Scalable fully implicit finite element flow solver with application to high-fidelity flow control simulations on a realistic wing design

Michel Rasquin[1][3]
mrasquin@alcf.anl.gov

Cameron Smith[2]
smithc11@rpi.edu

Kedar Chitale[2]
chitak2@rpi.edu

Seegyoung Seol[2]
seols@rpi.edu

Benjamin A. Matthews[3]
benjamin.a.matthews@colorado.edu

Jeffrey L. Martin[3]
jeffrey.l.martin@colorado.edu

Onkar Sahni[2]
sahni@rpi.edu

Raymond M. Loy[1]
rloy@alcf.anl.gov

Mark S. Shephard[2]
shephard@rpi.edu

Kenneth E. Jansen[3]
kenneth.jansen@colorado.edu

[1]Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439
[2]Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180
[3]Aerospace Engineering, University of Colorado Boulder, 429 UCB, Boulder, Colorado 80309

ABSTRACT

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. We consider in this work an active flow control application on a realistic wing design that could be leveraged only by a scalable fully implicit unstructured finite element flow solver and the access to high performance computing resources. After a brief introduction, we first describe the main objectives and promises of our active flow control application. We then summarize the main features in the implementation of our massively parallel flow solver, which can address turbulent flow phenomena on any arbitrary complex geometries. Finally, we demonstrate its excellent strong scalability at extreme scale. For that purpose, scaling studies are performed with unstructured meshes of 11 and 92 billion elements on the Argonne Leadership Computing Facility's Blue Gene/Q Mira machine with up to 786,432 cores and 3,145,728 MPI processes.

1. INTRODUCTION

Numerical methods for partial differential equations have reached a level of maturity for a range of physical problems including ones in fluid mechanics, solid mechanics, electromagnetics, biomechanics, to name a few, where application to cases of practical interest is now becoming feasible. These practical cases can be divided into two broad classes: (a) one class that involves challenging and inherently large problems, for example, engineering flows in complicated geometries (such as detailed aeronautical configurations) with complex physics (such as flow instabilities/turbulence or multiphase interactions) or (b) a second class of problems where the results are used in time-critical decisions (such as rapid design). Both of these classes of problems place extreme demands on the numerical solver. When the solution of interest is sensitive to small details of the complex geometry and/or if complex anisotropic solution features develop a wide range of length scales that cannot be determined *a priori*, adaptive unstructured meshes offer tremendous reduction in computational effort. Furthermore, many such solutions are transient in nature and contain time scales whose resolution can be critical to the analysis, strongly motivating the consideration of implicit time integrators whose numerical stability does not depend on the value of the time step. Such problems are otherwise prohibitively expensive to consider with contrasting methods using structured grids (meshes grow too large), explicit techniques (time step becomes too small) and/or procedures with limited scalability (time-to-solution takes too long).
To meet their full potential, the unstructured and implicit techniques must scale for extremely-large practical problems and also yield dramatic compression in time-to-solution(s) for a fixed-size problem. In this study, we present a parallelization paradigm and associated procedures that enables our implicit, unstructured mesh flow solver to achieve strong scalability on a leadership-class computing facility. Although the paradigm and procedures introduced herein are in the context of implicit, non-linear finite element methods, the techniques are amenable to other linear or nonlinear, explicit or implicit numerical methods for partial differential equations, for example, finite volume methods. We demonstrate the effectiveness of the current procedures that yield excellent strong scalability on the Argonne Leadership Computing Facility's Blue Gene/Q Mira system with up to 786,432 cores. The parallel paradigm is applied to an unsteady and turbulent incompressible viscous flow solver with application to active flow control over a swept and tapered wing that is presented in the next section. Cases considered use partitions exceeding 3.1 million parts and up to 92 billion finite elements.

