SAND2017-9775C

A Novel Shard-Based Approach for Asynchronous Many-Task Models for *In Situ* Analysis[*]

P.P. Pébaÿ, G. Borghesi, H. Kolla, J.C. Bennett[†]
Sandia National Laboratories
Livermore, CA, U.S.A.
pppebay,gborghe,hnkolla,jcbenne@sandia.gov

S. Treichler
NVIDIA
Santa Clara, CA, U.S.A.
sean@nvidia.com

ABSTRACT

We present the current status of our work towards a scalable, asynchronous many-task, *in situ* statistical analysis engine using the Legion runtime system, expanding upon earlier work, that was limited to a prototype implementation with a proxy mini-application as a surrogate for a full-scale scientific simulation code. In contrast, we have more recently integrated our *in situ* analysis engines with S3D, a full-size scientific application, and conducted numerical tests therewith on the largest computational platform currently available for DOE science applications. The goal of this article is thus to describe the SPMD-Legion methodology we devised in this context, and compare the data *aggregation* technique deployed herein to the approach taken within our previous work.

KEYWORDS

Programming Models, Asynchronous Many-Task, Parallel Computing, Computational Statistics

ACM Reference format:
P.P. Pébaÿ, G. Borghesi, H. Kolla, J.C. Bennett and S. Treichler. 2017. A Novel Shard-Based Approach for Asynchronous Many-Task Models for *In Situ* Analysis. In *Proceedings of ISAV Workshop, Denver, CO, U.S.A., November 2017 (ISAV'17),* 5 pages.
https://doi.org/10.xxx/xxxxxd

1 INTRODUCTION

1.1 Background

Placement and movement of data are becoming the key-limiting factors of both performance and energy efficiency for next generation high performance computing platforms. Furthermore, the increased quantities of data that the systems are capable of generating, in conjunction with insufficient improvements in the supporting input/output (I/O) infrastructure, are forcing applications away from the off-line post-processing of data towards techniques based on *in situ* analysis and visualization. These challenges are shaping how next-generation, effective, performant, and energy-efficient software will be designed and developed. In particular, these highlight the need for data-centric operations to be fundamental in the reasoning about scientific workflows on extreme-scale architectures.

In our first attempt at tackling this issue, with the stated goal of avoiding the bottlenecks of bulk-synchronous parallel communication, we devised and described in [8]an *asynchronous many-task* (AMT) model, based on a novel concept of *aggregation regions* as surrogates for parallel collective operations, implemented using the Legion system [2]. This preliminary work established that it was straightforward to port our existing MPI, Single-Program, Multiple-Data (SMPD) statistical analysis tool set to this new model, thanks in part to their original design pattern that separated computation from communication [12]. Furthermore, this approach exhibited optimal on-node parallel scaling, thereby taking advantage of the multiplicity of cores on each node. In subsequent work [10], we demonstrated that the additional overhead from the inclusion of the statistics computations *in situ* would not become a bottleneck. Furthermore, one of the noted benefits of the approach presented was that only a small set of well contained code was required to connect the analysis to the main simulation application, in this case the MiniAero computational fluid dynamics (CFD) mini-application [5].

However, given that the findings in these earlier results did not stem from a full-scale scientific code, it remained to be proven whether the nice portability features of our approach would extend to such a complex setting. The current article thus vastly extends our previous work, where the overall performance of the *in situ* system was only evaluated on with mini-applications ran on modestly-sized test platforms. Namely, we use S3D [7], a massively parallel DNS solver developed at Sandia National Laboratories, for which a Legion implementation already exists and is routinely run on the largest production platforms available at DOE.

