Performance of the MOSIX Parallel System for a Cluster of PC's

Amnon Barak and Oren La'adan *

Institute of Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel

Abstract. The scalable PC cluster at Hebrew University consists of 48 Pentium and Pentium-Pro servers that are connected by fast Ethernet and the Myrinet LANs. It is running the MOSIX operating system, an enhancement of BSD/OS with algorithms for dynamic resource sharing that are geared for performance scalability in a scalable computing cluster. These algorithms use a preemptive process migration mechanism for load-balancing and memory-sharing, in order to create a convenient multi-user time-sharing execution environment for HPC, particularly for applications that are written in PVM or MPI. This paper gives a brief overview of MOSIX and its resource sharing algorithms. Then the paper presents the performance of these algorithms as well as the performance of several large-scale, parallel applications.

1 Introduction

Scalable Computing Clusters (CC), ranging from a cluster of (homogeneous) servers to a Network of (heterogeneous) Workstations (NOW), are rapidly becoming the standard platforms for high performance and large-scale computing. The main attractiveness of such systems is that they are made of affordable, low-cost, commodity hardware, e.g. Pentium-Pro based Personal Computers (PC's), and fast LANs e.g. Myrinet [4]. Other advantages are that these systems are scalable, i.e., can be tuned to available budget and needs, and that they use standard software components, i.e., UNIX, PVM [6] and the MPI [7] parallel programming environments. Two examples of such systems are the UC Berkeley NOW project [1], which uses a cluster of SPARC based workstations connected by Myrinet, and NASA's Beowulf [3] "pile of PC's", which connects 16 Pentium-Pro PC's by multiple fast Ethernet boards.

A major drawback of many existing computing clusters is the lack of intelligent mechanisms for dynamic, network-wide resource sharing, that can respond to resource requirements and availability. Such mechanisms are necessary for performance scalability in clusters of servers and to support a flexible use of workstations, since the overall (network-wide) available resources in such systems are expected to be much larger than the available resources at any workstation or server. The development of such mechanisms is particularly important

* E-mail: {amnon,orenl}@cs.huji.ac.il WWW: http://www.cs.huji.ac.il/mosix

to support multi-user, time-sharing parallel execution environments, where it is necessary to share the resources and at the same time distribute the workload dynamically, to utilize the global resources efficiently. Throughout this paper we use the term computing cluster (CC) to refer to a collection of UNIX computers (nodes), ranging from a network of workstations to a cluster of servers on local area networks.

One mechanism that can perform dynamic work distribution and resource sharing, both efficiently and transparently, is preemptive process migration. This mechanism can move a process from one node to another dynamically, for load-balancing or memory-sharing, to improve the performance and the overall utilization of the CC. We use the term "memory-sharing" to describe the use of remote memory resources to avoid local paging and thus provide processes with the memory resources they require. This parallels load-sharing where the goal is to provide a process with the CPU resources it needs. In spite of the advantages of process migration, very few operating systems have implemented it for dynamic resource sharing. Instead, most existing systems provide explicit, user-controlled remote execution and migration, while other systems provide automated remote executions, but perform preemptive process migration at the explicit request of a user.

This paper presents the CC version of MOSIX, an enhancement of BSD/OS that supports preemptive process migration for load-balancing and memory sharing. These algorithms attempt to improve the performance by dynamic distribution of the workload and the available resources among all the processes. The goal is that the application programs do not have to know the current state of the resource usage. Parallel applications can be executed by simply creating many processes, just like a single-machine environment. The paper presents the performance of these algorithms and the performance of several, large scale parallel applications.

The paper is organized as follows: the next section presents MOSIX and its main properties. Section 3 presents the performance of some of the resource sharing algorithms of MOSIX. Section 4 presents the performance of several parallel applications. Our conclusions are given in Section 5.

2 What is MOSIX

MOSIX [2] is a set of enhancements of BSD/OS for supporting dynamic resource sharing in a CC. This section describes the hardware configuration for MOSIX and its main software characteristics.