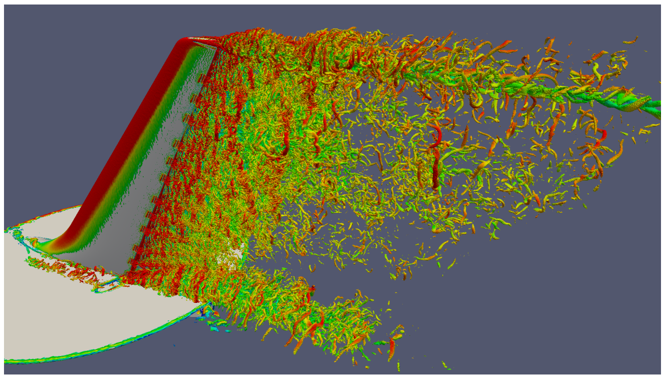
2. FLOW CONTROL APPLICATION

The aerodynamic simulations of this project involve modeling of active flow control to produce large-scale flow changes from micro-scale input, which are the key features of this emerging technology [1, 2]. These large-scale flow changes can range for instance from the re-attachment of separated flow on wing profiles under unfavorable conditions to virtual aerodynamic shaping of lifting surfaces. To name a few examples, experiments have already demonstrated that this micro-scale input can restore and maintain flow attachment and reduce vibrations in wind turbine blades during dynamic pitch, thereby reducing unsteady loads on gearboxes that are currently the prime failure point. In virtual-shape flight control surfaces for aerial vehicles (including commercial airplanes), conventional control surfaces such as flaps and rudder can be augmented or even replaced with active flow control over a wide range of operating conditions, thus improving their lift-to-drag ratio and/or control power.

In this work, we focus our attention to a realistic two component wing assembly with a swept back tapered main element and flap at a high deflection angle of 30° and a Reynolds number of 3.5×10^5. For such a configuration, flow control is known to have the capacity to augment the streamwise momentum near the flap suction peak where separation is typically observed to limit flap effectiveness for high deflection angles. This can result in a significant improvement of the aerodynamic performance in terms of lift and an extension of the flight envelope with higher deflection angles [3].

That said, the objective here is to achieve the same aerodynamic lift but with a smaller wing design so that significant jet-fuel reduction will result both through a drag and weight reduction. For that purpose, we consider an array of twelves synthetic jets located periodically in the spanwise direction of this wing assembly, at the junction between the main element and the flap. Concretely, these synthetic jets are produced by the periodic ejection and suction of fluid through a small orifice induced by the movement of a diaphragm at a high frequency inside a resonance cavity. The complete geometry of these jet cavities is taken into account in our geometric model and the resulting unsteady flow both inside the cavities and in the jet plumes is simulated with high accuracy in terms of grid resolution, temporal resolution and turbulence modeling. In particular, low-fidelity turbulence models such as RANS have shown their limitation for predicting such massively separated flow. Therefore a high-fidelity Delayed Detached Eddy Simulation (DDES) model [4] is used, which is particularly well suited for this application where flow separation is induced by a sharp angle in the geometry.

Our goal with these high-fidelity numerical simulations is to provide a complementary and detailed view of the flow interactions at a much higher Reynolds number than previous simulations, approaching engineering application scales for the first time. Figure 1 shows some of the scales that we could capture with our approach and highlights the vortical structures of the flow which develop downstream the deflected flap. We can concretely observe from this picture the well-known root and tip vortex on such a wing profile, along with the turbulent structures in the wake of the deflected flap and the synthetic jets arising from their respective orifice. The requirements in terms of temporal and grid resolution for accurate prediction of such complex flows highlight the need for efficient parallel discretization methods and HPC resources. We use for that purpose our open-source massively parallel finite element flow solver named PHASTA, as detailed in the next section.


(a) Full span view.

