In situ visualization with task-based parallelism

Alan Heirich
SLAC National Accelerator Laboratory
aheirich@stanford.edu

Elliott Slaughter
SLAC National Accelerator Laboratory
slaughter@cs.stanford.edu

Manolis Papadakis
Stanford University
mpapadak@stanford.edu

Wonchan Lee
Stanford University
wonchan@cs.stanford.edu

Tim Biedert
NVIDIA
tbiedert@nvidia.com

Alex Aiken
Stanford University
aiken@cs.stanford.edu

ABSTRACT

This short paper describes an experimental prototype of *in situ* visualization in a task-based parallel programming framework. A set of reusable visualization tasks were composed with an existing simulation. The visualization tasks include a local OpenGL renderer, a parallel image compositor, and a display task. These tasks were added to an existing fluid-particle-radiation simulation and weak scaling tests were run on up to 512 nodes of the Piz Daint supercomputer. Benchmarks showed that the visualization components scaled and did not reduce the simulation throughput. The compositor latency increased logarithmically with increasing node count.

KEYWORDS

in situ visualization, exascale, image compositor, task-based, Legion, sort-last

ACM Reference Format:
Alan Heirich, Elliott Slaughter, Manolis Papadakis, Wonchan Lee, Tim Biedert, and Alex Aiken. 2017. *In situ* visualization with task-based parallelism. In *Proceedings of In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, Denver, CO, USA, November 12–17, 2017 (ISAV'17),* 5 pages.
https://doi.org/10.1145/3144769.3144771

1 INTRODUCTION

The dominant scientific visualization paradigm is in transition from dedicated visualization clusters to *in situ* visualization on the largest exascale systems [7, 10]. Scalable visualization subsystems are based on sort-last graphics architectures and effective MPI-based implementations of sort-last have been developed [9, 11].

Task-based parallelism is an alternative to the message passing paradigm of parallel computing. The current paper describes experimental support for *in situ* visualization in one task-based framework, *Legion* [1]. This is accomplished by composing an existing fluid-particle-radiation simulation [5] with new tasks for rendering, compositing, and displaying imagery. The modified simulation was run on up to 512 nodes of the Piz Daint supercomputer. Results showed that the throughput of the modified simulation was unimpeded by the visualization tasks and that the compositor tasks scaled logarithmically.

The contribution of this paper is a model of scalable visualization in a task-based framework. The rest of this section describes the Legion framework and the simulation. Section 2 describes the visualization tasks that were composed with the simulation. Section 3 describes the experimental results, and Section 4 concludes with a discussion of future near term work.

1.1 Related work

Moreland provided a solution to the compositor problem for MPI based applications [11]. Biedert et al report on a task-based compositor for parallel volume rendering using OSPRay and HPX [4]. They segmented the problem in image space and performed back-to-front compositing, overlapping compositing with rendering.

1.2 Task-based Programming

Task-based programming is an alternative to the message-passing paradigm represented by MPI. In task-based programming models the programmer defines (parallel) tasks and also provides information about data dependencies between tasks. Armed with this information a runtime system can schedule tasks and data automatically to maximize parallelism and throughput. A programmer is relieved of the need to explicitly handle parallelism, synchronization and communication. Because tasks and dependencies are machine independent the programmer also has considerable flexibility in how a program is mapped on to a particular machine.

Task-based programming affords compositionality in that different task-based programs or modules can be easily composed. This paper reports an experiment of combining a visualization library with an existing scientific simulation. The visualization library is reusable by other future programs.

Legion is a framework for task-based parallelism with sequential semantics[1, 2, 12–17]. Legion programs are sets of tasks that operate on collections of data. Programmers define a sequence of tasks with explicit privileges describing the data used by each task. A runtime system derives the dependencies among tasks and automatically schedules task execution and data transfers to maximize system throughput while preserving the sequential semantics.

Application data is contained in *Logical Regions* that are analogous to relations in a relational data base. A Logical Region is based on a coordinate system called an *Index Space* that may be regular or irregular.

