Performance Visualization and Analysis of Parallel Discrete Event Simulations with Projections

Chee Wai Lee, Terry L. Wilmarth and Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
{*cheelee, wilmarth, kale*}*@cs.uiuc.edu*

Abstract

In designing software for Parallel Discrete Event Simulation (PDES), detailed performance analysis has proved essential to address the challenges of minimizing overhead, maximizing parallelism and achieving scalability. This paper describes Projections[12], a performance visualization and analysis tool for parallel applications in CHARM++*, and its usage in the analysis of PDES applications. We discuss PDES analysis from three points of view: PDES tool design, PDES simulation design and PDES model analysis. We describe enhancements to Projections that make PDES-specific performance visualizations possible. We also develop generic analysis capabilities that can be used by other PDES engines. We demonstrate the utility of these features with* POSE*[21, 22] applications.* [1]

1. Introduction

Optimistically synchronized Parallel Discrete Event Simulation (PDES) is notoriously difficult to scale to many processors, particularly when the granularity of events is fine[8]. The cost of executing such fine-grained events is often far outweighed by the overhead of checkpointing, rollback, cancellation, fossil collection and *global virtual time* (GVT) estimation. Strategies for all of these activities, including optimistic synchronization, must be efficient, scalable and should combine together well.

Analysis of each of these components in parallel can be quite challenging. While simply timing each component can give insight into the efficiency of the particular algorithms used, it may not reveal the bigger picture of how the components behave and interact in combination. Tools such as *gprof*[9] and its relatives which provide similar timing information as well as quantities of invocations and dependencies, may provide further insight, but do not capture the parallel interactions.

Thus, detailed performance analysis of parallel applications is a minimum requirement for analysis of PDES. Discrete event simulation is, however, straightforward enough to be amenable to a variety of analysis techniques.

We would like to analyze parallel discrete event simulations from three perspectives: the *PDES tool designer*, the *simulation developer* and the *model designer*. To clarify the relationship between these perspectives, the PDES tool designer would like to develop software for PDES that is scalable and handles fine-grained events elegantly. She would like an analysis tool to capture as much information as possible about all the components of the optimistic synchronization mechanism and clearly illustrate how these components interact in a parallel environment.

The simulation developer is faced with the significant challenge of translating a discrete model of concurrent interacting entities into code that is both efficient and accurate to the model. He is concerned with parallelism, utilization, load balancing and of course correctness. The simulation developer and the model designer are often the same person.

The model designer wishes to study or create some complex real-world system. He develops a discrete model of this system as best as he can and requires simulation to set the works in motion. He may have some specific desires with regard to the system, and may alter the model to meet his needs. Some aspects of the utility of a model can be captured and vi-

1 This work was supported in part by the National Institutes of Health (PHS 5 P41 RR05969-04) and the Department of Energy (B341494).

sualized in a general fashion for certain types of process-oriented systems. For example, we should be able to capture the peak operating efficiency of a factory assembly line and display it so that a factory designer can be satisfied with his design, or can subsequently improve upon it. Alternatively, in simulating an interconnection network designed for a very large parallel machine, we may discover that a tentative network model performs poorly for the particular traffic patterns of the applications the machine is being designed to handle, and thus choose to modify that model.

This paper begins by describing Projections, a post-mortem, trace-based performance visualization and analysis framework for parallel CHARM++ applications. We discuss our experiences using Projections to tune the performance of POSE, a CHARM++-based PDES tool. We describe the addition of performance data collection to POSE to visualize critical path parallelism for the analysis of simulation performance and behavior from the perspectives of simulation developers and model designers. Finally, we present enhancements to Projections that allow us to study PDES and model performance using this technique.

2. The Projections Performance Tool

Projections is a post-mortem, trace-based performance visualization and analysis framework for parallel CHARM++ applications.

The Projections user can control the amount of trace data generated at application runtime by selectively turning tracing on and off at key points. The user can also choose to generate cumulative summary information instead of full log data.

The visualization component of the Projections framework is implemented in Java and has the advantage of being generally portable across many platforms. The trace logs generated by the instrumentation modules in the CHARM++ run-time are platform neutral.

A key strength of Projections as a performance visualization tool is its ability to manage large volumes of data at various levels. Many application classes, including PDES, can generate copious quantities of information.

2.1. Overview of Projections Visualization Capabilities

A thorough description of basic Projections visualization capabilities is provided by *(name elided) et al*[12] which also describes optimizations to NAMD[18], a molecular dynamics simulation application, that were inspired by Projections visualizations of the performance data. We now present several views we feel to be particularly useful for optimizing PDES performance.