1.2 Data-centric AMT Approach with Legion

In a typical *in situ* framework, simulation and analysis codes must be explicitly connected, requiring manual data management and communication. Subsequent changes in how analysis is done thus requires rewriting parts of the simulation, and vice-versa. One of the key advantages of a data-centric AMT model is that it naturally supports composability of simulation and analysis code bases, thereby reducing the entanglement of the application and analysis codes to simply *what* data is being shared, and not *when*, *where*,

[*]This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the United States Department of Energy, under the guidance of Dr. Lucy Nowell.
[†]Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

or *how* this sharing is to occur. Analysis tasks should therefore be much easier to incorporate into simulation code, and re-usability by other codes should also be vastly improved. In addition, this 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 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 within either *in situ* or *in transit* frameworks. This approach also provides an opportunity for the run-time to efficiently co-schedule simulation and analysis tasks, allowing for the incorporation of the analysis workload into available gaps in the execution of the simulation, whereas this scheduling requires significant programmer effort in the MPI+X models [6]. Dynamic load balancing provided by AMT models has the potential to allow for more graceful handling of dynamic variability in analysis tasks and simulation.

1.3 The Legion Programming System and Run-Time

Legion is an AMT model making data and data-centric operations first-class programming constructs. A Legion application is organized as a hierarchy of tasks that declare which parts of the application data they will access or update. This model separates the functional description of the code (i.e., tasks and the data upon which they operate) from the way it is mapped to a given machine (i.e., tasks and data placement). Application data is contained in *logical regions*, which have neither an implied location within the memory hierarchy of the machine nor a fixed physical layout.

The Legion run-time leverages these data properties to issue data movement operations as needed, removing this burden from the developer. This run-time system detects pairs of tasks that have a data dependence (i.e., they may access the same data and at least one is making non-commutative modifications to it) and guarantees that the second task in the pair does not execute until it is safe to do so. This technique extracts (dynamic) task parallelism from the application, while preserving programmer-friendly "apparently-sequential" execution semantics. Legion run-time calls are thus deferred as needed, allowing the application code to issue tasks with dependencies immediately. This approach also allows for the dynamic execution of performance-related transformations (e.g., the replication of read-only data to increase parallelism), perhaps differently on different machines, without modifying the "machine-agnostic" functional description. Such run-time transformations also have the potential to alleviate the risks of early optimization as well as the many burdens of late-stage performance analysis that are commonplace in the bulk-synchronous SMPD context. Legion is therefore designed for two classes of users: advanced application developers and domain-specific library authors.

1.4 SPMD Parallel Statistics Engines

Between 2008 and 2013, some of the authors of this report have developed at Sandia National Laboratories a scalable parallel statistical analysis library [12]. These parallel statistics *engines* are a set of C++ classes based on SPMD data parallelism using MPI, which were designed with the dual intent of mimicking the predominant types of data analysis work flows, so that a data analyst using our framework would find it natural and intuitive to use, while being conducive to embarrassingly parallel implementations whenever possible. In order to meet these two overlapping but not exactly congruent design requirements, we isolated those parts of the analysis which by construction are not embarrassingly parallel (due to the mathematics of the statistical analysis itself, not due to our design) so that parallel design trade-offs be limited to those components where embarrassingly parallel implementations are not viable. Specifically, the statistical analysis work flow is split into 3 disjoint operations:

