parallel performance results

Figure 7 shows the speedup and Table 2 provides the execution time of PHASTA runs for AAA case over various massively parallel systems. Multiple executions (i.e., O(5) trials) of each case were carried out to collect the timing information, where less than 2% variation in execution times was observed (exceptionally low on Blue Gene systems; about 0.2%). The number of cores utilized in these runs range from 1,024 to 8,192 including three doublings in core counts, i.e., run on 1,024 cores is used as the base for each system. This range was chosen since in this range significant loss in scaling occurs in the case of custom Opteron supercluster at TACC and it also covers the full system in the case of BibBen at PSC. Execution time for AAA case (considering the time for base runs on 1,024 cores) is lowest on Cray XT3 containing cores with fastest clock rate of 2.6 GHz followed by that on Ranger at TACC with core frequency of 2.3 GHz (these runs were done after cores of Ranger were upgraded in June 2008). Highest execution time is observed on IBM BG/L that has slowest core with clock rate of 700 MHz, while that on IBM BG/P is slightly lower than BG/L as it has cores with clock rate of 850 MHz. Although the focus of this study is on parallel performance of PHASTA, it is worth mentioning that PHASTA applies blocking strategy to optimize for single core performance on various systems. Further, work is underway to con-

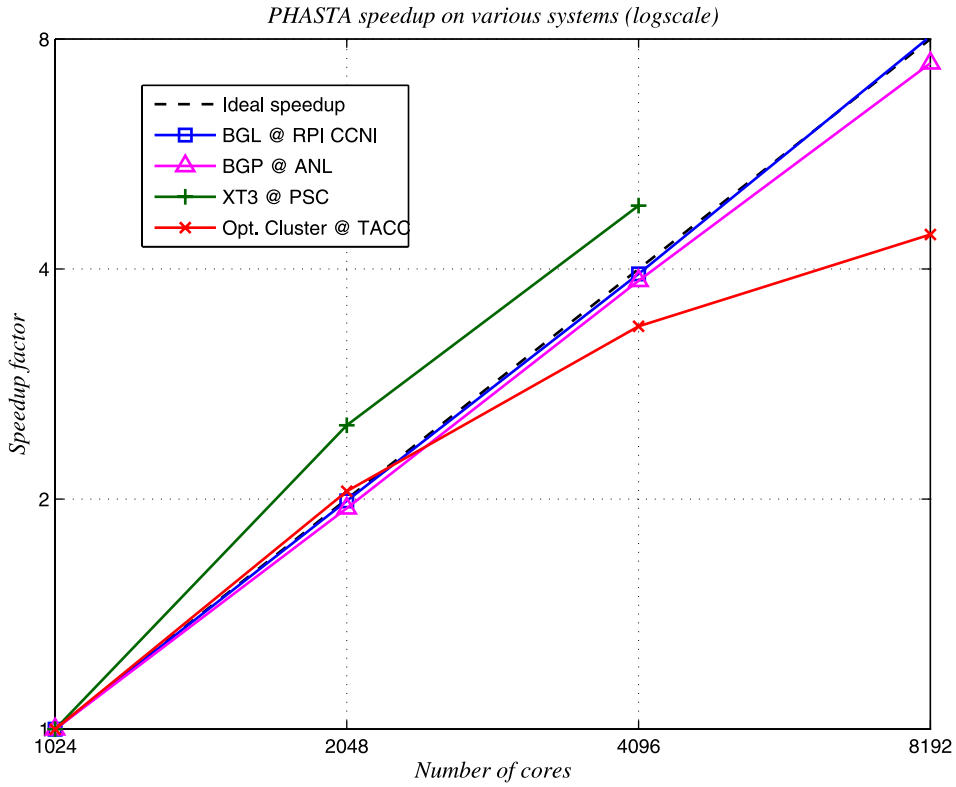


Fig. 7. PHASTA's strong scaling performance in an implicit solve on various massively parallel systems.

Table 2
Execution time (in seconds) of PHASTA runs for AAA case over 1,024 to 8,192 cores on various supercomputer systems

n_{cores}	IBM BG/L @ RPI CCNI	IBM BG/P @ ANL	Cray XT3 @ PSC	Opt. Cluster @ TACC
1,024	1052.4	942.34	406.94	527.87
2,048	529.1	484.37	162.94	257.82
4,096	267.0	244.08	84.04	156.80
8,192	130.5	126.74	–	118.93

duct core-level performance analysis based on hardware performance counters using PAPI, TAU or CrayPat.

