A Static Load Balancing Software for Parallel Applications

Adriano Joaquim de Oliveira Cruz[1], Cláudia Rita de Franco[2], Leonardo Silva Vidal[3]

[1]Department of Computer Science, Núcleo de Computação Eletrônica,
Universidade Federal do Rio de Janeiro
Cx Postal 2324, Rio de Janeiro, RJ, CEP 20001-970
{adriano@nce.ufrj.br}

[2]Department of Computer Science, Núcleo de Computação Eletrônica
Cx Postal 2324, Rio de Janeiro, RJ, CEP 20001-970
Universidade Federal do Rio de Janeiro
{crfranco@nce.ufrj.br}

[3]Department of Computer Science, Núcleo de Computação Eletrônica
Cx Postal 2324, Rio de Janeiro, RJ, CEP 20001-970
Universidade Federal do Rio de Janeiro
{leonardo@nce.ufrj.br}

Abstract—

This work describes the implementation and benchmarks applied to a load balancing software designed to improve performance of parallel applications running on networks of heterogeneous and non-dedicated workstations.

A user level mechanism to gather workload information about each node and the policy to treat this information in order to generate a precise snapshot of the workload of each node of the parallel machine are described throughout this work.

An analysis of the main issues concerning workload evaluation is provided, along with a brief explanation on the support offered by current operating systems and ways to overcome their problems.

Finally, results and interpretations of comparative tests made between *BEC/PVM* applications and *PVM* applications are presented.

Keyword— **Parallel Processing, Load Balancing.**

I. INTRODUCTION

Running parallel applications on computer networks is a cheaper alternative to expensive parallel computers. These networks are shared by a great number of users. But along the day, only some computers are used or remain idle for large periods of time. The ability to identify idle computers and spawn the processes of a parallel application on them can give a considerable performance improvement to any parallel application. Furthermore, most users do not generate heavy loads to their computers, so their computers can also host processes of the parallel applications.

A mix of fast and slow computers forms many networks. By giving preference to the faster computers, the performance of the parallel applications is further improved.

PVM (Parallel Virtual Machine) [GEI 94] is a tool that allows a heterogeneous collection of workstations and supercomputers to function as a single parallel computer, which is called a virtual machine. Each node of the virtual machine is called a host and the processes forming the parallel application are called tasks. PVM was chosen as our test bed for its high level of portability, simple message passing programming and ability to be installed by any user.

II. LOAD BALANCING

The main goal of load balancing a parallel application is to distribute work among the nodes of a parallel machine in order to obtain better performance. It is accomplished by giving a greater portion of work to the faster and less loaded nodes.

A balanced parallel application may have other benefits. All tasks will have comparable execution times since each one will receive a portion of work appropriated to its host capabilities. As a consequence, the execution time of a parallel application will not be delayed by the slower task.

We can divide load balancing in two categories. The simpler one is called **static load balancing**. This type of load balancing takes place before new tasks are spawned. It consists on choosing the best hosts to spawn the tasks that will work in the same host throughout their execution. Jackson and Humphres present a extension to PVM that provides static load balancing [JAC 97, HUM 95]. This extension requires that users run their own benchmarks in order to evaluate hardware performance and the results must be given to the system and it is implemented as a modification to the PVM source code.

Static load balancing cannot deal with changes in the workload of a host, since the best way to deal with them is to move the task to a less loaded host. This mechanism is called task migration and the load balancing that uses it is called **dynamic load balancing**. The mechanism of task migration is able to move a task to a faster host, if it becomes available, even when the workload is stable.

Task migration is a complex mechanism, which involves a great deal of operating system's work. MPVM [CAS 95] and CONDOR [LIT 97] are software tools that provide process migration and dynamic load balancing.

III. BEC

BEC [FRA 98] is a normal PVM application that accepts requests from other PVM applications in order to spawn their tasks. BEC neither changes PVM implementation, nor requires any special privileges within PVM. It has its own set of functions that replaces some PVM functions. BEC provides transparent load balancing to user applications. BEC´s current version is limited to static load balancing, but future versions may implement this mechanism.