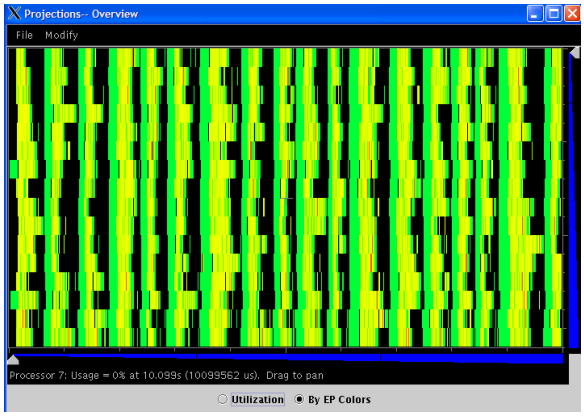


Figure 1: *Overview*: **A summary of overall parallel structure[5]**

The *overview* plot of Figure 1 can show the dominant activity within fixed time intervals along the horizontal-axis over each processor along the vertical-axis. It presents a good summary of the overall parallel structure of the application.

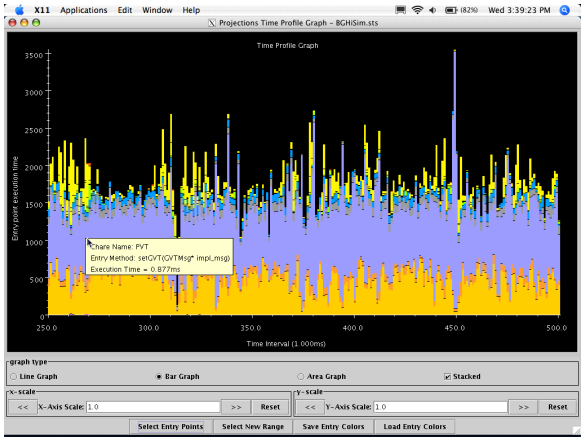


Figure 2: *Time Profile*: **Activity execution time summed across all processors**

The *time profile* view of Figure 2 displays, for each

application time interval, the total time spent by various activities summed across all processors. Moving the mouse pointer over a bar brings up a small window providing more information about that time interval. In Figure 2, we see the PDES activity `setGVT` (which encapsulates many other PDES activities) ran for 877 μs at around the 270 ms mark of the application. This view offers a good way of summarizing the changing nature of the PDES application as it executes.

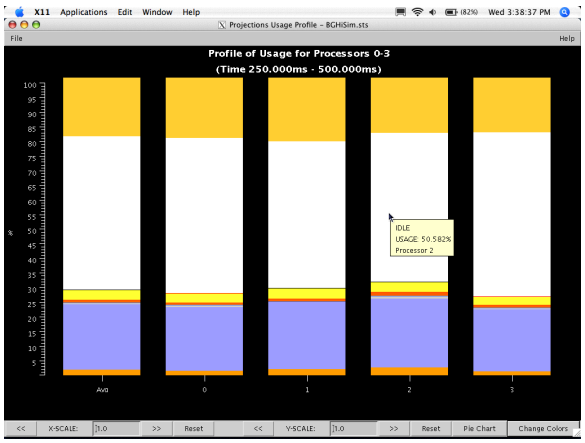


Figure 3: *Usage Profile*: **Processor utilization of various activities**

The *usage profile* view of Figure 3 displays, for each processor, the utilization percentage of all activities over a user-selected range of the application's execution time. This includes the percentage of idle time, when no useful work occurs. The left-most bar presents data averaged across all processors. This view is particularly useful for identifying the presence of load imbalance in the application, as well as its overall efficiency.

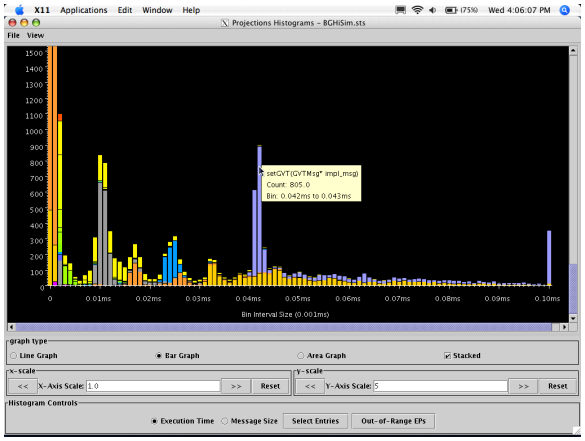


Figure 4: Histogram of activity grainsizes

The *histogram* view in Figure 4 counts the frequency of each instance of an activity binned by the time taken for each call. This shows the effective distribution of activity grainsize over the user-selected time range of study.