Similar speedup on both IBM BG/L and BG/P systems is noticed which is very close to linear (or ideal) performance out to 8,192 cores, see Fig. 7 (these runs were done under virtual-node mode on BG/L and quad-mode on BG/P using all cores within a node for both the IBM Blue Gene systems, and without any explicit double-hummer optimizations [6]). Cray XT3 system demonstrates super-linear scaling (as shown in Fig. 7) with over 20% extra efficiency on both runs over 2,048 and 4,096 cores (again, both cores of a node were used for runs on Cray XT3). This super-linear behavior is likely due to the increase in cache size on higher core counts, also note that the sub-linear scaling from 2,048 to 4,096 cores of Cray XT3 is indicative of the drift towards the loss of strong scaling. Performance on custom Opteron supercluster shows linear scaling on 2,048 cores but degrades on 4,096 and 8,192 cores resulting in parallel efficiency of around 84% and 54% respectively, see Fig. 7 (as other systems, all 16 cores of 4 quad-core Opterons within a node were used in these runs). In summary, parallel efficiency of PHASTA on IBM Blue Gene (both BG/L and BG/P) and Cray XT3 systems is near perfect (in fact super-linear in the case of Cray XT3) but a significant loss is observed in the case of custom Opteron supercluster at 4,096 and 8,192 cores.

Before analyzing the loss of scaling on Ranger and demonstrating scaling limitation due to OS interference, we show that the application is scalable (on a system with desirable attributes). In Fig. 8, we provide strong scaling for the same problem case on BG/L where the total number of cores used range from 512 to 32,768 (32K) cores including 6 doublings in core counts. It shows near perfect strong scaling out to 32,768 cores of IBM BG/L system (which is the full scale of system at CCNI, RPI). Note that the parallel efficiency is either 100%, or slightly above, up to 16,384 cores, and is about 93% on 32,768 cores. It is important to note that within a partition of 32K parts in total, the average number of mesh elements per part is close to 3,200 (about 700 mesh points per part), which is very low implying not only that computational load per core becomes insufficient when compared to communications (due to high surface-to-volume ratios in lightly loaded parts) but also that imbalances among parts is relatively higher (both in terms of communications and computations). This study demonstrates that BG/L system has very desirable attributes of massively parallel systems in the context of implicit scientific computation codes. IBM BG/L is the first system in the Blue Gene series which underscores the potential of upcoming high-performance systems for petascale computation. Efforts are underway to perform these studies on other systems and on larger core counts, for example, Cray XT5 and BG/P. In terms of OS noise, BG/L system is the current "gold standard" for being a virtually noiseless system [1]. Consequently, on near noiseless supercomputer systems, PHASTA is capable to achieve strong scaling for an implicit solve at the full system scale.

4.2. Parallel performance analysis

As previously indicated in Section 3, we associate the scaling loss on Opteron based supercluster due to OS jitter/interference. One could also consider interconnect as the potential reason for the loss of strong scaling (see discussions on two major system attributes that are relevant to strong scaling performance of PHASTA, or other scientific computation codes in general, using implicit techniques) but we show that this is not the case at the tipping point of strong scaling on Ranger. To test the current hypothesis of OS induced scaling limitation, we performed two sets of experiments. These experiments were carried out on 4,096 cores where the tipping of strong scaling is observed on Ranger for the AAA case considered in this study. Note that based on the clock rate of the underlying cores, tipping point of strong scaling on Ranger occurs relatively at fewer number of cores as compared to the scaling on the IBM BG/L system (shown in Fig. 8).

