RTI Performance on Shared Memory and Message Passing Architectures

Steve L. Ferenci
Richard Fujimoto, PhD
College Of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{ferenci,fujimoto}@cc.gatech.edu

Keywords:
HLA, RTI, Time Management, Shared Memory.

ABSTRACT: *This paper compares the performance of HLA time management algorithms across three computing architectures: 1) a shared memory multiprocessor (SGI Origin), 2) a cluster of workstations interconnected via a low latency, high-speed switched network (Myricomm's Myrinet), and 3) a traditional LAN using TCP/IP. This work is based on the RTI-Kit software package described in a paper presented at the Fall 1998 SIW. This software implements group communication and time management functions on the platforms described above. A time management algorithm designed to efficiently compute LBTS values on shared memory multiprocessors is described. This algorithm exploits sequentially consistent shared memory to achieve very efficient time management. We present results comparing the performance of this time management algorithm with a message-passing algorithm implemented over shared memory.*

1. Introduction

The U.S. Department of Defense (DoD) mandated that the High Level Architecture (HLA) be the standard architecture for DoD modeling and simulation programs [1]. This requires that the HLA span a wide range of applications, computing architectures, and communication paradigms.

Previously most implementations of the High Level Architecture (HLA) Run Time Infrastructures (RTIs) have been implemented on a Network of Workstations (NOWs). Typically these workstations are connected via TCP/IP networks or other high speed interconnects such as Myricomm's Myrinet [2].

Recently Shared Memory Multiprocessors (SMPs) have been considered as a platform for HLA RTIs. SMPs offer a different communication paradigm. An HLA RTI implemented over a NOW must use message passing to transfer information between workstations, e.g., using UDP or TCP packets over a TCP/IP network or Fast Messages (FM) [3] over a Myrinet network. In SMPs communication between processors is realized through the use of shared memory. For example, to transfer information from one processor to a second, the first processor writes to a memory location in shared memory and signals the second processor to read from the same location.

In this paper a shared memory time management algorithm is presented. This algorithm has been implemented as an RTI-Kit [4] library and its performance is compared to an existing message passing time management algorithm.

2. RTI-Kit

RTI-Kit is a set of libraries designed to support the development of Run Time Infrastructures (RTIs) for parallel and distributed simulation systems. Each library is designed so that it can be used separately or together with other RTI-Kit libraries.

RTI-Kit consists of three main libraries, see Figure 1. FM-Lib implements communication over the computer platform. TM-Kit implements time management functions. MCAST implements group communication. Both TM-Kit and MCAST use FM-Lib to carry out their respective functionality. By using a structure such as this it makes it possible to develop RTIs that are independent of the computer

platform. To implement an existing RTI on a new computer platform it only becomes necessary to build the appropriate FM-Lib library for that computer platform. To take full advantage of the properties of a particular architecture it is possible for MCAST or TM-Kit to bypass FM-Lib.

The libraries in RTI-Kit are self-contained i.e., the internal structure of one library has no dependence on the internal structure of another. The only dependence between libraries is the interface presented by one library to another. This makes it straightforward to develop a new library with the same functionality as an existing library. As long as the interface between libraries is maintained no changes to other libraries will have to be made.

For the experiments performed for this paper several variations of RTI-Kit were developed to operate on different computer platforms. Each of the variations differs only in the version of FM-Lib and TM-Kit used.

.

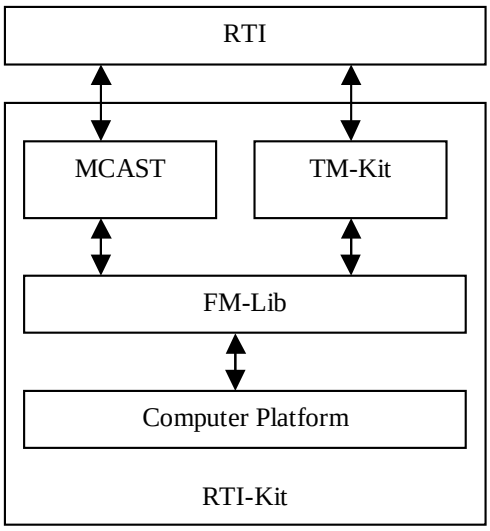

Figure 1 Architecture of an RTI using RTI-Kit

3. TM-Kit

TM-Kit is a library that implements time management functions. The primary purpose of TM-Kit is to compute the Lowest Bound Time Stamp (LBTS); LBTS is the lowest time stamp of a message that a processor can expect to receive from another processor in the future.

