Analysis of High Performance Communication and Computation Solutions for Parallel and Distributed Simulation*

Luciano Bononi, Michele Bracuto,
Gabriele D'Angelo, and Lorenzo Donatiello

Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna,
Via Mura Anteo Zamboni 7, 40126, Bologna, Italy
{bononi, bracuto, gdangelo, donat}@cs.unibo.it

Abstract. This paper illustrates the definition and analysis of a collection of solutions adopted to increase the performance of communication and computation activities required by the implementation and execution of parallel and distributed simulation processes. Parallel and distributed simulation has been defined, and a real testbed simulation scenario has been illustrated, based on the ARTÌS simulation framework. Three classes of solutions have been proposed to improve the performance of simulations executed over commodity off-the-shelf computation and communication architectures: multi-threaded software and Hyper-Threading support by the processor architectures, data marshalling solutions for shared-memory and network-based communications, and data structure optimization for simulation events' management. All the proposed solutions have been evaluated on a testbed evaluation scenario, under variable configurations. Results obtained demonstrate that a performance improvement can be obtained by adopting and tuning the proposed solutions.

1 Introduction

"A simulation is a system that represents or emulates the behavior of another system over time. In a computer simulation the system doing the emulating is a computer program" [13]. The computer simulation is a widely adopted technique to obtain "a priori" insights of behavioral issues, and performance evaluation of theoretic, complex and dynamic systems and system architectures. At an abstract level, a simulation can be seen as a process execution managing a huge set of state variables: each variable update is activated by a simulated event. Every update may require a complex state computation, and would represent a step in the behavior of a portion of the simulated system. The simulation can be implemented by one (single or monolithic) process, or more than one (parallel or distributed) processes. Monolithic simulations may suffer the bottleneck limitation of memory and computation, being executed over single CPUs. On the

* This work is supported by MIUR FIRB funds, under the project: "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

L.T. Yang et al. (Eds.): HPCC 2005, LNCS 3726, pp. 640–651, 2005.

other hand, parallel and distributed simulations could exploit aggregate memory and computation resources. Unfortunately, parallel and distributed simulations may suffer the bottleneck limitation of the communication system required to support the huge amount of synchronization and communication messages between multiple synchronized processes. The aim of this paper is to introduce main motivations and new dimensions for the research about scalability and efficiency issues of simulation frameworks executed over general purpose system architectures. Specifically, in this paper some recently introduced solutions for the processor architectures and communication network technologies have been preliminary investigated. The results presented involve the performance analysis and guidelines derived about the mixed adoption of Hyper-Threading processors, single-threaded and multi-threaded software architectures, data marshalling solutions for communications and data structure optimizations. Results obtained confirm that performance speedup can be obtained by considering and exploiting the proposed features, by offering experimental evidence to practical guidelines and limitations.

The paper structure is the following: section II illustrates some general concepts about the execution of simulation processes, that will be useful to define the assumptions and guidelines for subsequent work, section III will illustrate the state of the art in the field of parallel and distributed simulation and will introduce the simulation framework that we adopt as a testbed for our analysis (ARTÌS), section IV will illustrate the analysis of the Hyper-Threading features of processors in the context of parallel and distributed simulations, section V will illustrate the Data Marshalling concept and analysis to improve the communication efficiency, section VI will illustrate the simulation Data Structure optimization concepts and related analysis, and section VII will conclude the paper.

2 Simulation: Assumptions, Systems and Optimization

The aim of a simulation is to create a synthetic evolution of the system model, in order to capture data about the behavior of the model counterpart, that is, a real system behavior. The evolution of model entities could be defined as the history of state updates as a function of the simulated time. The system entities' evolution is emulated by a computer program that mimics their causal sequence of fine-grained state transitions, that is, the system events.