In the first set of experiments we demonstrate that the point-to-point network is not causing the loss of

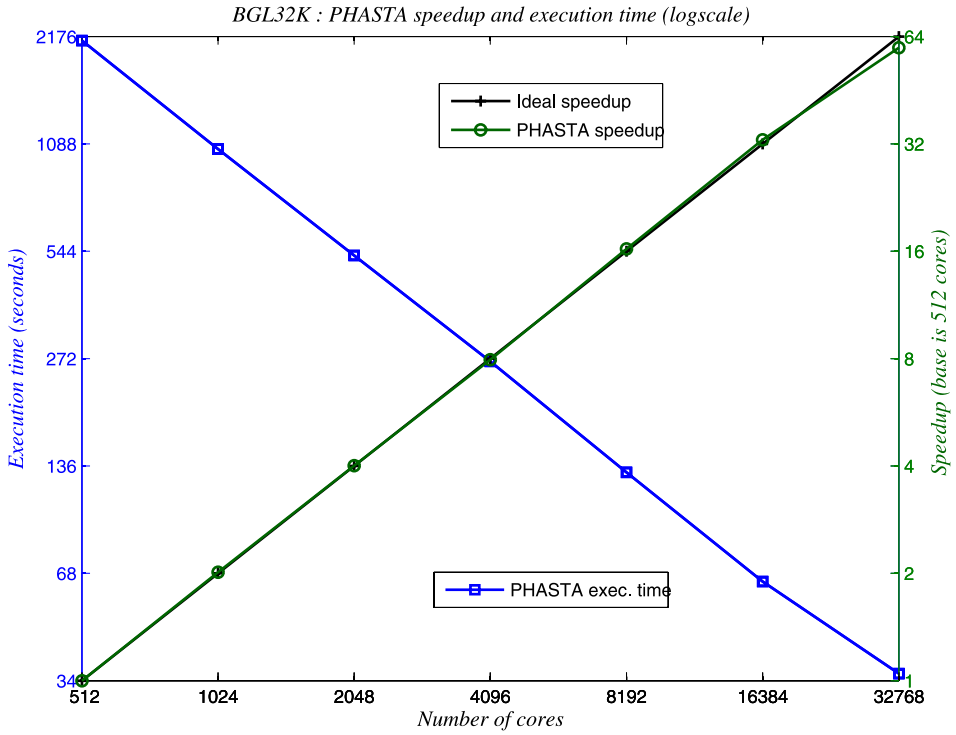
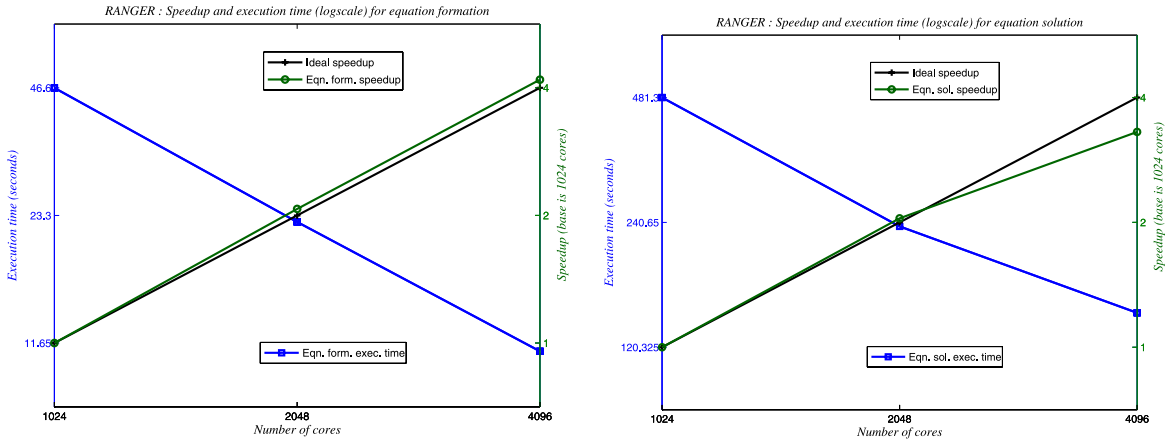


Fig. 8. PHASTA speedup and execution time out to 32,768 cores of IBM BG/L at CCNI, RPI.