The PVM's functions substituted are ***pvm_spawn*** and ***pvm_parent***. The former is responsible for the spawning of new tasks and the later informs a task the identity of its parent task. The function ***pvm_spawn*** is replaced by ***bec_spawn*** and the function ***pvm_parent*** is replaced by ***bec_parent***. The function ***bec_spawn*** is the point where PVM's non-balanced tasks spawning mechanism is substituted by BEC's load balanced task spawning mechanism.

BEC is written in C and has been ported to the following operating systems: *Linux*, *SunOS*, *Solaris* e *AIX*.

A. Architecture

BEC has three types of daemons: the **master daemon**, the **probe daemon** and the **creator daemon**. Figure 1 shows BEC architecture with PVM acting as a communication layer between BEC's daemons. ***pvmd3*** is the pvm daemon, ***becd*** is the master daemon, ***becpd*** is the probe daemon and ***beccd*** is the creator daemon.

B. Master Daemon

The master daemon is the centre of BEC's architecture. There is only one instance of it serving requests from tasks all over the virtual machine.

The master daemon spawns and controls the work done by the other BEC daemons, reacts to changes in the virtual machine and answers the requests from client applications.

At the start up, the master daemon connects to PVM, gets the virtual machine configuration, tries to spawn one probe daemon and one creator daemon in each host and requests PVM for notification on any change on the virtual machine. After start up, the master daemon starts listening to requests from client tasks and communications from the other BEC's daemons.

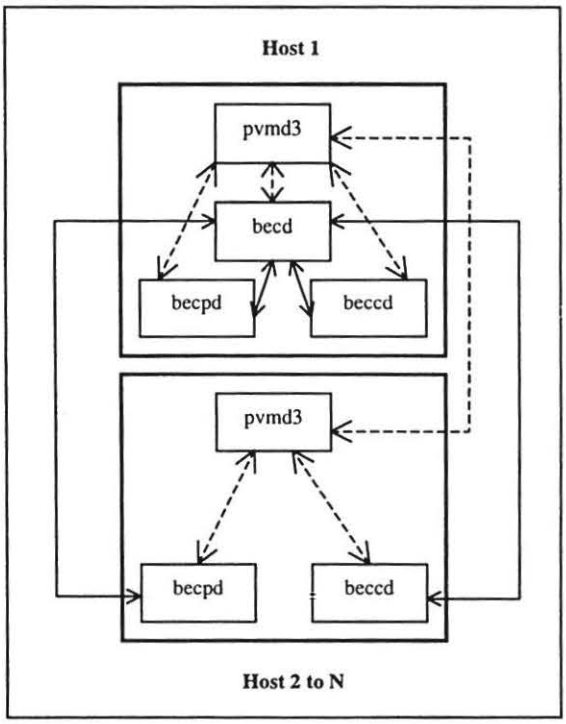


Fig. 1 BEC's architecture

C. Creator Daemons

The creator daemons are responsible for the spawning of new tasks in response to a call to bec_spawn. The tasks are spawned by a call to pvm_spawn specifying the local host as the target host. The creator daemons exist to speed up the spawning of new tasks on networks of any size.

If a creator daemon does not receive any message from the master daemon within a specified period of time, it sends a special message to the master daemon in order to discover if BEC is still running. If there is no answer, it stops.

D. Probe Daemons

The probe daemons gather information about hardware capabilities and workload information of each host in the virtual machine.

Should an attempt to send a message containing these statistics fails the probe daemon self-destructs.

IV. Performance Evaluation

In order to perform load balancing, BEC must have data about the performance of all hosts and must be able to make comparisons among them.

The current version of BEC combines all performance data from a host into a single number called **performance index**.

The range of possible parameters used to evaluate the performance is very broad. However, the two most important for BEC are the workload, which shows how much a host is being used, and the hardware speed, which prevents BEC from choosing empty but slow hosts.

All UNIX versions have some way to probe for the workload, but the policy to access this information usually involves some special privileges. The hardware speed is trickier to obtain from the operating systems. Therefore, BEC uses time benchmarks to obtain both values.