In the legacy approach for a computer simulation, the simulation software is executed on a single processing unit, by obtaining a monolithic simulation. Memory and computation bottlenecks may limit the model scalability and often require huge amount of time to complete the analysis. The need to evaluate complex systems with thousands or even millions simulated entities is often impossible to satisfy due to resources' limitations [12]. An alternative approach is based on the exploitation of parallel and distributed communication and computation systems [13]. The advantage of Parallel and Distributed Simulation (PADS) techniques is given by the exploitation of aggregate resources (memory and computation power of a collection of Physical Execution Units, PEUs) and

by the potential exploitation of the model concurrency under the model-update and state-computation viewpoint. This may translate in a reduction of the computation time required to complete the analysis of the model evolution in a given scenario. A PADS framework is composed by a set of cooperating computation units managing the model state evolution in distributed way. The simulation is partitioned in a set of Logical Processes (LPs), each one managing the evolution of a subset of simulated model entities.

3 Parallel and Distributed Simulation: State of the Art

The parallel and distributed simulation (PADS) field has been widely studied in the past, resulting in the design and implementation of many tools to assist the simulation processes. In recent years, a good number of PADS tools have been proposed: Maisie [6], PDNS [7], DaSSF [1],TeD [2,15], Glomosim/Qualnet [8]. Unfortunately, weak performances and the lack of a standard have limited the adoption and interoperability of tools, and the model reuse, with low potential impact of PADS technology on real world applications. After the year 2000, under the auspices of the US Department of Military Simulation Office (DMSO), the IEEE 1516 standard for distributed simulation (High Level Architecture, HLA) has been approved [5]. ARTÌS (Advanced RTI System) is a recently proposed middleware, designed to support parallel and distributed simulations of complex system models [10,9]. A number of existing runtimes compliant to the HLA IEEE 1516 standard are available. Some runtimes suffer of implementation problems, the source code is often not available, and they miss interesting features (as entity migration support, and security). These observations stimulated the design of the ARTÌS middleware. In ARTÌS, many design optimizations have been realized for the synchronization and communication protocols' adaptation over heterogeneous execution systems (basically, Local Area Network and Shared Memory mono- and multi-processor architectures). The communication and synchronization middleware in charge of the adaptation is completely user-transparent. The middleware is able to select in adaptive way the best communication module with respect to the dynamic allocation of LPs over the execution environment. The ARTÌS runtime (RTI) kernel is realized on the top of the communication modules, and it is composed by a set of management modules, whose structure and roles have been inherited by a typical HLA-based simulation middleware.

In next sections, by following a bottom-up approach, we will introduce some details about additional optimizations that can be applied to ARTÌS (and other PADS architectures), by exploiting high computation and communication performance solutions.

4 Exploiting Advanced Processor Features: HT

The Hyper-Threading technology (HT) is a new processor architecture recently introduced by Intel [14,4]. HT technology makes a single physical processor appearing as two logical processors at the user level. To achieve best performances,

the operating system should natively support the HT technology. In general, one physical execution resource (CPU) is shared between two logical processors. To obtain this effect, with low overheads introduced, the HT technology duplicates the high level portion of the architecture state on each logical processor, while logical processors share a subset of the physical processor execution resources. Some experimental results from Intel [14,4] have shown an improvement of CPU resources' utilization, together with higher processing throughput, for multi-threaded applications with respect to single-threaded executions. Under optimal assumptions and conditions, the performances shown a 30% increase. To the best of our knowledge, the influence of HT technology on PADS architectures and frameworks has not been investigated in detail. Thanks to simple heuristics, HT-enabled OSes should be able to adapt the process scheduling to the HT architecture, with the aim of optimizing the overall execution of processes. On the application side, it is quite common for parallel and distributed simulation frameworks to allocate a single LP for each available processor. This is based on the assumption that one single LP will be the only running process and would not cause context switches and other relevant overheads. On the other hand, each time the LP is blocked due to communication and synchronization delays, the CPU time would be wasted [11]. The effects of HT technology could significantly change the assumptions related to current implementation choices. Under the software architecture viewpoint, most of the modern PADS middlewares are based on multi-threaded implementation, and basically should take advantage of the HT support. It would be interesting to evaluate if the increasing in the number of logical processes could be exploited as a new dimension for PADS optimization: to concurrently run more LPs than the number of physical processors, under HT processor architectures.