When calculating LBTS a mechanism is needed to determine the global minimum time. In order to determine the global minimum time each processor needs some way to communicate its local minimum time to the other processors. Here we present two algorithms that use different communication paradigms to calculate LBTS. The first algorithm uses message passing to communicate, and the second algorithm uses shared memory.

Each of these algorithms is implemented as a TM-Kit library. The message passing algorithm will be referred to as Message Passing TM-Kit (MP TM-Kit), and the shared memory algorithm will be referred to as Shared Memory TM-Kit (SHM TM-Kit). The architecture of RTI-Kit using MP TM-Kit is exactly the same as that shown in Figure 1. However, SHM TM-Kit does not use FM-Lib, instead it is directly linked to the computer platform. In Figure 1 this would be seen as a double arrow between the TM-Kit box and Computer Platform box and no double arrow between the TM-Kit box and FM-Lib box.

3.1 Message Passing TM-Kit (MP TM-Kit)

When an LBTS computation is started using MP TM-Kit the processors have to send their local minimum time to the other processors by using communication primitives defined by FM-Lib. This algorithm works by having a processor send its local minimum time to another processor at each step of the LBTS computation. When a processor receives a message containing the local minimum time of another processor it computes the pair wise minimum between its local minimum time and the local minimum time contained in the message received. This continues for log N steps, where N is the number of processors. The computation is completed when a processor has received a message containing the local minimum time from log N other processors. The LBTS is the result of the final pair wise minimum operation performed. A more complete description of this algorithm can be found in [4].

3.2 Shared Memory TM-Kit (SHM TM-Kit)

Unlike the previous version of TM-Kit, SHM TM-Kit does not use the FM-Lib library to send messages between processors to calculate LBTS; all communication is done implicitly through the use of

variables in shared memory. As stated earlier, SHM TM-Kit communicates directly with the computer platform and not through FM-Lib.

This algorithm assumes a sequentially consistent memory model for memory access [5]. A sequentially consistent memory model is one in which memory accesses by a processor to memory appear to occur in some global ordering and the references by each processor appear to occur in the order specified by that processor. For example, if processor 1 issues memory references M_1, M_2, M_3 and processor 2 issues memory references M_a, M_b, M_c then M_1, M_a, M_2, M_b, M_c, M_3, is a sequentially consistent ordering of the memory references. However, M_a, M_c, M_1, M_2, M_b, M_3 is not because memory reference M_c appears before M_b, and this is not the order specified by the program. This memory model allows the development of simpler more efficient shared memory algorithms than would be possible using a message passing algorithm.

The computation of the LBTS is separated into three stages corresponding to the three critical procedures needed to implement the algorithm. They are 1) starting an LBTS computation, 2) sending messages, 3) reporting local minimum time and calculating LBTS. This algorithm is based on the Global Virtual Time algorithm presented in [6].

Required Variables

All of the communication required to compute the LBTS between the processors is done through three variables in shared memory: GVTFlag (an integer), PEMin (an array to store the local minimum time of each processor), and GVT (stores the newly computed LBTS). In addition to the variables in shared memory each processor needs two variables in local memory SendMin and LocalGVTFlag. Their use will be explained shortly.

Stage 1: Starting an LBTS Computation

To start an LBTS computation a processor first tests if GVTFlag is equal to zero. If GVTFlag is equal to zero it indicates that no other LBTS computation is in progress and therefore it is safe to start a new LBTS computation by setting GVTFlag equal to N, where N is the number of processors. The process of testing and setting GVTFlag is a critical section, i.e., only one processor is allowed to test and set GVTFlag at a time to eliminate the possibility that two processors can set GVTFlag equal to N simultaneously.