Logical Regions have neither an implied location within the memory hierarchy of the machine nor a fixed physical layout. Instead the runtime instantiates one or more *Physical Regions* containing the data when and where they are needed for task execution.

Legion provides two levels of programming: an API callable from C++, and a more concise high-level language called Regent [13]. Both levels use the same runtime system and have similar performance. The visualization subsystem was prototyped using both of these levels. The measurements presented in this paper are from the Regent implementation.

1.3 Simulation

Soleil-X is a turbulence/particle/radiation solver written in the Liszt-Ebb Domain Specific Language for execution with the Legion runtime [3, 5]. The simulation is an initial value problem consisting of a regular grid for solving the compressible Navier-Stokes equations, a set of particles that are advected by the fluid velocity field, and a radiation model.

Figure 1 shows an early frame from the simulation. The visualization shows the fluid velocity field with particles advected in the flow. Color coding is used to show the radiated quantities (e.g. temperature).

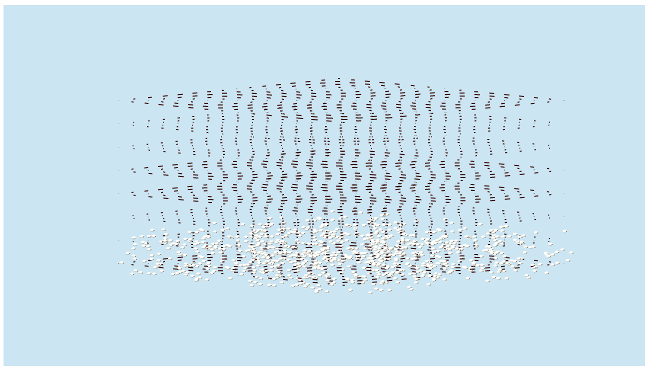

Figure 1: Simulation after 8 time steps showing velocity vectors and particles color coded according to temperature.

2 VISUALIZATION SUBSYSTEM

The visualization subsystem is implemented by three tasks that are composed with an existing Legion simulation. These tasks render imagery, composite the result, and display it.

A simulation proceeds through a number of time steps, issuing simulation tasks at each step. On some time steps the visualization subsystem is invoked in the form of two new tasks: Render and Reduce. These tasks work together to produce a global snapshot of system state in the form of a high resolution image. This image can be mapped to display devices using a third new task: Display.

The simulation tasks run as a set of OpenMP processes across 8 of the cores. The remaining cores are used for system processes. Since the visualization tasks do not run under OpenMP they also run on one of the remaining cores. This makes it possible to ensure that visualization is overlapped with simulation. This is accomplished by writing a custom mapping function that gives the render task a separate copy of the simulation data. This breaks a write-after-read dependence and effectively implements double-buffering but without requiring any application-level logic.

The goals of the subsystem are to be reusable and to scale. Render tasks are embarassingly parallel. Since each Render task is local to a node it generates a fixed size image.[1]

The Reduce tasks run in parallel to collectively composite the images that were just rendered. Their complexity depends on the algebraic properties of the compositing operator which may be any of the standard OpenGL blend functions, blend equations, or depth functions. For the most common compositing operators the task complexity is logarithmic, and for all other cases it is at most linear in the number of nodes.

2.1 Render

The goal of the Render task is to produce at each node a rendered image of the simulation data at that node. The image is in the form of a Legion logical region containing pixels with fields R,G,B,A,Z. The logical region has a 3D index space organized in width, height, layer where each layer corresponds to one node. Each node renders an image of width × height pixels to its corresponding layer of the logical region.

The Render task is application-specific in that every application will have different rendering requirements. Figure 1 shows the output of the Render task. It visualizes a vector field containing a set of particles. In a general purpose visualization package the Render task would be implemented by the package.