2.1 The hardware configuration

MOSIX is designed to run on Pentium and Pentium-Pro based platforms, including personal computers, file and CPU servers, that are connected by standard LANs or fast interconnection networks. Depending on the type of applications and the budget, MOSIX configurations may range from a small cluster of PC's

that are connected by Ethernet, to a high performance configuration, with a large number of Pentium-Pro based servers that are connected by a Gigabit/Sec. scalable LAN, e.g. Myrinet [4].

The main advantage of the above configurations is the use of standard, low-cost, off-the-shelf, commodity hardware components. For example, the multi-computer that we use has 16 Pentium-Pro 200MHz and 32 (high-end) Pentium based servers that are connected by Fast Ethernet and the Myrinet LANs. At a cost of less than $100,000 the above system delivers more than 2 GigaFLOPS (sustained), 8GB of main memory and 50GB of disk space.

2.2 The MOSIX operating system enhancements

MOSIX is a set of enhancements of BSD/OS with dynamic resource sharing algorithms that are geared for cluster computing. These algorithms are designed to respond dynamically to variations in resource usage among the nodes, by migrating processes transparently from one node to another, preemptively, to improve the performance. The granularity of the work distribution in MOSIX is the UNIX process. Users can run parallel applications on MOSIX by initiating multiple processes. Alternatively, MOSIX supports an efficient multi-user, time-sharing execution environment.

In MOSIX, each user interacts with the multicomputer via the user's "home" node (workstation or server), which is similar to the "home node" of Sprite [5]. The system image model is a computing cluster, in which all the user's processes seem to run at the home node, and all the processes of each user have the execution environment of the user's home node. Processes that migrate to other (remote) nodes use local (in the remote node) resources whenever possible, but interact with the user's environment through the user's home node. As long as the requirement for resources such as the CPU or main memory, are below certain threshold levels, all the user's processes are confined to the user's home node. When these requirements exceed the threshold levels, e.g., the load created by one CPU bound process or the size of the local memory, then the process migration mechanism migrates one or more processes to other nodes. The overall goal is to maximize the performance by efficient utilization of the network-wide resources.

The MOSIX enhancements are implemented in the BSD/OS kernel, without changing its interface, and they are completely transparent to the application level. Its main characteristics are:

- *Probabilistic information dissemination algorithms* - that provide each node with sufficient knowledge about available resources in other nodes, without polling or further reliance on remote information. Each node sends, at regular intervals, information regarding its available resources to a randomly chosen subset of nodes. At the same time it maintains a small buffer (window), with the most recent arrived information. The use of randomness supports scaling, even distribution of the information and dynamic configuration, e.g. overcome node failures.

- *Preemptive process migration* - that can migrate any user's process, any time, to any available node, transparently. The cost of the process migration includes a fixed cost, to establish a new process frame in the remote site, and an additional cost, proportional to the number of pages copied. In practice, only the page table and the dirty pages of the process are copied.
- *Dynamic load-balancing* - that continuously attempts to reduce the load differences between pairs of nodes by migrating processes from over-loaded nodes to under-loaded nodes. This scheme is decentralized in the sense that all of the nodes execute the same algorithms, and that the reduction of the load differences is performed independently by pairs of nodes. Other goals of the load-balancing algorithms are to respond to changes in the loads of the nodes, the runtime characteristics of the processes, and the number of nodes. The load-balancing algorithms prevail as long as there is no extreme shortage of resources such as free memory.
- *Memory sharing* - by a memory depletion prevention algorithm that is geared to place the maximal number of processes in the "network RAM" across all the nodes, to avoid as much as possible processes thrashing or the swapping out of processes. The algorithm is triggered when a node starts excessive paging due to insufficient free memory. In this case the algorithm overrides the load-balancing algorithm and it attempts to migrate a process to a node which has sufficient free memory, even if this migration results in an uneven load distribution.
- *Efficient kernel communication* - that was specifically designed to reduce the overhead of the system's kernel communications, e.g. between the process and its home node, when it is executing in a remote site. The new protocol is geared for a locally distributed system, e.g.,it does not support general inter-networking such as routing, and it assumes a relatively reliable media. The result is a fast, reliable datagram protocol with low startup latency and high throughput.
- *Decentralized control and autonomy* - each node is capable of operating as an independent system, i.e., it makes all its own control decisions independently, and there is no master-slave relationships between the nodes. This organization allows a dynamic configuration, where nodes may join or leave the network with minimal disruptions.
- *Scaling considerations* - that ensure that the system runs as well on large configurations as it does on small configurations. The main considerations include symmetry, and the use of randomness in the system control algorithms. Each node bases its decisions on partial knowledge about the state of the other nodes and it does not even attempt to determine the overall state of the cluster or any specific node.