Finally, the *timeline* view in Figure 5 presents to the user a detailed look at the interactions between application activity and communication between processors. The horizontal time-axis marks the passage of time in the application while the vertical axis represents processors. Moving the mouse pointer over an activity pops up a small window with specific information about various performance properties of that activity (eg. number of messages sent).

2.2. Related work

There are many performance tools available. These tools vary greatly in their scope and goals, which can be divided into four subdomains: instrumentation, data generation, visualization and analysis. The scope of many tools span more than one subdomain. Moore *et al*[16] give a good review of recent available tools.

Vampir[17, 3], *Jumpshot*[24] and *Paradyn*[15] all provide support for detailed instrumentation and visualization of performance data.

TAU (Tuning and Analysis Utilities)[14, 13] is a flexible framework for performance analysis with bindings for various languages and parallel programming models. It generates trace logs that can be used or converted into other open formats. These logs can then be processed by visualization tools like *ParaProf*[2] and Vampir or passed to trace analysis tools.

The *SvPablo*[7] performance tool is similar in nature to *TAU* in its emphasis on portability, extensibility and scalability for dealing with performance data. *SvPablo* is a component of the more general *Pablo*[19] performance analysis environment.

PVaniM-GTW[4] extends the more generic PVM visualization tool *PVaniM*[20] in order to expose PDES-related performance details of the GTW (Georgia Tech Time Warp) system[6].

Balakrishnan *et al*[1] describe a framework which uses a Workload Specification Language to express the performance attributes of PDES applications which can then be generated for further study and visualization.

Projections is designed along the lines of *TAU* and *SvPablo* in that it provides support for the performance analysis process from instrumentation to visualization. It is tightly coupled to the CHARM++ run-time and par-

allel applications. CHARM++ is, however, supported on many platforms and machines. Adaptive MPI [10] is an MPI implementation on top of CHARM++ which allows many MPI applications to also make use of the performance analysis support provided by Projections.

3. PDES Performance Analysis

In this section, we describe our experiences using and enhancing the functionality of Projections for visualizing and analyzing performance of PDES applications. As mentioned earlier, Projections was designed for performance visualization of parallel CHARM++ applications. As POSE is implemented in CHARM++, we can make full use of Projections' capabilities by using POSE for our demonstration. However, any PDES system could in theory produce output of the expected format and make use of much of the functionality described herein. In particular, the extended data formats for analyzing real- and virtual-time critical paths are orthogonal to the standard logs and unburdened by any CHARM++ run-time semantics.

3.1. Visualizing Performance of PDES Tools

Our first objective in the use of Projections with POSE was to improve POSE's performance as a PDES tool. As POSE is implemented in CHARM++, Projections automatically provides a display of all CHARM++ *entry methods* (asynchronous remote method invocations) and their interactions. More precisely, CHARM++ entry methods can be considered to be the *unit of visualization* of Projections.

While this certainly provides a clearer picture of the behavior of POSE than tools like *gprof* alone, it has severe limitations for displaying the key activities that occur in optimistic PDES, namely, forward execution of events, rollbacks, cancellations, fossil collection, etc. Since the only clearly denoted activities are those that involve CHARM++ entry methods, we are limited in our visualization capabilities. For example, Projections clearly displays POSE's GVT behavior because it is governed by exchanges of CHARM++ entry methods. It also displays event method *arrivals*, which are also implemented as entry methods. However, in throttled optimistic PDES, event arrival does not necessarily correspond to event execution. Thus, most of the activities of interest to us are hidden within the execution of a few entry methods.

For example, when a new GVT is calculated, an entry method `setGVT` is broadcast to all processors. Within

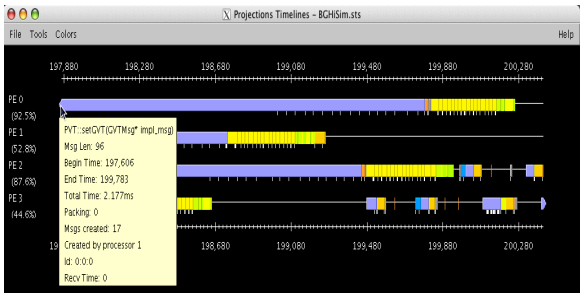


Figure 5: Projections timeline of POSE activity

the context of this entry method, each processor performs fossil collection and executes any events that were enabled by the new GVT estimate.

