SAND2016-0760C

Asynchronous Many-Task *In Situ* Data Analysis Using Legion

Philippe PÉBAŸ*, Janine C. BENNETT†, David HOLLMAN‡, Sean TREICHLER§,
Patrick S. MCCORMICK¶, Christine M. SWEENEY‖, Hemanth KOLLA**, and Alex AIKEN††
*†‡**Sandia National Laboratories, {*pppebay,†jcbenne,‡dshollm,**hnkolla}@sandia.gov
§††Stanford University, {§sjt,††aiken}@cs.stanford.edu
¶‖Los Alamos National Laboratory, {¶pat,‖cahrens}@lanl.gov

***Abstract*—We explore the use of asynchronous many-task (AMT) programming models for the implementation of *in situ* analysis towards the goal of maximizing programmer productivity and overall performance on next generation platforms. We describe how a broad class of statistics algorithms can be transformed from a traditional SPMD implementation to an AMT implementation, demonstrating with a concrete example: an measurement of descriptive statistics implemented in Legion. Our experiments to quantify the benefit and possible drawbacks of this approach are in progress, and we present some encouraging initial results on the (minimal) impact of the AMT-based approach on code complexity, task scheduling, and application scalability.**

***Index Terms*—Programming Models, Asynchronous Many-Task, Parallel Computing, Computational Statistics**

I. INTRODUCTION

Science and engineering codes are currently facing the daunting task of adapting to rapidly developing extreme-scale computer architectures. Relative to current practice, both the hardware and future algorithms will be characterized by dynamic behavior and lack of uniformity, with architectures that support increased concurrently by a factor of 40,000-400,000 (cf. [1], [2])), yet are constrained by energy and input/output (I/O).

A number of data analysis and visualization (DAV) research efforts are currently focused on mitigating these extreme-scale challenges, e.g. by developing *in situ* (cf. [3]–[5]) and *in transit* (cf. [6]–[8]) frameworks wherein raw simulation output is processed as it is computed, decoupling the analysis from I/O and storing only the analysis results (which are typically several orders of magnitude smaller than raw data).

Other research efforts seek to mitigate the complexities imposed by future architectures via novel programming models. Many high performance computing (HPC) applications have assumed static, homogeneous system performance, with dynamic parallelism requirements stemming solely from the work load, and thus typically follow the communicating sequential processes (CSP) programming model using MPI and MPI+X [9] approaches. However, the procedural and imperative nature of MPI requires application-level management of system performance heterogeneity, fault tolerance, and increasingly complex workflows. Asynchronous many task (AMT) models and *runtime systems*, cf. [10]–[16], a leading alternative to the CSP approach, attempt to mitigate these challenges by allowing the compiler and runtime system to assist with this management of these complexities. Furthermore, AMT models facilitate the expression of all forms of parallelism (pipeline, task, and data). In Section II we discuss these expected benefits in more detail - experiments to quantify them are a major part of our ongoing work.

In Section III, we demonstrate how a broad class of statistics algorithms can be transformed from their CSP formulation to AMT implementation, illustrating that a holistic solution (leveraging both programming models and DAV research) can jointly maximize programmer productivity and code performance on next generation computing platforms. We use the Legion programming model for this work – similar implementations are possible in other AMT models.

Finally, Section IV discusses some of the possible drawbacks that might arise from coupling analysis code with the simulation code in an AMT model. We present some encouraging early results that suggest these costs are small enough that the use of AMT models is viable.

II. BACKGROUND AND BENEFIT OF AMT MODELS

A. The Legion Data-centric AMT Model and Run-time

Legion (cf. [11]) is an AMT model that makes data and data-centric operations first-class programming constructs. A Legion application is decomposed into a task hierarchy, and tasks declare which parts of the application data they will access or update. The Legion runtime is able to reason about data usage of tasks, detect dependencies between tasks, and issue data movement operations as needed, removing these burdens from the developer. All runtime calls in Legion are deferred, allowing the application code to issue tasks with dependencies immediately – the runtime begins the execution of that task only when it is safe to do so (i.e. its data dependencies have been satisfied).

