XTracker, A Graphical Tool for Parallel Simulations

Steven Bellenot

Department of Mathematics
Florida State University
Tallahassee, FL 32306-3027
bellenot@math.fsu.edu

Li Duty

Department of Computer Science
Florida State University
Tallahassee, FL 32306-4019
ma@cs.fsu.edu

Abstract

A Motif based graphical tool XTracker is described. XTracker can show Gannt-like charts of the activities on each node or it can show the event messages as traffic between simulation objects. XTracker can take its data from sequential simulation runs and simulate a parallel execution under a number of simulation methods. XTracker can act as a performance modeling tool.

Introduction

XTracker is an attempt to graphically visualize the execution of a parallel simulation. XTracker has a simplified view of a simulation, for example it assumes there is no overhead, messages are infinitely fast, and context switches take zero time. XTracker shows graphically which object (if any) each node is currenting executing at any real time. Rectangles representing an object's execution at a given virtual time can have different colors. These colors are assigned depending on if this execution is the final one at given virtual time, or if it will be later be rollbacked and re-executed. Like many graphical tools, XTracker allows for re-scaling and has means of identifying boxes too small to hold a text label. XTracker has another view which shows the event messages as traffic between simulation objects. XTracker was based on Tracker [2].

As a motivation, suppose one has a correct simulation which does not get good parallel speedup. One would like to be able to use XTracker to view the parallel execution and tell why it is not getting good parallel performance. We have a couple of examples where XTracker can find the problem (see parallelism and pucks below).

As usual we assume a simulation is divided into simulation objects which schedule events for each other via time-stamped messages. Casualty requires that these events be executed in increasing virtual time order. Parallel simulation methods usually assign simulation objects to physical nodes which then execute their potion of the simulation according to some simulation method.

XTracker currently only supports optimistic methods, but there is no reason it could not also support conservative methods. For a good introduction to parallel discrete event simulation see [4].

While executing a parallel simulation, each processor does a number of activities which can be considered overhead. This overhead includes message traffic, synchronization to preserve casualty, and state saving overhead including GVT and fossil collection. (All of the simulation object's variables are stored in a data structure called its state.) XTracker currently ignores all of the above activities. XTracker views a simulation as an execution of objects with no overhead and infinitely fast messages. XTracker does allow for events to be preempted when an earlier time-stamped appears on a node.

Currently XTracker simulates three optimistic methods, two risky methods in the sense of [8], and a riskfree method like that in SPEEDES ([9] and [10]). Execution of event can generate events (messages) for other objects. Risky simulation methods immediately send these messages. Thus an object executing in a wrong future could send a incorrect message (positive message) which will need to be unsent. A risky method must provide a mechanism, often anti-messages or negative messages, to unsend an incorrect (positive) message. When a negative and positive message meet in an object's queue they annihilate each other and rollback the object. A parallel simulation method can be riskfree just by not sending messages until it is certain they are correct, no anti-messages are needed. The difference in XTracker's two risky methods is based on their cancellation strategies [7] which is either aggressive or lazy.

We used the sequential simulator and the benchmarks Pucks and Bank from the TWOS project. The sequential simulator was used to generate the data files needed by XTracker. The Time Warp Operating System (TWOS) is the operating system implementation of Time Warp [5] done at the Jet Propulsion Laboratory (JPL).

The Simplified World of XTracker

XTracker uses three ascii input files: an event trace file, a message log file and a configuration initialization file. The event trace file has one line for each event in the simulation. The format of the trace file is that generated by the -t option to the TWOS sequential simulator, but includes the virtual time, the objects name, the amount of real cpu time used and a count of the number of input and output messages. The format of the msglog file is that generated by the -m option to the TWOS sequential simulator, and has one

Obj	VT	RTime	Msgs (Dst@Receive Time)
A	100	20	D@150 C@200 B@300
D	150	20	D@250
C	200	10	A@500
D	250	20	none
B	300	40	A@400
A	400	10	A@500
A	500	20	none

Table 1: The SE simulation.

line for each message listing the sender, the send time, the receiver, the receive time and the message selector an application defined message type. The format of the configuration file is based on the configuration file needed for TWOS. A shell script is used to massage the above data into the format needed by XTracker.