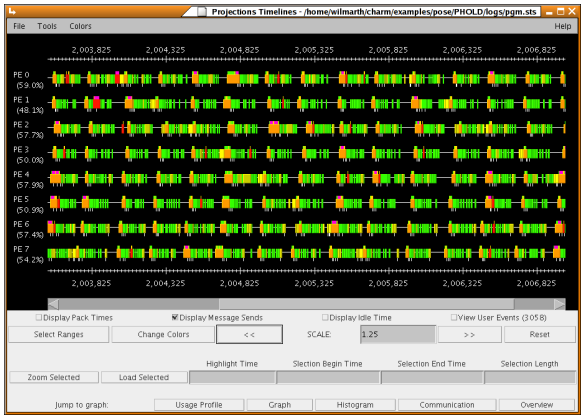
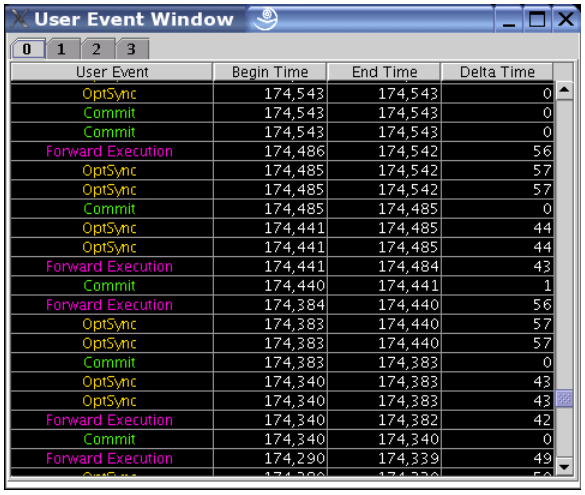


Figure 6: Projections timeline of POSE activity with user events

To obtain more information about what was occurring within these entry methods, we enhanced POSE with special trace function calls to the *user events* API of Projections. The *user events* API allows a Charm++ application developer to demarcate the start and end of arbitrary activities of interest, and have those displayed above the regular Projections timeline. For POSE, we specified user events for forward execution, fossil collection, rollbacks, spawning cancellations, handling cancellations, and optimistic synchronization. This made it possible to visualize what was happening with these important PDES events along the Projections timeline.

Figure 6 shows a Projections timeline with POSE-specific instrumentation visualized. These events are rendered as colored bars or ticks above the regular timeline bars. Note the richer information available when compared with the timeline Figure 5. Detailed information about these user events can also be obtained via a control button on the timeline window. This is presented in the form of a column-sortable table as shown in Figure 7. For each processor, the table shows the user



User Event	Begin Time	End Time	Delta Time
OptSync	174,543	174,543	0
Commit	174,543	174,543	0
Commit	174,543	174,543	0
Forward Execution	174,486	174,542	56
OptSync	174,485	174,542	57
OptSync	174,485	174,542	57
Commit	174,485	174,485	0
OptSync	174,441	174,485	44
OptSync	174,441	174,485	44
Forward Execution	174,441	174,484	43
Commit	174,440	174,441	1
Forward Execution	174,384	174,440	56
OptSync	174,383	174,440	57
OptSync	174,383	174,440	57
Commit	174,383	174,383	0
OptSync	174,340	174,383	43
OptSync	174,340	174,383	43
Forward Execution	174,340	174,382	42
Commit	174,340	174,340	0
Forward Execution	174,290	174,339	49

Figure 7: Table of User Events

events colored as they are on the Projections timeline, and shows start time, end time and elapsed time for each user event. The timeline of Figure 6 is particularly interesting as it exhibits the typical behavior of a PDES application with fine-grained events. The short, nearly-synchronized sections of events that repeat on that timeline are cycles of GVT calculation followed by forward executionsteps and finally follwed by reductions to be used for the next GVT calculation. A close-up of the user events above the timeline bar is shown in Figure 8. This zoomed-in section shows the processors receiving new GVT estimates at the start of one cycle. User events above the bars show work that was enabled by the new GVT being executed, along with fossil collection.

One performance improvement that resulted from the use of Projections was in how the GVT estimation was triggered. BigNetSim[5], a large-scale interconnection network simulation with very fine-grained events and dramatically varying load, displayed a lot of idle time as some processors ran out of events to handle, triggered GVT estimation, and had to wait for other, more loaded processors for the GVT estimation to complete and enable more events. This behavior is illustrated by a Projections overview shown in Figure 9. As mentioned earlier, the Projections overview shows time on the x-axis and processors on the y-axis. Here, we see stripes of work (colors) that start at roughly the same times, but minute imbalances cause some processors to run out of work before others and go idle (black).

We solved the problem by allowing GVT-triggering processors to interrupt event processing on busy processors so that they could report required information for the GVT estimation. This was achieved by inserting high priority entry methods on the processors that would be handled before additional events. This resulted in a dramatic improvement in processor utilization and run-time, in spite of an increase in the number of times the GVT estimation was performed. This improvement was easily visualized in the Projections overview of Figure 10, which shows much less idle time. In fact, we had to zoom in to this overview to see the idle time. The method is not perfect however, as processors cannot be interrupted in the midst of an individual event method execution. One long-running event can cause the occasional period of idle time on the other processors.

