1995

Characterization of the Paging Activity of NAS Benchmark Programs on the Intel Paragon

Kuei Yu Wang

Dan C. Marinescu

Report Number:
95-015

CHARACTERIZATION OF THE PAGING ACTIVITY OF NAS BENCHMARK PROGRAMS ON THE INTEL PARAGON

Kuei Yu Wang
Dan C. Marinescu

Computer Sciences Department
Purdue University
West Lafayette, IN 47907-1398

CSD-TR-95-015
March 1995

Characterization of the Paging Activity of NAS Benchmark Programs on the Intel Paragon*

Kuei Yu Wang and Dan C. Marinescu
Computer Sciences Department
Purdue University
West Lafayette, IN 47907-1398

March 1995

Abstract

In this work, a two prong dynamic paging activity data analysis is considered. The "skyline analysis" studies how similar/dissimilar is the paging activity of different node programs in SPMD execution mode. This analysis is done by determining the rate at which different events occur, by isolating the peaks of activity from the background, and by correlating the time of occurrence and the amplitude of these peaks. The second type of analysis is the "cumulative profile," used to determine the total load due to paging activity on the communication and the I/O sub-system.

Finally, we report the result of analysis of the paging activities of a representative class of applications, the NAS parallel benchmark programs.

*Work supported in part by NSF under grants CCR-9119388 and BIR-9301210, by a grant from Intel Corporation and by CNPq Brazil.

Contents

1 Introduction

Massively Parallel Processing Systems (MPPs) and clusters of workstations are two classes of distributed memory MIMD (Multiple Instruction Multiple Data) systems used to solve problems which require a substantial amount of computing cycles, main memory, and secondary storage.

Recently MPPs like the Intel Paragon and the IBM SP2 running commodity operating systems on all processing elements (PEs) have emerged. Demand paging is one common feature of commodity operating systems such as UNIX and OSF/1. The support for virtual memory on MPPs is a significant advance because it allows writing portable code without concern for the actual memory configuration of a specific system, and makes MPPs more attractive for broader classes of applications.

The support for virtual memory on MPPs also raises a series of questions not previously addressed in virtual memory for uniprocessor systems. A first example is the process scheduling. In the uniprocessor system, all processes are independent; the latency of a page fault is hidden by blocking the process currently experiencing the fault and then, by switching the context to another process ready do run. In multiprocessor systems, where processes of a process group cooperate to carry out a specific computation task, the method of hiding the latency of a page fault can be applied only if all processes of a group experience the page fault at the same time and the overhead associated with a group context switch is smaller than the page fault service time. Unfortunately, previous measurements reported in [5] suggest that neither condition is satisfied.

Another important issue is how to configure a parallel system to provide efficient virtual memory support, such as the number of I/O nodes needed and their optimal placement. This question can only be answered after observing and monitoring the paging activity of a substantial number of representative applications running on existing MPPs.

The focus of this paper is the study of the paging activity of parallel programs based on the SPMD (Same Program Multiple Data) paradigm, running on distributed memory MIMD systems, called in the following DMIMD systems. We present models of paging activity of a parallel program, a monitoring tool to collect data pertinent to the paging activity and a data processing tool to analyze the collected data.

In this work, a two prong dynamic paging activity data analysis is considered. The "skyline analysis" studies how similar/dissimilar is the paging activity of different node programs in SPMD execution mode. This analysis is done by determining the rate at which different events occur, by isolating the peaks of activity from the background, and by correlating the time of occurrence and the amplitude of these peaks. The second type of

analysis is the "cumulative profile," used to determine the total load due to paging activity on the communication and the I/O sub-system.

Finally, we report the result of analysis of the paging activities of a representative class of applications, the NAS parallel benchmark programs.

2 Characterization of Paging Activity of Parallel Programs on MPPs

A massively parallel system consists of compute nodes, I/O nodes, and service nodes connected by a high speed interconnection network. Network topologies used in existing systems are hypercubes (e.g., the Intel iPSC/860, the NCUBE), 2-D meshes (e.g., the Intel Paragon), 2-D tori (e.g., the Cray MPP), fat-trees (e.g., Thinking Machines CM5), or extra stage omega networks (e.g., IBM's SP1 and SP2).

Each compute node consists of one or more processors having a common memory, possibly co-processors (e.g., a message co-processor) and a network interface. In addition to the configuration mentioned above, an I/O node has I/O interfaces for devices like disks, and/or computer networks. Space and cost considerations limit the number of I/O nodes, and therefore the I/O bandwidth of current systems.

MPPs are partitioned statically; a partition is a number of compute nodes assigned to a computational task and shares with other partitions the set of I/O nodes. A parallel program runs in a partition of a size determined by the needs of that application.

To characterize the dynamic of paging of a program we define the *page fault profile* as the number of page faults as a function of time. In a uniprocessor system, the page fault profile of a process can be measured by having a cumulative counter of page faults and by sampling it periodically. In a parallel system, a program consists of a process group, a set of node programs, one process for each processing element. We are interested in the overall load placed on the shared resources (communication network and I/O nodes). To study this dynamics, we defines a *"cumulative paging profile"* of a parallel program by composing the individual profiles of individual node programs for relevant paging activities such as page faults, copy-on-write, page-ins, and page-outs.

2.1 A Model of Parallel Program Execution

In this section we are concerned with parallel programs for DMIMD systems. Such a parallel program consists of a set of tasks running concurrently, one task per node or

possibly multiple tasks per node for multiprocessing nodes. Each task can be either active or suspended. Each active task can be in one of three possible states:

(a) *Compute.* The task executes its own code and issues system calls other than those related to I/O and communication.

(b) *I/O.* The task has invoked an I/O system call and might be waiting for its completion.

(c) *Communication.* The task has invoked a communication system call and might be waiting for its completion.

The microscopic behavior of a parallel program consisting of N tasks, π_k, $k = 1, N$ each task going through a sequence of m_k states $S_{j,k}$ with $j = 1$, m_k, will be characterized by the average value $\lambda_{k,j}^{q_i}$ for each of the n parameters q_i with $i = 1, n$ relevant to the property of interest. For each parameter, for each task, and for each state we have a tuple ($t_{k,j-1}$, $t_{k,j}$, $\lambda_{k,j}^{q_i}$) with

$t_{k,j-1}$ – the time where task π_k performed its $(j-1)$ state transition, entering state $S_{j,k}$.

$t_{k,j}$ – the time where task π_k exited the state $S_{j,k}$.

$\lambda_{k,j}^{q_i}$ – the average rate of change of the global counter q_i given by

$$\lambda_{k,j}^{q_i} = \frac{q_i(t_{k,j}) - q_i(t_{k,j-1})}{t_{k,j} - t_{k,j-1}}$$

Several observations are in order. (a) The average rate of change of the parameter q_i is a good approximation of the temporal behavior of the task only if the lifetime of the corresponding state is short. (b) The model is amenable to performance monitoring. One could automatically detect the transition from one state to another, record the values of the parameters in the minimum set every time a state transition occurs, determine the lifetime of the state and compute the average rates. (c) This microscopic characterization is very costly in terms of the amount of information stored. (d) In some cases it will be difficult, if possible at all, to correlate events occurring different tasks. If different PEs have unsynchronized clocks, it is next to impossible to perform such a correlation.

The discrete time model introduced in this section can reduce significantly the amount of data necessary to characterize the behavior of a sequential or parallel program by filtering the raw data obtained through monitoring. For example, assume that parameter q_i has the behavior illustrated in Figure 1a. Figure 1b shows the discrete event representation of η_{q_i} supported by our model. If we accept that values of η_{q_i} lower than a given threshold, say

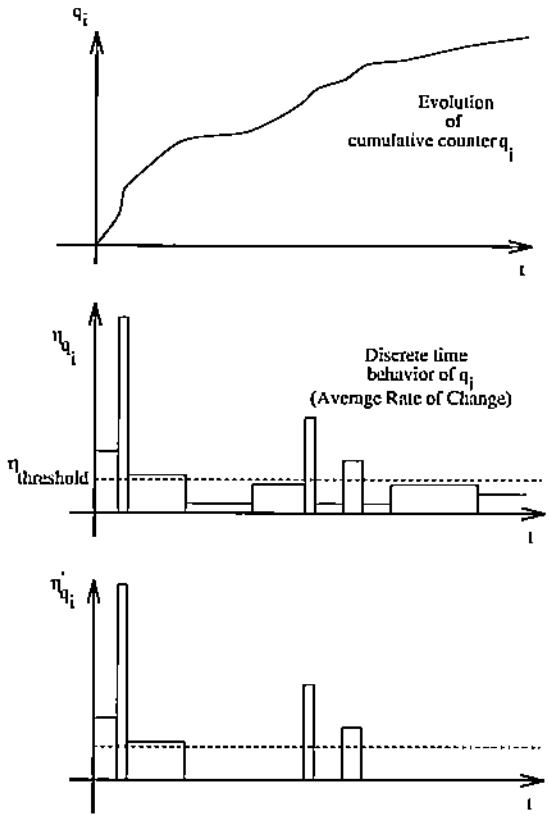


Figure 1: (a) The evolution of the cumulative counter q_i. (b) The discrete time behavior of q_i. The rate of q_i, η_{q_i} is plotted as function of time. $\eta_{threshold}$ indicates the threshold used to separate the peaks from the background. (c) The filtering out of the background and retention of the peaks of η_{q_i}.

$q_{threshold}$, can be neglected, then we can approximate η_{q_i} by a number of peaks rising above a background of level $q_{threshold}$. For example, when we want to correlate the paging activity of individual node programs, it seems reasonable to filter out the background and retain only the peaks of the page fault profile.

2.2 Modeling the Paging Behavior of SPMD Programs

A commonly used paradigm for solving problems that require a considerable amount of computing time using parallel systems is to partition the data in some manner and to run the same program in all the PEs assigned to the user, with each PE executing the

same program but with different data, therefore the name SPMD. This execution mode is compatible with distributed and shared memory MIMD architectures.