The most noticeable properties of executing parallel applications on MOSIX are the symmetry and flexibility of its configuration, and its dynamic resource distribution policy. The combined effect of these properties is that application programs do not need to know the current state of the system configuration. Users need not recompile their applications due to node or communication fail-

ures, nor be concerned about the load of the various processors. Parallel applications can simply be executed by creating many processes, just like a single-machine system.

3 Performance of the MOSIX algorithms

This section presents the performance of the process migration, load-balancing and memory sharing algorithms of MOSIX. The execution platforms were two computing clusters, with 16 identical Pentium and Pentium-Pro servers that were connected by fast Ethernet and the Myrinet LANs respectively.

3.1 Performance of the process migration mechanism

Process migration is carried out in two stages. The first stage involves establishing a connection and negotiations between two nodes and creating a process frame in the remote node. In the second stage the active memory of the process is transferred. The cost of the negotiations stage is fixed, while the data transfer cost is proportional to the amount of data transfered.

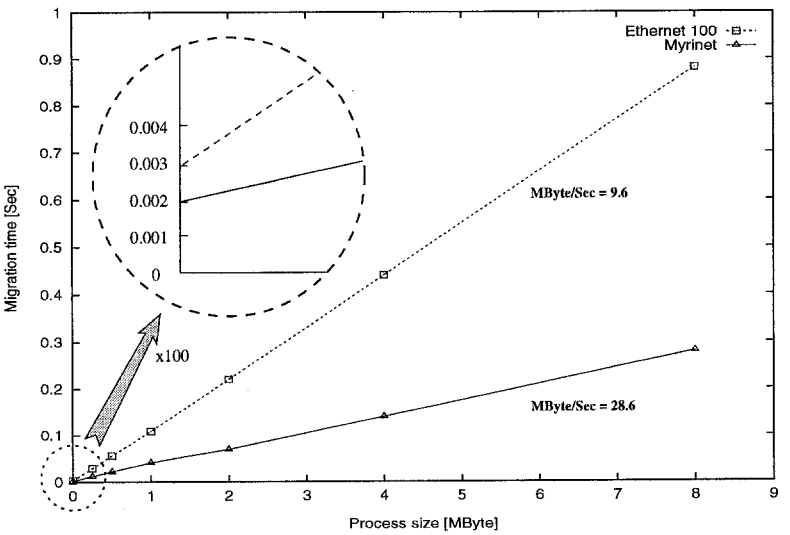

Fig. 1. *Migration times vs. process size*

The performance of the process migration mechanism between two Pentium-Pro 200MHz servers that were connected by Ethernet-100 and the Myrinet LAN, using the TCP/IP protocol are shown in Figure 1. From the figure it can be seen that the migration time is a linear function of the process size. It amounts to 76.8Mb/s for Ethernet-100 and 228.8Mb/s for the Myrinet. The corresponding migration latencies (magnified) are 3ms and 2ms respectively.