Thus the basic functionality of Projections and the addition of special user events to POSE make it possible to dissect performance of POSE from a PDES tool

Figure 8: Close-up of User Events

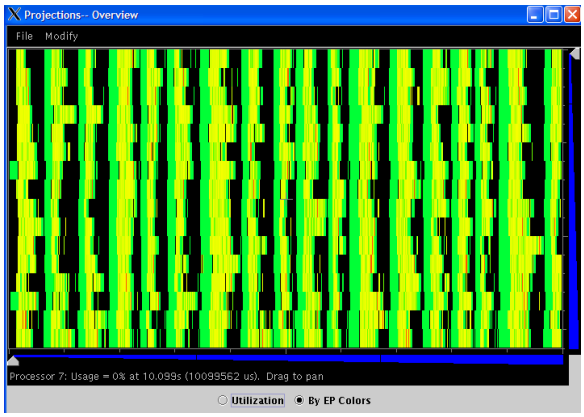


Figure 9: GVT-imposed idle time on processors[5]

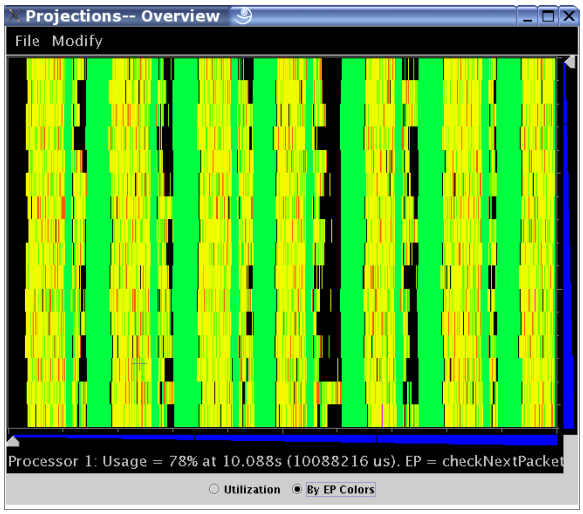


Figure 10: GVT triggering immediate update[5]

developer's perspective.

3.2. Critical Path Visualization

The same techniques that were used in the previous section can, of course, be used to analyze the actual PDES application itself. Information about both the model and its implementation can be gleaned from the available tools as they are. However, sometimes the quantity of information in a simulation with fine-grained events is too overwhelming for the level of detail provided by the Projections timeline tool. The Projections overview could be used for very general diagnoses of load imbalance and poor utilization, but could not expose problems that might be inherent in the model implementation or in the model itself. In particular, our BigNetSim[5] application was suffering from problems of poor utilization that we had no way to diagnose. It became clear that we needed some means to analyze how much *parallelism* is present in a simulation implementation or model.

3.2.1. Visualizing Real-time Critical Paths of PDES Applications Given an application, we have added code to trace the critical forward execution paths through the simulation, and output this information for post-processing. We plot the information in the form of the number of parallel objects processing events versus real time. This plot gives us an idea of how parallel the implementation actually is, and indicates trouble spots where we might expect low utilization. It can also give us an idea of the maximum number of processors we could expect the simulation to scale to, as well as a maximum number of processors beyond which no improvement in run-time could be achieved. In short, it provides clear and concise information about the performance of a simulation to a simulation developer.

To achieve this, we implemented a simple system of critical path tracing in POSE. This tracing keeps track of event start and end times along the critical path, excluding any time for optimistic PDES overheads and communication. Thus a critical path indicates constant activity, with almost no inactive periods, from start to finish. The only cause for a delay is when the different critical paths try to engage the same *logical processor* (LP) at the same time. Other critical paths may branch off at any point.

We illustrate an example of a possible critical path tree in Figure 11. We have drawn green real-time intervals to represent event execution, and red arrows to connect them in critical paths. There are four initial events with no incoming red arrows. These four critical paths branch out, performing work on all twelve LPs, however there are fourteen branches in the entire space shown. Of these, there are a maximum of nine that are active at a single point in time.