In the SPMD mode, the sequence of instructions executed by different PEs is different owing to data dependencies. Often, a special form of data dependency, the dependency of the identity of the PE, makes different PEs have a very different flow of control and allows the implementation of a worker-coordinator programming model. In this extended SPMD mode (ESPMD), the coordinator performs functions which are strictly sequential, such as reading the problem description, computing some initial values and distributing them to all the workers, and then at the end of the computation collecting statistical information from the workers. In the SPMD mode, one could reasonably expect similar, but not identical paging behavior for different PEs.

3 Parallel Program Profiling

Our methodology for monitoring the paging activity of a parallel program is based on detecting transitions from one state of the task to another and recording the paging activity data collected by the kernel at the time of the transition. A parallel program can be in one of the following states: *compute*, *I/O* and *communication*.

An event-driven *parallel profiling library* is built to monitor and profile the execution of the parallel programs on the Paragon^{TM} XP/S System running the OSF/1 Mach Operating System. Following the Mach terminology we call a node program a *task*. During the execution of each task snapshots of paging statistics at each state transition are collected. The set of parameters related to paging activities in the OSF/1 Mach kernel is found in the vm_statistic structure, collected at the time a state transition occurs.

Figure 2 summarizes the steps taken to monitor the parallel program execution. First the code is instrumented, then trace data is generated by executing the instrumented code and finally, the trace data is analyzed.

4 Data Analysis Methodology

Two types of analysis are performed on the trace data collected by the trace library: the skyline analysis and the cumulative analysis. The skyline analysis studies the amplitude and time correlation of the paging activities of the individual node programs, and the cumulative analysis aims to determine the total load of a parallel application due to paging activity on the communication and the I/O subsystem.

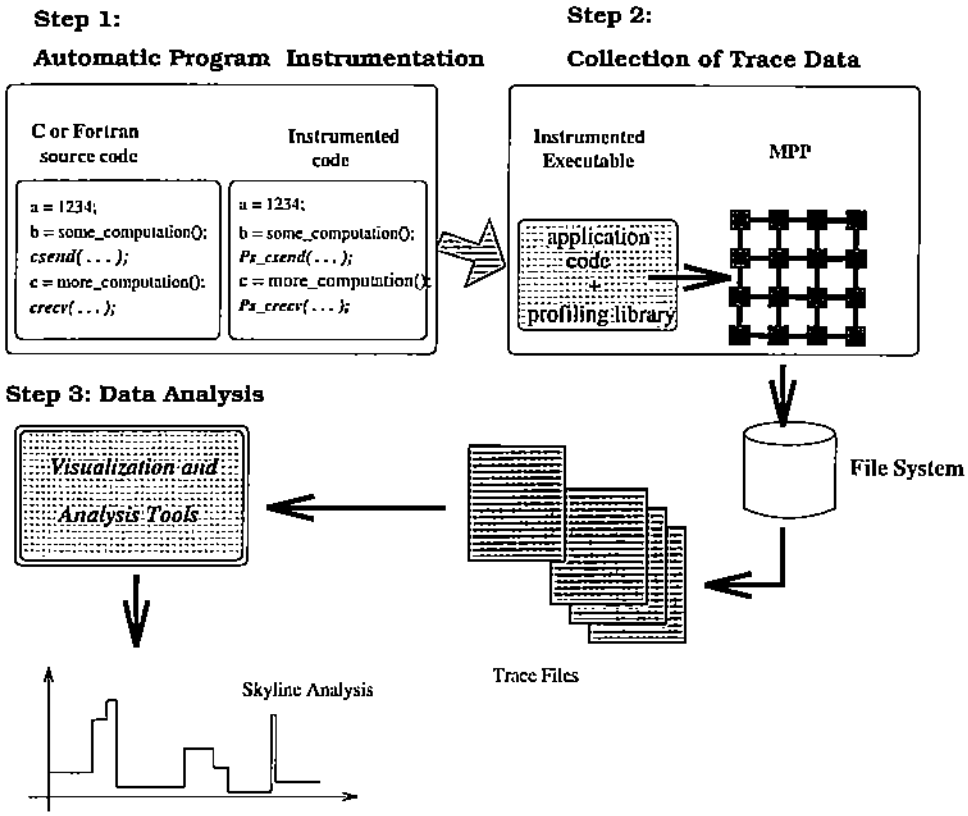


Figure 2: The Architecture of the Performance Monitoring Tool. The original source code is first instrumented (selected statements are substituted by calls to the profiling library), then a new executable is created (Step 1). The instrumented code is executed and trace data is collected in Step 2. And finally, the trace data is analyzed in Step 3.

The trace data collected during the profiling phase are first reduced to a simpler representation. The goals of data reduction are two. First, to expedite the identification of paging activity patterns of an application and to observe the system response under heavy paging work, and second, to reduce the amount of data needed to describe the program's paging behavior.

The data reduction is composed of the following three elements: *peak selection*, *background filtering*, and *skyline representation* for each individual trace data. The methodology for the reduction of trace data is described in [5] and [6]. The N highest peaks of activities are isolated and selected, and the rest are averaged as background activities. The graphical representation of peaks and background regions is called a *skyline*, representing the overall

paging behavior of a node program. The skyline representation is a more compact discrete time representation of an activity indicator.

4.1 The Amplitude and Time Correlation of the Paging Activities of the Individual Node Programs

The methodology used to study the correlation of the paging activities of individual PEs is to isolate the peaks of activity and to study how their amplitude at time of occurrence relate to each other.

We first look at the amplitude correlation among node programs in the three states: *compute*, *I/O*, and *communication*, and then the time correlation of the same data comparing the time of occurrence of each peak. The purpose is to see how similar or dissimilar the paging activities in different nodes are. This type of analysis is described in detail in [5] and [6].

4.2 The Cumulative Profile Analysis

The cumulative profile is performed for page-ins and page-outs indicators by merging and adding individual skylines into a single global load representation.

The cumulative profile is a graphical representation of the overall activity of a parallel program. The cumulative profile of an activity indicator, such as page-in and page-out rates, is derived from the skyline graphs of individual node programs of that indicator.

Not all paging statistics provided by the trace data imply in I/O activity. For instance, while a percentage of page faults converts into page-in operation, the remaining faults are satisfied by the virtual memory cache. The I/O load due to paging activity is reflected in page-ins and page-outs. A page-in occurs whenever a program generates a reference to an address in a page not in memory. Then the operating fetches the page from the secondary storage. A page-out occurs when the main memory frame holding a modified page in the inactive queue is needed by the virtual memory manager and the page contained in that frame is written to the external memory. The load imposed to a pager is the sum of all loads generated by the nodes that the pager serves.

After processing raw trace data and filtering out background, we obtain the skyline representation for the trace data of individual nodes. Since the skylines represent the profile of an activity *rate* and are synchronized along the time dimension (time stamped by a system-wide global clock), the cumulative profile of a certain activity indicator is built by summing up all the skylines of this indicator.

Because the individual skylines have the *"additive"* and *"commutative"* proprietics, a simple algorithm to produce the overall summation is adding skylines pairwise until there is only one resulting skyline. This algorithm is easily implemented in a parallel machine by distributing individual skylines to each node and then combining the partial sum of the skylines, resulting in one single overall cumulative profile. Figure 3 illustrates the cumulative profile of eight individual skylines using two processors.

5 The NAS Parallel Benchmarks

The Numerical Aerodynamic Simulation (NAS) Parallel Benchmarks [1] have been developed at NASA Ames Research Center for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five parallel kernel benchmarks and three simulated application benchmarks. The NAS application benchmarks mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

The five kernels are relatively compact problems, each emphasizing a particular type of numerical computation, and the three simulated CFD applications indicate the types of actual data movement and computation required in real CFD applications.

The following is a brief description of the NAS Parallel Benchmark programs.

5.1 Kernel Benchmarks

There are five kernel benchmarks which are intended to test specific features of parallel machines. Each benchmark has two problem sizes, classified as Class A and Class B, differing mainly in the size of principal arrays. Table 1 shows the problem sizes of NAS Parallel Benchmark programs.

EP Embarrassingly Parallel Benchmark

This benchmark provides an estimate of upper achievable limits for floating point performance on a particular system.

Two dimensional statistics are accumulated from a large number of Gaussian pseudorandom numbers, generated according to a particular scheme described in [1], and the number of pairs are tabulated in successive square annuli. This application does not require significant interprocessor communication, since the communication only occurs in the combination of sums from various processors at the end.