(a) (b)

Fig. 9. Strong scaling results on Ranger at TACC for two work components of an implicit solve. (a) Speedup in eqn. formation. (b) Speedup in eqn. solution.

scaling whereas show that collective communications incur significant delays in allreduce operations. This is done by analyzing the performance of both, equation formation and equation solution, work components of PHASTA. The equation formation component strictly executes MPI_Isend and MPI_Irecv operations (along with MPI_Wait and equivalent ones) without any global collective operations. As shown in Fig. 9(a), equation formation scales perfectly (slightly super-linear) out to 4K cores of Ranger, TACC. However, this is not the case for the equation solution component of PHASTA, shown in Fig. 9(b), because it involves global communications and we observe significant delays incurred in allreduce operations. To confirm these

delays in allreduce operations, we store the results of all allreduce operations (i.e., O(10,000) operations) during one execution, and then use these stored results for allreduce operations in a subsequent execution leading to a "re-run" of PHASTA with virtually zero-cost allreduce operations. The outcome of this test shows a "re-gain" in strong scaling of PHASTA at 4K cores of Ranger with 96% parallel efficiency. It is worth mentioning that the recorded allreduce data was retrieved based on an array via a running index leading to negligible cost.

The next question that arises is how to distinguish between the role of a poor performance of allreduce due to the interconnect (or software implementation) with that of OS interference. Our experimental solution to this issue (based on second set of tests) is two fold. First, we added a global barrier operation prior to each allreduce operation in the equation solution component of PHASTA and measure the time spent in allreduce operations as well as in the barriers. All timing data are collected using the `rdtsc` cycle counter instruction which is realized as inline assembly code [1]. The results from this simple exercise, show that the barrier operations absorb the delays (i.e., the time observed in allreduce operations is now spent in the barriers) and in turn the allreduce operations now show the same latency as in isolation without presence of any computational work. This leads to our final experiment and its results are shown in Fig. 10. In this final experiment, we constructed a simple, but vital allreduce performance test. The test is a tight loop consisting of a fixed number of multiply-add operations such as 1M MADDS which is followed by an allreduce operation

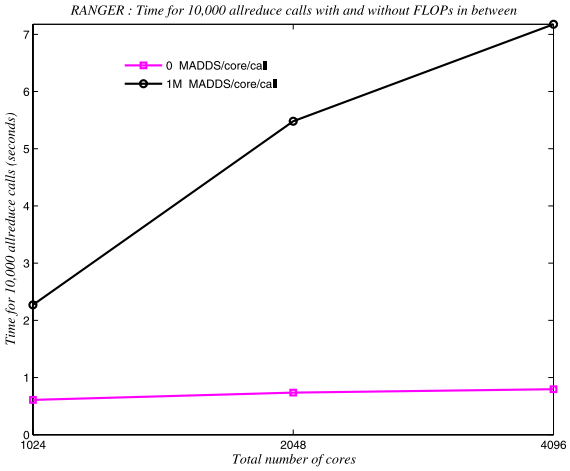


Fig. 10. Total time spent in allreduce operations with and without computational work in between.

in each iteration of the loop (note, 1M MADDS are performed on same scalars to avoid any influence of the memory subsystem). We find that when this fixed modest work is non-zero (i.e., 1M MADDS as compared to 0 MADDS) there is a significant increase in the time spent in the allreduce operations (note that the time spent in allreduce calls is accumulated by only wrapping the allreduce operation within the timer calls). This in our view captures the essence of the OS jitter phenomena (such as intermittent kernel interrupts) and shows its strong degrading impact on a real application which is an additional finding to the previous time-based studies [1,20,22]. As long as there is zero work as denoted in by the line for 0 MADDS in Fig. 10, the number of allreduce operation per OS interrupt is high. However, as fine grain amounts of work are added, there are fewer and fewer full allreduce/work cycles completed per OS interrupt. Additionally, as the number of cores increases, so does the probability that an allreduce operation will incur delay that describes the observed loss of scaling on Ranger (note that the impact of synchronized OS interrupts could be tolerable as shown by benchmarks in [1]). Detailed study on this issue will require use of specialized tools such as KTAU [20], along with significant effort from both the application and system teams to obtain detailed data on kernel interrupts. It is worth mentioning that TACC team recognized OS interference issue on PHASTA and applied improvements (by suppressing certain background processes) that lead to parallel efficiency of 84% on 4,096 cores with upgraded system which was around 77% prior to it.