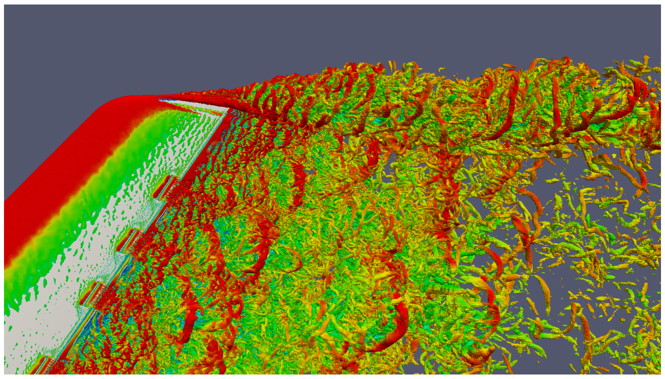

(b) Zoom near the tip.

Figure 1: Flow visualization in the wake of a deflected flap of a realistic two component wing assembly with active flow control through isosurface of instantaneous Q criterion colored by speed.

3. PARALLEL IMPLICIT SOLVER

In the next sections, we review the critical features of our flow solver to allow the discussion of the scaling study presented later in article. More details on the implementation of our flow solver are available in reference [5].

3.1 Basics of implicit solvers

The computational work involved in implicit methods for partial differential equations (e.g., finite element or finite volume 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 (e.g., for finite elements, element-wise integration based on numerical quadrature) are performed to form the system of equations, $\mathbf{Ax} = \mathbf{b}$ (where, $\mathbf{b}$ is the right-hand-side or residual-vector of the weak form and $\mathbf{A}$ is the left-hand-side or linearized tangent-matrix of $\mathbf{b}$ with respect to unknown solution coefficients $\mathbf{x}$). The resulting system is sparse but involves a large number of unknowns. The second work component involves computation of a solution to the formed system of equations. Here, pre-conditioned iterative procedures are suitable for large, sparse systems (e.g., GMRES [6, 7]).

In the current solver, the Navier-Stokes equations are discretized in space and time. Discretization in space is carried out with a stabilized finite element method. This step leads to a weak form of the governing equations, where the solution (and weight function) are first interpolated using hierarchic, piecewise polynomials [8, 9], and followed by the computation of integrals appearing in the weak form using Gauss quadrature. Implicit integration in time is then performed using a generalized-α method [8]. The resulting nonlinear algebraic equations are linearized to yield a system of equations which are solved using iterative procedures, e.g., GMRES is applied to the linear system of equations $\mathbf{Ax} = \mathbf{b}$.

3.2 Parallel paradigm

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 partition is used to indicate the collection of all parts; together all the parts within a partition comprise the aggregate mesh. Consequently, for a mesh with fixed element topology and order, a balanced partition implies that each part contains as close to the average number of mesh entities as possible [5].

One important point to consider during partitioning is that the computational load (in any part) during the system formation stage (i.e., during formation of $\mathbf{A}$ and $\mathbf{b}$) depends on the elements present in the part whereas in the system solution stage it depends on the degrees-of-freedom (*dofs*), or unknowns in the system of equations on that part. For example, in the case of linear finite elements of all the same topology, work involved in equation formation is proportional to the number of mesh elements in the part while during equation solution the work is proportional to the number of mesh nodes in the part since the unknowns are associated with the nodes.

Element-based partitioning is currently used as it is natural for the element-integration/equation-formation stage making it highly scalable. So long as the *dof* balance is also preserved, this partitioning also maintains the scalability in the iterative linear solve step. Typically, element balance (with sufficient load per part) and minimization of amount of communications during partitioning results in a reasonable *dof* balance as well. In element-based partitioning, each element is uniquely assigned to a single part but in turn leads to shared *dofs* or unknowns in the system of equations. Here, a shared *dof* (or unknown) is defined as one that appears on inter-part boundary, see solid dots in Figure 2, whereas a non-shared *dof* resides solely on one part (e.g., interior to a part). Therefore, the communication effort related to shared *dofs* is peer-to-peer, i.e., required only among connected neighboring parts, and grows with the number of *dofs* on the surface/boundary that is shared in segments with one or more neighboring parts. Note that in situations where the number of mesh elements per part is relatively small (in the order of few thousand), significant imbalance in *dofs* can result, since the balance of *dofs* is not explicitly requested. Furthermore, the percentage of shared *dofs* increases in situations where a fixed-size problem is spread over more and more parts as is the case during strong scaling studies and thus, eventually becomes detrimental to scaling [10].