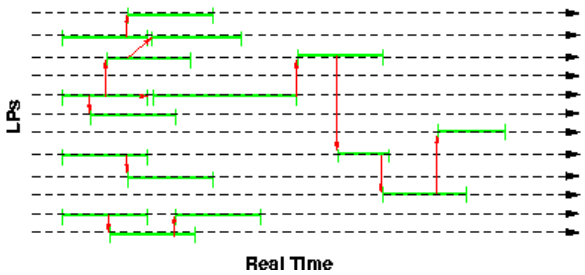


Figure 11: Real-time Critical Paths

We would like to visualize the critical path in terms of the number of active events at a given point in time. This will give us an idea of how much parallelism the simulation is capable of. To do this, we need to take the critical path tree and plot its length along the x-axis and plot simultaneous events along the y-axis. This involves translating real time into microseconds for the x-axis. Thus, an event that is active from t_1 μs to t_2 μs contributes a value of one to the plot for all times t_1 through t_2-1. In the special case that $t_1 = t_2$ we allow the event to contribute one to the t_1 time. Figure 12 shows a handmade graph for the real-time critical paths of Figure 11. It reflects the higher density of the critical path tree at the earlier times and the sparser activity at the later times. It also illustrates that the maximum amount of parallelism realized by the twelve LPs is only nine, and on average the parallelism is much worse.

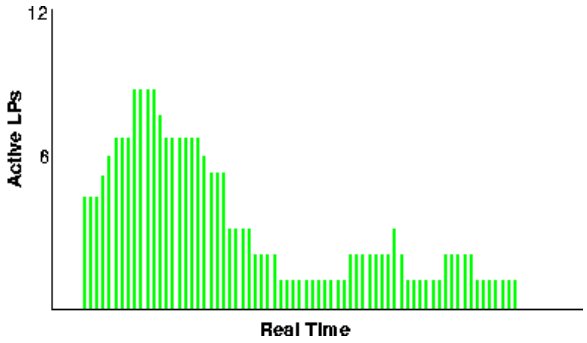


Figure 12: Real-time Active LPs

An early version of real-time critical path visualization for a real simulation is shown in Figure 13. This type of visualization has recently been integrated into Projections as an additional PDES-specific tool. The visualization shown is for an early version of a simulation of large-scale interconnection networks[11, 23]. We experienced poor utilization for this simulation in which ordinary Projections tools illustrated many components were simply waiting for work to do. This did not make sense since the network was supposed to be continuously and heavily loaded. The real-time critical path plot for this program did expose some mysterious time-periods during which the maximum parallelism was very low. This indicated that something about the implementation was poorly designed.

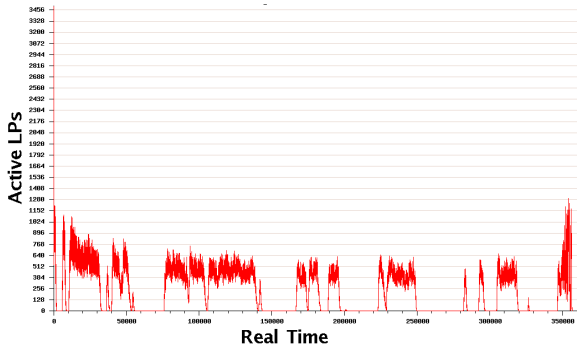


Figure 13: Real-time Active LPs for Network Simulation

We initially surmised that the routing algorithm was possibly in error, and was routing all messages through a single switch in the network. This would indicate that the model was not merely poorly implemented, but rather incorrectly implemented. However, we also examined the virtual-time critical path (discussed in the next section), which correctly indicated that the entire network was very busy with work that gradually tapered off, as shown in Figure 15. Thus the routing algorithm was behaving as expected.

We then surmised that certain individual simulation entities (called *posers* in POSE) had been designed to correctly simulate parallel activities in the model. While this is permitted in POSE, posers are meant to model sequential entities, and thus using a single poser to model parallel activities places unnecessary limits on the maximum possible parallelism of the simulation. A closer examination of the code revealed that this was indeed the case. A *switch* object was found to have a variety of complex behaviors. While it could model the parallel receiving, routing and sending of simultaneous packets, it did so serially, one event at a time. A redesign of the switch object, namely decomposing it into smaller posers, resulted in more predictable critical path behavior and significantly improved utilization[5]. In Figure 14, we show the latest version of the critical path tool as implemented in Projections. This example is for a smaller simulation, but the upper blue line shows that we now get much more consistent parallelism (and consequently utilization) from the network simulation application. The lower green line indicates the number of events making forward progress in virtual time, a feature we are currently experimenting with. Thus the real-time critical path visualization has proved useful from a simulation developer's perspective.