4.1 The Experimental Testbed

To give answers to the above questions about HT technology and PADS assumptions, we evaluated the performances of the real ARTÌS simulation framework on a real experimental testbed, instead of relying on synthetic CPU benchmarks. First, we defined a scalable model of a complex and dynamic system, whose definition contains many of the worst model assumptions that has been identified as stressing conditions under the PADS framework optimization and simulation execution performances viewpoints: the wireless mobile ad hoc network model. The model is composed by a high number of simulated wireless mobile hosts (SMHs), each one following a Random Mobility Motion model (RMM) with a maximum speed of 10 m/s. This mobility model is far from being real, but it is characterized by the completely unpredictable and uncorrelated mobility pattern of SMHs. The system area is modeled as a torus-shaped bi-dimensional grid-topology, 10.000x10.000 space units. The torus area, indeed unrealistic, allows to simulate a closed system, populated by a constant number of SMHs. The torus space assumption is commonly used by modelers to prevent non-uniform SMHs concentration in any sub-area. The simulated space is flat and open, without obstacles. The modeled communication pattern between SMHs is

a constant flow of ping messages (i.e. constant bit rate), transmitted by every SMH in broadcast to all neighbors within a wireless communication range of 250 spaceunits.

4.2 The Experimental Results

All the experiments and the analysis results shown in this paper are based on the parallel and distributed simulation of the wireless ad hoc model, under the control, optimization and management of the ARTÌS runtime (Section 3). We performed multiple runs for each experiment, and the confidence intervals obtained with a 95% confidence level (not shown in the figures) are lower than 5% the average value of the performance index. The experiments collected in this section have been executed over a PEU equipped by Dual Xeon Pentium IV 2800 Mhz, 3 GB RAM. The experiments have been divided in two groups: the first group is based on Hyper-Threading support enabled for the PEU (HT-ON), and the second one is based on Hyper-Threading support disabled directly by BIOS settings (HT-OFF). The ARTÌS implementation adopted in this analysis is itself multi-threaded, and takes advantage of the multi-threading support to manage the execution of LPs: each LP is composed by at least 3 threads (main, shared memory and network management). The standard ARTÌS implementation implements a timed wait synchronization mechanism between the threads that compose each LP. Alternative solutions for implementing communication between the threads could be based on busy waiting, or signalling-based implementations.

Figure 1 shows the wall-clock time (WCT) required to complete one simulation run, taken as a reference. The reference run is defined as the evolution of 1000 time-steps of simulated time for the wireless ad hoc model with 6000 wireless SMHs. The X coordinate (LPs) shows the number of concurrent LPs implementing the set of model entities for the reference scenario. When $LP = 1$, the simulation is strictly sequential and monolithic, that is, only one processor executes the single LP incarnating the execution of all the model entities of the simulated model. In the $LP = 2$ scenario, 2 LPs incarnate the set of model entities, and each LP is allocated on a different physical processor of the execution architecture. When $LP = [3..8]$ the ARTÌS framework introduces a load-sharing of model entities over LPs. In addition, ARTÌS supports communication layer adaptation, resulting in low latency communication between LPs. Thanks to load-sharing capability of ARTÌS, the time required for completing the simulation run (Wall Clock Time, WCT) for $LP = 2$ is better than the one obtained with one LP ($LP = 1$). When the number of LPs grows, this fact introduces overheads under the synchronization and data distribution management (DDM) viewpoints, while the concurrency at the CPU hardware level is stable. For this reason, the WCT increases when $LP \geq 2$ and shows additional overheads with HT-ON. On the other hand, results in Figure 1 show that the HT-enabled PEU (HT-ON) for $LP = 2$ gives slightly better results than the HT-disabled PEU (HT-OFF). For this experimental scenario, the minimum WCT for both HT-ON and HT-OFF curves is obtained for 2 LPs. The activa-