3.2 Performance of the load-balancing algorithms

We conducted two tests. First, we measured the total execution times of a set of identical CPU-bound processes under PVM, with and without the MOSIX load-balancing algorithms. This test was executed on a system with a background load, which reflects processes of other users in a typical time-sharing, multi-user computing environment. The background load was generated by a set of additional CPU-bound processes that were executed in cycles of a random computing period followed by an idle (suspend) period. The results of this benchmark are shown in Figure 2(a). These results show that the average slow-down of PVM vs. MOSIX is over 35%, with as much as 62% slow-down, in the measured range, for 20 processes.

A second benchmark, shown in Figure 2(b), presents the execution of parallel programs with unpredictable execution times, e.g. due to recursion, different amount of processing, etc., which are difficult to pre-schedule. We ran a set of CPU-bound processes that were executed for random durations, in the range 0 – 600 seconds. From the corresponding measurements it follows that the average slow-down of PVM vs. MOSIX is over 52%, with as much as 75% slow-down for 36 processes.

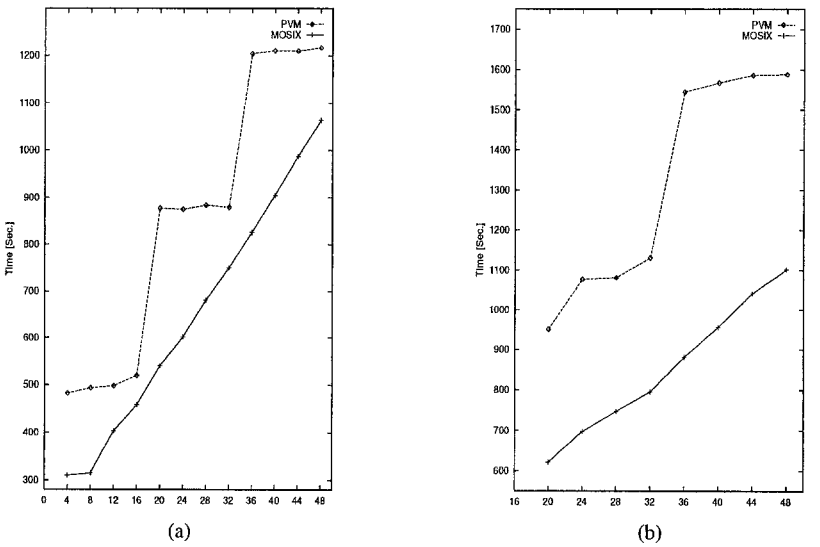

Fig. 2. MOSIX vs. PVM: (a) with background load, (b) random execution times

3.3 Performance of the memory sharing algorithm

This benchmark represents cases where the load of the nodes is balanced, but memory is not being used uniformly. We created a set of processes, with increasing average sizes, and distributed these processes evenly among the nodes

using PVM, such that some nodes run out of free memory. In this case, process migration should be motivated by memory considerations, to prevent excessive paging.

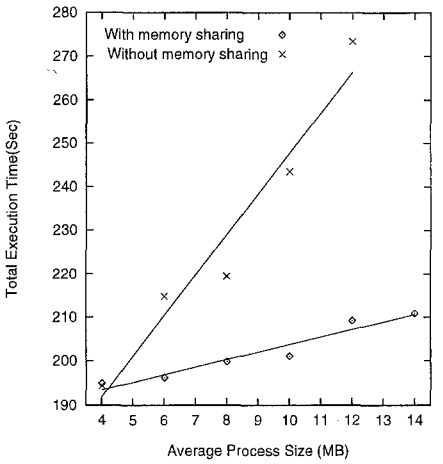

Fig. 3. *Performance of the memory-sharing algorithm*

The benchmark was executed in a load-balancing system with and without the memory-sharing algorithm. The results (see Figure 3) show that when the total requirement for memory is increased, the execution times of the load-balancing algorithm without memory-sharing are sharply increased due to excessive paging activities. The corresponding execution times with the memory-sharing have a significantly lower growth due to much lower paging activities.

