Porting a Molecular Dynamics Application on a Low-cost Cluster of Personal Computers running GAMMA

G. Ciaccio[1], V. Di Martino[2]

[1] DISI, Università di Genova
via Dodecaneso 35, 16146 Genova, Italy
E-mail: ciaccio@disi.unige.it

[2] CASPUR, c/o Università di Roma "La Sapienza"
P.le A.Moro 5, 00185 Roma, Italy
E-mail: vincenzo@caspur.it

Abstract. The Genoa Active Message Machine (GAMMA) is a low-cost cluster of Personal Computers (PCs) networked by commodity fast LAN (100base-T Ethernet) and running a Linux operating system kernel enhanced with low-latency, high throughput support to inter-process communication based on the Active Messages paradigm. The target of our work is the porting on GAMMA of a Molecular Dynamics (MD) code used to study polarizable fluids and written in FORTRAN with calls to PVM communication routines. Such porting required to extend the GAMMA programming library with FORTRAN stubs to GAMMA communication functions. Then the communication policy in the original PVM version of MD has been changed to fit the GAMMA Active Message-like communication style. As a third step, the performance of the obtained GAMMA version of MD has been measured and compared to the performance of the PVM version on the same hardware platform as well as to the performance obtained by the PVM version on a Memory Channel cluster of DEC SMP workstations of higher class of costs and on a eight-nodes IBM SP2. The results show the appealing cost/performance figure of GAMMA and the significant performance improvement exhibited by MD after being ported from PVM to GAMMA.

1 Introduction

Molecular Dynamics (MD) is one of the most frequent parallel applications in the scientific community. MD typically exhibits fairly good speed-up figures on a wide range of parallel computers with good intrinsic load balancing. This offers the opportunity to investigate the behaviour of large size samples of material by numerical simulation. Increasing the size of the samples as well as the time interval of the simulation allow to gain a deeper understanding of physical processes, hence the interest in running MD faster using parallel computing techniques.

The simulation of a large number of molecules became affordable with the advent of low-cost CPUs like the ones installed on modern workstations. The

memory bottleneck of such conventional computer architectures can be avoided by spreading the computation in parallel over a so called Network of Workstations (NOW). As performance of modern high-end Personal Computers (PCs) approaches performance of conventional workstations with very competitive costs, the use of PCs as computation nodes in a NOW platform becomes a viable alternative at a very convenient cost/performance ratio and with fairly good absolute performance level. In the near future low cost SMP-based PCs will be available, providing for increased computation power per node of a network of PCs at a competitive cost.

A serious obstacle to using MD on a NOW platform is the high communication latency exhibited by standard parallel programming environments like PVM [9] and MPI [10] running atop industry-standard communication protocols like TCP and UDP. Recently several teams have been engaged in producing efficient solutions using faster networks and optimized communication software to keep latency as low as possible. Many of such attempts gave rise to non-standard programming interfaces for high-performance communication, posing important issues as for porting existing parallel applications to such efficient parallel platforms. Porting a non-trivial parallel application on a non-standard communication layer may be an expensive task, especially in terms of human costs. However the better cost/performance ratio and the satisfactory absolute performance level achieved by a network of PCs may justify the porting effort.

In this paper we discuss the experience of porting an existing MD parallel program on GAMMA. GAMMA is an efficient Active Message-like communication layer implemented on a 100base-T Ethernet network of Pentium PCs running Linux, a POSIX-compliant Unix Operating System. The original MD code is a FORTRAN program with calls to PVM communication routines. Porting MD to GAMMA required replacing PVM calls with calls to communication routines from the GAMMA library, as well as changing some communication patterns in order to achieve better exploitation of the capabilities of the underlying network hardware fully exposed by GAMMA.