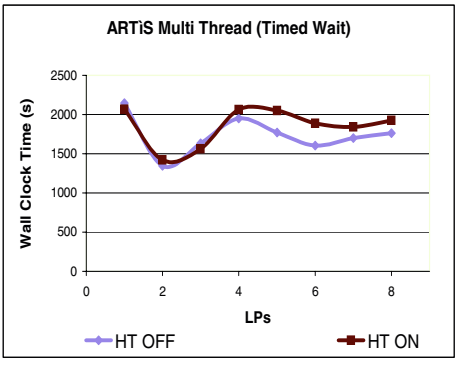

Fig. 1. PEU=1, SMH=6000, ARTÌS (Timed Wait)

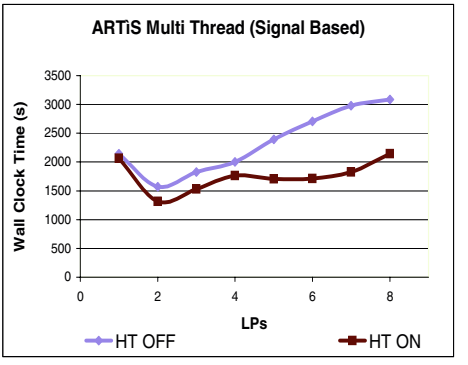

Fig. 2. PEU=1, SMH=6000, ARTÌS (Signal Based)

tion of HT does not change the optimal number of LPs, but has effects on the overall performance of the simulation processes. The reason for additional overheads with HT-ON could be due to the timed wait implementation of the thread synchronization implemented by ARTÌS. For this reason, in the following Figure 2, we performed the same investigation shown in Figure 1, with a modified version of ARTÌS. The latter version of ARTÌS is still multi-threaded, but the synchronization among threads is implemented with a signal based approach. In Figure 2 results show that the HT-enabled PEU (HT-ON) performs always better than the HT-disabled (HT-OFF) version. In addition, the HT-ON scenario shows an increase in the simulation scalability, with respect to HT-OFF. In Figure 3 the HT support is evaluated with respect to a mono-threaded version of the ARTÌS runtime architecture. This means that the ARTÌS runtime is based on single-thread, which is responsible to manage all the model entities' executions and to manage the communications. This test is interesting to evaluate the behavior and performances of HT architectures when executing mono-threaded software. Figure 3 shows that the HT support may slow down single threaded

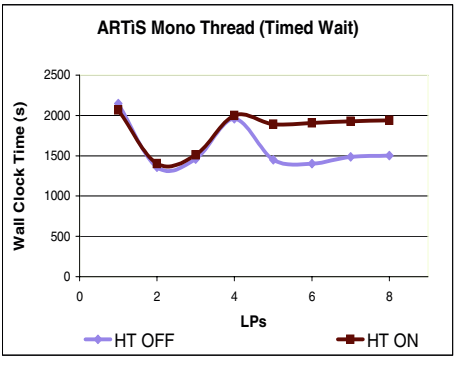

Fig. 3. PEU=1, SMH=6000, ARTÌS Monothread implementation

applications, by resulting in additional overhead. A quite strange behavior appears when $LP = 4$, showing a simulation slowdown (found also in Figure 2). The reason for this behavior requires further investigation. To summarize, in ARTÌS the HT support gives better results with signal based synchronization among threads, with respect to timed wait synchronization. On the other hand, HT support does not change the optimal number of LPs with respect to the underlying PEU architecture. When the LPs are mono-threaded, the HT support appears as not influent up to a given number of LPs (up to 4 LPs in the figures), and as an overhead when more than 4 LPs are executed (that is, when the model entities are load-shared among more than 4 LPs) in the considered PEU architecture.