The Legion model separates the functional description of the code (i.e. tasks and the data upon which they operate) from the way in which the code is mapped to a given machine (where to run tasks and place data). Application data is contained in

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Program Manager Lucy Nowell. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC04-94AL85000.

logical regions, which have neither an implied location within the memory hierarchy of the machine nor a fixed physical layout. This allows performance-related transformations (e.g., the replication of read-only data to increase parallelism) to be performed dynamically, and perhaps differently on different machines, without modifying the "machine-agnostic" functional description.

B. Data-centric AMT Approach Benefits

The data-centric AMT model supports composability of the simulation and analysis code bases. Traditional CSP simulations and analysis code bases must be explicitly connected, requiring manual data management and communication. Subsequent changes in how analysis is done will require rewriting parts of the simulation, and vice-versa. In contrast, a data-centric AMT formulation reduces the entanglement of the application and analysis code to simply what data is being shared, and not when, where, or how this sharing occurs. Analysis tasks should therefore be much easier to incorporate into simulation code and can be more easily reused by other codes.

In addition, the AMT approach provides performance portability in the face of increasing I/O cost and variability. Different pieces of code will likely have different spatio-temporal characteristics in terms of compute intensity, degree of parallelism, data access patterns and the task and data inter-dependencies. The decoupling of the functional code from computation and data placement allows an analysis code to easily be tuned for different machines or to be easily implemented as either *in situ* or in transit.

Finally, the AMT approach provides an opportunity for the runtime to make an efficient schedule of simulation and analysis tasks. It may be possible to incorporate the analysis workload into available gaps in the execution of the simulation, whereas this scheduling requires significant programmer effort (and is generally not performance-portable) in the MPI+X models. Dynamic load balancing provided by AMT models has potential to allow for more graceful handling of dynamic variability in analysis tasks and simulation.

III. Parallel Statistics in Legion

A. Parallel Statistics Engines

In earlier work, we described a scalable, parallel statistical analysis library, using SPMD data parallelism with MPI (cf. [17]–[19]). It was designed to mimic the predominant statistical analysis workflow (so that a data analyst would find it natural and intuitive to use) and to be conducive to embarrassingly parallel implementations when possible. In order to meet these requirements, those parts of the analysis which by construction are not embarrassingly parallel were isolated so that design trade-offs be limited to operations explicitly requiring parallel communication. The resulting analysis workflow comprises four disjoint operations, and a given analysis may use all or just a subset. Illustrated in Figure 1, the operations are as follows:

- Learn a model from observations,
- Derive statistics from a model,
- Assess observations with a model, and
- Test a hypothesis.

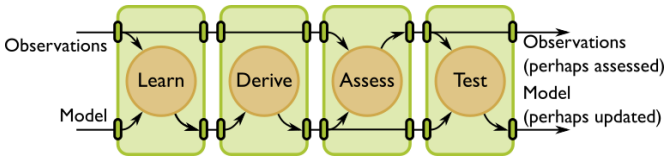

Fig. 1: The four operations of statistical analysis and their interactions with input observations and models. When an operation is not requested, it is eliminated by connecting input to output ports.

From the parallelism standpoint, the Learn operation is a special case of the `map-reduce` pattern [20], while the remaining operations are embarrassingly parallel. Specifically, all local values associated with the same key are merged by the `reduce` function to compute the global *primary model*. In some of the statistical algorithms, namely *moment-based*, it is not necessary to communicate the keys, so sending values alone is unambiguous and the number of such keys is typically very small. This allows for an implementation of the `reduce` function as an `AllGather` MPI collective. In contrast, for *quanta-based* algorithms, it is necessary to communicate keys , so the `reduce` function is implemented using a `Gather-Broadcast` scheme, as illustrated in Figure 2. This can cause problems as network size increases, thereby justifying the need to investigate AMT strategies.