BEC also considers some other parameters: the **number of users**, the **number of terminals** and the **number of active tasks spawned by BEC** on a host. These are complementary parameters used to modify the performance index of all the hosts.

The set of parameters used might be changed in future versions of BEC. The benchmark, the meaning, and the methods to obtain each parameter are explained below.

E. Benchmark

This benchmark measures the execution time of a set of tests. The tests are a summation of an integer series, a floating point summation and the copy of data blocks in memory. The summation involves all basic arithmetic instructions.

These tests represent the most common processing tasks carried out by parallel applications. The span of the tests is limited by the hosts' resources consumed during the tests and the need to exceed the usual Unix timeslice. Long duration tests would imply greater resource consumption, affecting other tasks and degrading the host's performance. Fast tests would not exceed the process' timeslice and the process would not be preempted, producing a false workload measure.

The benchmark is carried out by the probe daemons on their respective hosts and sent to the master daemon. The benchmark is repeated periodically to give an updated snapshot of the host's performance.

The probing interval can be changed by the function ***bec_probeinterval*** allowing the user to define the best probing interval for his needs.

Workload Index (WI) – it is the measure of how loaded with active processes a host is. It is proportional to the time elapsed during the execution of the benchmark. The time elapsed contains the time the process was effectively running and the time the process was stopped while the other active processes executed. It is a dynamic parameter collected periodically by the probe daemons.

Hardware Index (HI) – this parameter measures how fast the combination of hardware and operating system is, it is proportional to the processor holding time necessary to accomplish a task. The faster combinations should need less time to run, hence they have the lower values. It is obtained by the sum of the user time and the system time the process uses to complete the benchmark. The hardware index is a constant parameter taken once at the probe daemons start up.

Number of Users (NUser) – it is the number of users logged in a host. Each logged user consumes part of the resources of a host, so it is a good policy to avoid hosts being used by many users. It also prevents the parallel application from disturbing other users. This parameter is collected periodically by the probe daemons reading the system's *user login records.*

Number of Terminals (NTty) – it is the total number of terminals opened by the users of a host, this number includes terminals used in X sessions. More open terminals increase the possibility of workload peaks. This parameter is collected by the probe daemons by periodically reading the system's *user login records.*

Number of Tasks Spawned by BEC (NTask) – it is an important parameter that prevents BEC from spawning large numbers of tasks through successive calls to bec_spawn between performance data gathering. It is updated by the master daemon when task is spawned by a creator daemon or when a spawned task exits. By increasing this parameter after a task is spawned, the performance index gets worse and the host is less likely to be chosen again. When the task exits the host returns to its normal performance index. The updates in the parameter avoid the need to collect performance data again for the host.

F. Performance Index

The Performance Index for a host is the combination of the results of the benchmark on that host. The master daemon receives the results from the probe daemons and uses the following equation to evaluate the performance index:

$$\textit{Performance Index} = \textit{Scale} \times \textit{HI} \times \textit{WI} \times (1 + \textit{wu} \times \textit{NUser} + \textit{wy} \times \textit{NTty} + \textit{wt} \times \textit{NTask})$$

Scale is a range adjustment to the performance index. The factors **wu**, **wy** and **wt** are weights to the last three parameters. BEC provides the function ***bec_weights*** to allow the user to fine-tune these weights.

The lower the hardware index and the workload index are, the better is the performance of the host, therefore the better hosts are those with lower performance indexes.

V. TASK SPAWNING

The process of task spawning begins with a call to the function bec_spawn. This function forwards the request to the master daemon in a message containing all parameters passed by the user, some PVM environment variables and shell environment variables exported with PVM_EXPORT.

The master daemon selects the best hosts that match the user conditions and issue a request for the creator daemons of the selected hosts to spawn the desired number of tasks. All the request data is stored in a internal list of task spawning requests in execution, so the master daemon can continue to listen to other messages while it waits for the responses from the creator daemons. The algorithm to choose hosts will be explained latter.

The creator daemons register the requesting task's PVM environment in its own PVM environment to guarantee that the spawned tasks will inherit the correct environment, and then call pvm_spawn to locally spawn the tasks for its host. The resulting tasks' identifications are sent back to the master daemon.