4 Performance of parallel applications

This section presents our experience with the execution of several, large scale, parallel applications. The executions were carried on a MOSIX cluster with 32 Pentiums that were connected by fast Ethernet and 16 Pentium-Pro 200MHz that were connected by fast Ethernet and the Myrinet.

4.1 Global self organization of all known protein sequences

There are many efforts today to study sequences of proteins. Current efforts use algorithms for pairwise sequence comparisons, using a small number of sequences and "nearest neighbors" methods. Until recently, only few computational studies considered all, or many, of the known sequences. This project [9] deals with the universe of all protein sequences, by translating the space of these sequences

into an Euclidean space. Then a statistical, hierarchical clustering model is constructed. It offers additional insight into the large-scale organization and representation of the space of all protein sequences, and also reveals significant biological signatures of protein sequences.

The clustering algorithm associates each data point with the nearest centroid, where the centroids are reestimated to minimize the distortion within each cluster. This process is repeated until convergence to a (local) minimum of the distortion. At each iteration, the cluster of highest aspect ratio is split. The algorithm is performed on two randomly chosen subsets of the data, aborting every split on which the two processes "disagree". The process terminates when all attempted splits get aborted.

The clustering algorithm was executed on a 32-node MOSIX configuration. The input data consisted of about 540,000 points, in 200-dimension Euclidean space, that were initially split among several processes. In the execution, each process made the association of its (local) data points, and then calculated the first two moments (mean and covariance). For synchronization, a "master" process collects the results from all the "slaves", re-estimates the centroids and redistributes them. Since the algorithm is computational intensive, with only a small amount of communication, the speedup obtained was proportional to the number of processors. The total amount of memory was about 1GByte. The longest execution time was 20 days, i.e., over 15,000 hours, during which we executed a case with 150 clusters.

Observations: this is a straightforward example of a parallel application that can benefit from the multiplicity of resources, e.g. CPU cycles and memory of a CC. The main MOSIX advantage is its ability to execute a memory consuming application for 20 days, without swapping or excluding other jobs, by using the memory sharing algorithms.

4.2 Quantum simulations of large molecules

This project involves the development of novel parallel algorithms for quantitative simulations of large biological molecules, by computing the multidimensional, quantum wave-functions of these systems and studying their spectroscopic and dynamical properties. The algorithms use classical quantum mechanics methods to solve the time-independent and the time-dependent Schrödinger equations for systems of high dimensionality. The method uses as a first approximation a separation of variables, so that different degrees of freedom are handled by different processes. This is an intensive compute process, which requires a large amount of CPU time. The method was recently applied to calculate the ground-state wave-function and excited states of the protein BPTI.

The specific application that was executed on MOSIX was parallel molecular dynamics on chemical systems with over 10,000 degrees of freedom, that simulate molecules of the order of complexity of proteins. The algorithm, which includes correlation effects (between the different degrees of freedom) was applied on Bacteriorhodopsin. Figure 4 shows the performance of this algorithm

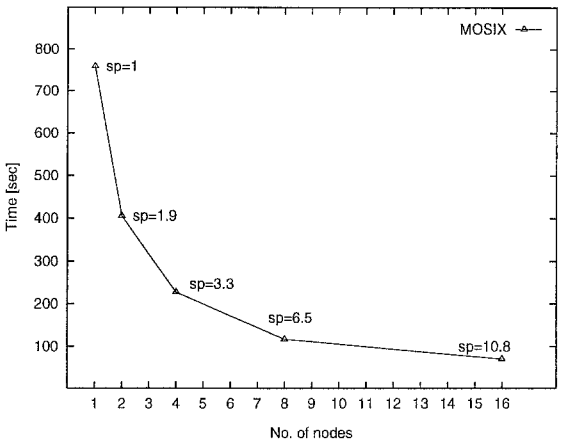

Fig. 4. *Speedup (sp) obtained in a classical molecular dynamics simulation*

and the speedup obtained, using a Pentium-Pro cluster with 16 nodes. The parallel algorithm uses a straight-forward spatial decomposition, with an internal mechanism for load-balancing between the different processes by modifications to the initial data decomposition. As the number of partitions increases, the communications overhead becomes very significant, since the CPU work of each partition decreases.