B. Legion Approach and Implementation

We now propose an AMT implementation of the parallel statistics engines. Again, we use the Legion programming model for this work, but expect that our experience would translate to other AMT models as well. The existing decomposition into independent operations and isolation of parts that do not require global communication from the others allows for a natural transition to an AMT model.

In the Legion implementation, data movement is no longer explicitly described (for instance with the dashed red arrow in Figure 2). Instead, a logical region is created to contain the primary and derived models, which we call the *aggregation region*. Sub-tasks launched by a top-level task pick up work on those data segments to which they are assigned, in a similar way to what is done by parallel processes in the SPMD context, at least for the Learn and Assess phases. The main difference between the Legion and SPMD implementations is that a broadcast of the global primary model is not necessary, as data movement operations are handled by the run-time. Instead, the annotation of data requirements for each task provides the run-time with sufficient information to address conflicts and to prevent incomplete or incorrect model updates. This works well for statistical aggregation operations (set unions, number additions and multiplications) as they are commutative, and therefore the primary model is guaranteed to be independent of the order in which tasks report their results.

The Derive operation is performed by the top-level task, and its results, also stored in the aggregation region, are logically

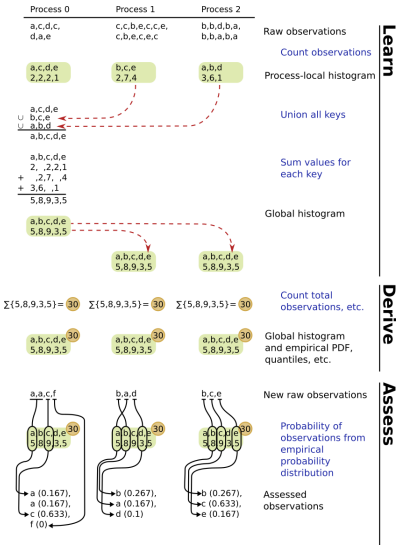

Fig. 2: An MPI implementation of the parallel order statistics; dashed red arrows indicate inter-process communication. In the map-reduce pattern, keys are the raw observations and values are the number of observations.

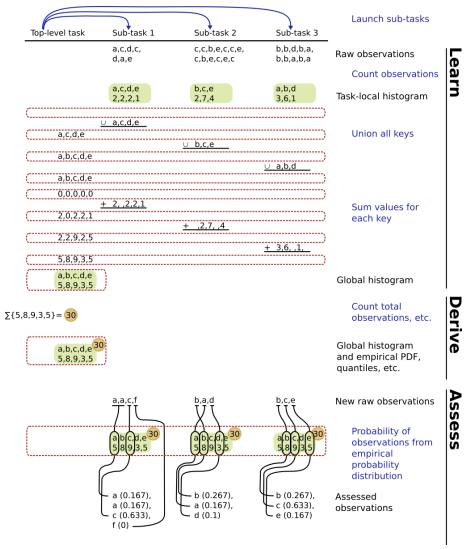

Fig. 3: A simplified example illustrating the operations of the task-based order statistics; solid blue arrows indicate task launches whereas dashed red symbolize the logical aggregation region. Sub-tasks are not obligated to terminate in this order, as both union and addition operators are commutative.

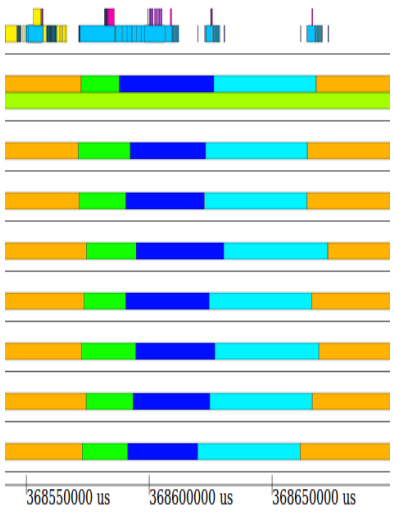

Fig. 4: Task scheduling timeline: Learn sub-tasks are displayed in light blue color.

available to any Assess sub-task launched from the top level. This asynchronous many-task Learn/Derive/Assess scheme is represented in Figure 3 in the case of order statistics. One of the benefits of this new approach is that the approach is valid for both quanta-based and moment-based statistics.