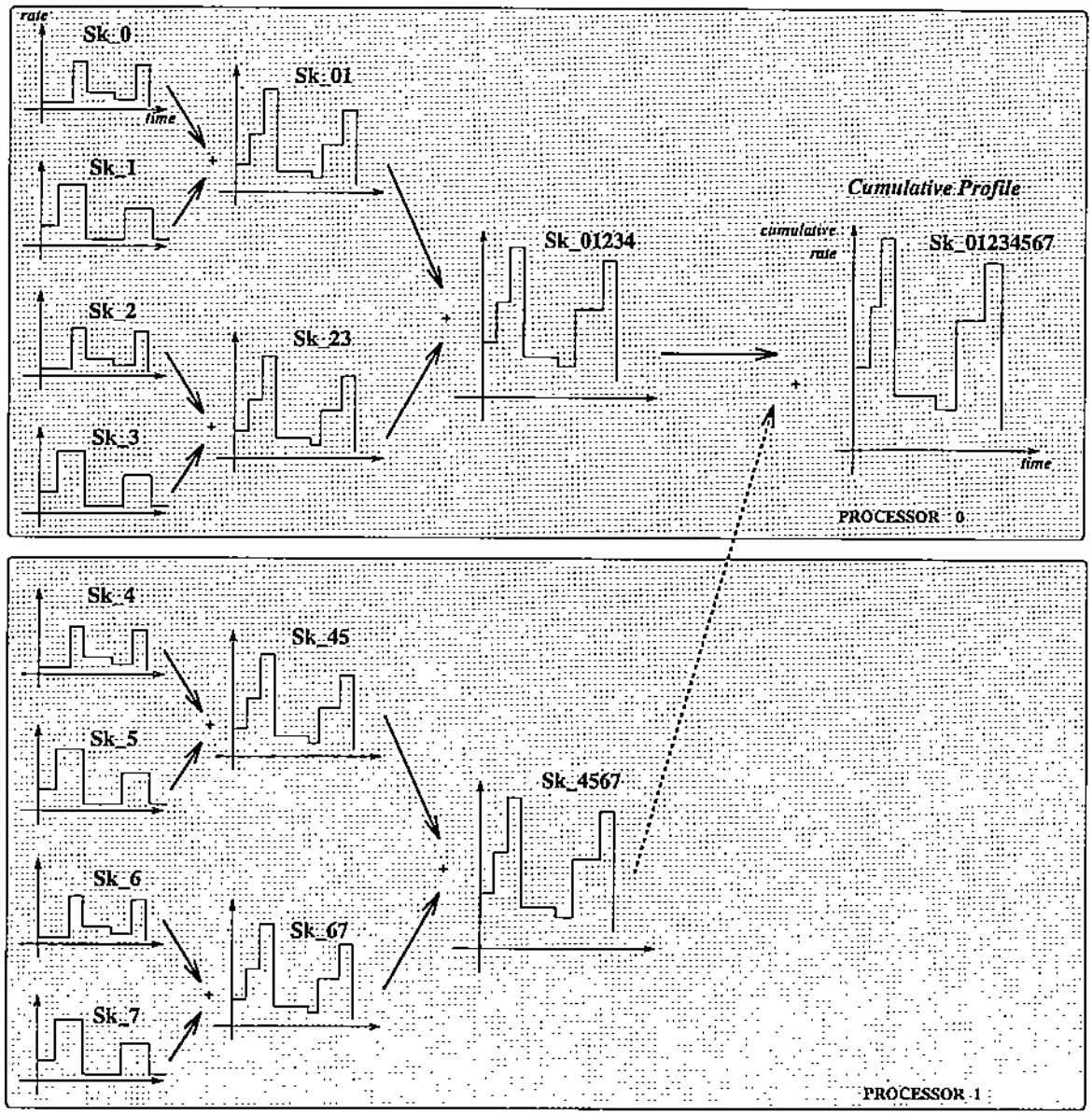


Figure 3: The parallel procedure to produce the cumulative profile of a paging activity indicator. Eight skylines of individual node programs are reduced to one *cumulative profile* using two processors.

Benchmark Name	Abbreviation	Nominal Size	
		Class A	Class B
Kernel Benchmarks			
Embarrassingly Parallel	EP	2^{28}	2^{30}
Multigrid	MG	256^3	256^3
Conjugate Gradient	CG	14,000	75,000
3-D FFT PDE	FT	128×256^2	512×256^2
Integer Sort	IS	$2^{23} \times 2^{19}$	$2^{25} \times 2^{21}$
Simulated CFD Application			
Lower-Upper diagonal benchmanrk	LU	64^3	102^3
Scalar Pentadiagonal benchmark	SP	64^3	102^3
Block Tridiagonal benchmark	BT	64^3	102^3

Table 1: The NAS Parallel Benchmark. The Class A and Class B problems are classified by the size of the principal arrays that the programs manipulate.

MG Multigrid Benchmark

This is a simplified (using constant coefficients) multi-grid kernel which solves a 3-D Poisson partial differential equations (PDE.)

Four iterations of the V_cycle multi-grid algorithm are used to obtain an approximate solution to the discrete Poisson problem

$$\nabla^2 u = v$$

on a $256 \times 256 \times 256$ grid with periodic boundary condition.

The code tests both short and long distance structured communication.

CG Conjugate Gradient Benchmark

This benchmark solves an unstructured sparse linear system by the conjugate gradient method and tests irregular long distance communication, employing unstructured matrix vector multiplication. The conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix.

FT 3-D FFT PDE Benchmark

This benchmark solves numerically a 3-D partial differential equation using forward and inverse FFTs.

The 3-D FFT steps require considerable communication operations as array transposition. It is a good test of long distance communication performance.

IS Integer Sort Benchmark

This kernel performs a large integer sort operation important for "particle method" codes. The "particle in cell" are applications of physics in which particles are assigned to cells and may drift out. The sorting operation reassigns particles to the appropriate cells.

This benchmark sorts N keys ($N = 2^{19}$ for Class A and $N = 2^{21}$ for Class B application) in parallel and tests both integer computation speed and communication performance.

5.2 Simulated CFD Application Benchmarks

Computational fluid dynamics (CFD) application involves numerical solution of a system of nonlinear partial differential equations in two or three spatial dimensions, with or without time dependency.

The Navier-Stokes equations represent the laws of conservation of mass, momentum and energy applied to fluid medium in motion. These equations, when supplemented by appropriate boundary and initial conditions, describe a particular physical problem.

Many algorithms of CFD applications present global data dependencies and are unable to extract performance out of current parallel supercomputers. Often, the data motion among processing elements is the bottleneck due to high latencies and inadequate bandwidth.

The NAS application benchmarks are **synthetic CFD application programs**, which lack the complexity of a real application, but retain all the essential computational structures. They are designed for the numerical solution of a synthetic system of nonlinear PDEs, using iterative techniques similar to those found in CFD applications without any of the pre- and post-processing required by the full CFD applications, or the interactions of the processors and the I/O subsystem.

The following is a brief description of the synthetic CFD application programs used to benchmark highly parallel systems.

LU The lower-upper diagonal benchmark

This benchmark performs a symmetric successive over-relaxation (SSOR) numerical scheme to solve a regular-sparse, block (5×5) lower and upper triangular system.

This problem exhibits limited amount of parallelism compared to the next two.

SP The scalar pentadiagonal benchmark

Solution of multiple independent systems of non-diagonally dominant, scalar pentadiagonal equations.

BT The block tridiagonal benchmark

In this benchmark, multiple independent systems of non-diagonally dominant, block tridiagonal equations with a 5×5 block size are solved.

SP and BT are similar in many respects and both involve global data dependencies. The fundamental difference between them is the communication-to-computation ratio, as BT has much higher communication-to-computation ratio than SP.

6 Case Studies

6.1 The Environment

The measurements of NAS Parallel Benchmark programs were performed on the Purdue Paragon, a Paragon XP/S model 10 with 140 compute nodes, one boot/service node, four service nodes, thirteen I/O nodes, three HiPPI/network nodes, and one Ethernet node. Compute nodes, service nodes, two of the three HiPPI nodes, and the boot node have 32 Mbytes of DRAM each. The thirteen I/O nodes, one HiPPI nodes, and the Ethernet node have 16 Mbytes each.

The Paragon operating system is a distributed version of the Open System Foundation's OSF/1 Unix system. The operating system is fully and transparently distributed across the system's nodes. A Mach 3.0-based microkernel resides on each node.

6.2 The Trace System

Our monitoring system is based on software probes inserted into a parallel program. During the execution of the instrumented program, trace data associated with every event is collected, stored in some internal buffers, and written to an external storage device in a trace file when these buffers are filled. The level of intrusion of this monitoring system depends on the overhead of gathering the data associated with each event and the amount of data collected. An intrusive system [4] will increase drastically the execution time of the parallel program and will alter the behavior of the parallel programs we want to observe.

In our system, as in many other cases, a significant level of intrusion is caused by the act of storing the trace buffers on some external device, mainly because of the limited I/O bandwidth of parallel systems, the high latency of I/O operations, and the contention for I/O devices and the interconnects. Several choices to reduce the level of the intrusion are discussed in [7]. In table 2 we present a summary of the most important versions of our profiling library.

The measurements reported in this paper are obtained using the least intrusive version of the profiling library, the MP/MP profile library – "Profiling using multiple processes per node that communicate through message passing mechanism" [7] –. In this library, there exist two processes on each compute node: one for executing the parallel application code (the compute process) and the other (the trace-writer process) for sending the performance data to the file system. The trace-writer process is responsible for all I/O operations caused by the monitoring (e.g. open the trace files, write the trace data and close the trace files). The communication between a compute and the corresponding trace-writer is done through IPC message passing mechanism in the same compute node. The trace-writer allows the compute process to proceed without being blocked by I/O write operations generated when writing the trace data.

The amount of data generated by the monitoring system depends on the characteristics of the parallel application under monitoring. Some applications, such as the EP benchmark, present few state transitions, thus a small number of trace data are collected. Other applications, such as the CG and BT benchmark programs, generate a few Mbytes of trace data for each node program quickly consuming the file system's free space.

For applications with low rate of state transition, the solution is to provide the uniform sampling mechanism in which trace data are collected at regular intervals in addition to the collection at the state transitions (MP/MP_sel.)

While the solution for the small amount of trace data is easily solved by the uniform sampling profiling library, the solution for the huge amount of trace data requires the user's intervention. The profiling library gives the user the ability of defining the number of nodes to be profiled (ranging from none to all nodes) and the period during the execution in which trace data are collected. This provides the user a certain extent of control over the amount of trace data generated during the program execution. The drawback with this additional control is the possible load unbalance due to data collection in some of nodes while others remain free of this task.

A summary of the execution time of NAS benchmark programs with and without monitoring is presented in Table 3. The programs were profiled with MP/MP_sel profiling library. From Table 3 we see that for applications with very high event rate, such as SP and BT,

Version	Name	Description
V1	NOPT	A trace record is generated at each state transition. Trace data are written to the Unix File System.
V2	UNIFORM	In addition to V1, uniformly sampling in the compute state, i.e. trace data are also collected in constant intervals during the compute state.
	PFS	Same as V1 except that trace data are written to PFS (the Intel Parallel File System, which strips file data across multiple I/O nodes) using buffering and asynchronous I/O.
V3		The trace data recording is done by a separate process.
V3/HN	HN	Host/node paradigm. The trace data collected by all nodes are sent to a *host* process in a service node.
V3/MP_MP	MP/MP	Multiple processes per node using message passing communication. Two processes at each compute node: the trace-writer and the compute process.
V3/MP_shm	MP/SM	Same as the MP/MP approach except the communication between trace-writer and compute processes is done through a shared memory area, instead of passing messages.
V3/MP_select	MP/MP_sel	Some selection options are provided in addition to the MP/MP approach, such as the number of nodes to be monitored, the monitoring time interval and activate or deactivate the uniform sampling mechanism.
V4	MPI/NX	Use of multiple partitions (Prototype). The trace data are recorded in the Unix File System by processes in another partition. This approach uses the MPI and the Nx communication library to provide inter-partition communication.