2 A brief account of Active Messages and GAMMA

Many modern high performance messaging systems have been derived from the Active Message communication paradigm [11]. Active Messages are aimed at reducing the communication overhead by allowing communication to overlap computation. The efficiency of the Active Message paradigm lies in the fact that it eliminates the need of intermediate copies of messages along the communication path, thus remarkably speeding up communications. In a traditional send-receive messaging system, messages delivered to a destination node need to be temporarily buffered waiting for the destination process to invoke a "receive" operation which will consume them. With Active Messages this is no longer true: as soon as delivered, each message triggers a function of the destination process, known as *receiver handler*, which will consume it immediately. Here "consuming" means integrating the message information into the ongoing computation of the destina-

tion process, notifying the reception to the destination process itself, and possibly setting some data structures in order to promptly "consume" the next incoming message as soon as it arrives.

The current prototype of the Genoa Active Message MAchine (GAMMA) [3, 4] is a pool of (currently 16) autonomous Pentium 133 MHz PCs networked by a 100base-TX Ethernet repeater hub and 3COM 3c595 as well as 3c905 network interface cards. Each PC runs the Linux operating system whose kernel is enhanced by an Active Message-like messaging system, which comprises a custom driver for the 100base-T Ethernet cards capable of managing both Active Messages and standard IP communications.

GAMMA Active Messages are inspired to Thinking Machines' CMAML library for the CM-5 [1], where each process has a number of *communication ports* and may attach both a receiver handler and a data structure to a single communication port in order to handle all messages incoming through that port. Moving the message from the network to the destination data structure attached to the communication port is efficiently and timely performed by the GAMMA custom network device driver.

Differently from the Generic Active Message paradigm [5], in GAMMA the sender process specifies a communication port of the destination process instead of the pointer to a receiver handler, and the receiver handler must not explicitly extract and store the incoming message. The programming paradigm of GAMMA is Single Program Multiple Data (SPMD), although the "port-oriented" approach makes GAMMA suitable for MIMD programming as well, as a common address space among cooperating processes is not required, multiprogramming is allowed, and a process may communicate with other processes running different programs.

Most of the GAMMA communication layer is embedded in the Linux kernel, the remaining part being placed in a user-level programming library. All the functionalities of the original Linux kernel were left unmodified. All the communication software of GAMMA has been carefully developed according to a performance-oriented approach. The adoption of the Active Message paradigm allows a true "zero copy" protocol, with no intermediate copies of messages along the whole user-to-user communication path. Differently from many other high performance communication layers running on NOWs, GAMMA yields high performance while still offering a fairly flexible and easy-to-use virtualization of the communication hardware in a multi-user, multitasking UNIX environment.

GAMMA exploits the Ethernet hardware broadcast features, thus providing for native broadcast/multicast communication besides point-to-point one.

The GAMMA messaging system is capable of delivering nearly the whole performance of the raw communication hardware to user applications, substantially outperforming any other known prototype developed on low-cost NOWs. One-way "ping-pong" user-to-user message latency is less than 13 μs, whereas asymptotic bandwidth raises 12.2 MByte/s. Half the asymptotic bandwidth is achieved with messages as short as 200 bytes. Such performance numbers are measured at application level, that is they represent the communication performance effectively delivered to user applications.

In terms of latency GAMMA rivals many much more expensive massively parallel platforms. Obviously GAMMA cannot compete with such platforms in terms of bandwidth as well as scalability. On the other hand no massively parallel computer can compete with GAMMA in terms of cost/performance ratio.

For a description of the GAMMA programming interface see [2, 4].

3 A description of the Molecular Dynamics application

Our MD application [6, 8] is a typical Molecular Dynamics code used for simulating the behaviour of polarizable fluids. The current release of MD is written in FORTRAN with calls to PVM routines. The application is structured as a SPMD parallel program. The I/O part has not been parallelised due to the lack of a standard implementation of parallel I/O across the various hardware platforms. All the computational code runs in parallel.