Observations: an intensive compute and communication bound parallel application can obtain a good speedup, with the increase in the number of nodes. In the current case, this speedup is impaired for 16 nodes due to the (relatively small) size of the problem.

4.3 Molecular Dynamics simulation

Molecular Dynamics (MD) simulation has been used extensively as a tool to study irradiation damage. We describe two parallelization methods of large scale simulations.

High energy MD: The first case is a physical system that consists of an energetic atom (in the range of 100 kev) impacting a surface, where a simulation for a large number of time steps and a large number $N > 10^6$ of atoms, is needed. The key issues are the different time scales involved and the number of particles.

The simulation applies a spatial decomposition into cubic cells. Most of the calculation is local, except the force calculation phase. In this phase each process needs data from all its 26 neighboring processes. Due to the use of non-local potentials, data communication between the processes became a major part of the execution. Hence, the decomposition attempted to minimize the surface to volume ratio. All communication was implemented using the PVM library.

The simulations were executed on a 16 Pentium-Pro MOSIX configuration, and the obtained results were compared with similar executions on the IBM

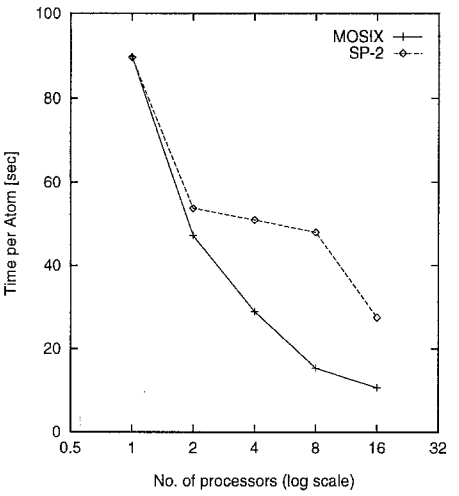

Fig. 5. *MD performance of MOSIX vs. the SP-2*

SP-2. The results of these executions are shown in Figure 5. From the figure it follows that MOSIX outperforms the SP-2 by a factor of 2.6 for 16 nodes. The corresponding factors for 4 and 8 nodes are even higher. We suspect that in these last two cases, slower ("thin") nodes were used in the SP-2. Note that in spite of the massive communication, MOSIX obtained a speedup of 8.4.

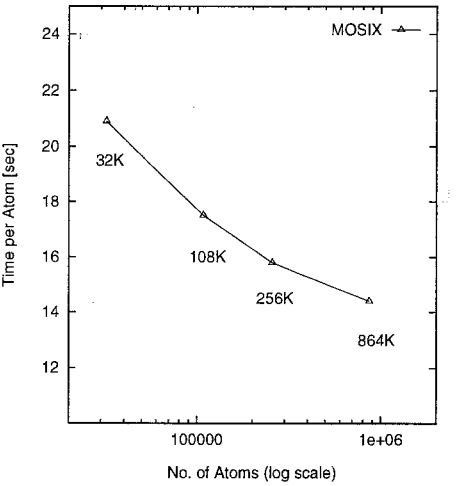

Fig. 6. *MD scalability in problem size on MOSIX*

Figure 6 shows the scalability in the problem size on MOSIX (using 8 nodes). Observe that the time per atom decreases with the increase in the problem size. This is due to a better initial load distribution and reduced communication

overhead per atom. Since the simulation scales well in size, we expect that a large number (over 4 million) of atoms can be simulated on a MOSIX system with 32 nodes.

Observations: a cluster of Pentium-Pro 200MHz PC's connected by Myrinet can outperform the SP-2 for heavy applications such as MD simulations, which are both CPU and communication intensive.