Table 2: The versions of the Parallel Profiling Library. There are four main versions. *V1* is the simplest library, without performance optimization. *V2* is a super-set of V1 containing the uniform sampling and the PFS versions, in addition to V1. *V3* is the multiple processes approach in which trace data are recorded by a separte process. *V4* is the multiple partition approach working with a *trace-process* partition and a *parallel-program* partition.

the intrusion of the monitoring system in the execution of the program increases with the number of nodes being monitored. The larger is the difference between the execution time with and without instrumentation, the more intrusive the monitoring system is. For programs with large event rates, we are forced to monitor the execution of only a subset of all the nodes in order to limit the effects of intrusion. For instance, in tracing the SP kernel, which generates about 1 Mbyte of trace data per node, the increase in execution time when the measurements were performed in the first four nodes out of 64 is substantial, about 33% (4_64 run) and in the first sixteen nodes out of 64 (16_64 run) is much higher, about 284%. On the other hand, applications with low event rate (e.g. FT) seem not prone to the effects of the monitoring intrusion.

6.3 The Amplitude and Time Correlation of the Paging Activities of the Individual Node Programs

All our measurements for different NAS Benchmark programs running on different numbers of PEs show that there is a substantial dissimilarity among node programs, even for SPMD programs. This confirms the conclusions we presented previously in [5]. The correlation in time of different peaks of activity and the correlation of the amplitude show great discrepancies for all the activity indicators. Figure 4 shows the page fault analysis for a subset of 16 nodes of the MG (Multigrid) program when running on 32 nodes.

6.4 The Cumulative Profile Analysis

In general, the NAS benchmark programs do not present I/O operations to disk. Most of them switch states between *Compute* and *Communication* state.

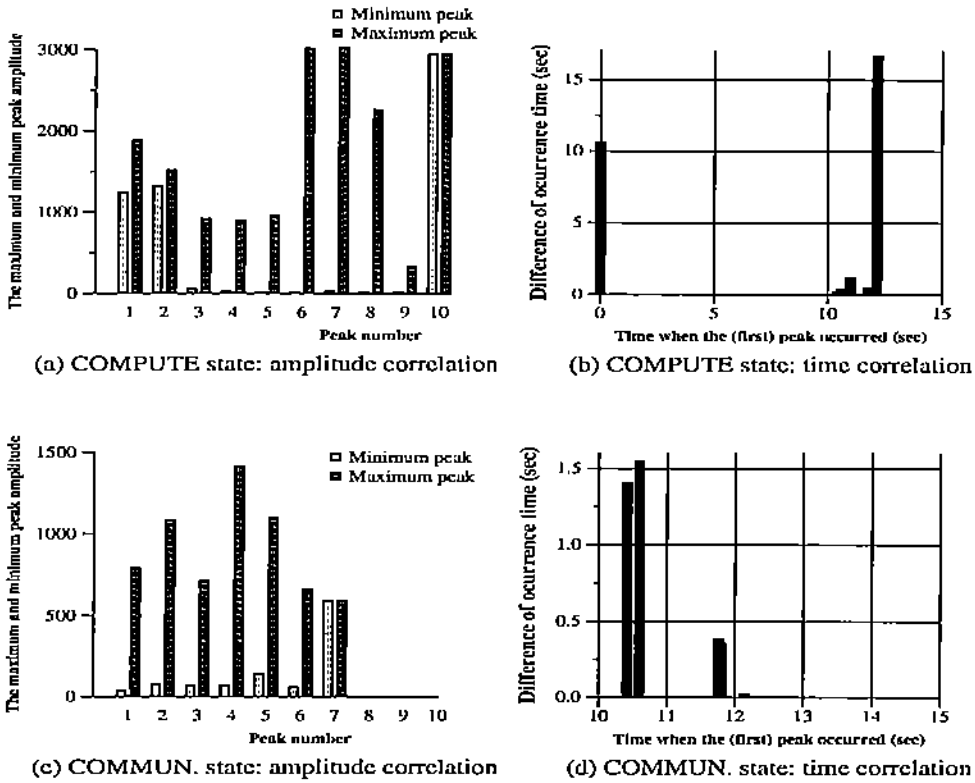
Figures 7 to 15 show significant graphs of the cumulative profile of selected NAS benchmark programs. The total execution time represented in the cumulative profile graph is not necessarily the same reported by the NAS benchmark programs since the measurement reported by the programs may correspond only to the execution of a portion of the program.

EP – Embarrassingly Parallel

The uncommon characteristic of EP benchmark is its lack of communication among compute nodes during the execution. Because of this, there are few state transitions, thus few trace records are generated. Even using the uniform sampling library, (MP/MP_sel,) almost no changes of paging activity indicators are reported by the skyline and cumulative profile analysis; this application consists mainly of in-core computation.

Benchmark	Number of Nodes	Execution Time (in seconds) without profiling	Execution Time (in seconds) with profiling	Trace File Size per Node (bytes)
EP	32	21.16	21.29	1.84 K
	Class A 64	10.45	12.16	1.70 K
	128	5.24	6.67	0.70 K
	Class B 64	42.40	44.82	33.08 K
	128	21.42	24.41	1.82 K
MG	32	16.35	(4_32) 17.99	~ 0.5 M
			(16_32) 17.82	
	64	8.39	(16_64) 9.89	~ 0.5 M
CG	32	10.93	(4_32) 25.47	> 3.0 M
	64	7.45	(4_64) 11.40	~ 3.7 M
FT	32	18.46	(16_32) 18.97	~ 180 K
			(32_32) 18.91	
	64	9.65	(16_64) 10.11	~ 360 K
			(64_64) 10.14	
IS	32	8.17	9.19	~ 137 K
	64	4.88	5.41	~ 267 K
	128	2.88	3.72	~ 529 K
LU	64	241.67	(4_64) 277.61	~ 1.8 M
SP	64	285.46	(4_64) 381.74	~ 1.0 M
			(16_64) 811.00	
BT	64	247.25	(4_64) 260.47	> 8.0 M

Table 3: The execution time of NAS Benchmark programs. The NAS programs were profiled with MP/MP_sel profiling library. The notation (n_m) indicates that the application was executed on m nodes but only n $(n \leq m)$ nodes were profiled. When $n \neq m$, the first n nodes carry out measurements and the other $(m - n)$ nodes do not collect trace information.



(a) COMPUTE state: amplitude correlation

(b) COMPUTE state: time correlation

(c) COMMUN. state: amplitude correlation

(d) COMMUN. state: time correlation

Figure 4: The page fault analysis. The correlation of amplitude and time of occurrehce for the peaks of page fault activity of the MG Multigrid program of a subset of 16 nodes of running on 32 nodes.

Figures 5 and 6 illustrate the evolution of page faults of a Class B EP program running on 64 and 128 nodes. Note that both graphs have similar behavior and both present a constant cumulative page fault rate (about 105.5 faults per second for 64 nodes and about 214.5 faults per second) for most of the execution period.

MG – Multigrid

As the size of trace file in each node remains the same when running on 32 or 64 nodes ($\sim$ 0.5 Mbytes per node), Table 3 shows the correlation between the intrusion of the trace system and the number of nodes used for computing the program. While the increase in the execution time for 32 nodes is about 8 $\sim$ 9 %, the increase for 64 nodes is about 17.89%. This can be explained by the higher contention, for 64 nodes, for shared resources such as

the I/O and the communication subsystems.

The cumulative analysis shows that the PEs are not exactly synchronized, although the skyline analysis shows that their activity indicators have similar shapes.

Figures 7 and 8 illustrate the evolution of page faults of the MG program running on 32 and 64 nodes, respectively. They are cumulative skylines of a subset of 16 nodes when running on 32 and 64 nodes.

CG – Conjugate Gradient

The huge number of trace records generated by the CG Benchmark causes the increase of execution time of instrumented program, even when only few nodes are being monitored. The load unbalance caused by the collection scheme in addition to the global synchronizations of the application during the execution have contributed for the increase of the execution time.

From the cumulative profile of faults in communication state (Figure 9), we can clearly see the points of global synchronization and the period of intense message exchange among nodes.

FT – 3-D FFT PDE

In this application most of the paging activities happened in the beginning of the execution.

This application shows that selecting some nodes (not all) for monitoring is a valid approach since the cumulative profile analysis of a 16_64 run (collecting 16 nodes out of 64 nodes), Figure 10, and the cumulative profile of a 64_64 run, Figure 11, have similar shapes, where the profile of a 16_64 run scales down from the cumulative profile of a 64_64 run.

IS – Integer Sort

Because of the communication pattern (an n-to-n communication), the number of message generated during the program execution increases with the number of nodes and likewise the size of trace files. IS is the program used for the intrusion analysis and the trace system optimization reported in [7].

There are few bursts of activity in the compute state all in the beginning of the execution (Figure 12), in contrast with the periodic paging activities in the communication state (Figure 13) which are spread all over the execution time.

LU – Lower-Upper Diagonal

The periodicity of paging activity is noticeable in all activity indicators. For page fault indicator, except the first peak of activity in the beginning of the execution, all the peaks are in the same range of amplitude. Other indicators, such as page-ins (Figure 14) and page-outs, are mostly regular in the peak amplitude.

SP – Scalar Pentadiagonal

Although many trace records are generated, the paging activity happened only in the beginning of the execution (the first 1% of the execution time).

BT – Block Tridiagonal

This application generates the biggest trace files among all NAS benchmark programs because of its high communication-to-computation rate. From the analysis of cumulative profile of a 4_64 run (Figure 15), we can see that the paging activity is "regularly" spread over the execution time (faults and page-ins in compute state), but it is risky to conclude that the same result would be obtained from a 64_64 run.

7 Conclusions

The first step to understand the paging and the I/O requirement of the parallel application is a detailed characterization of the paging behavior of both parallel application code and existing demand paging systems, such as the Virtual Memory system on the Intel Paragon XP/S.