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

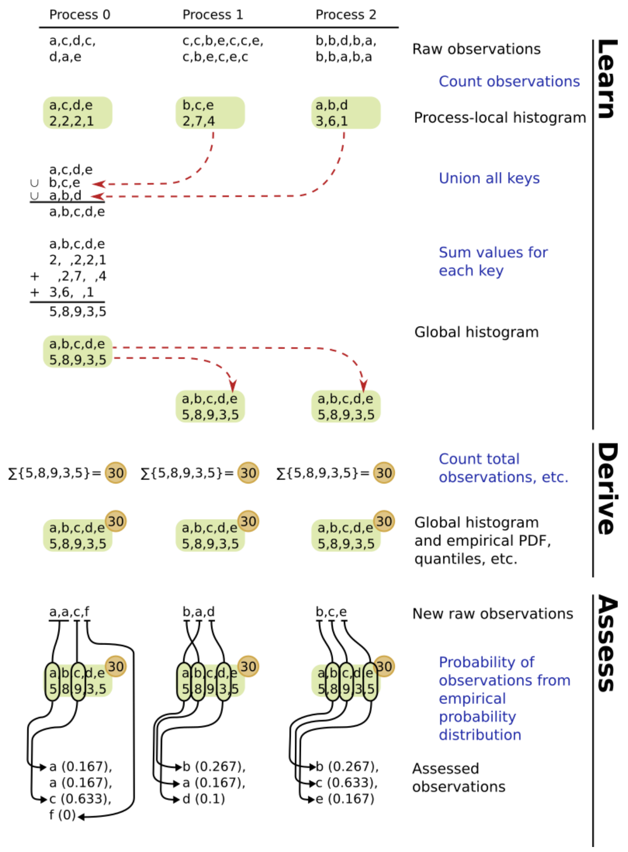
Process 0
Process 1
Process 2
a,c,d,c, d,a,e
c,c,b,e,c,c,e, c,b,e,c,e,c
b,b,d,b,a, b,b,a,b,a
Raw observations
Count observations
a,c,d,e 2,2,2,1
b,c,e 2,7,4
a,b,d 3,6,1
Process-local histogram
a,c,d,e ∪ b,c,e ∪ a,b,d a,b,c,d,e
Union all keys
a,b,c,d,e 2, ,2,2,1 + ,2,7, ,4 + 3,6, ,1 5,8,9,3,5
Sum values for each key
Global histogram
a,b,c,d,e 5,8,9,3,5
Learn
Σ{5,8,9,3,5}= 30
Count total observations, etc.
Global histogram and empirical PDF, quantiles, etc.
Derive
a,a,c,f
b,a,d
b,c,e
New raw observations
Probability of observations from empirical probability distribution
a (0.167), a (0.167), c (0.633), f (0)
b (0.267), a (0.167), d (0.1)
b (0.267), c (0.633), e (0.167)
Assessed observations
Assess

Figure 1: A simplified example illustrating the operations of the parallel order statistics; dashed red arrows indicate inter-process communication. In terms of the map-reduce pattern, keys are the raw observations (represented by letters a, b, c, d, e) and values are the number of observations.

These operations, which need not all be executed, occur in order as illustrated in Figure 1, for a univariate analysis where the Learn operation builds a global histogram, along with derived statistics such as empirical PDF and quantiles. From the parallelism standpoint, this subdivision of the work flow reduces the Learn operation to a special case of the map-reduce pattern [4], while the remaining two are embarrassingly parallel. Specifically, the Learn operation belongs to the map-reduce pattern in that the parallel algorithm computes a set of distributed (key,value) pairs. All local values associated with the same key are then merged by the reduce function to compute the global *primary model*. In some of our statistical algorithms, namely *moment-based*, it is not necessary to communicate the keys, as there is a fixed number of these. In addition, the number of such keys is typically very small, which allowed us to implement the reduce function as an AllGather MPI collective; all subsequent operations are performed locally without further communication. In contrast, for *quanta-based* algorithms, an arbitrary number of key-value pairs must be communicated and different keys may be present on each process. For these cases, the reduce operation uses two steps (Gather, Broadcast) involving only a small number of reduction nodes. Numerous systematic, multiple-parameter scalability studies our SPMD/MPI implementation with up to 10^5 cores have demonstrated that this design allows for optimal scalability of all moment-based engines when data is evenly distributed. We also established that for quanta-based statistics, our design trade-offs allowed for optimal strong and weak scaling when the input data is adequatly quantized. Furthermore, those parallel engines have been successfully applied to the analysis of large-scale turbulent, reacting flow simulation data [3].

2 PARALLEL STATISTICS IN LEGION

After this stamement of the problem, we now present the two main approaches that we have explored to address it.