Integration using MD: This problem consists of the statistical analysis of the desorption probability of an atom from a surface due to a nuclear stimulated desorption. Prediction of the spatial distribution of a desorbing atom requires simulations over a wide range of physical parameters. The time needed for each simulation varies considerably with the initial parameters. A single case takes about 40 minutes (average) on an "SGI" workstation, resulting in up to a month execution time for a simulation with 10^3 different initial conditions.

The parallelization consists of concurrent execution of a large number of independent simulations, using different initial conditions. When some jobs end, others are created, as needed, until a sufficiently small variance in the observed variable was achieved. The main execution requirement is to keep the system load balanced, in spite of the unpredictable execution times and total number of jobs. The MOSIX system enabled to completely disregard this issue and relieved the program from all assignment related decisions.

Observations: The MOSIX load-balancing scheme simplifies the execution of jobs with unpredictable execution parameters, e.g., time and number of processes, especially in the presence of other users' jobs.

5 Conclusions

This paper presented the performance of the MOSIX parallel system for a cluster of PC's. MOSIX supports dynamic resource sharing among the PC's for performance scalability of parallel applications in a multi-user, time-sharing environment. The net result is that users do not have to know the current state of the resource usage and the load of the various workstations, or even the system configuration. Parallel applications can be executed by simply creating many processes, almost like in an SMP. The performance of the CC MOSIX shows a good utilization of the resources, relatively good speedups in a scalable configuration and competitive results vs. the SP-2. A major advantage of MOSIX over other CC environments is its ability to respond at run-time to unpredictable requirements by many users and irregular resource usage, e.g., execution times, memory usage or the number of processes - MOSIX adapts well to all such cases.

The main outcome of this paper is that it is possible to build a low-cost, scalable multicomputer system from commodity components, such as PC's and PVM, as an alternative to traditional mainframes and MPP's. With an intelligent operating system that supports global resource sharing, such as MOSIX, these multicomputer systems offer a convenient, general-purpose environment for executing large scale, demanding sequential and parallel applications.

The CC MOSIX is fully compatible with BSD/OS [8]. It has been operational for over 4 years. It is used for student course work, research of operating systems for scalable CC, and the development of parallel applications. A limited (up to 6 processors) version of MOSIX, called MO6, is available on the Internet for experimentation at URL `http://www.cs.huji.ac.il/mosix`.

Acknowledgments

Special thanks to A. Shiloh for his valuable work. We would also like to thank Y. Ashkenazy, A. Braverman, E. Fredj, B. Gerber, I. Gilderman, I. Kelson and G. Yona for their contributions.

This research was supported in part by the Ministry of Defense and the Ministry of Science.

References

1. T.E. Anderson, D.E. Culler, and D.A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.
2. A. Barak, S. Guday, and R.G. Wheeler. The MOSIX Distributed Operating System, Load Balancing for UNIX. In *Lecture Notes in Computer Science, Vol. 672*. Springer-Verlag, 1993.
3. D.J. Becker, T. Sterling, D. Savarese, J.E. Dorband, U.A. Ranawak, and C.V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. In *Inter. Conf. on Parallel Processing*, 1995.
4. N.J. Boden, D. Cohen, R.E. Felderman, A.K. Kulawik, C.L. Seitz, J.N.Seizovic, and W-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.
5. F. Douglis and J. Ousterhout. Transparent Process Miration: Design Alternatives and the Sprite Implementation. *Software – Practice & Experience*, 21(8):757–785, August 1991.
6. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM - Parallel Virtual Machine*. MIT Press, Cambridge, MA, 1994.
7. W. Gropp, E. Lust, and A.Skjellum. *Using MPI*. MIT Press, Cambridge, MA, 1994.
8. R. Kolstad, T. Sanders, J. Polk, and M. Karles. *BSDI Internet Server (BSD/OS 2.1) Release Notes*. Berkeley Software Design, Inc., Colorado Springs, CO, January 1996.
9. M. Linial, N. Linial, N. Tishby and G. Yona. Global self organization of all known protein sequences reveals inherent biological signatures. *Molecular Biology, to appear*, 1997.