The simulation of material samples with larger number of molecules turns the behaviour of MD from communication intensive to computation intensive. In our investigations the number of molecules has been kept as low as 4000 to stress the communication side. With such a small number of molecules, important obstacles to speed-up are high communication latency, especially with fast CPUs, and low aggregate throughput of the interconnection network. However with the current prototype of GAMMA, which exploits not particularly fast CPUs (Pentium 133 MHz), we expect the main limiting factor to be communication latency, although the poor aggregate throughput of the 100base-TX Ethernet repeater hub and the possible network contention that may arise on such shared LAN hardware could in principle affect performance as well.

MD performs a standard Lennard-Jones calculation plus the solution of the induced polarizability on each molecule taking in account first dipole momentum. Each step of MD consists of evaluating the induced dipoles $\bar{p}_i$ consistent with the values of $\bar{E}_i^{(q)}$ due to a given distributions of the point charges. This part of the calculation requires an iterative procedure with small computation time and many communications to exchange the values of the induced polarizability at each iteration among all processors. For a small number of molecules the cutoff radius is of the same size as the replicated box and the number of force vectors between molecule pairs grows almost quadratically with the total number of molecules. In such a situation any domain decomposition technique based on the spatial position of each molecule in the box is not feasible.

In the parallel implementation each processor maintains a copy of the position of each molecule. However each processor will compute force pairs only on a predefined subset of molecules which has been previously assigned to it. In this way the list of interacting particles, which is by far the larger data structure of MD, could be partitioned among the computation nodes and the total memory occupancy per processor is expected to decrease with increasing number of computation nodes.

To avoid doubling the computation of the force acting on a pair of molecules the second Newton law is applied once the force is calculated on one side. This

may require that the force computed on a given molecule be communicated to the processor owning the partner molecule.

Our parallel organization does not reorganize the local molecule indexes, at the expense of more communication between processors. For this reason MD stresses the communication of the parallel platform when run with as few as 4000 molecules.

When using high-latency communication primitives like PVM's ones, an important optimization is to keep the number of distincts messages as low as possible in order not to pay too much for the communication start-up costs. This is achieved by packing all the variables to be communicated (i.e. forces,virial,energy) in a single outgoing message whenever possible. Applying such optimization technique the message size in the exchange phase is:

$$\text{number\_of\_byte} = 8 * 3 * \text{Number\_of\_molecules}/\text{n\_procs}.$$

Since the resulting message size was in the order of KByte the negotiation of network access was not addressed at the programming level. For larger messages the available buffers on the communication patterns may require to pay attention to the switching overhead that a process can incur when simultaneously receiving multiple point-to-point communications.

Keeping the number of distinct messages as small as possible reduces the possibility of using multicast/broadcast communication primitives, as in PVM such collective communications are implemented as bare repetitions of point-to-point communications. Almost all communications were point-to-point ones, but a few of them, i.e. the exchange of the new coordinates of the molecules.

During the variables exchange, a storm of messages in the number of

$$\text{n\_procs} * (\text{n\_procs} - 1)/2$$

compete on the network. This is a typical situation for many well balanced problems, where the computation time is almost the same for each processor. As told before there was no evidence of such competitions in the speed-up data although it was hard to isolate this contribution to the performance degradation.

4 Porting strategy

In order to port MD from PVM to GAMMA, the GAMMA programming library has been extended with FORTRAN stubs to the original GAMMA communication C functions in a straighforward way.

The GAMMA platform is based on low-cost shared 100base-TX Ethernet hardware. This implies that the above mentioned communication mechanism may well cause lots of collisions on the Ethernet channel, with heavy delays in the communication phase. This could be partially avoided if the Fast Ethernet hub be replaced by a switch. The alternative is to explicitly program a proper serialization of network accesses at the application level and to take best advantage of the Ethernet's hardware broadcast facility that the GAMMA programming

interface directly exposes. The serialization of communications during collective all-to-all data exchanges has been obtained in MD by considering all processes as circularly ordered by instance number and implicitly granting transmission right to a process after it has received messages from all its predecessors.