5. Conclusions

We demonstrated that on *properly balanced* supercomputer systems, unstructured implicit codes are capable of achieving strong scaling on full scale of the system; we showed strong scaling out to 32,768 cores on full IBM BG/L system at CCNI, RPI. On one hand implicit codes based on unstructured meshes are considered challenging but are an important class of methods for petascale systems since they allow efficient consideration of many interesting real-world problems that would be prohibitively expensive to consider with contrasting methods using structured grids (mesh grows too large) and/or explicit methods (time step becomes too small and in turn increases the total number of time steps for a fixed time interval). For such a method based on unstructured and implicit tech-

niques, achieving scalability on a balanced system not only enables the solution of extremely large problem cases but also allows for significant compression in the solution time for a given problem (by a speedup factor directly proportional to the aggregate computing power of the system used). We also showed that an unbalance in a system attribute, specifically compute node operating system (OS), can impact strong scaling of a real application. In particular, systems with lightweight OS kernels on compute node (and relatively much lower overheads) were shown to exhibit excellent strong scaling of implicit schemes (for example, out to 32,768 cores of BG/L) while systems with more complete OS kernels on compute nodes (and relatively higher overheads) were shown to achieve limited strong scaling. This result was observed in the real application code and was verified with a microbenchmark code that showed the degrading impact of OS jitter/interference on parallel performance of collective operations.

Acknowledgments

We would like to acknowledgment funding support from NSF (under PetaApps project, OCI-0749152) and computing resources support from CCNI-RPI (BGL-CCNI), TeraGrid (Ranger at TACC and Bigben at PSC) and ALCF at ANL (Intrepid). We would also like to acknowledge that the results presented in this article made use of software components provided by ACUSIM Software Inc. and Simmetrix Inc.

References

[1] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan and A. Nataraj, Benchmarking the effects of OS interference on extreme-scale parallel machines, *Cluster Comput.* **11** (2008), 3–16.

[2] M. Behr, M. Nicolai and M. Probst, Efficient parallel simulations in support of medical device design, in: *NIC Series*, vol. 38, NIC, Jülich, 2007, pp. 19–26.

[3] Argonne Leadership Computing Facility, ANL, Argonne, IL, available at: http://www.alcf.anl.gov.

[4] Computational Center for Nanotechnology Innovations (CCNI), Rensselaer Technology Park, North Greenbush, New York, available at: http://www.rpi.edu/research/ccni.

[5] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, Reading, MA, 1995.

[6] IBM XL Compiler Development Group, Exploiting the dual FPU in Blue Gene/L, in: *Online IBM Support Documentation*, 2006.

[7] T.J.R. Hughes, L. Mazzei and K.E. Jansen, Large-eddy simulation and the variational multiscale method, *Comput. Vis. Sci.* **3** (2000), 47–59.

[8] K.E. Jansen, Unstructured grid large eddy simulation of flow over an airfoil, in: *Annual Research Briefs*, NASA Ames/CTR Stanford University, 1994, pp. 161–173.

[9] K.E. Jansen, A stabilized finite element method for computing turbulence, *Comput. Method. Appl. M.* **174** (1999), 299–317.

[10] 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, 2002. (40th AIAA Annual Meeting and Exhibit.)

[11] K.E. Jansen, C.H. Whiting and G.M. Hulbert, A generalized-α method for integrating the filtered Navier–Stokes equations with a stabilized finite element method, *Comput. Method. Appl. M.* **190** (1999), 305–319.

[12] K.E. Jansen, Unstructured grid large eddy simulation of wall bounded flow, in: *Annual Research Briefs*, NASA Ames/CTR Stanford University, 1993, pp. 151–156.

[13] P. Jetley, F. Gioachin, C. Mendes, L.V. Kale and T. Quinn, Massively parallel cosmological simulations with ChaNGa, in: *Proc. of IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, IEEE, Miami, FL, April 2008.

[14] A.K. Karanam, K.E. Jansen and C.H. Whiting, Geometry based preprocessor for parallel fluid dynamics simulations using a hierarchical basis, *Eng. Comput.* **24**(1) (2008), 17–26.

[15] G. Karypis and V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, *SIAM Rev.* **41** (1999), 278–300.

[16] D.K. Kaushik, D.E. Keyes and B.F. Smith, High performance parallel implicit CFD, *Parallel Comput.* **27** (2001), 337–362.

[17] J.C. Lasheras, The biomechanics of arterial aneurysms, *Annu. Rev. Fluid Mech.* **39** (2007), 293–319.