XTracker simulates a parallel execution (under one of the three methods above) and gives a simplified view of how the parallel simulation might execute. Only the "correct" events and messages are in the data files XTracker uses. Thus XTracker cannot simulate "wrong" events or messages, only "correct" ones that are executed out of order. As all simulations, it might not give realistic picture of any parallel execution of the simulation. However, if a simulation is not overloading the overhead of a parallel execution, then it is likely that XTracker will give a reasonable picture of this parallel execution. It is possible that the pictures drawn by XTracker will be too complex to provide simple answers.

XTracker as a teaching tool

We feel that XTracker has potential as an teaching tool to illustrate how parallel simulations methods execute. The simple example, SE, will also highlight some of the features of XTracker. Consider the following simple simulation given by Table 1. In Table 1, the first column gives the objects name, the second column the virtual time of the event, the next column is the amount of real time needed to execute the event and finally, the last column is a list of messages sent in the form destination at receive time. So the first line says object A executes at virtual time 100 for 20 units of real time and sends messages for object D at virtual time 150, for object B at virtual time 200 and for object C at virtual time 300.

The next two figures give snapshots of XTracker on this simple example simulation. Figure 1 is the aggressive method overview, note there are two executions of object A at virtual time 500. The darker color of the first indicates premature out-of-order execution, where as the lighter color indicates in-order or correct execution. The same simulation executed using the riskfree method would have the out-of-order execution for A at virtual time 500 at a later real time, because the message from C at virtual time 200 cannot be released until the event for D at virtual time 150 is done. (Object D could have sent a message to C before virtual time 200, which could have made C change its message to A.)

Figure 2 shows the message overview mode of

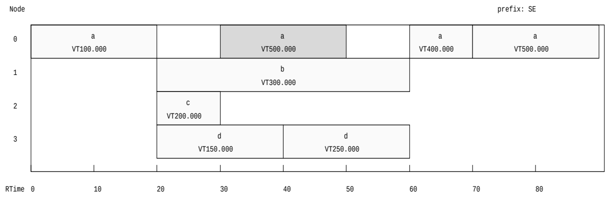

Figure 1: XTracker SE display for aggressive case.

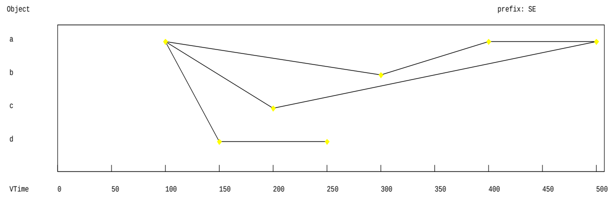

Figure 2: XTracker SE message display.

XTracker. It shows all the messages between objects showing the virtual times of sending and receiving. For SE it shows the simulation structure at a glance.

XTracker and Parallelism

Not all simulations are good candiates for parallel execution. If a simulation is inheritly sequential, no parallel method will make the simulation run significantly faster. Yet it is often hard to determine the amount of parallelism in a given simulation. Consider the benchmark Bank [1], with 8 objects bank_00 to bank_07 and message density 1. In this simulation there are 8 independent messages traveling from bank to bank. When a message arrives, both its receiver and receive time are randomly choosen and it is sent onward until the receive time exceeds a cutoff time.

How much parallelism does this version of bank have? Clearly it is less than 8, because one cannot expect the 8 messages to randomly spread over all 8 objects. Mathematically, one can show if one puts 8 balls into 8 boxes the expected number of non-empty boxes is 5.25 and hence the parallelism is likely less than 5.25. Figure 3 shows XTracker's display of this simulation in overview mode, which shows all events. This view is for aggressive cancellation. Figure 3 clearly shows that bank_01 is a sequential bottleneck. The maximal speedup is thus the sum of the execution times divided by the sum of the bank_01 execution times which is about 3.3. If we ignore the initialization phase at the start, the maximal speedup drops to 3.0.