By studying the paging requirements of parallel application, we determine the overall performance requisite of I/O subsystem caused by the demand paging, the acceptable system configuration, the data storage volume and data transmission rate. Similarly, estimating the performance limitations of current virtual memory system and its response to the application requests provide a base for the performance optimization of both parallel application and existing systems.

In this work, we have selected the NAS Parallel Benchmark programs to assess the paging requirement of a class of representative parallel applications. The NAS benchmark programs are unique because of their lack of I/O operations and the mainly in-core computation. Although the benchmarks differ from each other, the general conclusion from the

skyline analysis and the cumulative profile analysis confirms what we have observed in a previous work [5], studying another class of applications, the structural biology applications.

We noted in both works, that there is a substantial dissimilarity of paging activity among the node program even for SPMD programs. We also observed that, because of the poor performance of current virtual memory systems in massively parallel systems, the parallel program designers exchange the convenience of virtual memory for better performance by implementing their own data management, as described in [3]. To understand demand paging in MPPs better, we still need further evaluation of the same parallel systems running applications with real use of virtual memory system.

The two trace data analysis methodologies, the *skyline analysis* and the *cumulative analysis*, provide both quantitative and qualitative measures of similarity of the paging activity among individual node programs. The *skyline analysis* shows quantitatively the difference between the peak amplitudes and the time of occurrence of peaks among individual node programs. The *cumulative analysis* shows, in addition to the total paging load, the overall pattern of paging activity of a parallel program. We can see, for example from Figure 14, that there are some points of global synchronization and the peaks of activity happened at similar but not exactly the same times.

Currently, we are in the process of enhancing the graphical performance monitoring and analysis tool. The tool will support automatic instrumentation of parallel programs using a variety of communication and I/O primitives. Analysis tools will allow to correlate paging and I/O activities of individual node programs and to study the overall load placed on the I/O and paging systems.

The instrumentation and characterization of paging activities of parallel programs allow the creation of models for realistic evaluation of alternatives of scheduling mechanism and design of I/O subsystems. Our next step is to study the interaction between demand paging and gang scheduling, and provide alternatives for hiding the paging latency.

Acknowledgements

The authors want to express their gratitude to Intel Scalable Systems Division, in particular Dr. Jerry Baugh, for providing us with a version of the NAS Parallel Benchmark programs running on the Paragon.

References

[1] D.Bailey, E.Barszcz, J.Barton, D.Browning, R.Carter, L.Dagum, R.Fatoohi, S.Fineberg, P.Frederickson, T.Lasinski, R.Schreiber, H.Simon, V.Venkatakrishnan and S.Weeratunga. *The NAS Parallel Benchmarks*, RNR Technical Report RNT-94-007, March 1994

[2] D.Bailey, E.Barszcz, L.Dagum, and H.Simmon. *NAS Parallel Benchmark Results 10-94*, NAS Technical Report NAS-94-001, October 1994

[3] M.A. Cornea-Hasegan, D.C. Marinescu, and Z. Zhang, "Data Management for a Class of Iterative Computations on Distributed Memory MIMD Systems," *Concurrency: Practice and Experience*, vol 6(3), pp. 205–229, 1994.

[4] S.R.Sarukkai and A.D.Malony. Perturbation Analysis of High Level Instrumentation for SPMD Programs. *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993, pp. 44–53.

[5] K.Y.Wang and D.C.Marinescu. Correlation of the paging activity of individual node programs in the SPMD execution mode. *Proceedings of HICSS'28, IEEE Press*, **vol 1**, 1995, pp. 61-71

[6] K.Y.Wang and D.C.Marinescu. *"An analysis of the paging activity of parallel programs. Part I: Correlation of the paging activity of individual node programs in the SPMD execution mode."* Technical Report CSD-TR-94-042, Purdue University, Department of Computer Sciences, June 1994.

[7] K.Y.Wang. *"A study of intrusion of a software trace system in the execution of parallel programs"* Technical Report CSD-TR-94-078, Purdue University, Department of Computer Sciences, Nov. 1994.

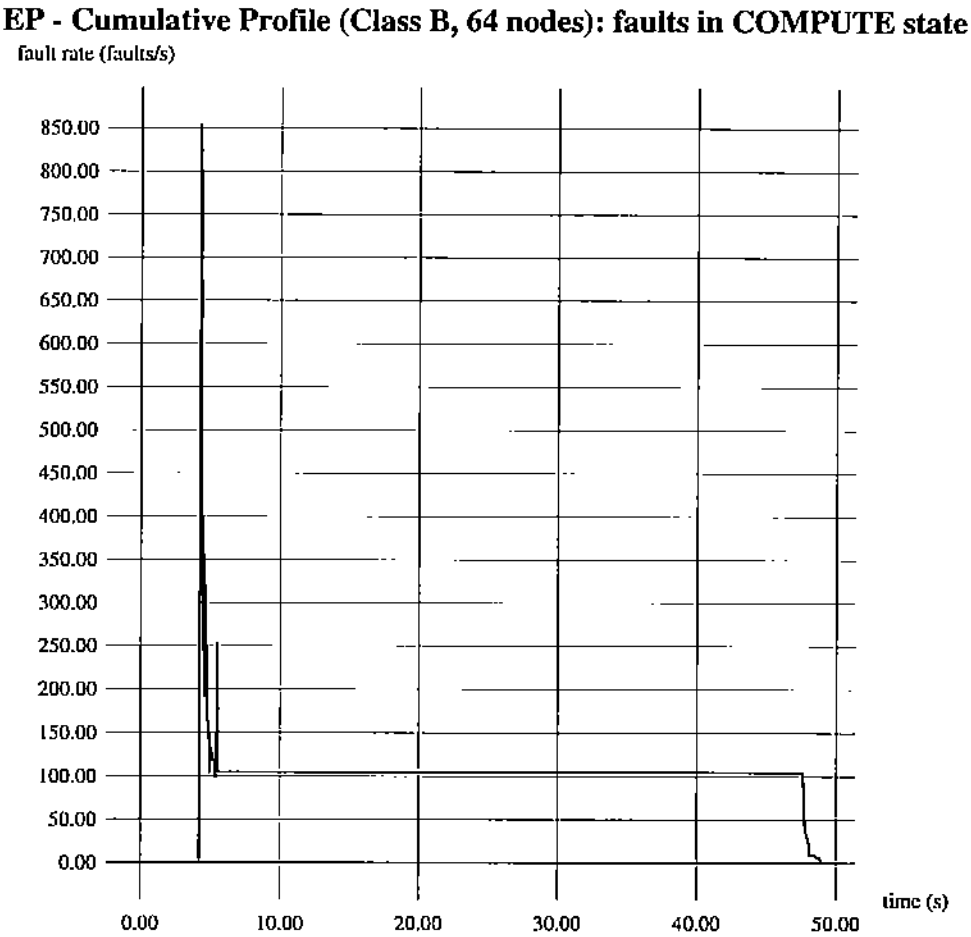


Figure 5: The cumulative fault rate in the compute state for 64 nodes of a class B EP Embarrassingly Parallel benchmark. The total execution time was 48.90 seconds and the time corresponding to the NAS benchmark measurement was 44.82 seconds.

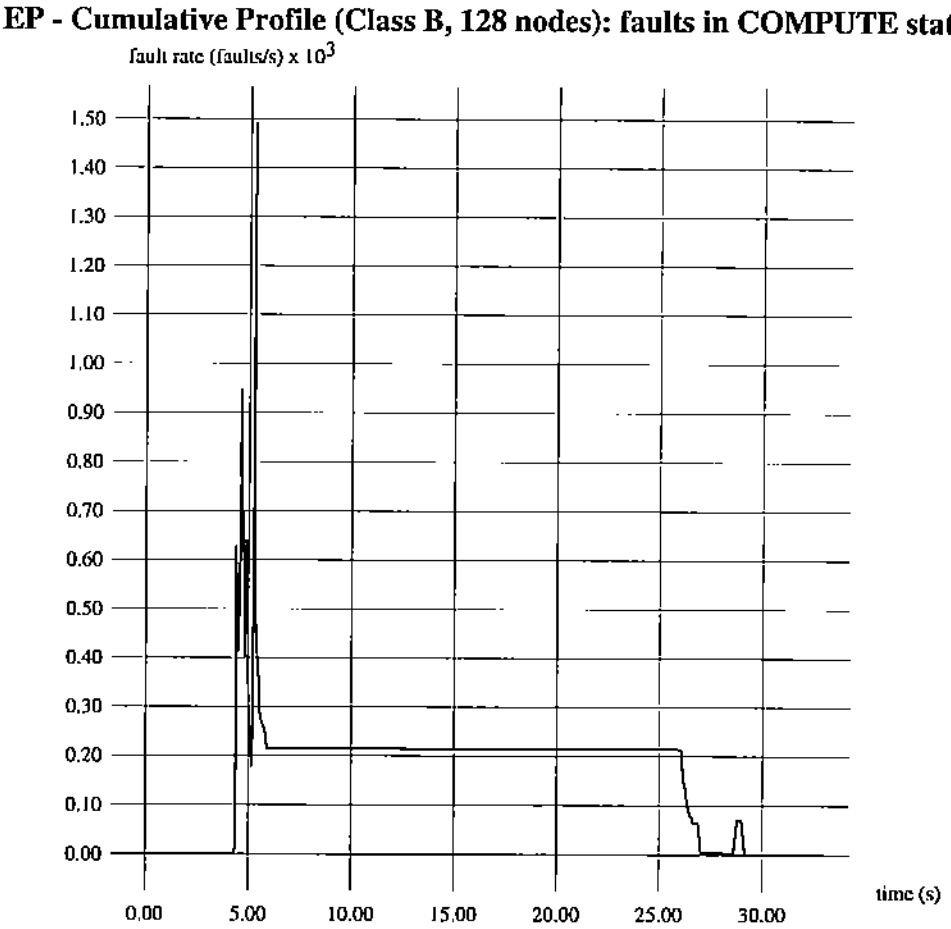


Figure 6: The cumulative fault rate in the compute state for 128 nodes of a class B EP Embarrassingly Parallel benchmark. The total execution time was 32.55 seconds and the portion corresponding to the NAS benchmark measurement was 24.41 seconds.