To keep the debugging overhead of the GAMMA version of MD as low as possible, we first derived a modified PVM version of MD from our original PVM version. On such modified version all the communications are performed with multicast/broadcast primitives, still according to a plain SPMD style. The price to pay is larger message size, the advantage is a zero collision communication phase once moving from PVM to GAMMA. Another potential source of overhead with this modified version comes from a reduction of overlap between computation and communication because of a predefined order in the message exchange between processes.

We measured performance of the modified PVM version of MD on a cluster of four DEC 4100 four-processors SMP workstations connected by a 60 MByte/s Memory Channel hub. Performance results are shown as the speed-up curves depicted in Figure 1. The limiting factor to ideal speed-up is the available bandwidth on the memory channel hub as already shown in previous work [7]. This modified version send the same number of total messages but with larger dimensions, the rate of the packet dimensions in the two cases is:

$$\text{new\_packet\_size}/\text{original\_packet\_size} = (\text{n\_procs} + 1)/2$$

This means that in the worst case when n_procs = 16 the exchanged packet size is about 8 times larger. If the PVM implementation at hand takes advantage of the hardware broadcast capability offered by the network the overhead for larger packet size could be recoverd by a better implementation of multicast, i.e. when one of the four process running on a 4100 workstation needs to multicast data to the four processes running on another 4100 a smart implementation can avoid that four different messages transitate on the memory channel hub that, as it has been pointed out, have a smaller total bandwidth respect to internal shared memory bandwidth. This is not the case, and we experience the degradation in the performances shown in the speed-up graphics.

Another source of performance degradation with MD is the need of application-level temporary storage for incoming messages. Even with a "zero-copy" messaging system like GAMMA, MD must implement a temporary storage for received messages, because some broadcast messages carry information to be scattered among many processors and summed component-wise to existing local information arranged as arrays. Such additional copies of messages have no relevant impact on overall performance when using a high-latency communication system like PVM. Indeed in the case of the PVM version of MD, the test performed on the 4100 cluster did not show any relevant performance degradation, only few percents of the total time for n_procs = 4. With GAMMA the performance degradation may be more relevant, however we could not find any reasonably easy way to circumvent the problem without rewriting MD from scratch.

A potential problem with GAMMA is that the receiver is forced to accept messages at any time the sender starts a communication. This may cause race

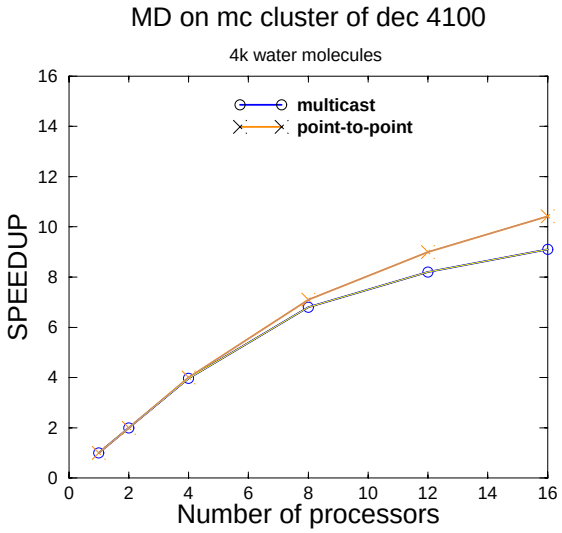

Fig. 1. Point-to-point versus broadcast MD implementation, measured on a shared 60 MByte/s Memory Channel cluster of DEC 4100 workstation running PVM.