Could we have predicted Figure 3? Another mathematical calculation about 8 balls into 8 boxes shows that the expected number of balls in the box with the most balls is 2.6 which is about 1/3 of 8. (Which suggests a maximal speedup of 3.) At the onset at least we can expect a third of the simulation to be sequential. At least in the simulation of Figure 3, the simulation never recovers from this initial imbalance. (Overall bank_01 receives 15 of the 54 messages or about 28% of the total.) Thus something which looks on the face value might have lots of parallelism is easily shown to be much more problemmatic. Figure 3

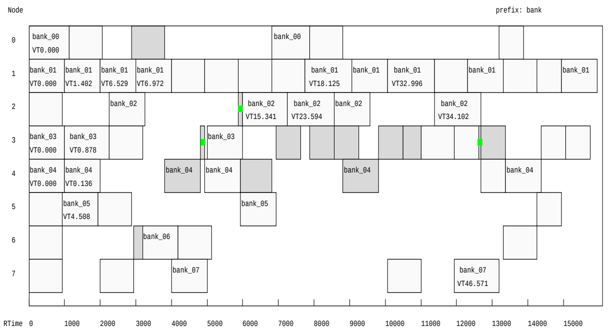

Figure 3: XTracker display for Bank.

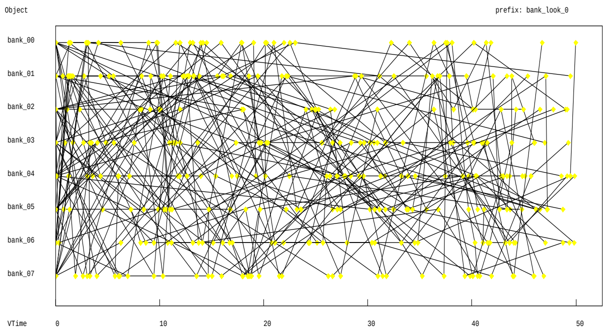

Figure 6: Riskfree Bank Messages with Lookahead 0.

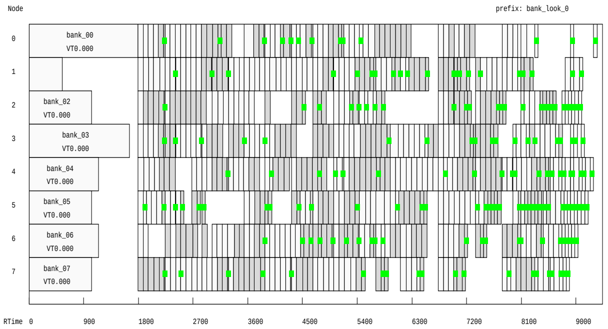

Figure 4: Riskfree Bank with Lookahead 0.

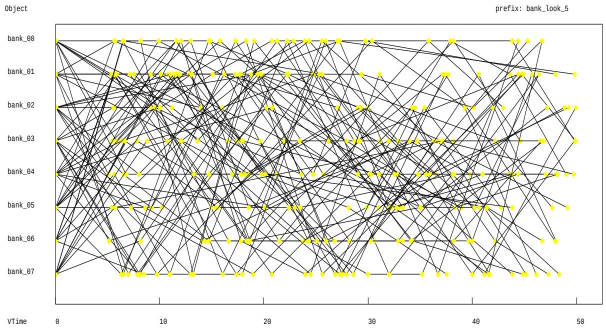

Figure 7: Riskfree Bank Messages with Lookahead 5.

shows XTracker can find sequential roadblocks in parallel executions.

Riskfree Lookahead

The lookahead of an object is the minimum virtual receive time minus the virtual send time of each message (event) it schedules. A simulation has good lookahead if the lookahead value is large compared to the usual event spacing. Good lookahead is needed to get good performance in conservation simulation methods [4] and some optimistic methods [1]. In general, good lookahead is a plus for any simulation method.

