Final Technical Report

for

A Unified Data-Driven Approach for Programming In Situ Analysis and Visualization

DOE Award DE-SC0012426

Period of Performance: 9/1/2014 – 8/31/2017

Institution: Stanford University

PI: Prof. Alex Aiken

Executive Summary

The placement and movement of data is becoming the key limiting factor on both performance and energy efficiency of high performance computations. As systems generate more data, it is becoming increasingly difficult to actually move that data elsewhere for post-processing, as the rate of improvements in supporting I/O infrastructure is not keeping pace. Together, these trends are creating a shift in how we think about exascale computations, from a viewpoint that focuses on FLOPS to one that focuses on data and data-centric operations as fundamental to the reasoning about, and optimization of, scientific workflows on extreme-scale architectures.

The overarching goal of our effort was the study of a *unified data-driven* approach for programming applications and *in situ* analysis and visualization. Our work was to understand the interplay between data-centric programming model requirements at extreme-scale and the overall impact of those requirements on the design, capabilities, flexibility, and implementation details for both applications and the supporting *in situ* infrastructure. In this context, we made many improvements to the Legion programming system (one of the leading data-centric models today) and demonstrated in situ analyses on real application codes using these improvements.

Comparison of Goals and Accomplishments

The project was structured into six tasks, with milestones for the each of the tasks for each of the three years. The full project was a joint effort between Los Alamos National Laboratory and Stanford, with the the Stanford team being primarily responsible for the development of the Legion infrastructure and features required to support the project's tasks. The six tasks were (taken from the proposal):

1. *Pipeline abstractions*: Design and develop the data model for structured and unstructured meshes and an accompanying *push-based* data flow execution. An initial algorithm API will be evaluated via a set of small proxy applications.

2. *Execution model*: Explore the use of partitions and mappers to couple applications and algorithms, allowing mesh/grid ghost-cell discrepancies to be automatically handled.

3. *Ray tracing and interactivity*: Design and develop a ray tracer using the Legion programming model. This will require us to explore support for latency-sensitive operations in Legion as well as the design and implementation of acceleration structures.

4. *Data transformation algorithms*: Design and develop topological (merge-tree) and statistical (principal component analysis) data transformation algorithms in Legion along with supporting proxy applications.

5. *Sublinear algorithms*: There are two key features required in Legion to support the proposed sublinear algorithm topics. First support must be added for speculative execution, specifically the ability to precompute results that are very likely, but not guaranteed, to be needed. Secondly, exploration and evaluation of using Legion as a query-based infrastructure will be design and prototyped using proxy applications.

6. *In situ co-design:* Design and develop a set of abstractions for simplifying the interfaces between MPI and Legion. In preparation for working with full applications, small proxies will be used to evaluate and refine this interface.

All of the work planned by the Stanford team in support of these goals was accomplished within the time frame of the project. All of the resulting implementation work was integrated into the main Legion source code repository and released as open source. The primary research results were published in refereed conferences and workshops.

Summary of Research Progress

In this section we summarize the specific tasks that the Stanford team worked on over the course of the project.

Experiments with in situ analysis in Legion. As part of the ongoing work on S3D, a major combustion chemistry simulation, we added an in situ analysis to detect the critical transition points in the reactions. In a typical S3D simulation, the vast majority of the calculation is spent in a low temperature regime where reactions are occurring but the energy release is relatively low. These lower temperature reactions produce species that eventually ignite a high temperature reaction, where the majority of energy in the entire reaction is released. Detecting the transition to high energy ignition is important because, typically, the interesting scientific questions concentrate on analysis of the relatively small part of the simulation spent in high temperature ignition. Thus, there is significant value in having an in situ analysis for this purpose, rather than using the more traditional approach of occasional checkpointing and then rerunning from the last checkpoint once high temperature ignition is observed.

In the case of S3D, the optimal mapping for the simulation is to put as much computation as possible on the GPU(s) attached to a node, which leaves the CPUs available for other purposes. We confirmed that we could add the in situ analysis to detect high temperature ignition on the CPUs and overlap it with both the GPU

computations and the transfer of the data to be analyzed from the GPU to the CPUs; that is, the in situ analysis was essentially free (less than 1% cost) in terms of wall clock time. This is very encouraging, as we expect the same basic framework (critical path computations on the GPU, followed by offloading in situ analysis to the CPUs) to apply in many other applications.

Redesign of the mapping interface. We undertook a major overhaul of the Legion mapping interface. Early experience with the old interface had revealed a number of limitations, and we felt we had learned enough to design a much better and longer lasting interface. The implementation of this effort took longer than anticipated but was completed within the duration of the project and incorporated into the main line Legion code. Old mappers are still supported through a reimplementation of the old interface on top of the new interface. We also ported the S3D mapper to the new mapper API.