5 Communication Optimization: Data Marshalling

The communication efficiency is one of the main factors determining the efficiency of a parallel or distributed simulation. The current optimization scheme in ARTÌS is based on a straightforward incremental policy: given a set of LPs on the same physical host (that is, with shared memory), such processes always communicate and synchronize via read and write operations performed within the address space of LPs, in the shared memory. Two or more LPs located on different hosts (i.e. no shared memory available) on the same local area network (or even on the Internet) would rely on standard TCP/IP connections. In ARTÌS, every interaction between LPs for synchronizing and distributing event messages is immediately performed over shared memory or network infrastructures. This kind of implementation generates much more transmissions on the communication channel, and replicates the message overheads (headers and trailers) and the channel accesses. A reduction of the overheads and channel accesses could result in increased channel utilization and reduction of the communication bottlenecks. In the following we will investigate if and how a message marshalling approach could reduce the simulation WCT. The data marshalling approach consists in the concatenation of more than one logical message in the same communication messages. In order to control the inverse trade-off degradation in the average communication latency, the data marshalling process is controlled by a timer: once every a maximum time limit the messages buffered on the LP are sent in a data marshalling packet (or frame). The proposed optimization has been applied both to shared memory and TCP/IP communications. Figure 4 shows the results for the optimization applied to a distributed simulation architecture. The hardware architecture is composed by 2 homogeneous PEUs equipped by Dual Xeon Pentium IV 2800 Mhz, 3 GB RAM, with HT-enabled, interconnected by a Gigabit Ethernet (1 Gb/sec). The simulated model for tests is the wireless ad hoc network model described in the previous section. Different scenarios have been modeled by varying the number of simulated entities (SHMs) in the model: from 3000 up to 9000 simulated mobile hosts. For each experiment, the data shown include the WCT time obtained with data marshalling ON and OFF, respectively. The data marshalling applied to this simulation testbed increased the

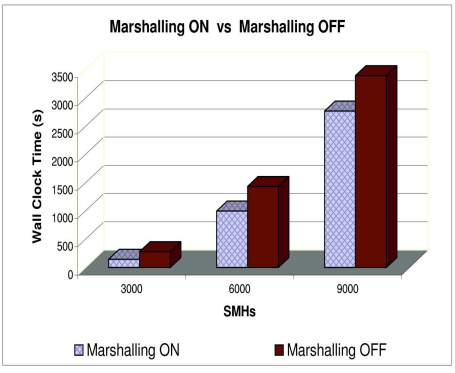

Fig. 4. PEU=2, Wall Clock Time with Marshalling ON and OFF

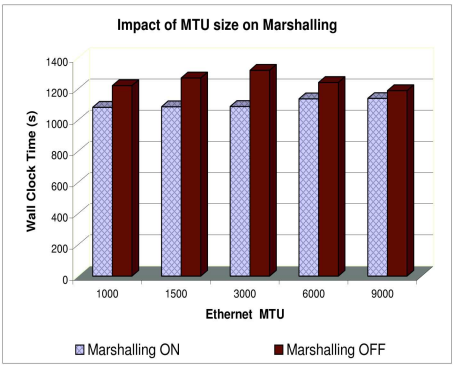

Fig. 5. PEU=2, SMH=6000, Wall Clock Time with Marshalling ON and OFF

WCT simulation performances: 48% for 3000 SMHs, 30% for 6000 SMHs, and 18% for 9000 SMHs (see Figure 4). When the number of SMHs increases, the percentage gain reduces: this happens because the computation required for updating and managing the states of many more SMHs becomes the predominant simulation bottleneck in this system (in the place of communication bottleneck).

The Maximum Transmission Unit (MTU) of local area network communications is another factor that could be managed under the data marshalling viewpoint. The Ethernet standard frame size has been "de facto" limited to 1500 bytes. In recent years the Ethernet bitrate has greatly increased, but the MTU size is substantially the same. Very large MTU size could reduce communication overheads (i.e. the percentage effect of headers) and could increase the network utilization. This approach is usually referred to as the adoption of "jumbo frames". For this test we have interconnected two homogeneous PEUs (defined above) by a cross-linked Gigabit Ethernet network cable. The simulation model is the wireless ad hoc model with 6000 SMHs. In Figure 5 results show that the data marshalling ON can reduce the WCT with respect to data marshalling OFF. On the other hand, results are only slightly influenced by the variation of the MTU size (from 1000 up to 9000 Bytes). The experiments shown that the adoption of jumbo frames slightly increased performances (up to 3000 Bytes) when data marshalling is OFF. When marshalling is ON, the simulation performance was almost constant up to 3000 bytes, and slightly increasing with more than 3000 Bytes.