conditions in the memory of the receiver process: Since the all-to-all exchange phase of MD is a two-steps operation with two communication steps interleaved by one computation step where the fresh data are manipulated i.e. summed to previous data, it may happen that more than one sender modify the same receiving buffer with no possibility of verifying the corruption of data. To avoid such race conditions we had to implemented FIFO queues of receiver buffers for incoming GAMMA messages rather than binding single buffers to the receiver ports. Freshly received messages are stored at the head of the queue by the GAMMA communication system while older messages are extracted at the tail of the queue by the receiver process to be processed. The FIFO queue is limited to three positions. From our experiments such upper bound appeared to be high enough for our purposes, at least as long as overall workload is distributed uniformly across the PCs in the platform. Under such conditions the possibility of race conditions inside the memory of the receiver process is quite low even with single buffers instead of FIFOs, thanks to the explicit serialization of communications performed by MD in the contention-free all-to-all message exchanges. An alternative approach is the definition of a larger number of communication ports. This solution must be avoided for two reasons: firstly, because it does not scale up with the number of processors; secondly, it is better to exploit as few as

possible receive buffers and to reuse them when possible since GAMMA requires them to be locked into physical RAM.

Since the definition of a GAMMA communication port takes several CPU cycles we choose to reuse the same ports for all the time step iterations. This is not a problem for MD because the communication patterns are well defined and iteratively reused as the simulation evolves in the time steps.

To avoid a possible overlapping of messages on the same receiver buffer we use the *gamma_attach_buffer* GAMMA library call in the receiver handlers in order to promptly switch to a fresh receive buffer upon each message receive. Such GAMMA function allows to attach a fresh buffer to a given GAMMA communication port. It is a lighter-weight alternative to re-programming a GAMMA port from scratch in the case of changing only the receiver buffer.

GAMMA guarentees implicit flow control based on the fact that the Fast Ethernet throughput is less than the throughput achieved by the DMA engines when moving data between RAM and network devices. Such implicit flow control has proved to work also in the case of a non-trivial application with complex communication patterns like MD. Previous experience of porting code to GAMMA has been done only on specific applications where the computational behaviour were defined in a better way, i.e. the number of CPU operations was low and well synchronised with data communication.

5 Performance results and conclusions

We have measured performance of both GAMMA and PVM versions of MD on our low-cost hardware platform (shared 100base-T Ethernet network of Pentium 133 MHz PCs).

Figure 2 reports the obtained speed-up curves. The nearly optimal speed-up curve with GAMMA is mainly due to the following reasons:

- the relatively poor floating-point computational power of Pentium 133 MHz CPUs
- the high efficiency of GAMMA inter-process communications
- the fine tuning of the communication patterns in the GAMMA version of the application, based on the knowledge of features (broadcast) and limitations (shared LAN) of the underlying communication hardware.

The slow-down exhibited by the PVM version of MD on the same hardware platform as the number of processors increases beyond eight is clearly apparent. Given the low computational power of Pentium 133 MHz CPUs, such behaviour accounts for the low efficiency of the PVM messaging systems involving many temporary copies of messages as well as the traversal of many layers of communication protocols. On the other hand we have still to verify whether the PVM version of MD be already tuned for a shared interconnection network. If this is not the case, then the PVM version running on many processors might cause collision storms on the Ethernet with lots of dropped packets and corresponding

dramatic degradation of communication performance, which could be eliminated by proper re-organization of the communication patterns.

Figure 3 reports the average completion time per time-step for the GAMMA version of MD. The average completion time per time-step measured with the PVM version run on our cluster of PCs as well as on a eight-"thin-nodes" IBM SP2 and on the DEC 4100 Memory Channel cluster are reported too. When reading such curves it is important to pay attention to both the absolute performance and the cost of the hardware platform. As for absolute performance, MD atop GAMMA shows the ability to scale up to many more processors compared to PVM, and the absolute performance of GAMMA appears to overcome the one of IBM SP2 if more than twelve processors be engaged in the computation. It is worth noting the excellent performance level raised by the DEC 4100 cluster with PVM, probably due to memory being shared amond CPUs at the level of each single workstation and to powerful floating-point units equipping the Alpha 400 MHz CPUs. As for costs, the current cost on the marketplace of the whole 16-nodes GAMMA leveraging shared 100base-T Ethernet and Pentium 133 MHz CPUs is comparable to the cost of one single workstation. As a conclusion MD atop GAMMA indeed provides a very favourable cost/performance ratio with satisfactory absolute performance levels.