We now provide a high-level description of our Legion implementation of the scheme outlined in Figure 3 for the descriptive statistics use case. The reader interested in a more detailed description is invited to read [21]. In the case of moment-based descriptive statistics, the primary statistical model computes the following double-precision values: sample size, minimum, maximum, mean, and centered M_2, M_3 and M_4 aggregates, using the *online* versions of the process-local update formulas for these quantities, whereas the aggregation with the global model is computed by means of the *pairwise* versions (cf. [22]). In turn, Learn tasks are launched by the top-level task as sub-tasks for each member of the input data partition. Learn tasks data dependencies include reading their subset of the input data and reading and writing the output region where results are to be aggregated.

The Derive task is launched only by the top-level task, for it only needs to read the small set of primary statistics in order to compute the derived statistics which, for descriptive statistics, are the variance, standard deviation, skewness and kurtosis estimators. This operation is typically negligible in computational terms for most statistical analyses. Derive task data dependencies include read-only access to the first field of the logical region used to store the primary statistical model, and write access to the second field for the derived model.

IV. Assessment

Experiments to quantify the expected benefits of the AMT approach for this style of *in situ* analysis are in progress, but such a significant change in programming style is not without potential risks either. As a "sanity check", we have performed some initial experiments focused on these risks. A poor result in any one of these assessments would indicate that the approach might not be viable.

Code complexity: Some additional complexity is necessary when coupling simulation and analysis code *in situ*, but this complexity should be contained and minimized as much as possible.

Heterogeneous task scheduling: An AMT runtime must be able to schedule both simulation and analysis tasks, ideally reducing, but certainly not introducing idle CPU periods.

Impact on scalability: The use of an AMT runtime must not introduce any new scalability bottlenecks into the application.

These initial experiments were conducted by inserting our Legion prototype implementation into a Legion port of MiniAero, a Mantevo mini application (cf. [23]–[25]), as a surrogate for a full-scale scientific application.

A. Code Footprint

The operation of our analysis prototype *in situ* requires that *adaptor code* (code to create and launch the Legion tasks) be added to the MiniAero proxy application. The amount of adaptor code should be kept as small possible.

The first part of MiniAero/Legion modified was its internals that hard-code one index per type of task allowed to be created at run-time: we thus had to create new entries for the Learn, Derive, and Assess tasks, as well has a couple of ancillary tasks. We also modified the user interface of MiniAero/Legion in order to allow for the specification of additional parameters

in the command line arguments to be passed to the *in situ* analysis.

In addition, the main file as well as the top-level task of MiniAero/Legion was modified to create, register, launch, and delete the statistics tasks. It takes twelve lines to specify the aggregation region for primary statistics in the top-level file. Note that another aggregation region is created similarly for the derived statistics. The scheme, illustrated in Figure 3, is actually invoked from inside the time-loop and takes seven lines to initialize statistics, launch Learn and Derive tasks and optionally launch a Dump task. We conclude this brief overview of the adaptor code that is currently needed by indicating that it was only necessary to add a total of *ca.* eighty lines of C++ code in six different files of MiniAero/Legion. See [21] for details.