In order to ensure a well balanced partition at extreme scale, our procedure relies on two libraries called PUMI and ParMA, which provide respectively a parallel infrastructure with a general unstructured mesh representation and dynamic load balancing operations to reduce potential *dof* imbalance [11, 12]. Typically, we can maintain the part imbalance in terms of both element and vertices below 15%, which guarantees a good performance and scalability of our flow solver even for complex geometries.

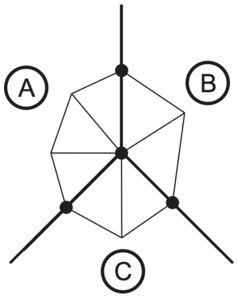


Figure 2: Portion of a mesh on three parts. Solid dots indicate shared *dofs*.

Once the mesh partition is complete, the interaction between neighboring parts in PHASTA is defined based on shared *dofs*, as shown in Figure 2, where 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 [5], see Figure 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 the owner image of each shared *dof* to be "in-charge" for data accumulation and update and in turn limits communications to exist only between owner and non-owners (i.e., non-owner images do not communicate to each other). Furthermore, data exchange is done only for vector entries as this is sufficient to advance the computations, which means that incomplete entries in the matrix $\mathbf{A}$ will be implicitly updated during the iterative solve and construction of Krylov vectors. In summary, all the steps in the numerical analysis phase operate from a physical decomposition of the mesh. Typically, with one part per processor-core (or process), each core executes a copy of the analysis code to handle the computational work and interactions corresponding to its part. Collectively, all the cores have the same information as in

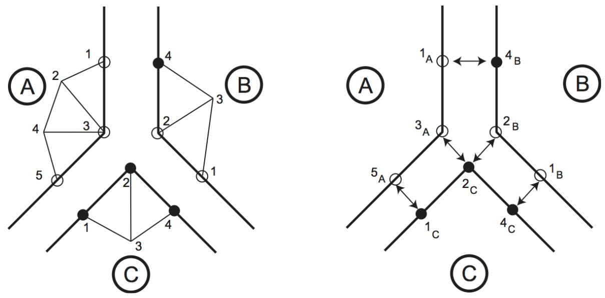


Figure 3: Control relationships are established between multiple images of shared *dofs* (*dofs* are locally numbered on each part). Solid dot denotes an owner image whereas hollow ones indicate non-owners.

the unpartitioned or serial case but no one core holds or has knowledge of the entire tangent matrix, **A**, nor the residual vector **b**. Thus, to be able to process the computations in parallel, interactions between shared *dofs* are completed via communications. The following attributes are therefore possessed by the current approach to maintain strong scalability to 786,432 cores [5]:

- Partitioning is maintained among all compute stages of the solver including equation-formation and equation-solution and thus, no re-distribution of data is required in between.
- The critical step of balancing computations and communications is followed throughout the analysis phase with the help of a distributed partition-graph that describes interaction of every part with its neighbors. Communications are dominated by non-blocking point-to-point operations that are bounded to neighboring parts only whereas blocking collective operations are required to compute norms during iterative solves within the linear algebra routine.
- There is no information required in a global view such as global numbering of *dofs* or equation number; as each part handles elements (and *dofs*) allocated to it in a local manner and exchanges data associated with shared *dofs* through communication steps and structures defined in terms of local images.
- Control relationships are established between images of each shared *dof* residing on multiple parts (i.e., two or more parts). Such a relationship allows owner image of each shared *dof* to be in-charge for data accumulation and update and thus, limits communications to exist only between owner and non-owners (i.e., non-owners do not communicate to each other). Furthermore, exchange of data is done at a part-level (and not for each and every pair of owner and non-owner), and is done only for vector entries (e.g., in **b**, the right-hand side of the global system, and Krylov vectors of the iterative linear solver).