When the master daemon receives a response from a creator daemon, it updates the corresponding entrance on the list of task spawning requests in execution and updates the performance index for the host. Once all the responses pending for the request arrive, the master daemon sends the number of spawned tasks and their identifications to the requesting task.

BEC's heuristics to distribute tasks among the hosts uses the idea that it is better to create more tasks on a faster host then adding a slower host to the set of hosts. Therefore, if host A has a performance index that is half the performance index of a host B, then host B should get half or less the number of tasks assigned to host A.

Task spawning through BEC is slower than task spawning via PVM, but the gains in performance fully compensate this disadvantage.

VI. COMPARATIVE TESTS

This section presents the results of tests comparing the performance of PVM parallel applications running with or without BEC's support.

The data for the first and fourth tests were extracted from a program that generates pi. These tests were designed to evaluate workstation's global performance avoiding influences of network load. The second test was a program that calculates a scalar product and the third a program that computes the average and the standard deviation of a large set of points. These tests, besides evaluate the workstation's global performance, measure the delays caused by large exchange of data on common networks.

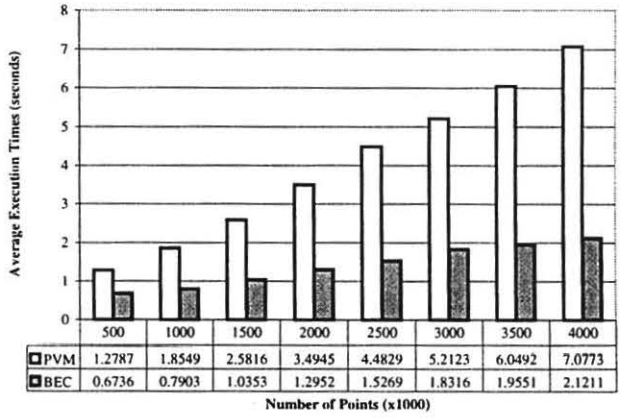
Each test was run 50 times on two different networks and produced the average execution times shown here. The first network comprised five Sun4m workstations running Solaris and four Sun4c workstations running SunOS. Nine Pentium PCs of equal configuration running Linux composed the second one. Both networks remained fully operational.

During the tests, other researchers and students used an average of four workstations. The workload probing was repeated in a 120 seconds interval.

The main goal of the tests on the SUN network was to prove that BEC is able to choose the faster machines among a pool of hosts of different performance. Whereas the main goal of the tests on the Linux network was to show that BEC can choose the least loaded hosts to improve performance even when all machines have the same hardware characteristics.

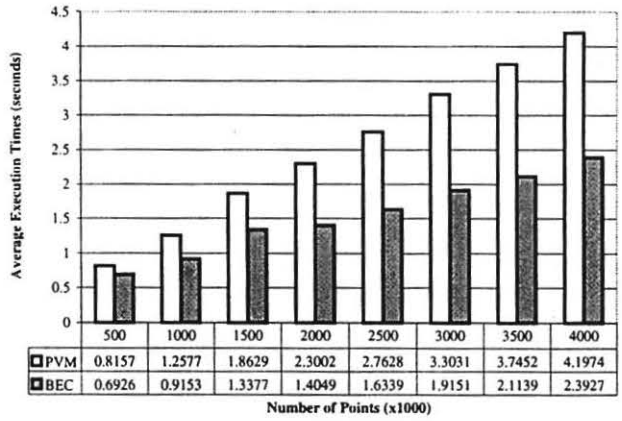
G. Numeric Computation of Pi Varying the Number of Points

The purpose of this test is to evaluate the behaviour of a processor intensive parallel application. The network load has minor influence in the execution times since only the borders of the intervals assigned to the slave tasks and the partial results are exchanged between the tasks.



