Asynchronous programming with Tarragon*

Pietro Cicotti
pcicotti@cs.ucsd.edu

Scott B. Baden
baden@cs.ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego, La Jolla CA 92093-0404, USA

Abstract

Tarragon is an actor-based programming model and library for implementing latency tolerant asynchronous event driven simulations. It is novel in its support for meta data describing run time virtualized process structures, which may be optimized as a free-standing object. We demonstrate early results with a synthetic benchmark, and observe that Tarragon can mask communication costs with ongoing computation.

1 Introduction

Event driven simulations model discrete entities that make scheduled changes over time. However, scalable simulators can be difficult to implement owing to high synchronization costs associated with fine grained event signaling and in maintaining causality. In addition, workloads often exhibit irregular time dependent behavior, and the required workload balancing disrupts locality, since processor boundaries tend to cut through highly active regions of event signaling. The skills required to cope with these details are beyond the reach of many programmers, and are difficult even for the expert.

We are investigating Tarragon, a run time library designed to overcome the above obstacles, which implements a non-bulk synchronous, actor-based programming model. We describe work in progress in using Tarragon to tolerate latency on clustered systems with low-cost switches where communication is at a premium. Tarragon is novel among asynchronous programming models [6] in its support for meta data, which carries information about run time virtual process structures that may be optimized to improve performance. We show that Tarragon can hide significant amounts of communication, so long as there is sufficient computational work to be overlapped. We plan to apply Tarragon to the MCell cell microphysiology simulator [2].

*This work was supported by NSF contract ACI-0326013

2 Tarragon

Tarragon supports self-timed process structures, reflecting the underlying behavior of event driven simulations. Fine grain events are streamed across processes, and the run time system adjusts communication, and hence event triggering granularity, at run time. The run time system runs in the background, and it offloads the overhead of managing parallelism and events from the simulator, removing much of the activity from the critical path of execution.

Tarragon's actor based execution model simplifies communication tolerant algorithm design by eliminating the need for complicated control flows traditionally used to choreograph communication and computation. Instead, the programmer expresses a partial ordering of tasks, and the Tarragon run time system handles the choreography automatically.

Tarragon employs virtualized process structures [4] where tasks are mapped many to a processor. Virtualization is useful in realizing overlap [6] and in load balancing [3]. A *TaskPlan* describes these process structures as a directed graph: vertices correspond to tasks to be executed, and edges to data dependences between the tasks. The TaskPlan is a form of computational meta data [5]. It may be decorated with attributes and subsequently used to improve performance, for example, to set scheduling preferences, aggregate messages, etc. The effect is to permit implementation policies like scheduling to be factored out of the application and isolated in separate software modules.

3 Implementation

Tarragon's run time services ("Services") manage actor semantics and run anonymously as logical background threads. The mapping of these logical threads to physical ones is installation dependent, and hidden from the view of the casual programmer. The Services form a separate communicating subsystem, detecting completed and runnable tasks, and managing synchronization needed to detect ter-

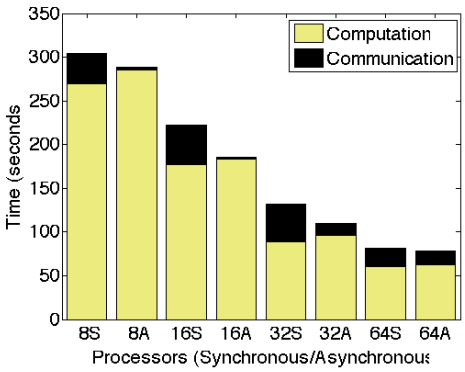

Figure 1. Tarragon overlaps communication with computation. The suffixes **S** and **A** refer to the synchronous and asynchronous variants, both running under Tarragon.

mination and manage causality. Services may run on one or more threads, with the exact number a tunable parameter. Although the Services consume resources, we have found that the cost is modest compared with the benefits.

When implementing distributed event queues, it is important that the simulator not have to enter a critical section each time it polls for a new event. The Tarragon Services update event queues without interfering with ongoing simulation, while also ensuring timely progress on incoming events. The strategy relies on computational meta data.

4 Current Status

A prototype of Tarragon has been implemented, which support the overlap of communication with computation, but not process virtualization. We use a synthetic benchmark called *Cells* to validate Tarragon's ability to realize overlap. Cells realizes a simplified version of the parallel time stepped version of the MCell cell microphysiology simulator. Molecules (like particles) carry out random walks in two dimensions, congregating and dispersing irregularly, migrating across processor boundaries.

Cells was run on the FWGrid cluster located in the UCSD CSE Department, comprising a mix of dual processor Xeon and Opteron nodes connected by 1 Gigabit ethernet[1]. Fig. 1 shows how running times are reduced when we overlap communication with computation, on up to 32 nodes. We are currently investigating the inability to realize overlap on 64 nodes. These results are encouraging: Tarragon is able to a significant portion of data transfer delays. A system builder can use this capability to forgo an expensive switch in favor of added processing and memory capacity.

[1] http://fwgrid.ucsd.edu

5 Discussion and Conclusions

A run time library called Tarragon has been proposed that supports the design of asynchronous, latency tolerant formulation of event driven simulations. Tarragon's programming methodology isolates scheduling, communication, and load balancing from simulation logic, offering performance and coding advantages compared with Bulk Synchronous Parallelism.

Formulating a highly scalable, latency-tolerant discrete event simulation is an open problem[2]. While there are results for algorithms arising in classical partial differential equations and numerical linear algebra [1, 7], significant and intrusive programmer intervention is required to reformulate the algorithm. Tarragon's asynchronous execution model is well matched to the task at hand, and provides a flexible testbed for experimenting with performance tradeoffs in implementing scalable event driven simulators.

References

[1] S. B. Baden and S. J. Fink. Communication overlap in multi-tier parallel algorithms. In *Proc. SC '98*, Orlando, Florida, November 1998.

[2] Gregory T. Balls, Scott B. Baden, Tilman Kispersky, Tom Bartol, and Terrence J. Sejnowski. A large scale monte carlo simulator for cellular microphysiology. In *Proceedings 18th International Parallel and Distributed Processing Symposium*, pages 42–51, Santa Fe, NM, April 2004.

[3] J. E. Flaherty, R. M. Loy, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Parallel structures and dynamic load balancing for adaptive finite element computation. *Applied Numerical Maths.*, 26:241–263, 1998.

[4] L. V. Kalé. The virtualization model of parallel programming : Runtime optimizations and the state of art. In *LACSI 2002*, Albuquerque, October 2002.

[5] P.H.J. Kelly, O. Beckmann, A. Field, and S. Baden. Themis: Component dependence metadata in adaptive parallel applications. *Parallel Processing Letters*, 11(4):455–470, December 2001.

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

[7] A. Sohn and R. Biswas. Communication studies of DMP and SMP machines. Technical Report NAS-97-004, NAS, 1997.

[2] Richard Fujimoto, private communication.