Stage 2: Sending Messages

In order to calculate the LBTS a processor keeps track of the minimum time stamp of sent messages; this value is stored in SendMin. Though SHM TM-Kit does not use FM to send any messages other libraries of RTI-Kit may use FM so it is vital that SHM TM-Kit be aware of these out going messages to accurately compute the new LBTS. Recording the minimum time stamp of out going messages is only done when an LBTS computation is in progress, GVTFlag is greater than zero, and the processor has not yet reported its local minimum time.

Stage 3: Reporting the Local Minimum Time and calculating the LBTS

Periodically each processor has to give time to TM-Kit to test GVTFlag to determine if an LBTS computation is in progress. The most natural location to place this test is in the main event scheduling loop of the simulation. If the processor does not perform this test often the completion of an LBTS computation will be delayed degrading performance.

Each processor first saves a local copy of GVTFlag by setting LocalGVTFlag equal to GVTFlag. The processor then processes any pending incoming messages. Despite the fact that SHM TM-Kit does not use FM-Lib for communication the other libraries of RTI-Kit may be using FM. It is important to allow the processor to process any messages that have arrived via FM as these messages may affect the local minimum time of the simulation. Then if LocalGVTFlag is greater than zero and if this processor has not yet reported its local minimum time it reports the local minimum time as the minimum of SendMin and the time reported by the RTI. The minimum time reported by the RTI is obtained via a call back into the RTI. The local minimum time is placed in the processor's entry in PEMin.

Next, GVTFlag is decremented and the processor to decrement GVTFlag to zero is responsible for computing the new LBTS. The new LBTS is the minimum of all the entries in PEMin and this value is placed in GVT. This process of reporting the local minimum time and calculating the new LBTS is a critical section so only one processor can perform this operation at any given time. Once the new LBTS value is computed each RTI is notified through a call back into the RTI.

Constants

```
int N;     /* number of processors */
int Pid;   /* processor id number   */
```

Global Variables

```
int GVTFlag;
TM_Time  PEMin[N];  /* local minimum of each
                       processor */
TM_Time GVT;          /* computed LBTS */
```

Local Variables

```
TM_Time SendMin;
int LocalGVTFlag;
```

Procedure to start an LBTS computation

```
StartLBTS()
 begin critical section
     if (GVTFlag =  0) then
         GVTFlag = N;
     end if
 end critical section
```

Procedure to be called when sending a message. TS is the time stamp of the message being sent.

```
TM_Out(TS)
   if ((GVTFlag > 0) and
     (haven't already computed local min)) then
         SendMin = min(SendMin,TS);
   end if
```

Procedure to be called in the main event processing loop. RTIMinTime is the minimum time reported by the RTI.

```
TM_Tick()
   LocalGVTFlag = GVTFlag;
   Process any pending incoming messages
   if ((LocalGVTFlag > 0) and
      (haven't already computed local min)) then
       begin critical section
           GVTFlag = GVTFlag – 1;
           PEMin[Pid] = min(SendMin,
                            RTIMinTime);
           if (LocalGVTFlag = 0) then
              GVT = min(PEMin[0] … PEMin[N]);
           endif
        end critical section
    endif
```

Figure 2 Implementation of LBTS algorithm in SHM TM-Kit

4. FM-Lib Library

The FM-Lib library provides the means to allow federates to communicate with each other. Different versions of FM-Lib have been implemented to operate on different computing platforms and communication networks. An RTI can be made to run on a different platform with no change to the RTI by simply plugging in the appropriate FM-Lib for the platform in use.

Three version of FM-Lib have been implemented Myrinet FM-Lib (MYR FM), TCP FM-Lib (TCP FM), and Shared Memory FM-Lib (SHM FM). All three versions of FM-Lib implement the FM interface.

MYR FM uses FM written for Myrinet. Myrinet is a high speed, low latency switched interconnect network with a bandwidth of 640 Mbits/sec. TCP FM uses a traditional TCP/IP network for communication.

Unlike the previous two versions of FM-Lib SHM FM does not use a network to communicate instead communication is achieved through the use of shared memory. An SMP is a computer with N (>1) processors and a pool of memory that each processor has access. Each processor that is participating in the simulation allocates memory from the shared memory to act as a message in buffer. When processor *a* wants to send a message to processor *b*, processor *a* writes to processor *b*'s in buffer. When processor *b* detects a message in its in buffer it reads the message. By using this recipe SHM FM achieves the same behavior inherent to MYR FM and TCP FM.