	500	1000	1500	2000	2500	3000	3500	4000
PVM	1.2787	1.8549	2.5816	3.4945	4.4829	5.2123	6.0492	7.0773
BEC	0.6736	0.7903	1.0353	1.2952	1.5269	1.8316	1.9551	2.1211

Fig. 2 Numeric Computation of Pi on a Heterogeneous Network



	500	1000	1500	2000	2500	3000	3500	4000
PVM	0.8157	1.2577	1.8629	2.3002	2.7628	3.3031	3.7452	4.1974
BEC	0.6926	0.9153	1.3377	1.4049	1.6339	1.9151	2.1139	2.3927

Fig. 3 Numeric Computation of Pi on a Homogeneous Network

As Figure 2 shows, BEC's performance gains on the heterogeneous network increased from 47% to 70% as the number of points increased. Similarly, Figure 3 shows that BEC's performance gains on the homogeneous network

increased from 15% to 44% as the number of points increased.

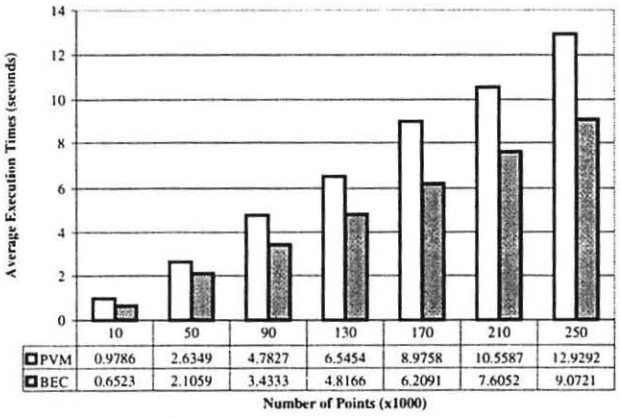
The performance gains increased with the number of points, because the time taken with data exchanging between tasks is constant, so the total execution time becomes more dependent on the performance of the host as the number of points increase. BEC chose the faster hosts, thus giving more performance gains.

H. Scalar Product Varying the Size of the Vectors

This is a data intensive test, involving sending large vectors to the slave tasks and receiving a floating point number as the partial result. The processing done by the slave tasks comprised two products and a sum per point. So this test measures the behaviour of BEC with an application requiring large data exchange and low processing.

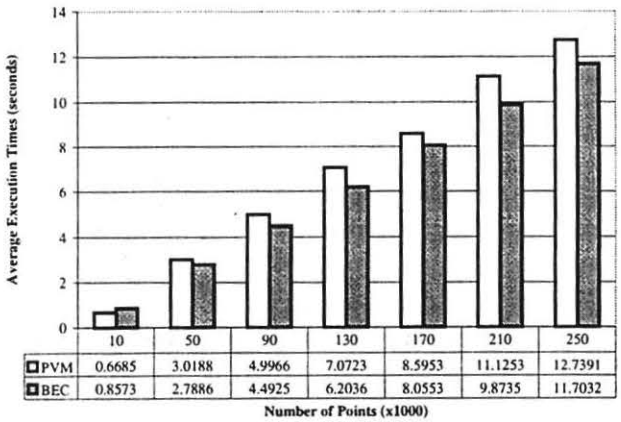
Most of the execution time is taken by the time to transfer data, so this application is highly network dependent.

BEC's performance gains on the heterogeneous network ranged between 20% to 33%, as shown in Figure 4.



	10	50	90	130	170	210	250
PVM	0.9786	2.6349	4.7827	6.5454	8.9758	10.5587	12.9292
BEC	0.6523	2.1059	3.4333	4.8166	6.2091	7.6052	9.0721

Fig. 4 Scalar Product of Two Vectors on a Heterogeneous Network



	10	50	90	130	170	210	250
PVM	0.6685	3.0188	4.9966	7.0723	8.5953	11.1253	12.7391
BEC	0.8573	2.7886	4.4925	6.2036	8.0553	9.8735	11.7032