3.3 Parallel implicit solve

In a transient, non-linear problem, multiple linearizations are performed for each time step to obtain a converged non-linear solution at that step before advancing to the next time step. Each linearization includes two primary work components of equation-formation and equation-solution. To allow the advancement of computations in parallel, we employ the communication steps and structures defined above.

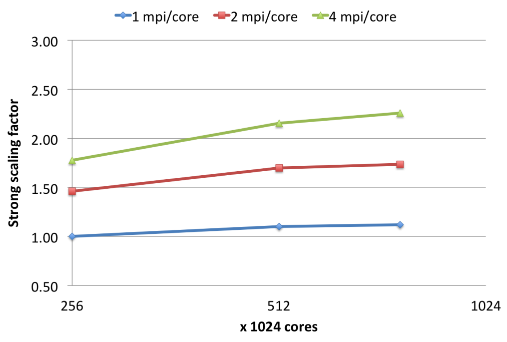
Each processor first performs interpolation and numerical integration over the (interior and boundary) elements on its local part to form the linearized equations, i.e., the tangent matrix (**A**) and the residual vector (**b**). At this point, the entries associated to shared *dofs* in **A** and **b** are incomplete and contain the contributions from elements local to the part. As a following step two communication stages (accumulation from the non-owner to the owner nodes and update from the owner to the non-owner nodes) are carried out to obtain complete values (only) in the residual vector (**b**) on each core. Although one could elect to communicate the on-part entries of the tangent matrix (**A**) to make them complete, our approach does not and limits communications to vector entries only [5]. The next step in an implicit solve involves finding the solution update vector (**x**).

We currently use Krylov iterative solution techniques to find **x** (e.g., GMRES [6, 7]) in two steps. First, a Poisson's system is solved for the pressure with a Conjugate Gradient method and the solution vectors are saved. Then, GMRES is applied to the coupled momentum and continuity equations while simultaneously maintaining the strong satisfaction of the continuity established via the Poisson solve through deflation of the saved vectors. This complex solver has proven highly efficient for incompressible flows. These Krylov techniques employ repeated products of **A** with a series of vectors (say, **p**) to construct a orthonormal basis of vectors used to approximate **x**. In this series, the outcome of any matrix-vector product is another vector **q** (=**Ap**) which is used to derive the subsequent vector in the series while the first vector in the series is derived from the residual vector (**b**) which contains complete values at this point of the solve. Even though **A** contains only on-part values for shared *dofs*, it is still possible to perform the basic kernel of (sparse) matrix-vector product, i.e., **q** = **Ap**, provided vector **p** contains complete values (as is the case for the first vector derived from vector **b**). Note that due to the distributive property of **Ap** product, the resulting vector **q** will contain on-part (incomplete) values. In other words, values from a local **Ap** product can be assembled to obtain complete values in **q**. Therefore, after a local **Ap** product we can apply our two-pass communication strategy to obtain complete values in **q** (provided **p** contained complete values). Before proceeding to the next product in the series, it is important to note that the computation of norms is required to perform orthonormalization. In this step, the norm of vector **q**, and its dot-product with the previous vectors in the series, are computed. To compute a norm or dot-product, first a local dot-product is computed (requiring no communication) but then, to obtain a complete dot-product, a sum across all cores is needed. A collective communication (of allreduce type) is used to carry out the global summation. It is important to point out that while computing a local 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 are carried out along with communications to obtain complete values and to perform orthonormalization. This series of steps leads to an orthonormal basis of vectors which is used to find an approximate update vector **x** and marks the end of a non-linear iteration step [5].