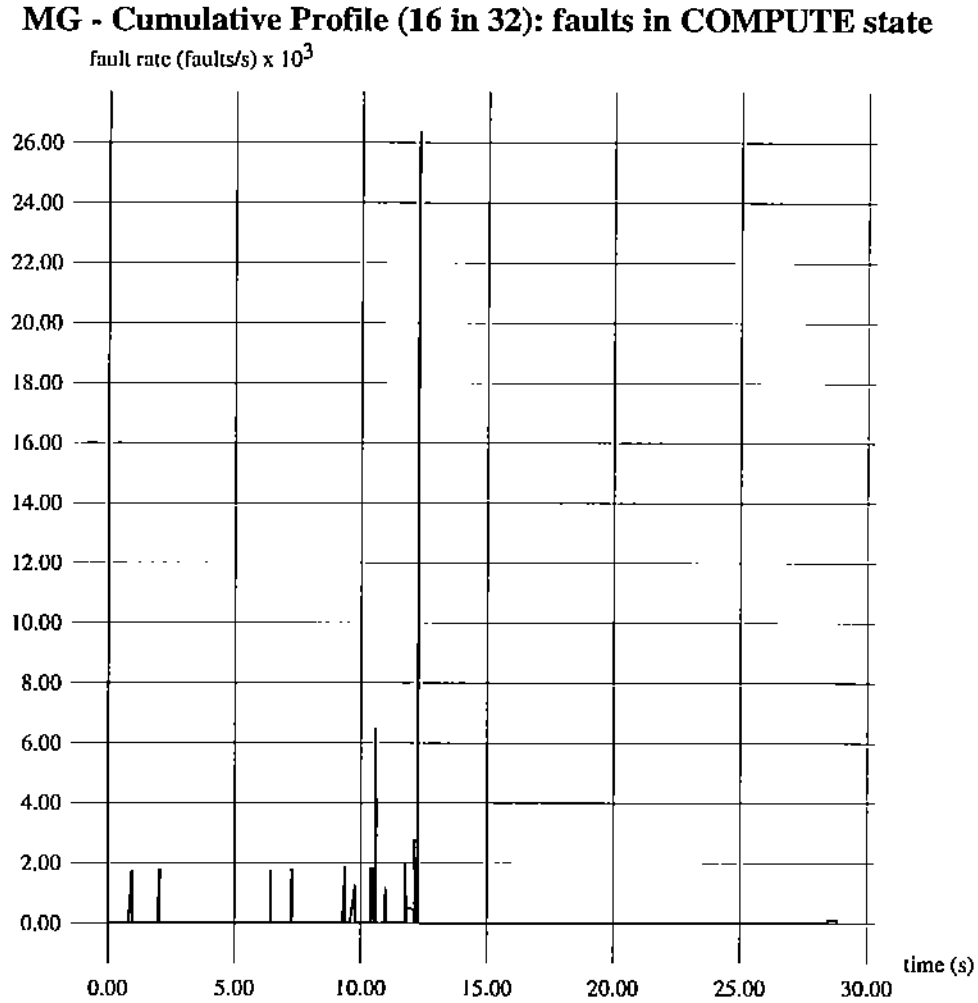


Figure 7: The cumulative fault rate in the compute state for 16 nodes of a class A MG Multigrid benchmark running on 32 nodes. The total execution time was 28.84 seconds and the portion corresponding to the NAS benchmark measurement was 17.82 seconds.

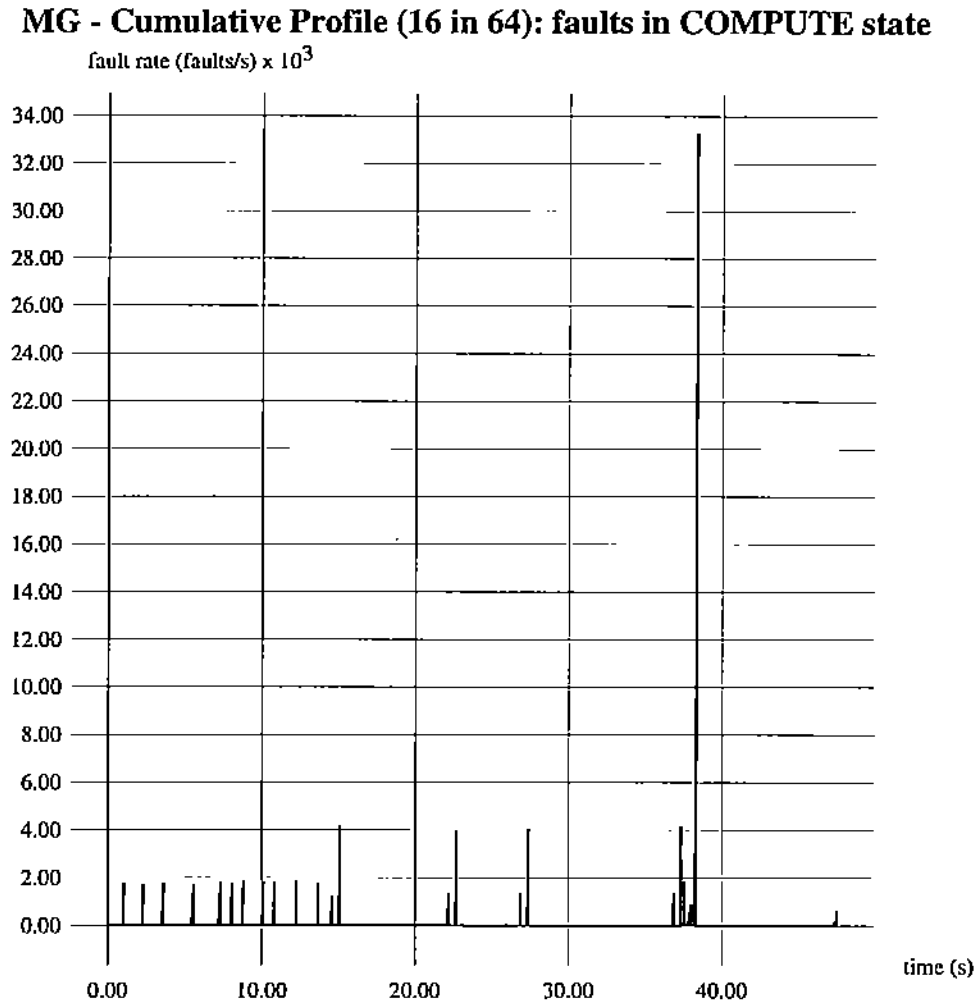


Figure 8: The cumulative fault rate in the compute state for 16 nodes of a class A MG Multigrid benchmark running on 64 nodes. The total execution time was 47.46 seconds and the time reported by the NAS benchmark was 9.89 seconds.

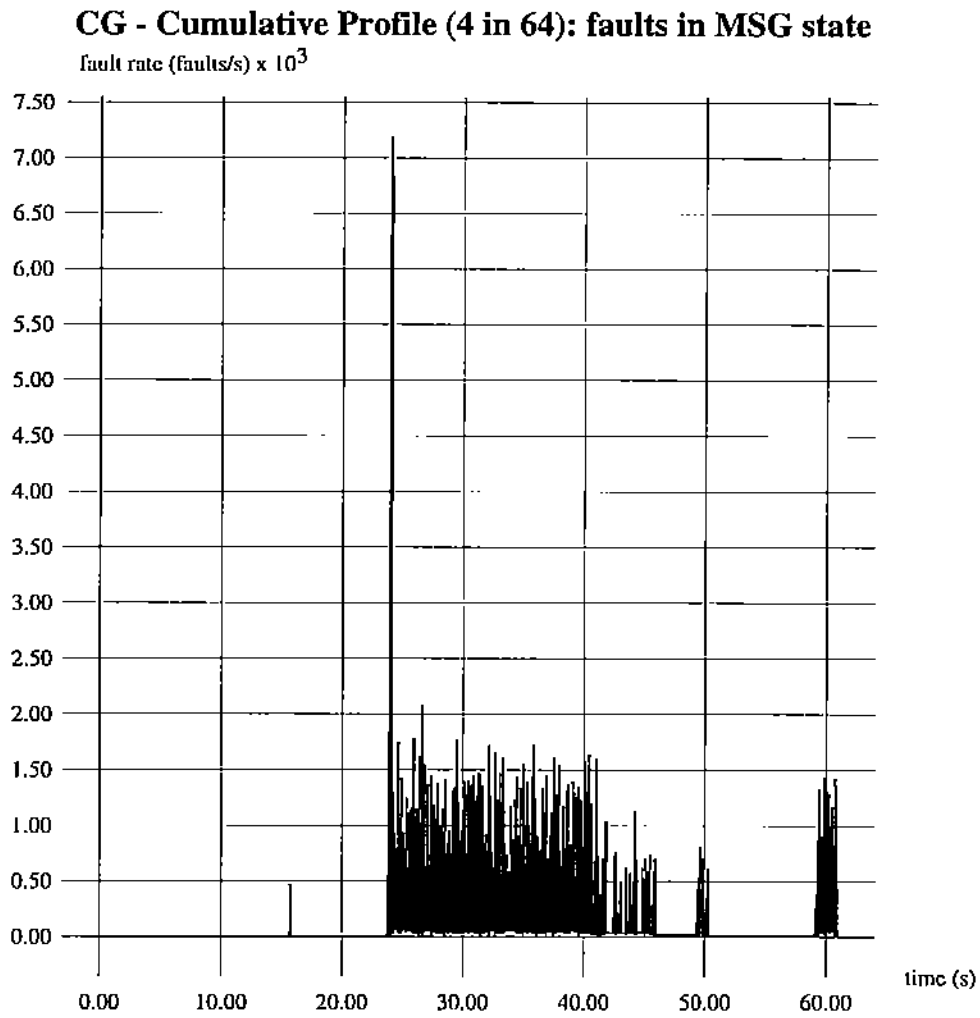


Figure 9: The cumulative fault rate in the communication state for 4 nodes of a class A CG Conjugate Gradient benchmark running on 64 nodes. the The total execution time was 60.92 seconds and the time reported by NAS benchmark was 11.40 seconds.