B. On-Node Parallelism

C. On-Node Parallel Scalability

As an early check of scalability impacts, we measured the scaling behavior of just the Learn task in an on-node setting, performing an *in situ* analyses of the initial mass values of the 3D Sod problem, cf. [26], which comes with the MiniAero distribution. Although what ultimately matters is the scalability of the overall combination of simulation and analysis workloads, the combination can scale no better than any of its individual components. The test platform was a single Linux server, containing two 8-core Xeon E5-2670 2.6GHz CPUs. Scalability was measured by running test cases on different numbers of cores (the MiniAero simulation code constrains these choices to powers of 2), either using the same input size (i.e. strong scaling) or with an input size proportional to the core count (i.e. weak scaling). Each test was run 20 times in order to capture any performance variability. Three separate series of timings were retained, corresponding to the the shortest ("best"), average, ("mean") and longest ("worst") *in-situ* analysis execution speeds. Figure 5 presents weak ($R(p)$) and strong ($S_N(p)$) scaling numbers for each of these series, normalized with respect to its own timing at $p = 1$, obtained with a $128 \times 256 \times 8$ reference grid, scaled up along the x-axis for the weak scaling analysis.

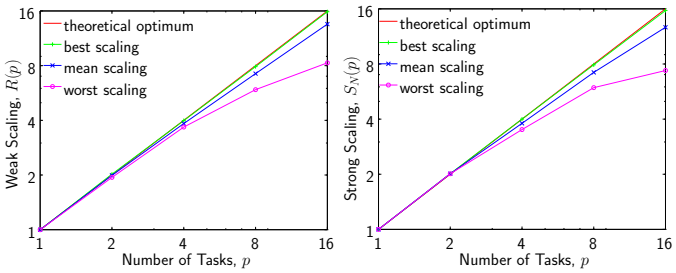

Fig. 5: Scaling of Learn tasks on a single node: left, weak scaling with a $128 \times 256 \times 8$ grid; right, weak scaling with a $128 \times 256 \times 8$ grid per task.

These results reveal optimal on-node scaling, both weak and strong, except when the node is fully subscribed: in this case, with 16 Learn tasks. This is not surprising, as the top-level task is still running as all Learn tasks are launched; therefore, when 16 sub-tasks are executed, at least one core must handle two tasks concurrently. Moreover, background OS perturbations are to be expected and their relative impact increases when all cores of the CPUs are utilized. These results however demonstrate that our approach of using aggregation regions instead of bulk-synchronous inter-thread communication performs as well as possible when running on a single node, and therefore should not impact scalability of the overall application.

V. Conclusion

We see great promise in an approach to *in situ* data analysis and visualization based on the use of AMT programming models. Our case study implementing established statistics algorithms in this model is a work in progress, but initial results are very encouraging. Porting the analysis code from an MPI implementation into a Legion one was straightforward (thanks in part to an initial design that has separated computation from communication for other reasons). The additional code required to connect the analysis code to the main simulation code is well contained, and early performance tests raise no concerns related to task scheduling or impact on scalability. Further experiments will attempt to quantify the benefits of the AMT approach to *in situ* analytics, measuring overall (i.e. simulation and analysis) application performance and scalability in comparison to an MPI-based implementation. We expect these benefits to become increasingly important as scientific simulations and their associated analyses run at extreme scale.

References

[1] R. Stevens *et al.*, "Architectures and technology for extreme scale computing," U. S. Department of Energy, Tech. Rep., 2009. [Online]. Available: http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Arch_tech_grand_challenges_report.pdf

[2] S. Ahern *et al.*, *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*, 2011. [Online]. Available: http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf

[3] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma, "In-situ visualization for large-scale combustion simulations," *IEEE Computer Graphics and Applications*, vol. 30, pp. 45–57, 2010.

[4] J.-M. F. Brad Whitlock and J. S. Meredith, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[5] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen, "The paraview coprocessing library: A scalable, general purpose in situ visualization library," in *Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, October 2011, pp. 89 –96.

[6] V. Vishwanath, M. Hereld, and M. Papka, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, October 2011.

[7] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky, "Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging," in *Proc. of 20th International Symposium on High Performance Distributed Computing (HPDC'11)*, June 2011.