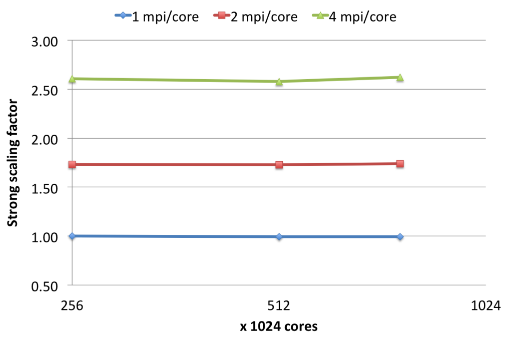
Note that the computation of the eddy viscosity for the DDES turbulence model requires an additional system to be formed and solved, following the same procedure as described in the previous sections. Since there is only 1 *dof* per vertex for this scalar system, the cost for the formation and resolution of this scalar system represents about 20% of the cost of the main system associated with the momentum and continuity equations. For the sake of conciseness, the scalability for the formation and resolution of the main system only is presented in the next section.

4. SCALING RESULTS

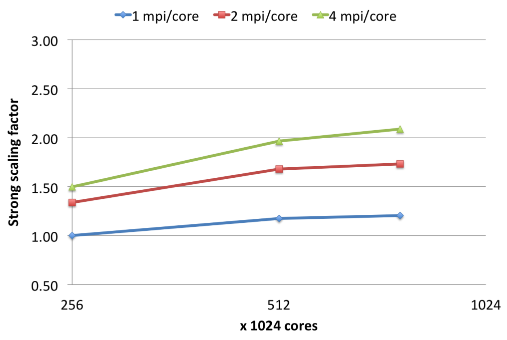
We now present a scaling study performed on the Argonne Leadership Computing Facility's Blue Gene/Q Mira system



(a) Overall (system formation+solve).

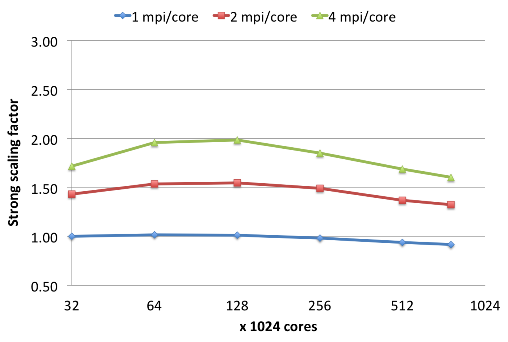


(b) System formation.

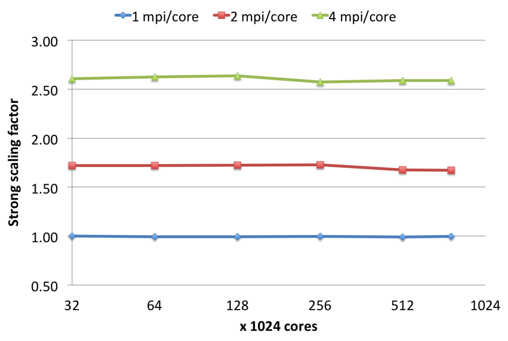


(c) System solve.

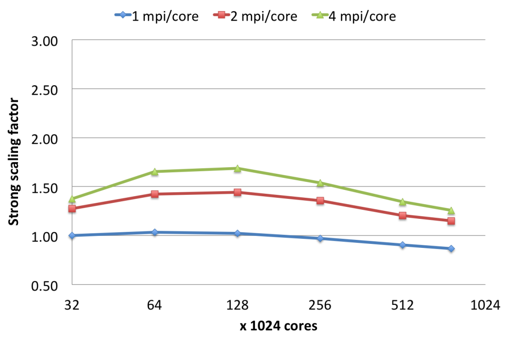
Figure 4: Strong scaling factor for the 92 billion element mesh (base case: 256k cores with 1 MPI per core).



(a) Overall (system formation+solve).



(b) System formation.



(c) System solve.

Figure 5: Strong scaling factor for the 11 billion element mesh (base case: 32k cores with 1 MPI per core).

with up to 786,432 cores. Furthermore, Mira can support even more acceleration by running up to 4 threads per core, leading to 3,145,728 MPI processes at full machine scale. We consider in this scaling study two tetrahedral unstructured meshes with 92 billion and 11 billion elements, resulting in linear systems with approximately 63 billion and 8 billion *dofs*, respectively.