Fig. 5 Scalar Product of two Vectors on a Homogeneous Network

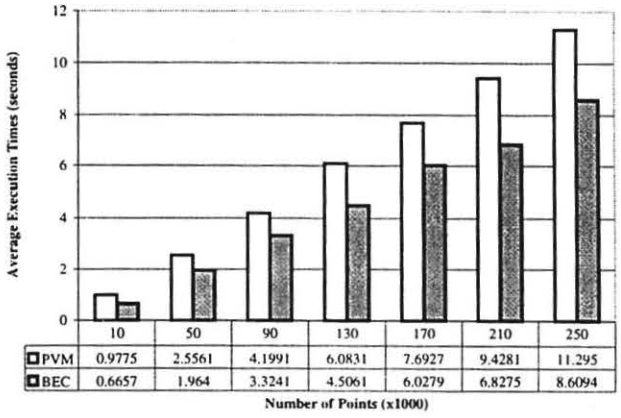
Figure 5 refers to the same tests run on a homogeneous network. BEC's performance gains ranged between 6% and 12%, except for 10,000 points where BEC increased execution time by 28%. This is explained by the execution time being too small to overcome the extra time to spawn tasks required by BEC.

I. Average and Standard Deviation of a Set Varying the Number of Points

This is another data intensive test. It consists on sending parts of a set of points to the slave tasks and receiving back the partial sum to compute the average. The average is sent to the slave tasks to compute the partial standard deviation.

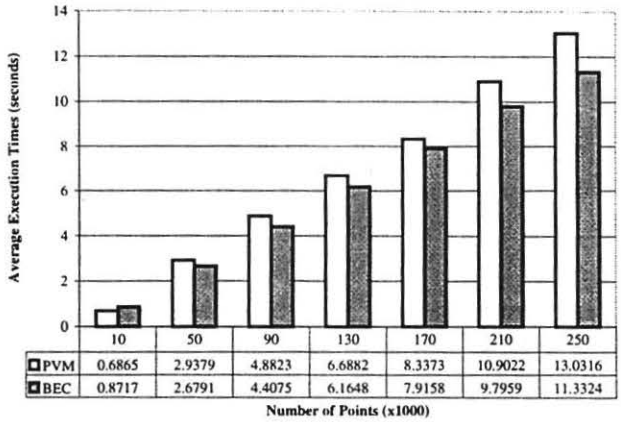
As Figure 6 shows, the performance gains attained by BEC, on the heterogeneous network, were between 22% and 32%.

Figure 7 shows that for 10,000 points BEC increased the execution time by 28% on the homogeneous network. For higher number of points, performance gains were between 5% and 13%.



	10	50	90	130	170	210	250
PVM	0.9775	2.5561	4.1991	6.0831	7.6927	9.4281	11.295
BEC	0.6657	1.964	3.3241	4.5061	6.0279	6.8275	8.6094

Fig. 6 Average and Standard Deviation of a Set of Points on a Heterogenous Network



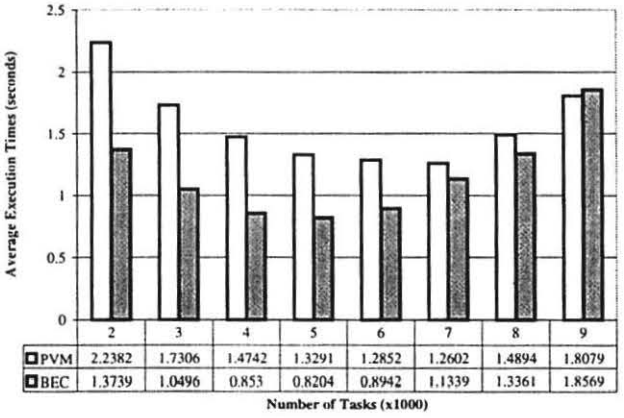
	10	50	90	130	170	210	250
PVM	0.6865	2.9379	4.8823	6.6882	8.3373	10.9022	13.0316
BEC	0.8717	2.6791	4.4075	6.1648	7.9158	9.7959	11.3324

Fig. 7 Average and Standard Deviation of a Set of Points on a Homogeneous Network

J. Numeric Computation of Pi Varying the Number of Tasks

This test shows how BEC's performance gains are affected by increasing the number of tasks, which forces BEC to choose slower hosts.