2.1 The Aggregation Regions Approach

As described in §1.4, our SPMD models can take two different incarnations depending on the considered type of statistics to be computed, with either an AllGather MPI collective to reduce the distributed local models into the global primary model, or a Gather-Broadcast two-step process, for the sake of reducing inter-process communication as the size of local models is not guaranteed to be negligible. In the AMT model however, it is no longer necessary to express data movement explicitly as is done in the SPMD model, for instance with the dashed red arrow in Figure 1. Instead, task parallelism can readily replace the jobs performed by the distributed processes. Instead of specifying movement of data from processes to a number of reduction nodes, it is instead sufficient to define a logical region of data where the equivalent of MPI collectives will be performed. The separation of the logical from the physical representation of data inherent to Legion makes it especially apt at supporting this purely data-driven scheme, which should work, albeit maybe not efficiently, with any generic mapper.

Our proposed approach thus replaces communication with a logical region that contains all model information, both primary and derived, 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 manner to what is done by parallel processes in the SPMD context, at least for the Learn and Assess phases. However, a first noticeable difference is that no communication of data is expressed (even though it will occur under the

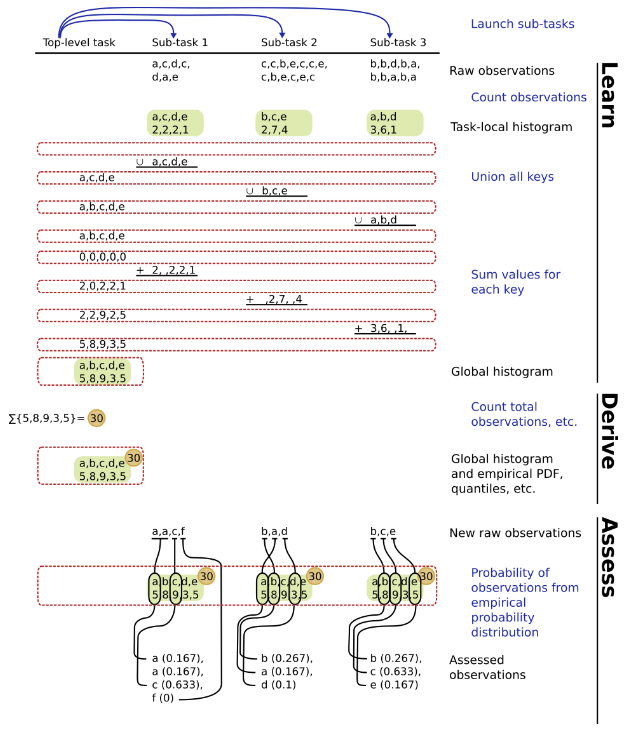

Figure 2: A simplified example illustrating the operations of the task-based order statistics; solid blue arrows indicate task launches, dashed red rectangles represent the logical aggregation region. Sub-tasks of the top-level task are not obligated to complete in this order, as both union and addition operators are commutative.

hood; how this happens is entirely the responsibility of the runtime system). Instead, each tasks aggregates the primary statistics it has computed over its data segment, with those already stored in the aggregation region, changing them in-place. As a result, no broadcast of the global primary model is necessary, for all tasks using it (for instance, a new set of Assess sub-tasks) will directly access the aggregation region to retrieve the values of the statistic. Moreover, the Derive operation is now performed by the top-level task, for its results, also stored in the aggregation region, will be also logically available to any Assess tasks launched from the top level. This asynchronous many-task Learn/Derive/Assess scheme is represented in Figure 2, again for the case of order statistics.

Note that another benefit of this AMT model is that it allows us to unify the two SPMD implementations (AllGather and Gather-Broadcast) into a single paradigm, valid for both quanta-based and moment-based statistics. Also, because all aggregation operations (set unions, number additions and multiplications) are commutative, the learned primary model is guaranteed to be independent of the order in which tasks report their results. However strict locks must be enforced to prevent incomplete model updates: once a task t_i has read the values in the aggregation region, no other task can access this region before t_i has written its own results there.