The Render task was implemented using hardware accelerated OpenGL with EGL as described in [8]. After rendering the task reads back the color buffer and the depth buffer (using glReadPixels) and then copies the values from both buffers into the logical region.

2.2 Reduce

The goal of the Reduce task is to collectively composite all of the rendered images into a common display image. The visualization subsystem supports all of the OpenGL depth functions, blend functions and blend equations. These functions fall into three categories: *commutative-associative* functions include all depth functions; *noncommutative-associative* functions include the most popular blend functions; *noncommutative-nonassociative* functions include some less popular blend functions.

Each of these categories requires a different implementation structure: commutative-associative reduction is implemented in a tree with unconstrained leaf ordering; noncommutative-associative reduction in a tree with specified leaf ordering; and noncommutative-nonassociative in a serial chain. The tree takes $O(\log n)$ steps while the chain takes $O(n)$.

A full C++ API implementation was developed and tested. The reduction operation performs a brute-force pixel-wise composite of two images to produce a result image. The actual compositing operations were implemented in C++ via a shim where the C++ code was copied from the C++ API implementation. The resulting

[1]Section 4 discusses extensions to arbitrarily large tiled displays.

implementation was tightly coupled to the Soleil-X simulation. Section 3 reports on the resulting performance up to 512 nodes.

2.3 Display

At the end of the Reduce tasks a fully composited image is left in layer 0 of the logical region. One or more Display tasks then transport the image pixels to their destination which may be either a file system or a live (tiled) display. It is straightforward in Legion to target a rectangular region of pixels to a specific Display task. In principle this makes it easy to write a high resolution result to multiple displays. The measurements presented in Section 3 were taken using a file proxy as the display device. Final images were written to files in PPM format and converted for offline viewing.

3 RESULTS

The modified simulation was run on the Piz Daint supercomputer in configurations from 4 to 512 nodes. Piz Daint is a Cray XC40/XC50 petaflop class supercomputer that contains GPU and CPU nodes. Each GPU node is a 12-core Intel Xeon E5-2690 coupled with an NVIDIA Tesla GPU. Each CPU node contains a pair of 18-core Intel Xeon ES-2695s. The nodes are connected by an Aries Dragonfly network. Simulations were run on the XC50 GPU partition.

Figure 1 shows the early conditions of the simulation. The simulation was run for 9 time steps and profiling data was collected and analyzed. The initial startup interval and the first two time steps were ignored to allow the program to achieve steady state.

Figure 2 shows the simulation and visualization subsystem scaled from 4 to 512 nodes. This shows the scaling of the simulation was not affected by the addition of the visualization tasks.

Figure 3 compares the latency of the reduce operation compared to the latency of Ice-T, illustrating that Ice-T times are many times lower than what was achieved here. There are several reasons for this that are orthogonal to the relative strengths and weaknesses of task-based and message passing-based systems. Ice-T is highly optimized, employs multiple compositing algorithms, and compresses images (and work) using Run Length Encoding (RLE). The SimpleTiming benchmark used to measure Ice-T draws a Rubik's cube centered in an empty screen which affords ample opportunity for RLE. Byte pixels are used and read back through Pixel Buffer Objects (PBOs).

In contrast the Reduce task is an unoptimized brute-force composite over all pixels. Float pixels were used and were read back through glReadPixels. This is slower than reading byte pixels through PBOs.

It is interesting to node that while the latency for the unoptimized task-based version increases logarithmically this does not affect the speed of the overall simulation. In fact, the latency can exceed the length of a time step and still not slow down the simulation as the task-based approach does not require the simulation and visualization systems to run in lock-step.[2] Note that 512 nodes demonstrated a latency increase (likely a performance bug) but this did not impact the overall runtime.

Figure 4 shows the profiler user interface for node 0 of a 256 node run. The second line from the top is labeled "CPU Proc 3". This processor runs the visualization tasks and this line shows a series of long Render tasks (yellow) separated by short Reduce tasks (purple). The bottom line is labeled "OpenMP Proc 5". This represents 8 cores running the simulation in OpenMP mode. Additional cores are used to run utility, IO and DMA processors. The OpenMP ABI is implemented by Legion so there is no conflict between these OpenMP cores and the cores assigned to visualization tasks.