We have two figures to illustrate the value of lookahead in the riskfree case. The simulation is bank as in [1], this time with 64 messages or an message density of 8. The first run, Figure 4, has lookahead of zero (as does the run in Figure 3.). While the second run, Figure 5, has a lookahead of 5.

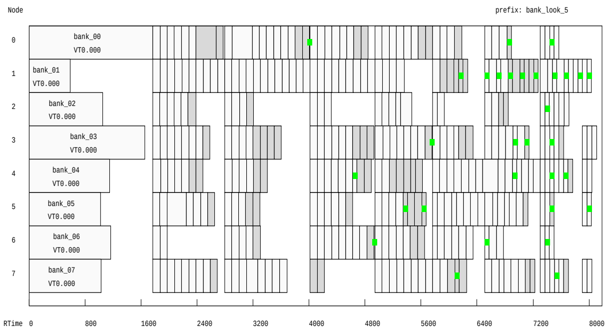

Figure 5: Riskfree Bank with Lookahead 5.

The difference in these two simulations runs by XTracker are several. The most notable is the lookahead 0 case takes over 15% longer to run than the lookahead 5 cases. The two simulations are not the same, the lookahead 0 case has more messages and events (about 12%) more, but the sum of the event run times is longer (about 4%) for the lookahead 5 case. We see more idle time in Figure 5, and more events which will be rollback in Figure 4. The message view of XTracker gives a good estimate of the lookahead in this case. The lookahead 0 case, Figure 6, has vertical lines with very steep slopes; at least in comparison with the slopes in Figure 7, the lookahead 5 case.

Assumptions made by XTracker in Simulating Parallel Executions

XTracker makes many simplifications in how simulations act. Besides the no overhead kinds of things mentioned before, there are significant assumptions on how out-of-order object executions behave. Most of these assumptions are cause by limitations in the data that would be hard to overcome. That is, to be truly accurate, one would have to collect kinds of data that would be hard to obtain. XTracker gives increasingly idealized views as it switches from riskfree to aggressive to lazy methods.

Benchmark Performances

Next we tried XTracker on the Benchmark Pucks [6]. This version of Pucks had the usual 128 sectors and 48 cushions but only 14 pucks and all of the XTracker drawings are done on 14 nodes, so that each node has exactly one puck. This is an attempt to see

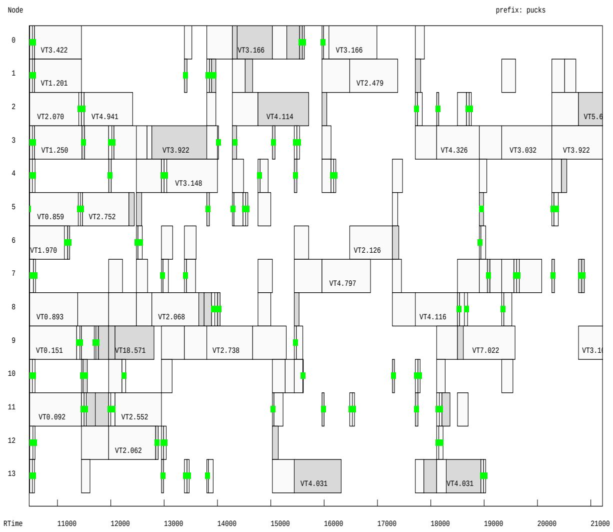

Figure 8: Pucks – Riskfree.

