Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

EXTREME SCALE OPTIMISTIC PARALLEL DISCRETE EVENT SIMULATION WITH DYNAMIC LOAD BALANCING

Peter D. Barnes, Jr.
David R. Jefferson
Markus Schordan
Dan Quinlan
Lawrence Livermore National Laboratory
7000 East Avenue
Livermore CA 94550 USA

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy NY 12180 USA

Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
201 North Goodwin Avenue
Urbana IL 62801 USA

ABSTRACT

The recent world record PHOLD performance result suggests optimistic parallel discrete event simulators (OPDES) should be able to deliver superior performance at extreme scale in many application domains. In fact, programming for OPDES today is extremely hard because of the necessity to write reversing and commit methods, in addition to the normal forward method of a conservative implementation. A second issue in extreme scale simulation is dealing with dynamic load imbalance. In this paper we will describe our approach to addressing these issues, using: source-to-source compiler tools to create optimistic forward, reverse and commit methods solely from the conservative method implementation; the ROSS OPDES simulator; and the Charm++ run time platform for dynamic load balancing.

1 INTRODUCTION

Recently we reported strong scaling results of the PHOLD benchmark in the ROSS simulator on Lawrence Livermore National Laboratory's Sequoia Blue Gene/Q supercomputer.(Barnes et al. 2013) The benchmark, with 251 million PHOLD logical processes, executed in up to 7.86 million MPI tasks on 1,966,080 cores, processing 33 trillion events in 65 seconds, yielding a sustained speed of 504 billion events/second, which is by far the highest event rate reported by any discrete event simulation to date, running PHOLD or any other benchmark. The significance is that direct simulations of planetary-scale problems are now, in principle at least, within reach. Our benchmark had *only* 251 million LPs in order to fit in RAM at the smallest scale. The largest scale had enough RAM for 100 times as many PHOLD LPs, *i.e*. 25 billion! By comparison, there are only 7B people and only 4B IPv4 addresses.

Developing models for OPDES and obtaining sustained performance on real problems remain great challenges, both in writing models and dealing with dynamic load imbalance. First we discuss the coding difficulty inherent in OPDES, which we have solved with a new approach to incremental state saving and new compiler tools to automatically create reverse source code. Second, we discuss use of the Charm++ parallel runtime system to bring dynamic load balancing to the ROSS OPDES.

Because a conservative simulator guarantees event execution in strict timestamp order within an LP, the modeler can think (and code) in strictly sequential terms. The key idea in optimistic simulation is speculative execution at the event (function call) level. As new events arrive from other processes in a distributed implementation, the simulator may discover that some already executed events were premature. In this case the simulator has to undo the effects of those events to *roll back* in time and re-execute them in the proper order. For rollback to work the sequential event method $E()$ has to be split into a forward method, $E^{+}()$, and a rollback method, $E^{-}()$. In practice, some operations can't be reversed; they must be deferred to a commit method, $E^{*}()$. Turning $E()$ into the triplet $E^{+}()$, $E^{-}()$, $E^{*}()$ has historically been a prohibitive software development and engineering burden. For optimistic simulation to become a

mainstream tool, it is essential to develop automated tools to generate the optimistic methods from the sequential code *E*() directly. Our solution to this problem is called Backstroke.(Vulov et al. 2011) Backstroke leverages the source-code level transformation capabilities of the ROSE compiler toolkit.(Quinlan et al. 2013) It acts as a preprocessor, using ROSE to parse the sequential method *E*() and automatically generate the associated forward/reverse/commit functions in source code form, after which compilation proceeds as normal. To actually implement the reversal, we have invented a new form of incremental state saving. In essence at all assignments to plain old data we push the target address and original data value pair, (*&*,*v*) onto a *rollback stack*, before executing the assignment. To reverse, we just pop the (*&*,*v*) pairs off the rollback stack restoring the original values. We have shown that this method enables us to reverse essentially all of the C++11 language in source code form. This mechanism will be fully described in a forthcoming paper.

Turning to the issue of load balance, we note that large models will almost always have dynamic imbalance. Even a well-behaved billion-entity model with a Gaussian work-per-event distribution will have a 12 sigma spread between fastest and slowest entity. While simulating more objects enables better determination of average or typical behavior, paradoxically it is more liable to become bogged down simulating at the *extrema* of the population distribution. In OPDES models even the usual definitions of load need to be revisited, since LPs experiencing a large number of rollbacks are not doing useful work, despite consuming many CPU cycles. Dynamic load measurement in PDES is almost completely unexplored territory, as are approaches to dynamic load balancing.

To make a scalable PDES framework actually execute real models efficiently at scale, we have to solve the dynamic load balancing problem. To this end we have adapted the ROSS optimistic simulation engine to use the Charm++ distributed runtime system in order to obtain dynamic load balancing capability. The Charm++ system at its core provides efficient and scalable asynchronous message passing between C++ objects.(Kalé and Zheng 2013) In addition it provides application-independent object migration, fault tolerance, power awareness, and automatic overlap of communication with computation. Most of these capabilities rely on a generic facility for serializing and deserializing objects. An initial version of ROSS running on Charm++ is now working and producing approximately the same performance. Significant optimization work is in progress.

In summary we have described our approach to automate the creation of optimistic forward, reverse and commit methods in source code solely from the conservative method implementation. We have adapted the ROSS OPDES simulator to use the Charm++ asynchronous message passing run time platform, to take advantage of Charm++ dynamic load balancing. For the future, a key research question is what load balancing instrumentation, triggers, and policies work best in different DES model domains. There is little prior work in this area for PDES, and the impact of different load balancing policies is largely unknown. We also plan to implement an asynchronous global virtual time (GVT) algorithm allowing GVT calculation to proceed in parallel with execution of model events.(Wilmarth 2005)

Ultimately our interest in tackling real world models, planetary in scope and scale, are truly extreme in the world of PDES; hence we describe our efforts as working towards XPDES: eXtreme scale Parallel Discrete Event Simulation.

REFERENCES

Barnes, P. D., Jr., *et al.* (2013). Warp Speed: Executing Time Warp on 1,966,080 Cores. PADS 2013: ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. Montreal, Canada, ACM.

Kalé, L. V. and G. Zheng (2013). "The Charm++ Programming Model," in Parallel Science and Engineering Applications: the Charm++ approach. L. V. Kalé and A. Bhatele, Eds., CRC Press.

Quinlan, D., *et al.* (2013). ROSE Compiler Infrastructure. Livermore, CA, LLNL.

Vulov, G., *et al.* (2011). The Backstroke framework for source level reverse computation applied to parallel discrete event simulation. the 2011 Winter Simulation Conference (WSC)**:** 2960-2974.

Wilmarth, T. L. (2005). "Pose: Scalable general-purpose parallel discrete event simulation."