In what follows, we define the scaling factor associated with an execution time t measured on a number of cores nc as

$$sf = (t_{base} \times nc_{base})/(t \times nc), \qquad (1)$$

where t_{base} and nc_{base} are the execution time and the number of cores of the base case on the lower core count with 1 MPI process per core. For 1 MPI process per core, this scaling factor takes an exact value of 1 beside the reference when scaling is perfect (i.e. linear). Since Eq. 1 utilizes the number of cores and not the number of processes, the scaling factor associated with 2 and 4 MPI processes per core represents the acceleration factor when using multiple hardware threads per core with respect to the base simulation.

4.1 92 billion element mesh

For the 92 billion element mesh, 3 core counts are considered with respectively 256k[1], 512k and 768k cores. PHASTA currently requires a number of parts in the partitioned mesh equal to the number of processes, each part being treated by one single MPI process. Therefore, we have partitioned

[1]1k = 1,024.

the 92 billion element mesh from 256k to 3,072k parts in order to run with 1, 2 and 4 MPI processes per core on the considered core counts. The case with 256k cores and 1 MPI process per core is considered as the base case ($sf = 1$) for all the other configurations. Note that the maximum element and vertex imbalance among all the parts is less than 10% for the largest partitioning in 3,072k parts, which is an essential feature of our partitioning tools to ensure the strong scalability of unstructured solvers.

Figure 4(a) shows excellent overall scaling as the number of cores increases from 256k to 768k cores, whether 1, 2 or 4 MPI processes per core are considered. On 768k cores, overall scaling factors equal to 1.12, 1.74 and 2.26 are indeed observed for respectively 1, 2 and 4 MPI processes per core. Breaking down the total computation time into its two main contributions, the system formation and system solve represents respectively about one third and two thirds of the total computation time. As highlighted in Figure 4(b), system formation demonstrates almost perfect scaling ($sf = 1$) with 1 MPI process per core. Besides, the acceleration when using multiple MPI processes per core is also constant for all three core counts. On the other hand, the system solve in Figure 4(c) shows a super linear effect, which is typically due to better utilization of the memory subsystem. As multiple MPI processes per core are used, the acceleration appears to be larger for the system formation in Figure 4(b) than for the system solve in Figure 4(c).

4.2 11 billion element mesh

We present now the performance results on core counts ranging from 32k to 768k cores for a 11 billion element mesh. Considering the decreasing part size across the partition, down to about 3650 elements per part for the 3,072k partition, excellent mesh element and node balance is maintained since both peak element and vertex imbalance is limited to 12% and 14% respectively.

The overall strong scalability of PHASTA is again illustrated in Figure 5(a). Considering as a reference 32k cores with 1 MPI process per core, PHASTA is able to achieve a 0.92 scaling on 768k cores (namely 24 times more cores) with still 1 MPI process per core. We also note that the acceleration factor with 4 MPI processes per core reaches 1.60 at full machine scale and 1.99 on 128k cores. Breaking down again the total computation time in its two main components, the element formation appears again to scale perfectly in Figure 5(b). When using multiple MPI processes per core, the acceleration factor for the system formation is significant and almost constant throughout all core counts. In particular, we observe a similar acceleration factor as for the 92B element mesh, with a value larger than 2.50 with 4 MPI processes per core. However, the scalability of the system solve in Figure 5(c) degrades slightly down to 0.87 on 768k cores with 1 MPI process per core and 1.26 with 4 MPI processes per core. The slight loss of scaling for the system solution observed for the 11B element case and currently investigated is attributed to (i) the increased node imbalance at smaller part sizes, (ii) the subsequent growth of the shared *dofs* on the part boundaries, (iii) the increased communication-to-computation ratio, and (iv) the memory access for the **Ap** product of the Krylov iterative procedure.