[18] D.J. Mavriplis, M.J. Aftosmis and M. Berger, High resolution aerospace applications using the NASA Columbia supercomputer, *Int. J. High Perform. C.* **21**(1) (2007), 106–126.

[19] J. Mueller, O. Sahni, X. Li, K.E. Jansen, M.S. Shephard and C.A. Taylor, Anisotropic adaptive finite element method for modeling blood flow, *Comput. Method. Biomec.* **8**(5) (2005), 295–305.

[20] A. Nataraj, A. Morris, A.D. Malony, M. Sottile and P. Beckman, The ghost in the machine: Observing the effects of kernel operation on parallel application performance, in: *Proc. of the ACM/IEEE Conference on Supercomputing*, ACM/IEEE, Reno, NV, 2007.

[21] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier and T. Goodale, Scientific application performance on candidate petascale platforms, in: *Proc. of IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, IEEE, Long Beach, CA, March 2007.

[22] F. Petrini, D.J. Kerbyson and S. Pakin, The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q, in: *Proc. of the ACM/IEEE Conference on Supercomputing*, ACM/IEEE, Phoenix, AZ, 2003.

[23] D. Porter and P. Woodward, Bursts of stellar turbulence, in: *Proj. in Sc. Comput.*, PSC, 2007.

[24] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7** (1986), 856–869.

[25] O. Sahni, K.E. Jansen, M.S. Shephard, C.A. Taylor and M.W. Beall, Adaptive boundary layer meshing for viscous flow simulations, *Eng. Comput.* **24**(3) (2008), 267–285.

[26] O. Sahni, J. Mueller, K.E. Jansen, M.S. Shephard and C.A. Taylor, Efficient anisotropic adaptive discretization of cardiovascular system, *Comput. Method. Appl. M.* **195**(41–43) (2006), 5634–5655.

[27] O. Sahni, C.H. Whiting, M.S. Shephard and K.E. Jansen, Scalable finite element flow solver for massively parallel computers, in preparation.

[28] E.S. Seol and M.S. Shephard, Efficient distributed mesh data structure for parallel automated adaptive analysis, *Eng. Comput.* **22**(3) (2006), 197–213.

[29] F. Shakib, T.J.R. Hughes and Z. Johan, A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis, *Comput. Method. Appl. M.* **75** (1989), 415–456.

[30] M.S. Shephard, K.E. Jansen, O. Sahni and L.A. Diachin, Parallel adaptive simulations on unstructured meshes, *J. Phys. Conf. Ser.* **78** (2007), 012053.

[31] G. Staffelbach, L.M.Y. Gicquel and T. Poinsot, Highly parallel large eddy simulations of multiburner configurations in industrial gas turbines, in: *Proc. of the Cyprus Intl. Symp. on Complex Effects in LES*, Univ. of Cyprus/CTR at Stanford Univ., Limassol, 2005.

[32] IBM Blue Gene team, Overview of the IBM Blue Gene/P project, *IBM J. Res. Dev.* **52**(1,2) (2008), 199–220.

[33] A.E. Tejada-Martínez and K.E. Jansen, Spatial test filters for dynamic model large-eddy simulation on finite elements, *Commun. Numer. Meth. En.* **19** (2003), 205–213.

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

[35] 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, *Comput. Method. Appl. M.* **194**(9–11) (2005), 1225–1248.

[36] A.E. Tejada-Martínez and K.E. Jansen, A parameter-free dynamic subgrid-scale model for large-eddy simulation, *Comput. Method. Appl. M.* **195** (2006), 2919–2938.

[37] V. Venkatakrishnan, Implicit schemes and parallel computing in unstructured grid CFD, in: *Proc. 26th CFD VKI Lect. Series*, VKI, Rhode-Saint-Genese, 1995.

[38] C.H. Whiting and K.E. Jansen, A stabilized finite element method for the incompressible Navier–Stokes equations using a hierarchical basis, *Int. J. Numer. Meth. Fl.* **35** (2001), 93–116.

[39] C.H. Whiting, K.E. Jansen and S. Dey, Hierarchical basis in stabilized finite element methods for compressible flows, *Comput. Method. Appl. M.* **192** (2003), 5167–5185.

[40] Zoltan, Zoltan: parallel partitioning, load balancing and data-management services, available at: http://www.cs.sandia.gov/zoltan.