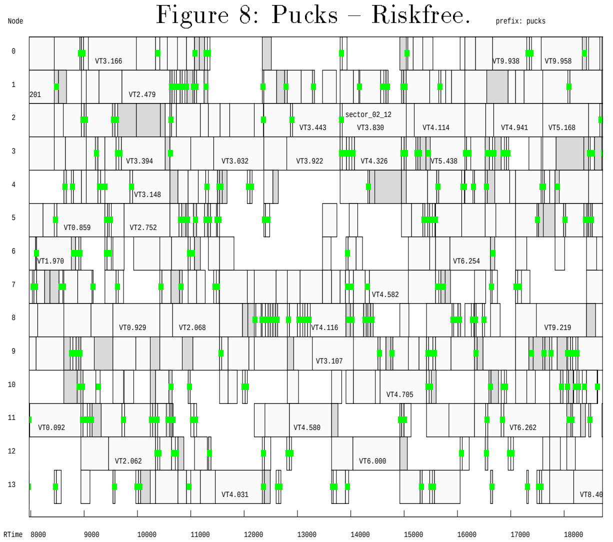

Figure 9: Pucks – Aggressive.

if XTracker can find the well known differences of running Pucks under different simulation methods. In [1], it is shown that Pucks is poorly designed for running under a riskfree simulation method.

The next couple figures show the part of the XTracker view of the Pucks simulation. Each figure has the same scale and shows the same amount of real time. Each figure is viewing the start of the simulation of the same simulation object at virtual time 111.xx on node 11. Figure 8 shows this XTracker drawing for Pucks in the riskfree mode. Figure 9 show this XTracker drawings for Pucks in the aggressive mode with the same scale as Figure 8. Not only is the amount of idle space much larger in Figure 8, but the compute run time in Figure 8 is over 2 times longer than either of the other risky methods.

Implementation and Conclusions

XTracker was implemented in C++ using Motif and a toolkit described in a textbook by Douglas Young [11]. This toolkit is example code from the text and is publically available. XTracker was developed on Sun Workstations using both SunOS 4.1.x and 5.x.

XTracker is an evolving graphic tool to help visualize parallel simulation execution. It provides some insight into the parallel workings of a simulation, perhaps too improve performance. It its useful in studying toy simulations and teaching simulation methods. For the model writer, if can point out limits in parallelism in some simulations. Also it can indicate which parallel simulation method might produce the best speedup for a given simulation.

References

[1] S. Bellenot, "State Skipping Performance with the Time Warp Operating System," 6th Workshop on Parallel and Distributed Simulation (PADS92), *SCS simulation series* Vol. 24, pp. 53-61, 1992.

[2] S. Bellenot, "Performance of a Risk Free Time Warp Operating System," 7th Workshop on Parallel and Distributed Simulation, *SCS simulation series* PADS-93-1, pp. 155-158, 1993.

[3] R. Fujimoto, "Performance measurements of distributed simulation strategies," *Trans. Soc. for Compul. Simul.* Vol. 6,2, pp. 89-132, 1989.

[4] R. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM,* Vol. 33 no 10, pp. 30-53, 1990.

[5] D. Jefferson, B. Beckman, *et. al,* "Distributed Simulation and the Time Warp Operating System," *Proc. 12th SIGOPS - Symposium of Operating Systems Principles,* pp. 77-93, 1987.

[6] P. Hontalas, B. Beckman *et. al,* "Performance of the colliding pucks simulation on the time warp operating systems (Part 1: Asynchronous behavior and sectoring)," Distributed Simulation, *SCS simulation series* Vol. 21, pp. 3-7, 1989.

[7] P. Reiher, R. Fujimoto, *et. al,* "Cancellation strategies in optimistic execution systems," Distributed simulation *SCS simulation series* Vol 22,1, pp. 112-121, 1990.

[8] P. Reynolds, "A Spectrum of Options for Parallel Simulation Protocols," *Proc of ACM Winter Simulation Conference,* pp. 325- 332, 1988.

[9] J. Steinman, "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation," Advances in Parallel and Distributed Simulation, *SCS simulation series* Vol. 23, pp. 95-103, 1991.

[10] J. Steinman, "SPEEDES: A Unified Approach to Parallel Simulation," 6th Workshop on Parallel and Distributed Simulation (PADS92), *SCS simulation series* Vol 24, pp. 75-84, 1992 .

[11] D. Young, *Object-Oriented Programming with C++ and OSF/Motif,* Prentise Hall, 1992.