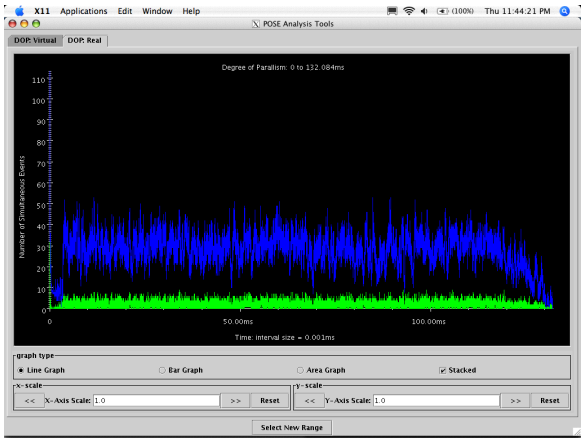


Figure 14: Real-time Active LPs for Latest Network Simulation (upper blue line)

3.2.2. Visualizing Virtual-time Critical Paths of PDES Models We present another view that can aid the model designer in the verification and optimization of their discrete models. This technique is particularly useful for models that are designed with productivity optimization in mind. Whatever the simulation domain, if the desired result is to maximize throughput, or minimize time, this view makes it possible to

locate and diagnose bottlenecks in the system and to determine where more production components are best placed to improve the outcome.

Virtual-time critical paths are similar to real-time critical paths, only we pay attention to what is overlapping in the model, i.e. in virtual time, rather than what is overlapping when we run the simulation on a parallel computer. We plot active LPs versus virtual time to obtain a similar visualization. In Figure 15, we show the early version of the virtual-time visualization for the interconnection network simulation. The network was populated by having each processor send a fixed number of messages, and then letting all the packets be delivered before the simulation was terminated. Had the virtual-time visualization displayed a more erratic behavior than the smooth curve shown, we would have been alerted to problems in the way we were modeling the network.

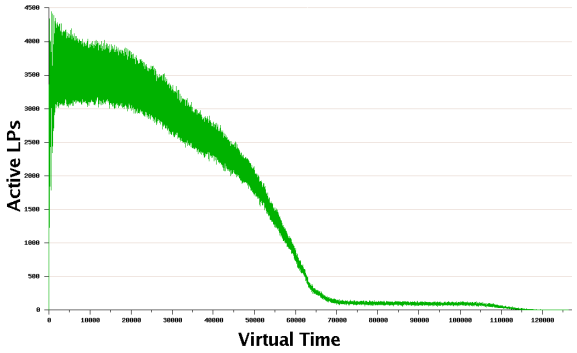


Figure 15: Virtual-time Active LPs for Network Simulation

3.2.3. Critical Path Visualization in Projections We have recently implemented both critical path visualizations as Projections PDES-specific tools. Because the number of data points to plot will become unwieldy as the size of a simulation increases, Projections built-in capabilities to handle large data sets are indispensable. Projections is capable of selecting time ranges to examine, and condensing times into intervals, displaying averages for intervals. A condensed view can be displayed at first, and then specific areas of interest selected for closer examination. Figure 16 shows virtual-time overlap averaged over intervals of size 50 (μs). Larger intervals smooth out some of the details of the peaks and troughs as the overlaps are averaged over each interval. They do, however, provide a quick glimpse into how much average and maximum parallelism a simulation has coded into it. Figure 17 shows the same data, but with maximum detail at an interval size of 1 (μs).

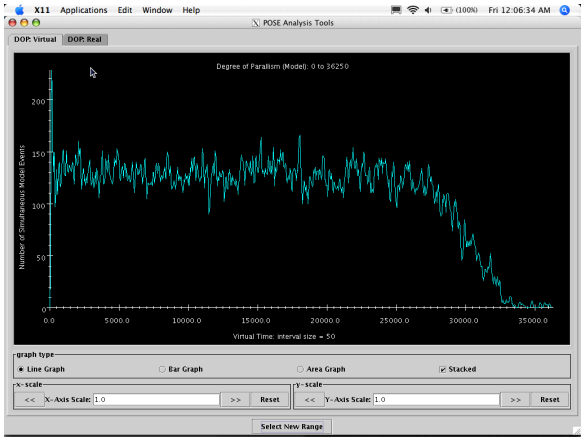


Figure 16: Larger interval sizes smooth out some detail

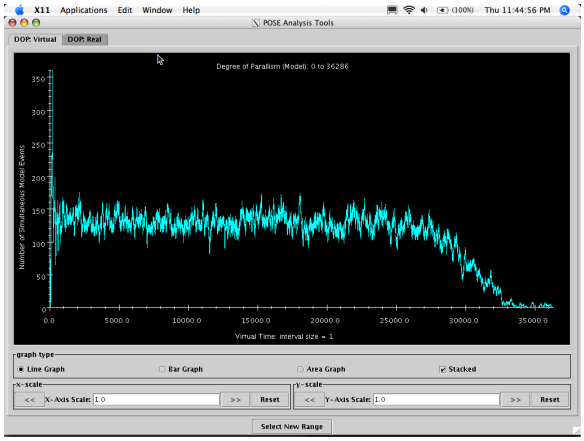


Figure 17: Smallest interval size shows most detail

4. Conclusions and Future Work

We have described the performance visualization and analysis tools provided by Projections and extended the functionality to support PDES-specific visualizations. These tools have proved extremely useful in developing and improving the PDES environment POSE, and in improving simulations and models.