5. RTI Implementations

Using the two TM-Kit and three FM-Lib libraries four implementation of an RTI were developed to gather performance results. The RTI used, called DRTI, implements a subset of the HLA Interface Specification services. For example, Time Advance Request, Time Advance Grant, Next Event Request, Update Attribute Values, Reflect Attribute Values are implemented in DRTI.

DRTI also implements the use of Attribute Handle Value Pair Sets as the mechanism to the Update Attribute Values service. Attribute Handle Value Pair Sets are simply a container to hold the attribute's handle (its name) and the attribute's value. The container holding this information is what is transmitted when performing attribute updates.

The four implementations implemented differ only in the FM-Lib and TM-Kit used. Figure 3 shows each of the four implementations.

Implementation	FM-Lib	TM-Kit
MYR DRTI	MYR FM	MP TM-Kit
TCP DRTI	TCP FM	MP TM-Kit
MP DRTI	SHM FM	MP TM-Kit
SHM DRTI	SHM FM	SHM TM-Kit

Figure 3 Various implementations of DRTI

MYR DRTI was executed on a network of eight Ultra Sparc workstations connected via Myricomm's Myrinet network. Myricomm's Myrinet network is a high speed, low latency switched interconnect network with a bandwidth of 640 Mbits/sec. TCP DRTI was executed on a network of eight Ultra Sparc workstations connected via TCP/IP network. The TCP/IP network is a standard 10BaseT Ethernet with a bandwidth of 100 Mbits/sec. Both MP DRTI and SHM DRTI were executed on an SGI Origin 2000 Symmetric Multiprocessor [7]. The SGI Origin 2000 used is configured with sixteen 195MHz R10000 processors with 4Mbytes of secondary cache and 4Gbytes of shared memory and provides a sequentially consistent model for memory access.

6. Performance Measurements

Three benchmarks are used to evaluate the performance of the various implementations of DRTI. The first benchmark established the performance of the underlying communication medium used. The other two benchmarks, [8], directly compare the performance of the different implementations of DRTI. The Latency benchmark measures the performance of the communication services of an RTI. The Time Advance Grant benchmark measures the performance of the time management services of an RTI.

6.1 Underlying Communication Performance

The FM-Lib library used makes a significant impact on RTI performance. To better understand RTI performance a comparison of the three FM-Lib libraries is in order. Figure 4 shows the one-way latency of sending a message of size N. The one-way latency is obtained by measuring the round-trip latency of sending a message of size N between two nodes and dividing by two. SHM FM achieves the lowest latency for sending a message followed by MYR FM and then TCP FM.

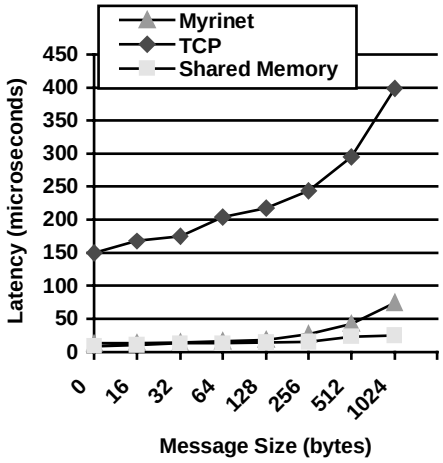

Figure 4 Latency measurements for three FM-Lib implementations

6.2 The Latency Benchmark

The latency benchmark measures the latency for best effort, receive ordered communications. This program uses two federates, and round-trip latency was measured. Specifically, latency times report federate-to-federate delay from when the UpdateAttributeValues service is invoked to send a message containing a wallclock time value and a payload of N bytes until the same federate receives an acknowledgement message from the second federate via a ReflectAttributeValues callback. The acknowledgement message has a payload of zero bytes. The one-way latency time is reported as the round trip time divided by two.

Figure 5 shows the results for the two implementations running on the network of Ultra Sparc workstations. As expected the performance of MYR DRTI is an order of magnitude better than is TCP DRTI. Myrinet achieves this speed up by avoiding the cost of having to marshal data through a protocol stack.