For example, in the case of *in situ* descriptive statistics, sub-tasks numbered 1, 2, and 3 in Figure 2 are implemented as a method templated on the type of the Legion *accessor* used by the run-time to access data within a physical region. Legion pointers do not directly reference data, but instead name an entry in an index space, and are used when accessing data within accessors for logical regions. The accessor is therefore associated with the field being accessed and the pointer names the row entry. In this way Legion pointers are not tied to memory address spaces or physical instances, but instead can be used to access data for any physical instance of a logical region created with an index space to which the pointer belongs. The computation of the process-local model is done using the *online* versions of the respective update formulas for the quantities above, whereas the aggregation with the global model is computed by means of the *pairwise* versions, cf. [11] for details.

The *in situ* statistics class provides an implementation of the Derive operation, to be called only by the top-level task, for it needs only a single pass over the small set of primary statistics in order to compute the derived statistics, as illustrated in Figure 2. Namely, for descriptive statistics, these are the variance, standard deviation, skewness and kurtosis estimators. This operation typically has negligible cost and is purely task-local. Note that in this case the Legion region requirements are a read-only access to the first field of the logical region used to store the statistical model, and write permission for the second field of the same region. sThe Assess operation also needs only be launched in as many sub-tasks as necessary. In the context of this study whose goal was to assess the validity of our proposed AMT design, there was therefore no point in testing it in addition to the Learn and Derive operations. Scaling results with the MiniAero/Legion implementation are presented and discussed in detail in [9, 10]. These demonstrate optimal on-node scaling, both weak and strong, except when nodes are fully subscribed. However, we observed wide wall clock variations between different runs of the same case, a finding which we could not be easily attributed to a single cause, although several were suggested. Further studies were thus warranted in order to definitely conclude in this regard, hereby confirming the need to explore other programming paradigms.

2.2 The Legion-SPMD Approach

The aggregation region approach provides an elegant solution that abides by a SPMD implementation while still benefiting from any asynchronous execution. However there are a few drawbacks to this approach that could potentially be improved upon: it represents an *all-to-one* model of collectives where all contributions to a collective are funneled to one-point (the aggregation region), and the result is not immediately available in the tasks contributing to the collectives but rather to the parent task.

For instance, in Figure 2 the sub-tasks 1, 2, 3 launched during the Learn phase contribute to the global model, but will not have access to it during their respective lifetimes. The global model is logically available only in the top-level task at the end of the Learn phase. Moreover, any Assess phase tasks that use this global model will require a fresh SPMD launch. This fork-join model may be seen as a "pinching" of the SPMD approach preventing the creation of sub-tasks with perfect SPMD form, that can run through the entire lifetime of the application. Nearly all *in situ* analyses require some form of global communication and most applications themselves might require frequent collectives, which can make the aggregation region approach incur noticeable overhead.

Legion and similar systems provide a sequential abstraction. A program consists of a sequence of tasks, and the system is responsible for finding parallelism that respects the original program order. Because tasks obey a sequential ordering, they need to be analyzed in order. In Legion, this currently happens on a single node, leading to a performance bottleneck. In general, if it is intended to launch N tasks where N is a function of the number of nodes in the machine, the overhead of the run-time system will be $\mathrm{O}(N)$. This bound is inherent given the chosen abstractions and even a distributed analysis of tasks suffers from this problem. For example, StarPU [1] follows an approach where each node independently filters the set of tasks to just those to be executed by that node. But because every node must still consider all tasks, the overall cost of this approach is $\mathrm{O}(N)$ as in Legion.