References

1. Connection Machine CM-5 Technical Summary. Technical report, Thinking Machines Corporation, Cambridge, Massachusetts, 1992.
2. G. Chiola and G. Ciaccio. GAMMA: Architecture, Programming Interface and Preliminary Benchmarking. Technical Report DISI-TR-96-22, DISI, Università di Genova, November 1996.
3. G. Chiola and G. Ciaccio. Implementing a Low Cost, Low Latency Parallel Platform. *Parallel Computing*, (22):1703–1717, 1997.
4. G. Ciaccio. Optimal Communication Performance on Fast Ethernet with GAMMA. In *Proc. International Workshop on Personal Computer based Networks Of Workstations (PC-NOW'98), to appear (LNCS)*, Orlando, Florida, April 1998. Springer.
5. D. Culler, K. Keeton, L.T. Liu, A. Mainwaring, R. Martin, S. Rodriguez, K. Wright, and C. Yoshikawa. Generic Active Message Interface Specification. Technical Report white paper of the NOW Team, Computer Science Dept., U. California at Berkeley, 1994.
6. V. Di Martino. Computer Simulation of Polarizable Fluids. In *First European PVM Meeting*, Rome, October 1994.
7. V. Di Martino and G. Ruocco. Molecular dynamics on Hybrid Memory Machines. In *Fourth PVM-MPI European Meeting*, Krakow, Poland, November 1997.
8. V. Di Martino, G. Ruocco, and M. Sampoli. Molecular dynamics of polarizable fluids on parallel systems. In *HPC-ASIA '95*, Taipei, Taiwan, September 1995.
9. V. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, pages 315–339, December 1990.
10. The Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Technical report, University of Tennessee, Knoxville, Tennessee, 1995.