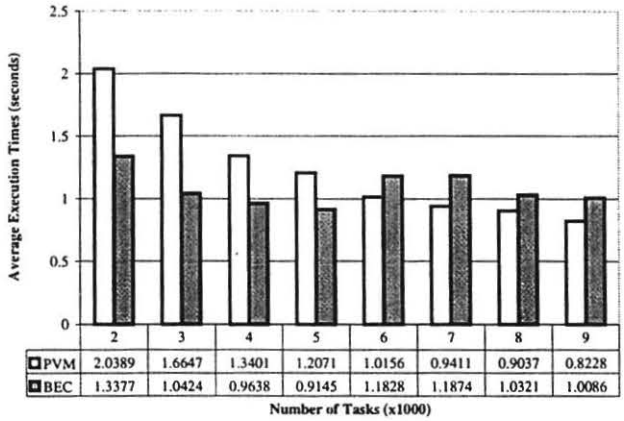
Figure 8 shows the test made on the heterogeneous network. BEC obtained performance gains until the number of tasks increased to five, above this value BEC was forced to use the hosts with slower hardware. The pure PVM's applications could get performance gains a little further because of BEC's task spawning time. BEC attained the best result with five tasks distributed over the five Solaris workstations.



	2	3	4	5	6	7	8	9
PVM	2.2382	1.7306	1.4742	1.3291	1.2852	1.2602	1.4894	1.8079
BEC	1.3739	1.0496	0.853	0.8204	0.8942	1.1339	1.3361	1.8569

Fig. 8 Numeric Computation of Pi on a Heterogeneous Network

Figure 9 shows the test made on the homogeneous network. Pure PVM applications gained performance as the number of tasks increased because the amount of work per task decreased. BEC also gained performance while there were free hosts. As the loaded hosts were added, the performance was reduced because the task spawning time on these hosts is longer.



	2	3	4	5	6	7	8	9
PVM	2.0389	1.6647	1.3401	1.2071	1.0156	0.9411	0.9037	0.8228
BEC	1.3377	1.0424	0.9638	0.9145	1.1828	1.1874	1.0321	1.0086

Fig. 9 Numeric Computation of Pi on a Homogeneous Network

VII. Conclusions

The results obtained show the value of load balancing to squeeze even more performance from computational resources. These performance gains can be used to solve even more complex problems.

BEC is completely functional and has been successfully used in undergraduate classes learning parallel programming and research activities.

The main advantage of BEC is the reduction of the execution time for parallel applications. Another result is the ability of BEC to avoid busy workstations, decreasing the impact of parallel applications to other users.

Work results also showed that the standard deviation of the execution times was reduced by 80% by comparison to pure PVM applications. This allows better prediction of the completion times when parallel applications are executed repeatedly.

The implementation of BEC allows easy porting of PVM applications, which can be done by replacing a small number of function calls, with no changes on programming methods. BEC does not require special privileges to be installed and used, allowing any user access to the gains of performance provided by load balancing.

REFERENCES

[CAS 95] CASAS, Jeremy; CLARK, Dan; KONURU, Ravi; OTTO, Steve; PROUTY, Robert; WALPOLE, Jonathan. *MPVM: A Migration Transparent Version of PVM*, Technical Report CSE-95-002, Dept. of Computer Science and Engineering, Oregon Graduate Institute of Science & Technology, February 1995.

[FRA 98] FRANCO, Cláudia Rita de; VIDAL, Leonardo Silva. *BEC - Balanceador Estático de Carga para o PVM,* B. Sc. Project, Federal University of Rio de Janeiro, 1998.

[GEI 94] GEIST, Al; BEGUELIN, Adam; DONGARRA, Jack; JIANG, Weicheng; MANCHECK, Robert; SUNDERAM, Vaidy. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press.

[HUM 95] HUMPHRES, Chris W. *A Load Balancing Extension for the PVM Software System,* M. Sc. Thesis, University of Alabama, 1995.

[JAC 97] JACKSON, David J; HUMPHRES, Chris W. *A Simple Yet Effective Load Balancing Extension to the PVM Software System.* Parallel Computing, vol. 22, Issue 12, February 1997, pp 1647-1660, North Holland

[LIT 97] LITZKOW, Michael; TANNENBAUM, Todd; BASNEY, Jim; LIVNY, Miron. *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, Technical Report #1346, University of Wisconsin-Madison Computer Sciences, April 1997.