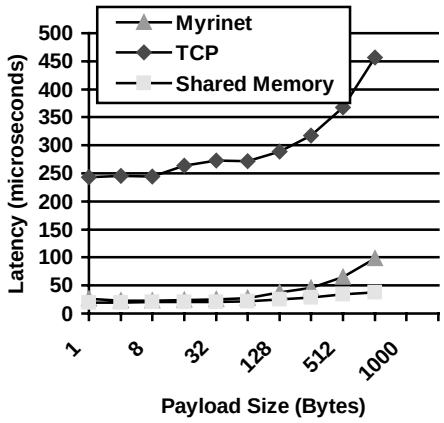

Figure 5 Latency measurements for Myrinet DRTI and TCP DRTI

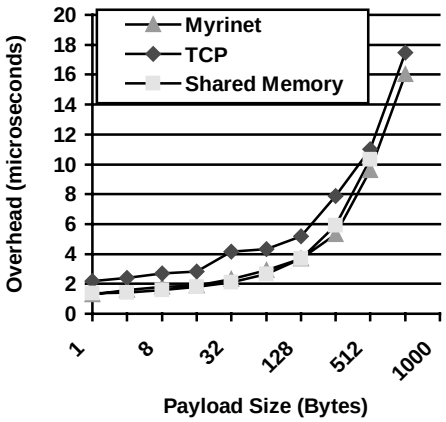

Figure 6 Time to copy one N byte attribute into an Attribute Handle Value Pair Set

The results for the two implementations running on the Origin 2000 are shown in Figure 6. The performance of the two implementations is nearly identical. Since the time required to send attribute updates is being measured and both versions are using SHM FM latency times are expected to be the same.

DRTI implements Attribute Handle Value Pair Sets as the mechanism to perform the Update Attribute Values service. It is required to first copy the attribute value into an Attribute Handle Value Pair Set and then calling the Update Attribute Values service. Copying this data incurs some overhead and can make a significant impact on performance depending on which FM Lib is in use and the size of the attribute. Figure 6 shows the overhead of copying one N byte block of data into an attribute handle value pair set. Figure 7 shows the overhead of copying M eight byte attributes.

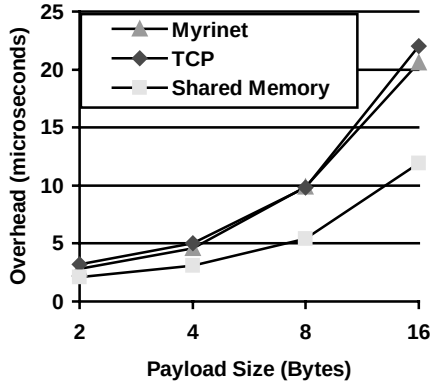

Figure 7 Time to copy M 8 byte attributes into an Attribute Handle Value Pair Set.

6.3 The Time Advance Request Benchmark

The TAR benchmark measures the performance of the time management services, and in particular, the time required to perform LBTS computations. This benchmark contains N federates, each repeatedly performing TimeAdvanceRequest calls with the same time parameter, as would occur in a time stepped execution. The number of time advance grants observed by each federate, per second of wallclock time is measured for up to eight processors.

Figure 8 shows the number of TAGs per second achieved by MYR DRTI and TCP DRTI. Again the

performance of MYR DRTI is orders of magnitude better than TCP DRTI and as before is due to the Myrinet's low communication overhead.

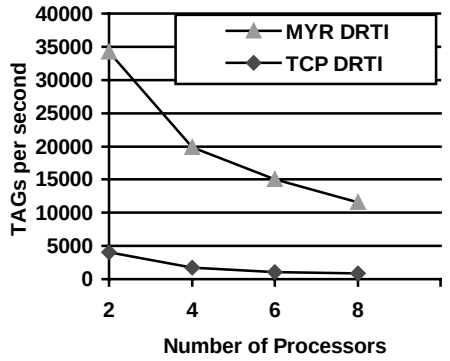

Figure 8 Time Advance Request measurements for Myrinet DRTI and TCP DRTI

Figure 9 shows the number of TAGS per second achieved by MP DRTI and SHM DRTI. In both implementations SHM FM is used yet SHM DRTI achieves higher performance. The difference between the two implementations is the time management algorithm, SHM TM-Kit as opposed to MP TM-Kit. By using a time management algorithm tailored to take advantage of shared memory SHM DRTI is able to achieve higher performance. However, when run on eight processors SHM DRTI does not perform as well as MP DRTI.