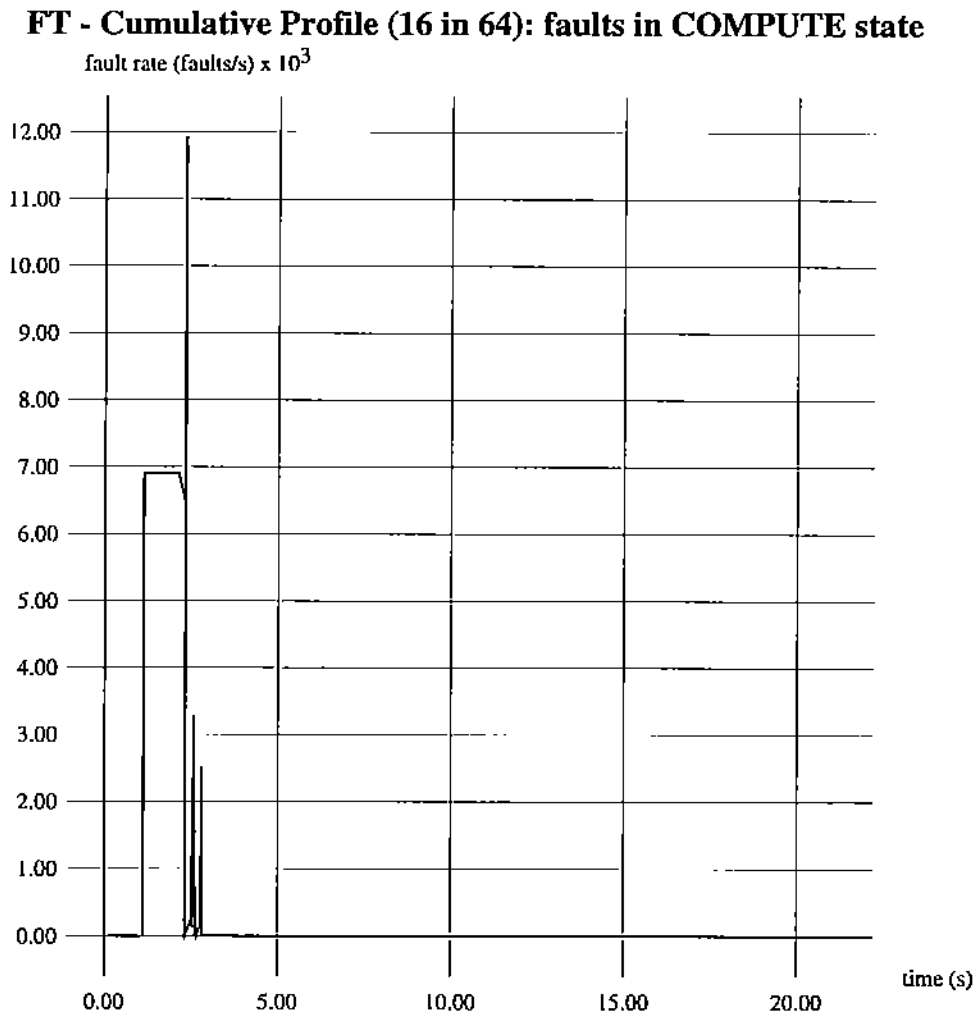


Figure 10: The cumulative fault rate in the compute state for 16 nodes of a class A FT 3-D FFT PDE benchmark running on 64 nodes. The total execution time was 21.13 seconds and the time reported by the NAS benchmark was 10.11 seconds.

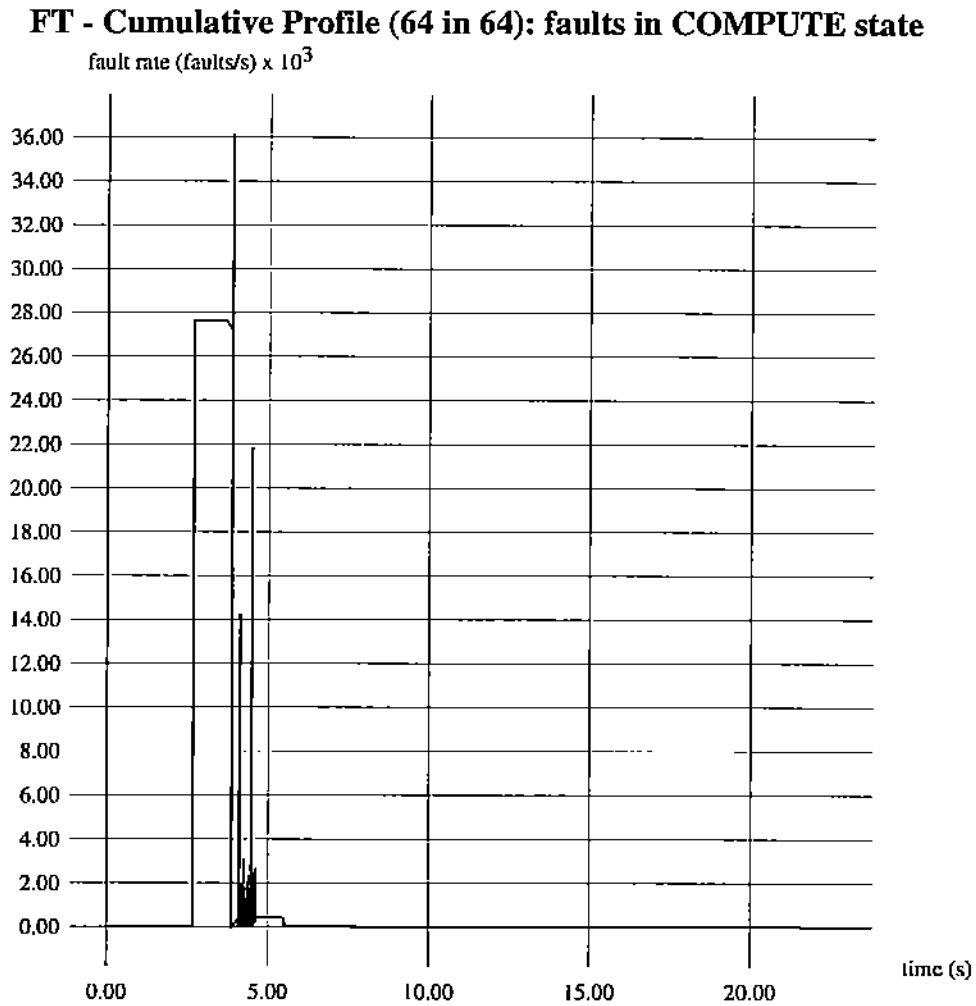


Figure 11: The cumulative fault rate in the compute state for 64 nodes of a class A FT 3-D FFT PDE benchmark running on 64 nodes. The total execution time was 22.64 seconds and the time reported by the NAS benchmark was 10.14 seconds. Note that this graph is similar to the previous graph, the cumulative fault rate for 16 nodes out of 64 nodes, differing in the fault rate scale.

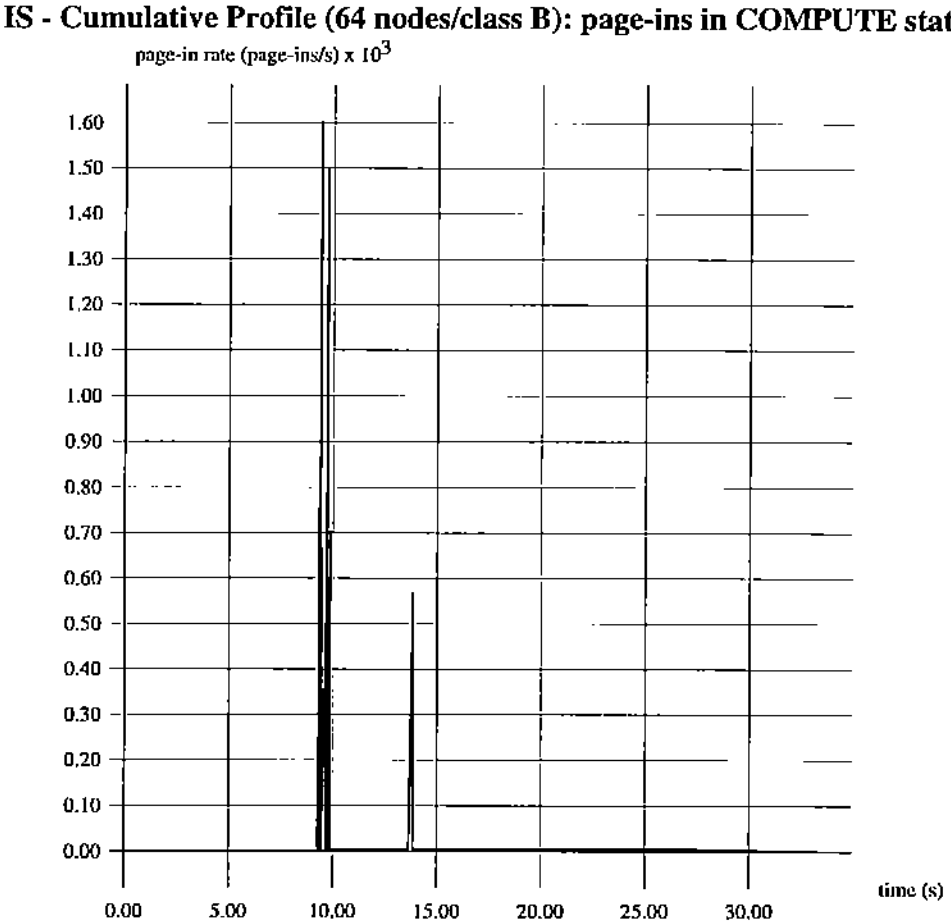


Figure 12: The cumulative page-in rate in the compute state of a class B IS Integer Sort benchmark running on 64 nodes. The total execution time was 33.26 seconds and the time reported by the NAS benchmark was 19.53 seconds.