Figure 5 shows a closeup of the same profile covering the duration of two render tasks. It shows that there is some idleness in the processor running the visualization system. It also shows small idle periods in the simulation that coincide with the end of a Render task. This is a result of data transfers that occur in support of the ensuing Reduce tasks.

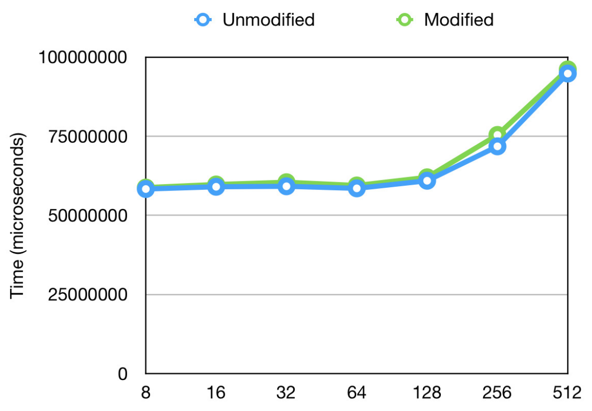

Figure 2: Weak scaling of the modified simulation (green line) and unmodified simulation (blue line) from 4 to 512 nodes.

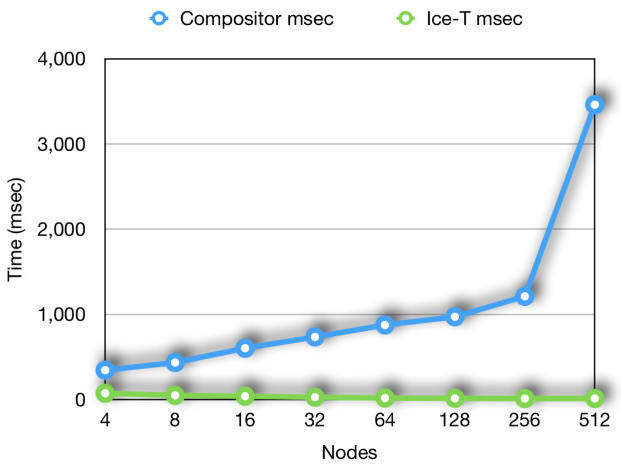

Figure 3: The unoptimized compositor (blue line) versus Ice-T (green line) from 4 to 512 nodes (weak scaling).

[2]Optimizations would be necessary if we desired interactive performance.

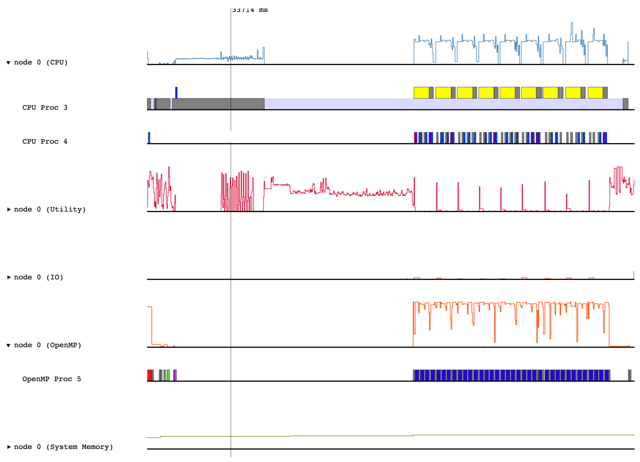

Figure 4: Performance profile of node 0 in a 256 node run for 9 time steps.