7. Conclusions

The experiments here show that it is possible to implement RTIs on SMPs using time management algorithms specifically designed to take advantage of sequentially consistent shared memory. For less than eight processors, the performance of an RTI using the shared memory time management algorithm when compared to a message passing time management algorithm is better. However, on eight processors the shared memory algorithm does not perform as well.

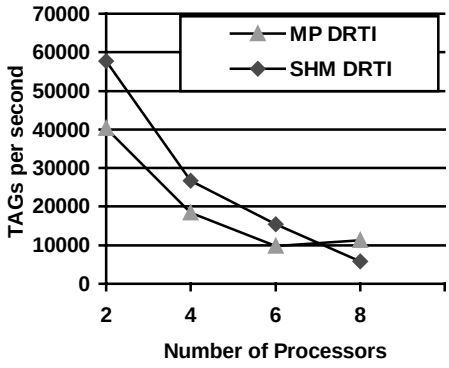

Figure 9 Time Advance Grant measurements for MP DRTI and SHM DRTI

We believe the decrease in performance for SHM TM-Kit on eight processors is due to contention for locks used in the critical sections of the shared memory algorithm. The more processors that participate in an LBTS computation increase the likelihood that there will be contention for the lock. Another cause for the degradation in performance as the number of processors is increased is traffic between the processors and memory. More processors mean more information will be transferred between the shared memory and the individual processors. Just as in a traditional network the more entities that try to send and receive information degrades overall system performance.

However, the use of SMPs to run HLA RTIs is promising. With more efficient implementations of time management algorithms (and efficient group communication algorithms,) SMPs will offer an efficient platform on which to execute RTIs.

8. Acknowledgments

Thanks to Ivan Tacic for the TCP implementation of FM-Lib. Thanks to Debbie Esslinger for her implementation of DRTI.

9. References

[1] Defense Modeling and Simulation Office Web Site, http://hla.dmso.mil.

[2] The Myrinet – A brief technical overview, http://www.myri.com.

[3] S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, J. Prusakova, A. Chein "Fast Message (FM) 2.0 User documentation" Dept of Computer Science University of Illinois at Urbana Champaign.

[4] R. M. Fujimoto and P. Hoare: "HLA RTI Performance in High Speed LAN Environments" Proceedings of the Fall Simulation Interoperability Workshop, September 1998.

[5] L. Lamport, "How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs," *IEEE Transactions on Computers*, Vol. C-28, No. 9, pp. 690-691, 1979.

[6] R. M. Fujimoto and M. Hybinette: "Computing Global Virtual Time in Shared-Memory Multiprocessors" *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 4, pp. 425-446, 1997.

[7] SGI Origin 2000 Technical Specifications Web Site, http://www.sgi.com/origin/2000_specs.html.

[8] J. Dahmann, R. Weatherley et al, "High Level Architecture Performance Framework", ITEC, Lausanne, Switzerland April 1998

Author Biographies

Steve L. Ferenci is a graduate student in the College of Computing at the Georgia Institute of Technology. He received a B.A. in Computer Science and Mathematics from Rutgers University is 1996. He is in his second year of graduate study and is a student of Richard Fujimoto.

Dr. Richard Fujimoto is a professor in the College of Computing at the Georgia Institute of Technology. He received the Ph.D. and M.S. degrees from the University of California (Berkeley) in 1980 and 1983 (Computer Science and Electrical Engineering) and B.S. degrees from the University of Illinois (Urbana) in 1977 and 1978 (Computer Science and Computer Engineering). He has been an active researcher in the parallel and distributed simulation community since 1985 and has published over 100 technical papers on parallel and distributed simulation. He led the working group responsible for defining the time management services for the DoD High Level Architecture (HLA) effort. Fujimoto is an area editor for ACM Transactions on Modeling and Computer Simulation. He also served on the Interim Conference Committee for the Simulation Interoperability Workshop in 1996-97 and was chair of the steering committee for the Workshop on Parallel and Distributed Simulation (PADS) in 1990-98.