A proper solution, achieving $\mathrm{O}(1)$ analysis cost per node, requires abstractions that are independent of machine size. Legion already provides this via *index launches*, which describe a set of tasks to be executed in parallel, although the current run-time implementation does not fully take advantage of this. Regent, a compiler for the Legion programming model, can automatically *control replicate* these index launches to produce long-running tasks called *shards* that amortize the cost of this analysis. Critically, the compiler can determine from the sequence of index launches the required patterns of communication for the application on a distributed-memory machine, and can automatically generate the appropriate code for that communication with no additional user input. Regent control replication has been demonstrated to achieve good scalability to 1024 nodes for a variety of structured and unstructured applications. An implementation for the Legion run-time itself is also in progress [13]. The Legion implementation of S3D relies on the "SPMD style" of Legion, which is currently seen as a practical way to achieve scalability, in particular for applications based on main tasks that exist throughout all or most of the run. This style is based on *dynamic collectives*, i.e., non-blocking barriers advanced on a specified number of *arrivals*. Those allow the developer to define reduction objects implemented as *fold* method of a dynamic collective, when programming in the SMPD style of Legion, provided the desired reduction operations be associative and commutative.

The configuration investigated consisted of a temporally evolving jet between *n*-dodecane and diluted air, including 35 chemical species. Input values were chosen to be higher than those usually observed in practical diesel sprays downstream of the evaporation region in order to guarantee the existence of flow regions containing the most reactive composition at the time of ignition. The full simulation was conducted on a computational grid initially with $1200 \times 1500 \times 1000$ points in the transverse, stream-wise, and spanwise directions respectively. The grid spacing was uniform in the latter two directions, whereas in order to reduce the total grid count, a discretization with a uniform central region and stretched edges was used in the transverse direction. This region was periodically enlarged as the jet expanded along the in-homogeneous direction during its temporal evolution: at the end of the simulation, the number of grid nodes in the transverse direction was 2400.

2.3 Results

We evaluated our approach with Titan, the premier DOE computational cluster at the time of writing, with demonstrated peak performance above 10PFLOPS, at 17.59PFLOPS [14]. Because of the hybrid CPU/GPU architecture of Titan, existing science applications that had been historically developed in a CPU-only context had to undergo substantial changes. The version of S3D that ported to Titan thus has greater performance than the CPU-only version.

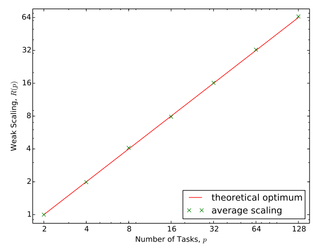

Figure 3: Weak scaling of descriptive Learn tasks on Titan, averaged over an ensemble of 20 runs with a 32^3 grid per task, and a single task per node.

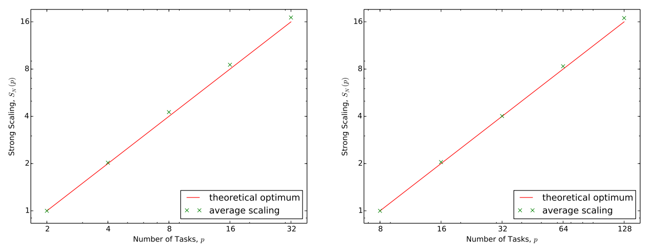

Figure 4: Strong scaling of descriptive Learn tasks on Titan, averaged over an ensemble of 20 runs, with a $64 \times 128 \times 64$ (left) and $256 \times 128 \times 128$ (right) grid and a single task per node.

Figure 3 shows results of a weak scalability experiment for the descriptive Learn task, scaling up to 128 tasks on as many distinct compute nodes of Titan, with a constant load per task, and Figure 4 the results of strong scalability experiments, with two different meshes of constant global size, and a single task per node. These reveal optimal scaling, both weak and strong, across the considered example space. We acknowledge however that we could not approach the Amdahl limit for strong scaling, as it turned out that a problem size allowing to approach it for descriptive statistics Learn tasks would be too large for the main processing computation. In other words, this finding demonstrates that our *in situ* implementation is more scalable than the overall application and therefore does not impact its scalability.