6 Simulation Data Structures Optimization

One of the most important data structures in a computer simulation is the repository of event descriptors. Both monolithic and distributed simulations, require that future events are collected and executed in timestamp order. Every simulation may include the management of millions of events. For this reason, it

is important to find a really efficient data structure at least in the support of a subset of management operations. The most frequent operations in a simulation process are: insertion of a new event descriptor (*insert()* operation), and extraction of the event with the minimum time stamp (*extract_min()* operation). Both operations should have a really low computational complexity and should be easy to implement. Some useful data structures can be adopted to assist in the implementation of the event repository definition and management: lists, hash tables, calendars and balanced trees. In most cases, the better solution is to adopt the heap data structure. "A heap is a specialized tree-based data structure. Let A and B be nodes of a heap, such that B is a child of A. The heap must then satisfy the following condition (heap property): $key(A) \leq key(B)$. This is the only restriction of general heaps. It implies that the greatest (or the smallest, depending on the semantics of a comparison) element is always in the root node. Due to this fact, heaps are used to implement priority queues" [3].

Given a binary heap, the worst case computational complexity for the *insert_heap()* and *extract_min_heap()* is $O(log_2 n)$ (because after the extraction the heap requires a heap re-organization (heapify) algorithm execution). We call a classical binary heap as the "base heap" data structure. The Base Heap (BH) demonstrates good performances in general, to implement the event repository for a simulation process. On the other hand, by considering common assumptions related to the event management and event characteristics in the simulation field, we could design even more efficient heap-based data structures. Event descriptors are usually organized as heap elements ordered by time-stamp (key). The set of events generated during a conservative simulation is usually characterized by sequential time-stamp values. Moreover, the time management of a simulation process could be time-stepped, which means that all the time-stamps of events located in the same timestep are equal. By exploiting these common properties of simulation processes it would be possible to implement an enhanced version of the heap data structure. Each node of the Enhanced Heap (EH) is now composed by a pointer to a linked list of events, including all the descriptors of events with the same time-stamp value (see Figure 6). Thanks to the principle of time-locality in the references to event descriptors in a simulation process, the access to event descriptor with time-stamp value t is followed with high probability by the accesses to event descriptors with the same time-stamp value. For this reason, by caching the pointer to the linked list of simultaneous events in the simulation, the management of the EH data structure is much more efficient than replicating search operations on the BH. This optimization allows to avoid frequent heapify operations, by working on the cached linked lists associated with heap elements. Hence, by calling a hit the insertion or extraction of one event descriptor to/from the cached list associated to current simulated time t, the *insert_heap()* and *extract_min_heap()* operations can be performed in the majority of cases with $O(1)$ complexity (given the time-locality simulation assumption, resulting in high hit ratio) in the linked list. The complexity is $O(log(k))$ in the worst case (that is, cache miss), being k the number of different event timestamps (keys) inserted in the EH. In general, with EHs, the number