5. CONCLUSIONS

We demonstrated how the development of massively parallel computation, along with scalable numerical methods, provides the capacity to perform high-fidelity simulations on a time scale that enables incredible scientific breakthroughs in computational mechanics. In particular, we consider in this work DDES simulations of turbulent flows with application to flow control, which has shown its capacity to improve significantly the aerodynamics performance of realistic wing profiles. We presented for that purpose the procedures, data structures, and algorithms for the exceptional strong scaling of the PHASTA implicit, unstructured finite element solver with 3,145,728 MPI processes on the multi-petaflop Mira system at the Argonne Leadership Computing Facility.
Strong scaling of 92% is achieved on 768k cores for a 11 billion tetrahedral element mesh with respect to a 32k core, one MPI process per core baseline, which corresponds to over four and a half doublings in the number of cores. Using four MPI processes per core, the scaling or acceleration factor reaches 160% on 768k cores for the same 11 billion element mesh. For a 92 billion element mesh, strong scaling of 112% and 226% with respectively one and four MPI processes per core is achieved on 768k cores with respect to a 256k core, one MPI process per core baseline.
A time compression factor of about 38.5 yielded from only a factor of 24 increase in processing cores for the 11 billion element mesh demonstrates the ability to apply the workflow for time critical applications. 92 billion element mesh results demonstrate the workflows ability to efficiently scale on massive problem sizes, which is critical for applications with a large disparity in characteristic length scales.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the use of the resources of the 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, U.S. Department of Energy under grant DE-SC00066117 (FASTMath SciDAC Institute) and in part by the National Science Foundation under grant 0749152. This work also utilized the Janus supercomputer, which is supported by the National Science Foundation (award number CNS-0821794) and the University of Colorado Boulder. The Janus supercomputer is a joint effort of the University of Colorado Boulder, the University of Colorado Denver and the National Center for Atmospheric Research. The solution presented herein made use of the Acusim linear algebra solution library provided by Altair Engineering Inc. and meshing and geometric modeling libraries by Simmetrix Inc. We also acknowledge Daniel Ibanez from RPI for his contribution to the meshing tools used in this work.

7. REFERENCES

[1] A. Glezer and M. Amitay, “Synthetic Jets,” *Annual Review of Fluid Mechanics*, vol. 34, no. 1, pp. 503–529, 2002.

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

[3] M. Amitay and A. Glezer, “Controlled transients of flow reattachment over stalled airfoils,” *International*

Journal of Heat and Fluid Flow, vol. 23, no. 5, pp. 690–699, 2002.

[4] P. R. Spalart, S. Deck, M. Shur, K. Squires, M. K. Strelets, and A. Travin, "A new version of detached-eddy simulation, resistant to ambiguous grid densities," *Theoretical and computational fluid dynamics*, vol. 20, no. 3, pp. 181–195, 2006.

[5] O. Sahni, M. Zhou, M. S. Shephard, and K. E. Jansen, "Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 68:1–68:12, 2009.

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

[7] 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.*, vol. 75, pp. 415–456, 1989.

[8] 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," *Comp. Meth. Appl. Mech. Engng.*, vol. 190, pp. 305–319, 1999.

[9] C. H. Whiting, K. E. Jansen, and S. Dey, "Hierarchical basis for stabilized finite element methods for compressible flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 192, no. 47-48, pp. 5167 – 5185, 2003. Linear model problem;.

[10] M. Zhou, O. Sahni, M. Shephard, K. Devine, and K. Jansen, "Controlling unstructured mesh partitions for massively parallel simulations," *SIAM J. Sci. Comp.*, vol. 32, no. 6, pp. 3201–3227, 2010.

[11] S. Seol, C. W. Smith, D. A. Ibanez, and M. S. Shephard, "A parallel unstructured mesh infrastructure," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pp. 1124–1132, 2012.

[12] M. Zhou, O. Sahni, T. Xie, M. Shephard, and K. Jansen, "Unstructured mesh partition improvement for implicit finite element at extreme scale," *Journal of Supercomputing*, vol. 59, no. 3, pp. 1218 – 28, 2012/03/.