We are currently working on an enhanced version of the critical path tool which will display stacked color bars, where each color corresponds to an event type. This should be very helpful in pinpointing any problem areas in both simulation implementations and model designs.

We also plan to look into visualizing event timelines, both in real and virtual time. For real time, one could view the critical path timeline, colored by event type, or one could view the actual timeline for the program run, again colored by event type, with PDES activities de-

emphasized. For such a display, it would be nice to be able to visualize event dependencies. Projections already has this ability for the entry methods of CHARM++ programs, so we should be able to create traces of a similar format from POSE simulations.

References

[1] V. Balakrishnan, P. Frey, N. B. Abu-Ghazaleh, and P. A. Wilsey. A framework for performance analysis of parallel discrete event simulators. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 429–436, New York, NY, USA, 1997. ACM Press.

[2] R. Bell, A. D. Malony, and S. Shende. A portable, extensible, and scalable tool for parallel performance profile analysis. In *Proc. EUROPAR 2003 conference, LNCS 2790*, pages 17–26, Berlin, 2003. Springer.

[3] H. Brunst, H.-C. Hoppe, W. E. Nagel, and M. Winkler. Performance optimization for large scale computing: The scalable vampir approach. In *ICCS '01: Proceedings of the International Conference on Computational Science-Part II*, pages 751–760, London, UK, 2001. Springer-Verlag.

[4] C. D. Carothers, B. Topol, R. M. Fujimoto, J. T. Stasko, and V. Sunderam. Visualizing parallel simulations in network computing environments: a case study. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 110–117, New York, NY, USA, 1997. ACM Press.

[5] N. Choudhury, Y. Mehta, T. L. Wilmarth, E. J. Bohm, and L. V. Kalé. Scaling an optimistic parallel simulation of large-scale interconnection networks. In *Proceedings of the Winter Simulation Conference*, 2005.

[6] S. R. Das, R. Fujimoto, K. S. Panesar, D. Allison, and M. Hybinette. GTW: a time warp system for shared memory multiprocessors. In *Winter Simulation Conference*, pages 1332–1339, 1994.

[7] L. DeRose and D. A. Reed. Svpablo: A multi-language architecture-independent performance analysis system. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1999.

[8] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[9] S. L. . Graham, P. B. Kessler, and M. K. McKusick. GPROF: a call graph execution profiler. *SIGPLAN 1982 Symposium on Compiler Construction*, pages 120–126, June 1982.

[10] C. Huang, O. Lawlor, and L. V. Kalé. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, pages 306–322, College Station, Texas.

[11] P. K. Jagadishprasad. Parallel simulation of large scale interconnection networks used in high performance computing. Master's thesis, University of Illinois at Urbana-Champaign, 2004.

[12] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, number to appear, 2005.

[13] A. D. Malony and S. Shende. Performance technology for complex parallel and distributed systems. pages 37–46, 2000.

[14] A. D. Malony, S. Shende, R. Bell, K. Li, L. Li, and N. Trebon. Advances in the tau performance system. pages 129–144, 2004.

[15] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *Computer*, 28(11):37–46, 1995.

[16] S. Moore, D. Cronk, K. London, and J. Dongarra. Review of performance analysis tools for mpi parallel programs. In *8th European PVM/MPI Users' Group Meeting, Lecture Notes in Computer Science 2131*, pages 241–248, Berlin, 2001. Springer-Verlag.

[17] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[18] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*, Baltimore, MD.

[19] D. A. Reed et al. Scalable performance analysis : The pablo performance analysis environment. In *Proceedings of the Scalable Parallel Libraries Conference*, pages 104–113. IEEE Computer Society, 1993.

[20] B. Topol, J. T. Stasko, and V. Sunderam. Pvanim: a tool for visualization in network computing environments. *Concurrency: Practice and Experience*, 10(14):1197–1222, 1998.

[21] T. Wilmarth and L. V. Kalé. Pose: Getting over grainsize in parallel discrete event simulation. In *2004 International Conference on Parallel Processing*, pages 12–19, August 2004.

[22] T. L. Wilmarth. *POSE: Scalable General-purpose Parallel Discrete Event Simulation*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[23] T. L. Wilmarth, G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. V. Kale. Performance prediction using simulation of large-scale interconnection networks in pose. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*, pages 109–118, 2005.

[24] O. Zaki, E. Lusk, W. Gropp, and D. Swider. Toward scalable performance visualization with Jumpshot. *The International Journal of High Performance Computing Applications*, 13(3):277–288, Fall 1999.