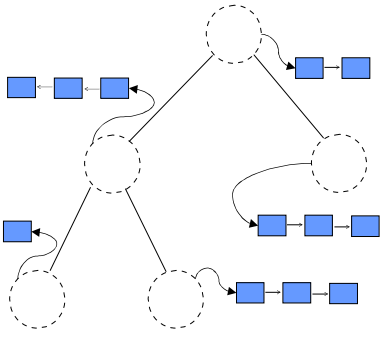

Fig. 6. An Enhanced Heap (EH) data structure

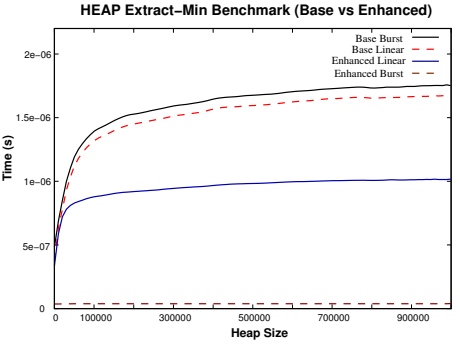

Fig. 7. Benchmarks: synthetic environment, base vs. enhanced versions

of three nodes can be reduced, by adopting only one placeholder node for the set of events with the same timestamp (key). The data structure size can be reduced by eliminating the time-stamp informations from all the event descriptors in the same list (whose time-stamp is implicitly defined by the corresponding EH node).

The Figure 7 shows the results obtained by a benchmark application (that is, not a simulation) that has been defined to test the efficiency of the EH data structure management. The curves show the average time required to insert (and to extract) a group of 1000 heap nodes in base (BH) and enhanced (EH) heap data structures, when the initial heap size has the value indicated on the X coordinate. Two kinds of heap insertion operations have been tested: the Burst insertion is defined as the insertion of a group of 1000 heap nodes with the same key (time-stamp value), while the Linear insertion is defined as the insertion of a linear sequence of 1000 heap nodes with incremental key (time-stamp value). The data shows that the management of BH with bursts insertions (Base Burst curve in the figure) obtains the worst performance, as expected. The BH with linear insertions (Base Linear curve in the figure) performs a little better than Base Burst. A great improvement in the performance is obtained with the Enhanced Heap (EH). The EH with linear insertions (Enhanced Linear curve in the figure) performs better than Base Burst (50% time reduction). The EH with burst insertions (Enhanced Burst curve in the figure) obtains almost constant performances, independent from the heap size, and results in a very good performance index.

By testing the Enhanced Heap data structure on the simulation testbed, we performed the simulation of the wireless ad hoc simulation model with variable number of SMHs over the multi-threaded version of the ARTÌS framework. The experiments have been executed over a PEU equipped by Dual Xeon Pentium IV 2800 Mhz, 3 GB RAM, with Hyper-Threading support activated. The Figure 8 shows the impact of the Heap type (Base vs. Enhanced) on the WCT performance for the simulations, with variable number of simulated entities (SMHs). As expected the WCT increases, but the effect of the Heap type is marginal.

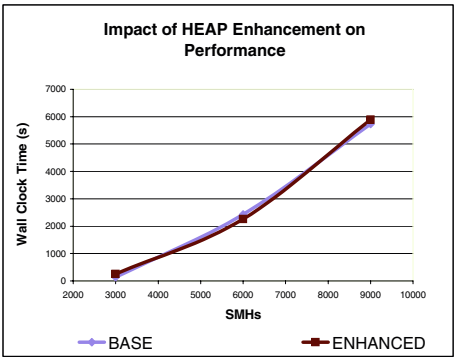

Fig. 8. Standard wireless model (computation intensive)

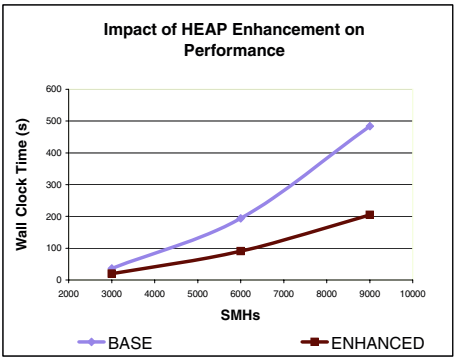

Fig. 9. Modified wireless model (low computation)