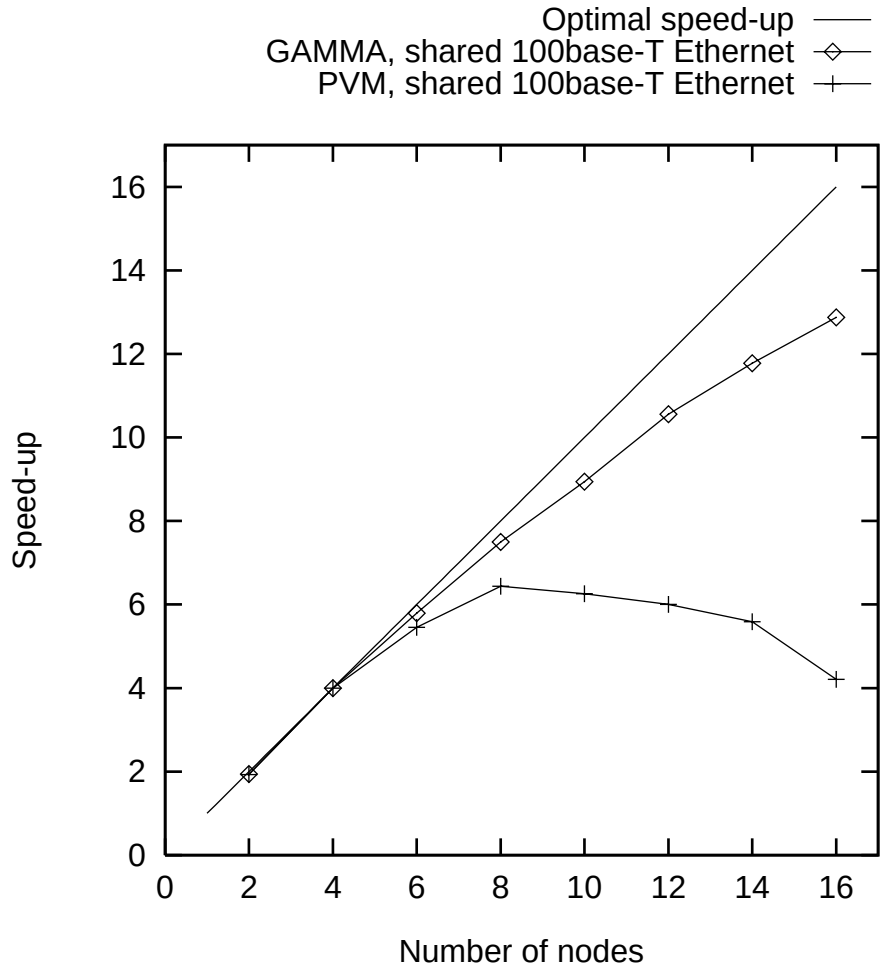

Fig. 2. Molecular Dynamics, GAMMA vs. PVM: speed-up comparison with same hardware platform (shared 100base-T Ethernet network of Pentium 133 PCs).

---

11. T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92)*, Gold Coast, Australia, May 1992. ACM Press.

This article was processed using the LaTeX macro package with LLNCS style

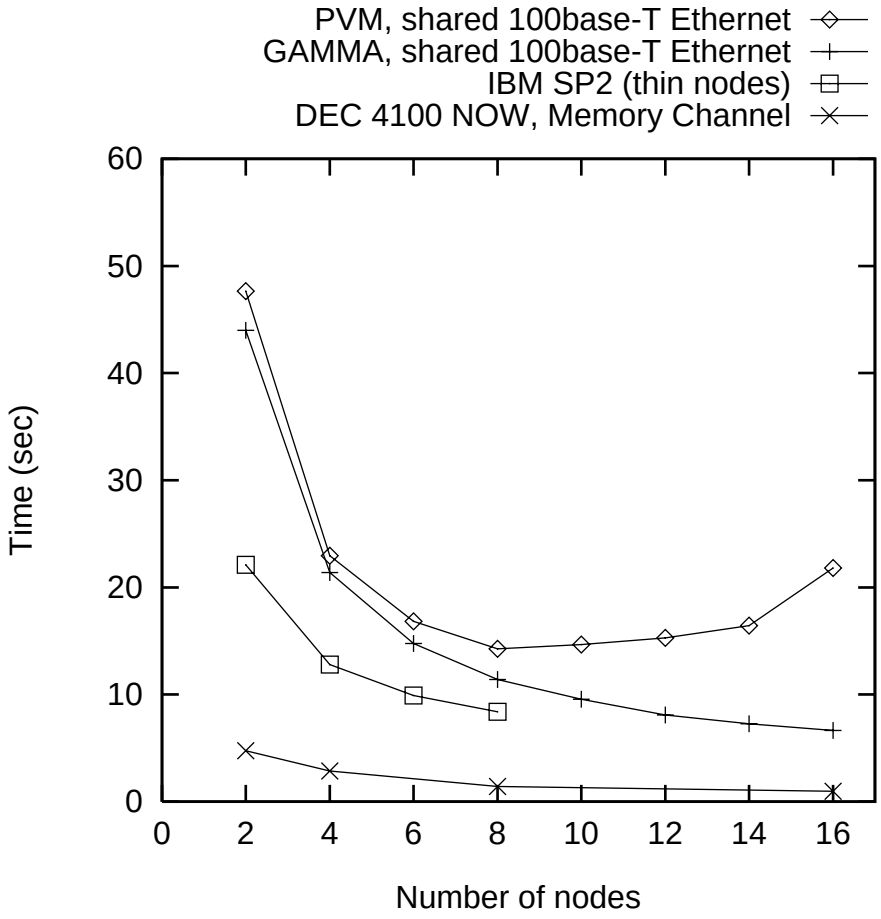

Fig. 3. Molecular Dynamics: average completion time per time-step on various parallel platforms including GAMMA.