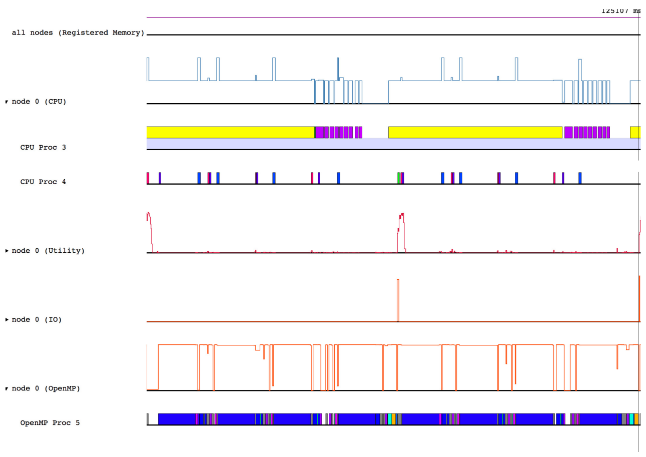

Figure 5: Closeup of Figure 4.

4 DISCUSSION

The goal of the visualization subsystem is to offer a reusable framework that can support a variety of *in situ* visualization requirements. Although this paper reported results from a hybrid C++/Regent implementation future plans are based on the C++ implementation alone. This implementation will be reusable with any Legion program regardless of whether the program is Regent-based or based on the C++ API.

4.1 Visualization applications

One issue to consider is coupling to open source visualization applications [6, 18]. The Legion runtime contains explicit representations of all of the data in a Legion program. This information can be provided to the visualization application to facilitate data access.

4.2 Large displays

The measurements reported here used a fixed size display of 3840 × 2160 pixels. This was chosen as the rendering size at each node. Large tiled displays will require images that are larger than can be rendered by a single node. This raises two related problems: generating pixel values at all of these locations, and skipping empty pixels. In the current framework these problems are unsolved. The following proposal is being evaluated as a possible solution.

When a Reduce task is launched two *projection functors* access the subsets of pixels in the logical region that are required for the reduction. These functors know the location of the pixels in the overall image. The size of a functor subregion is tunable and generally on the order of one or more scanlines. The functors are able to calculate whether the pixels fall outside of the rendered subregion from a local Render task.

The proposal is to allow projection functors to indicate whether their pixels fall within the locally rendered subregion. If both functors have valid pixels then the Reduce task would proceed. If only one functor has valid pixels then the Reduce task becomes a copy operation. And if neither functor has valid pixels then the Reduce task would not execute. We are evaluating this and other proposals.

5 CONCLUSION

This paper has demonstrated that a task based model allows noninvasive composition of visualization tasks with an existing simulation. The simulation showed the same scaling before and after the composition and the reduction operation scaled logarithmically up to 512 nodes. Visualization was overlapped with the simulation.

The problem of empty space skipping on large tiled displays is not currently solved. It appears that a voting scheme based on projection functors may provide a solution. This proposal and others are being evaluated.

The composition of tasks was accomplished noninvasively in that only two lines needed to be added to the existing simulation (one to initialize, the other to visualize). In contrast an MPI application would have to deal explicitly with issues of synchronization and communication.

6 ACKNOWLEDGEMENTS

This project has depended crucially on the inputs of the Legion team members, in particular Sean Treichler and Mike Bauer. Pat McCormick provided feedback on an early draft of this paper. This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID d51. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of ASCR, under the contract number DE-AC02-76SF00515.

REFERENCES

[1] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: Expressing Locality and Independence with Logical Regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12).* IEEE Computer Society Press, Los Alamitos, CA, USA, Article 66, 11 pages. http://dl.acm.org/citation.cfm?id=2388996.2389086
[2] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2014. Structure Slicing: Extending Logical Regions with Fields. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14).* IEEE Press, Piscataway, NJ, USA, 845–856. https://doi.org/10.1109/SC.2014.74
[3] Gilbert Louis Bernstein, Chinmayee Shah, Crystal Lemire, Zachary Devito, Matthew Fisher, Philip Levis, and Pat Hanrahan. 2016. Ebb: A DSL for Physical Simulation on CPUs and GPUs. *ACM Trans. Graph.* 35, 2, Article 21 (May 2016), 12 pages. https://doi.org/10.1145/2892632