[8] J. C. Bennett *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake Convention Center, Salt Lake City, UT, USA, November 10–16, 2012*, J. Hollingsworth, Ed. IEEE Computer Society Press, 2012, pp. 49:1–49:9. [Online]. Available: http://conferences.computer.org/sc/2012/papers/1000a089.pdf

[9] W. Gropp, "MPI at exascale: Challenges for data structures and algorithms," in *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 3–3. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03770-2_3

[10] A. Buss, H. Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger, "STAPL: standard template adaptive parallel library," in *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, 2010, pp. 1–10.

[11] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: expressing locality and independence with logical regions," in *SC '12: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2389086

[12] T. Heller, H. Kaiser, and K. Iglberger, "Application of the parallex execution model to stencil-based problems," *Comput. Sci.*, vol. 28, pp. 253–261, 2013.

[13] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," in *OOPSLA 1993: 8th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, 1993, pp. 91–108.

[14] J. D. D. S. Germain, S. G. Parker, C. R. Johnson, and J. McCorquodale, "Uintah: a massively parallel problem solving environment," 2000. [Online]. Available: http://content.lib.utah.edu/u?/ir-main,29551

[15] E. A. Luke, "Loci: A deductive framework for graph-based algorithms," in *Computing in Object-Oriented Parallel Environments (3rd ISCOPE'99)*, ser. Lecture Notes in Computer Science (LNCS), S. Matsuoka, R. R. Oldehoeft, and M. Tholburn, Eds. San Francisco, CA, USA: Springer-Verlag (New York), Dec. 1999, vol. 1732, pp. 142–153.

[16] T. Mattson, R. Cledat, Z. Budimlic, V. Cave, S. Chatterjee, B. Seshasayee, R. van der Wijngaart, and V. Sarkar, "OCR: The Open Community Runtime Interface," Tech. Rep., June 2015. [Online]. Available: https://xstack.exascale-tech.com/git/public?p=xstack.git;a=blob;f=ocr/spec/ocr-1.0.0.pdf;hb=HEAD

[17] B. Wylie and J. Baumes, "A unified toolkit for information and scientific visualization," in *Proc. SPIE 7243, Visualization and Data Analysis*, San Jose, CA, U.S.A., Jan. 2009. [Online]. Available: http://doi.org/10.1117/12.805589

[18] P. Pébay, D. Thompson, J. Bennett, and A. Mascarenhas, "Design and performance of a scalable, parallel statistics toolkit," in *Proc. 25^{th} IEEE International Parallel & Distributed Processing Symposium, 12^{th} International Workshop on Parallel and Distributed Scientific and Engineering Computing*, Anchorage, AK, U.S.A., May 2011. [Online]. Available: http://doi.org/10.1109/IPDPS.2011.293

[19] I. Kitware, *The VTK User's Guide, version 5.4*. Kitware, Inc., 2010.

[20] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, Dec. 2004.

[21] P. Pébay and J. Bennett, "An asynchronous many-task implementation of in-situ statistical analysis using Legion," Sandia National Laboratories, Sandia Report SAND2015-10345, Nov. 2015.

[22] P. Pébay, "Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments," Sandia National Laboratories, Sandia Report SAND2008-6212, Sep. 2008. [Online]. Available: http://infoserve.sandia.gov/sand_doc/2008/086212.pdf

[23] K. Franko, T. Fischer, P. Lin, and S. Bova, "CFD for next generation hardware: Experiences with proxy applications," in *Proc. 22^{nd} AIAA Computational Fluid Dynamics Conference*, Dallas, TX, Jun. 2015.

[24] J. Bennett, R. Clay *et al.*, "ASC ATDM Level 2 Milestone #5325: Asynchronous many-task runtime system analysis and assessment for next generation platforms," Sandia National Laboratories, https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/ATDM-AMT-L2-Final-SAND2015-8312.pdf, Sandia Report SAND2015-8312, Sep. 2015.

[25] "Mantevo." [Online]. Available: http://mantevo.org/

[26] G. A. Sod, "A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws," *J. Comput. Phys.*, no. 27, pp. 1–31, 1978.