The major conceptual change in the new mapping API is that it is constraint-based, meaning that the application now states only what constraints it wants tasks and region instances to satisfy, and the mapper is then free to satisfy those constraints in any way it sees fit. This frees the application from specifying anything that is not performance critical, while also allowing the mapper to incorporate intelligence of its own for situations in which it may be better positioned to judge what is best than the application. The application can still have as little or as much control as desired.

Support for speculation. To support the work on sublinear algorithms we need to allow for speculative execution, where execution of the task graph is allowed to continue past control flow decision points. Once the control flow decision is known, if there was misspeculation then that portion of the graph that should not have been executed can be discarded and execution backed up to the point of the decision and restarted. Work was finished on support for speculation in Legion programs in limited circumstances.

Software framework for in situ dataflow. We participated in the design with other team members of a framework that can handle statically described dataflows involving per-rank analysis of simulation state followed by a hierarchical reduction of the data to a root rank as well as a hierarchical broadcast of the global results back to individual ranks to augment the local data.

New DMA subsystem. A key aspect of the Legion runtime is its ability to move data efficiently from one part of a large machine to another, including managing the layout and any layout conversions required en route. This is especially important for in situ analyses, where decisions are made dynamically about what data to move where. The number of possible distinct cases for data movement is very large (being quadratic in the number of different kinds of memories and layout combinations); in the previous DMA subsystem there were fast paths written by hand for a number of common cases and then a slow, general code path for

everything else. We designed and implemented a much more aggressive DMA system that is substantially faster, handles all layout transformations automatically, and can also plan and execute multi-hop moves (for cases where two memories are not directly connected by a communication channel but instead require multiple moves through intermediate points in the machine). This new system is also designed to interact positively with the new mapping API, as the ability to plan moves allows for more intelligent choices in cases where the application has left the layout or destination of data less than fully constrained. The development of the new DMA subsystem was completed and integrated into the standard Legion code base as part of this project.

Prototype Visualization Subsystem. We successfully prototyped a visualization subsystem that has been used with Soleil-X, a major simulation in Legion being developed as part of the PSAAP-II project at Stanford. The visualization subsystem builds on all of the previous features described, using them to implement a stand-alone visualization library in Legion that can be used with any Legion application. An interesting outcome of the design is that Legion's latency tolerance meant that there was no need to optimize the latency of the visualization, i.e., the Legion implementation tolerates any amount of delay between when the data is produced by the simulation and is consumed by the visualization, so long as the visualization maintains adequate throughput to keep up with the rate at which data is produced. This greatly simplifies the design and obviates the need for extensive optimization of the visualization subsystem.

Legion Bootcamps, 2014-16. The first Legion bootcamp was sufficiently popular that we continued the bootcamps for two additional years. The two day event attracted over 70 people from the DoE, Stanford and local industry each year. We used the same model as the first bootcamp, with a day of lectures on new features, future directions and user experiences, and a second day spent on a hackathon to introduce people to programming in the Legion model. As of 2017 we have transitioned to tutorials that we teach elsewhere; e.g., annually at SuperComputing.

Maintenance of Legion system source code github repository. We continue to make regular updates to the Legion repository. Growth in the number of contributors has necessitated the addition of more source code processes, including adding a regression test suite and standards for examples and merges into the main branch.

Legion teaching/consulting. As part of the long-term effort, we spent significant time teaching people outside the project how to build, program and run Legion applications. We also helped other members of the team debug their Legion applications and regularly fixed a number of bugs in the Legion implementation uncovered by Legion programs written for the project.

Products

The project produced a number of peer-reviewed publications, listed below. In addition, all of the software developed under this project has been released as open source at legion.stanford.edu.

1. Integrating External Resources with a Task-Based Programming Model. Z. Jia, S. Treichler, G. Shipman, M. Bauer, N. Watkins, C. Maltzahn, P. McCormick and A. Aiken. Proceedings of the International Conference on High Performance Computing, Data, and Analytics, December 2017.
2. In situ Visualization with Task-Based Parallelism, A. Heirich, E. Slaughter, M. Papadakis, W. Lee, T. Biedert and A. Aiken. Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization, November 2017.
3. Integrating External Resources with a Task-Based Programming Model. Z. Jia, S. Treichler, G. Shipman, M. Bauer, N. Watkins, C. Maltzahn, P. McCormick and A. Aiken. Proceedings of the International Conference on High Performance Computing, Data, and Analytics, December 2017.
4. Dependent Partitioning. S. Treichler, M. Bauer, R. Sharma, E. Slaughter and A. Aiken. Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications, November 2016.
5. Towards Asynchornous Many-Task In Situ Data Analysis Using Legion, P. Pebay, J. Bennett, D. Hollman, S. Treichler, P. McCormick, C. Sweeney, H. Kolla and A. Aiken. Proceedings of the International Parallel Processing and Distributed Processing Symposium, May 2016