[4] Tim Biedert, Kilian Werner, Bernd Hentschel, and Christoph Garth. 2017. A Task-Based Parallel Rendering Component For Large-Scale Visualization Applications. In *Eurographics Symposium on Parallel Graphics and Visualization*, Alexandru Telea and Janine Bennett (Eds.). The Eurographics Association. https://doi.org/10.2312/pgv.20171094

[5] Thomas D. Economon and Ivan Bermejo-Moreno. 2017. Soleil-X. (2017). https://github.com/stanfordhpccenter/soleil-x

[6] Charles Hansen and Chris Johnson. 2005. *The Visualization Handbook.* Elsevier.

[7] Kwan-Liu Ma. 2009. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Comput. Graph. Appl.* 29, 6 (Nov. 2009), 14–19. https://doi.org/10.1109/MCG.2009.120

[8] Peter Messmer. [n. d.]. EGL Eye: OpenGL Visualization without an X Server. ([n. d.]). https://devblogs.nvidia.com/parallelforall/egl-eye-opengl-visualization-without-x-server/

[9] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. 1994. A Sorting Classification of Parallel Rendering. *IEEE Comput. Graph. Appl.* 14, 4 (July 1994), 23–32. https://doi.org/10.1109/38.291528

[10] Kenneth Moreland. 2016. The Tensions of In Situ Visualization. *IEEE Comput. Graph. Appl.* 36, 2 (March 2016), 5–9. https://doi.org/10.1109/MCG.2016.35

[11] Kenneth Moreland, Wesley Kendall, Tom Peterka, and Jian Huang. 2011. An Image Compositing Solution at Scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11).* ACM, New York, NY, USA, Article 25, 10 pages. https://doi.org/10.1145/2063384.2063417

[12] Philippe Pebay, Janine C. Bennett, David Hollman, Sean Treichler, Patrick S. McCormick, Christine M. Sweeney, Hemanth Kolla, and Alex Aiken. 2016. Towards Asynchronous Many-Task in Situ Data Analysis Using Legion. *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* 00 (2016), 1033–1037. https://doi.org/doi.ieeecomputersociety.org/10.1109/IPDPSW.2016.24

[13] Elliott Slaughter, Wonchan Lee, Sean Treichler, Michael Bauer, and Alex Aiken. 2015. Regent: A High-productivity Programming Language for HPC with Logical Regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15).* ACM, New York, NY, USA, Article 81, 12 pages. https://doi.org/10.1145/2807591.2807629

[14] Elliott Slaughter, Wonchan Lee, Sean Treichler, Wen Zhang, Michael Bauer, Galen Shipman, Patrick McCormick, and Alex Aiken. 2017. Control Replication: Compiling Implicit Parallelism to Efficient SPMD with Logical Regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17).* ACM, New York, NY, USA.

[15] Sean Treichler, Michael Bauer, and Alex Aiken. 2013. Language Support for Dynamic, Hierarchical Data Partitioning. *SIGPLAN Not.* 48, 10 (Oct. 2013), 495–514. https://doi.org/10.1145/2544173.2509545

[16] Sean Treichler, Michael Bauer, and Alex Aiken. 2014. Realm: An Event-based Low-level Runtime for Distributed Memory Architectures. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT '14).* ACM, New York, NY, USA, 263–276. https://doi.org/10.1145/2628071.2628084

[17] Sean Treichler, Michael Bauer, Rahul Sharma, Elliott Slaughter, and Alex Aiken. 2016. Dependent Partitioning. *SIGPLAN Not.* 51, 10 (Oct. 2016), 344–358. https://doi.org/10.1145/3022671.2984016

[18] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel in Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV '11).* Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 101–109. https://doi.org/10.2312/EGPGV/EGPGV11/101-109