3 CONCLUSION

The results presented herein are promising, and the statistics package is planned for deployment at-scale in upcoming S3D science runs. Planned future work includes integrating the *in situ* kernels developed into a more comprehensive and complex workflow. In particular, we are currently working on integrating *order statistics* as another analysis engine for S3D-Legion, in order to provide scientists with the ability to derive empirical probability density functions, and evince outlying features that deviate from models computed by the already integrated descriptive statistics engine.

REFERENCES

[1] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent, and S. Thibault. 2016. *Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model.* Technical Report RR-8927. INRIA Bordeaux Sud-Ouest.

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

[3] J. Bennett, R. Grout, P. P. Pébaÿ, D. Roe, and D. Thompson. 2009. Numerically Stable, Single-Pass, Parallel Statistics Algorithms. In *Proc. 2009 IEEE International Conference on Cluster Computing.* New Orleans, LA, U.S.A. http://doi.org/10.1109/CLUSTR.2009.5289161

[4] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation.* San Francisco, CA.

[5] K. J. Franko, T. C. Fisher, and P. Lin. 2015. CFD for next generation hardware: Experiences with proxy applications. In *Proc. 22rd AIAA Computational Fluid Dynamics Conference.* Dallas, TX, U.S.A.

[6] W. Gropp. 2009. MPI at Exascale: Challenges for Data Structures and Algorithms. In *Proc. 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface.* Springer-Verlag, 3. https://doi.org/10.1007/978-3-642-03770-2_3

[7] Evatt R Hawkes, Ramanan Sankaran, James C Sutherland, and Jacqueline H Chen. 2005. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. *Journal of Physics: Conference Series* 16, 1 (2005), 65. http://stacks.iop.org/1742-6596/16/i=1/a=009

[8] P. P. Pébaÿ and J. Bennett. 2015. *An Asynchronous Many-Task Implementation of* in situ *Statistical Analysis using Legion.* Sandia Report SAND2015-10345. Sandia National Laboratories.

[9] P. P. Pébaÿ and J. Bennett. 2016. *Scalability of Asynchronous Many-Task* In Situ *Statistical Analysis on Experimental Clusters: Interim Report.* Sandia Report SAND2016-1487. Sandia National Laboratories.

[10] P. P. Pébaÿ, J. Bennett, D. Hollmann, S. Treichler, P. McCormick, C. Sweeney, H. Kolla, and A. Aiken. 2016. Towards Asynchronous Many-Task *in situ* Data Analysis Using Legion. In *Proc. 30th IEEE International Parallel & Distributed Processing Symposium, High Performance Data Analysis and Visualization Workshop.* Chicago, IL, U.S.A.

[11] P. P. Pébaÿ, T. B. Terriberry, H. Kolla, and J. Bennett. 2016. Numerically Stable, Scalable Formulas for Parallel and Online Computation of Higher-Order Multivariate Central Moments with Arbitrary Weights. *Computational Statistics* 31, 4 (2016), 1305–1325. http://dx.doi.org/10.1007/s00180-015-0637-z

[12] P. P. Pébaÿ, D. Thompson, J. Bennett, and A. Mascarenhas. 2011. Design and Performance of a Scalable, Parallel Statistics Toolkit. In *Proc. 25th IEEE International Parallel & Distributed Processing Symposium, 12th International Workshop on Parallel and Distributed Scientific and Engineering Computing.* Anchorage, AK, U.S.A. http://doi.org/10.1109/IPDPS.2011.293

[13] E. Slaughter, W. Lee, S. Treichler, W. Zhang, M. Bauer, G. Shipman, P. McCormick, and A. Aiken. 2017. Control Replication: Compiling Implicit Parallelism to Efficient SPMD with Logical Regions. In Submission.

[14] TOP500. 2013. Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x. (2013). https://www.top500.org/system/177975