Only a little advantage is shown with 6000 SMHs. The reason for this fact is the high degree of computation that is required in this model for managing each event involving a subset of SMHs. Without entering in details, this simulation model is defined in order to be computation intensive. For this reason the advantages in the management of event descriptors obtained with Enhanced Heaps is hidden by the overwhelming amount of computation that follows every new event extraction or insertion. To confirm this fact, we implemented a light computation version of the same simulation model, where the computation caused by each event is reduced more or less of a factor 10. The results shown in Figure 9 confirms our expectations. In this figure, the advantage of adopting the Enhanced Heap is clear. The increase in the model complexity (number of simulated mobile hosts, SMHs) results in the increasing advantage of adopting the Enhanced Heap support for event management. The low computation required in this model, for each event, has emphasized the effect of the event management complexity in the simulation process evolution.

7 Conclusions and Future Work

In this paper we illustrated the definition and analysis of a collection of solutions adopted to increase the performance of communication and computation activities required by the implementation and execution of parallel and distributed simulation processes. Parallel and distributed simulation has been defined, and a real testbed simulation scenario has been illustrated, based on the ARTÌS parallel and distributed simulation framework. Three classes of solutions have been proposed to improve the performance of simulations executed over commodity off-the-shelf computation and communication architectures: multi-threaded software and Hyper-Threading support by the processor architectures, data marshalling solutions for shared-memory and network-based communications, and data structure optimization for simulation events' management. All the proposed

solutions have been evaluated on real testbed evaluation scenarios, and under variable configurations. Results obtained demonstrate that a performance improvement, summarized by the Wall Clock Time (WCT) required to complete the simulation processes, can be obtained by adopting and tuning the proposed solutions in opportune way. The experimental analysis has provided some interesting guidelines about the way to adopt and to compose the proposed solutions, under the considered simulation testbed. Some guidelines indicate that the system bottleneck could change depending on the model and system assumptions. Most of the guidelines have been commented under the general context assumptions, and could be considered generally extensible to other simulation frameworks, models and execution scenarios. Future works include the analysis of more wide simulation scenarios, and the detailed analysis of resource utilization metrics.

References

1. Dartmouth SSF (DaSSF). `http://www.cs.dartmouth.edu/research/DaSSF/`.
2. GTW/TeD/PNNI. `http://www.cc.gatech.edu/computing/pads/teddoc.html`.
3. Heap, From Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Heap`.
4. Hyper-Threading Technology on the Intel Xeon Processor Family for Servers. `http://www.intel.com/business/bss/products/hyperthreading/server/ht_server.pdf`.
5. IEEE Std 1516-2000: IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules, - federate interface specification, - object model template (OMT) specification, - IEEE recommended practice for high level architecture (HLA) federation development and execution process (FEDEP).
6. Maisie Programming Language. `http://may.cs.ucla.edu/projects/maisie/`.
7. Parallel / Distributed ns. `http://www.cc.gatech.edu/computing/compass/pdns/`.
8. SNT: QualNet. `http://www.qualnet.com`.
9. PADS: Parallel and Distributed Simulation group, Department of Computer Science, University of Bologna, Italy. `http://pads.cs.unibo.it`, 2005.
10. L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. ARTÌS: a parallel and distributed simulation middleware for performance evaluation. In *Proceedings of the 19-th International Symposium on Computer and Information Sciences (ISCIS 2004)*, 2004.
11. L. Bononi, G. D'Angelo, M. Bracuto, and L. Donatiello. Concurrent replication of parallel and distributed simulation. In *Proceedings of the nineteenth workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 2005.
12. L. Bononi, G. D'Angelo, and L. Donatiello. HLA-based Adaptive Distributed Simulation of Wireless Mobile Systems. In *Proceedings of the seventeenth workshop on Parallel and distributed simulation*. IEEE Computer Society, 2003.
13. R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc., first edition, 2000.
14. D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture: A hypertext history. *Intel Technology Journal*, 2002.
15. J. Panchal, O. Kelly, J. Lai, et al. Parallel simulations of wireless networks with TED: radio propagation, mobility and protocols. *SIGMETRICS Perform. Eval. Rev.*, 25(4):30–39, 1998.