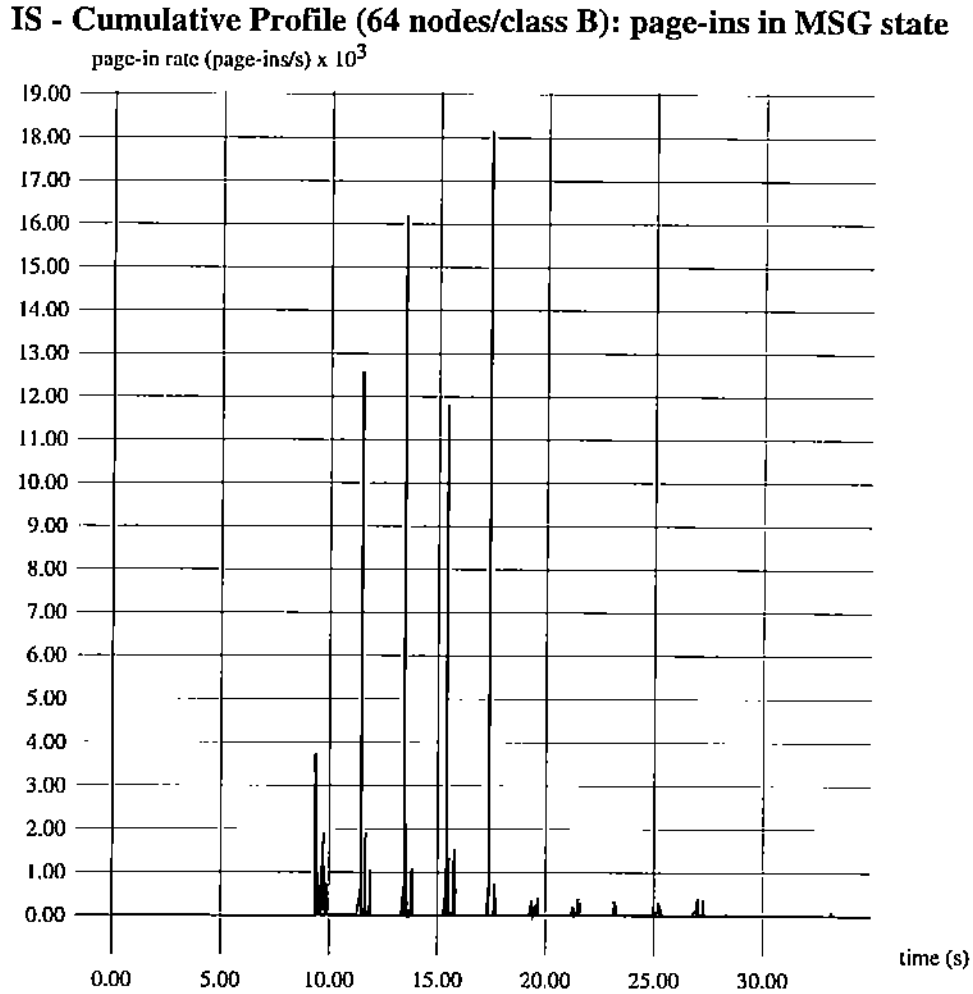


Figure 13: The cumulative page-in rate in the communication state of a class B IS Integer Sort benchmark running on 64 nodes. The total execution time was 33.25 seconds and the time reported by the NAS benchmark was 19.53 seconds.

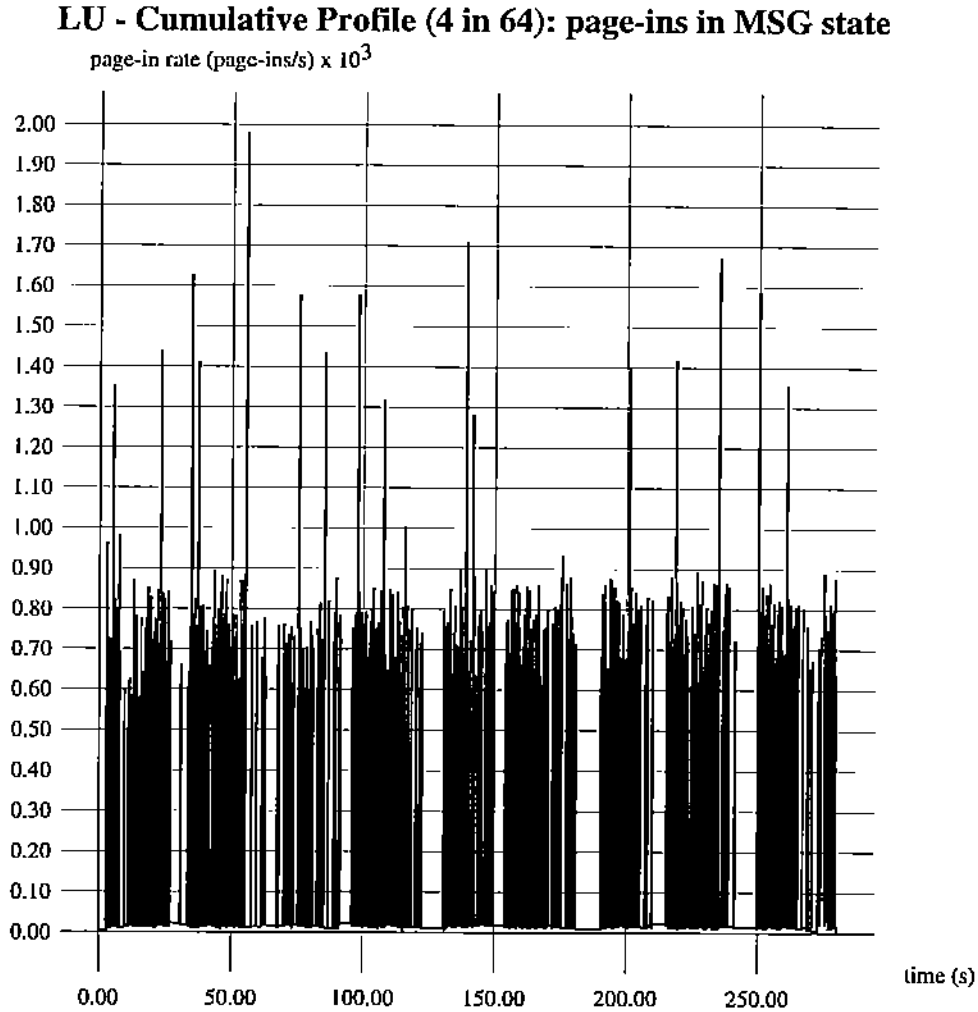


Figure 14: The cumulative page-in rate in the communication state for 4 nodes of a class A LU Lower-Upper diagonal benchmark running on 64 nodes. The total execution time was 279.97 seconds and the time reported by the NAS benchmark was 277.61 seconds.

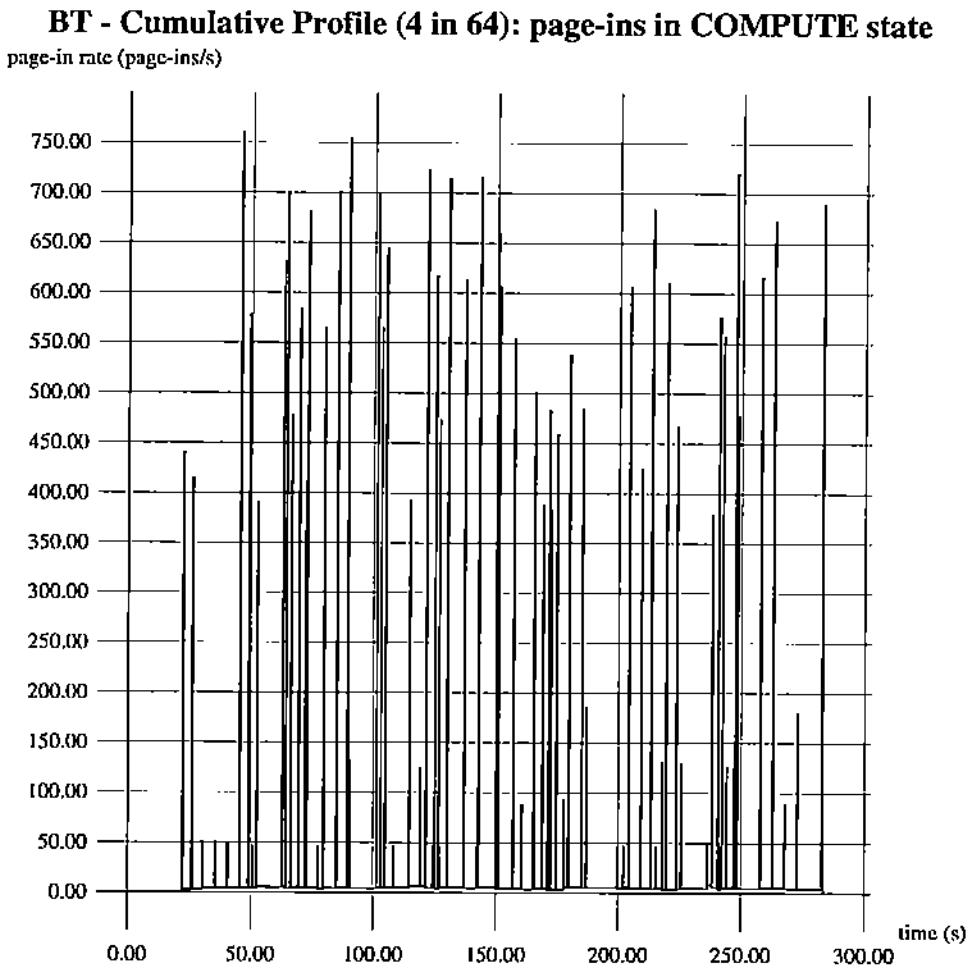


Figure 15: The cumulative page-in rate in the compute state of a class A BT Block Tridiagonal benchmark running on 64 nodes. The total execution time was 287.89 seconds and the time reported by the NAS benchmark was 264.47 seconds.