Amoeba on a Multiprocessor

Leo J. M. van Moergestel
Henri E. Bal[1]
Frans Kaashoek
Robbert van Renesse[1]
Gregory J. Sharp
Hans van Staveren
Andrew S. Tanenbaum

Dept. of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam, The Netherlands
Internet: leovm@cs.vu.nl or amoeba@cs.vu.nl

ABSTRACT

Amoeba was originally designed for a loosely-coupled network of machines. In this paper we describe some preliminary results about bus contention in a multiprocessor implementention of Amoeba.

Computing Reviews categories: C.2.4, D.4

Keywords: Multiprocessors, VMEbus, Operating systems, Distributed systems, Distributed operating systems

1. INTRODUCTION

The Amoeba project is a research effort aimed at understanding how to connect multiple computers together in a seamless way (Mullender and Tanenbaum, 1986; Tanenbaum et al., 1986; Tanenbaum and van Renesse, 1985). The basic idea is to provide the users with the illusion of a single powerful timesharing system, when, in fact, the system is implemented on a collection of machines, potentially close together, in a single rack in the machine room, or far apart, in different countries. This research has led to the design and implementation of the Amoeba distributed operating system, which is being used as a prototype and vehicle for further research. In this paper we will describe some hardware aspects as well as an application using a language, named *Orca*, for parallel applications running on a multiprocessor.

Amoeba was originally designed and implemented at the Vrije Universiteit in Amsterdam, and is now being jointly developed there and at the Centre for Mathematics and Computer Science, also in Amsterdam. The chief goal of this work is to build a distributed system that is *transparent* to the users. This concept can best be illustrated by contrasting it with a network operating system, in which each machine retains its own identity. With a network operating system,

1. This research was supported in part by the Netherlands Organization for Scientific Research (N.W.O.) under grant 125-30-10.

each user logs into one specific machine, his home machine. When a program is started, it executes on the home machine, unless the user gives an explicit command to run it elsewhere. Similarly, files are local unless a remote file system is explicitly mounted or files are explicitly copied. In short, the user is clearly aware that multiple independent computers exist, and must deal with them explicitly.

In a transparent distributed system, in contrast, users effectively log into the system as a whole, and not to any specific machine. When a program is run, the system, not the user, decides the best place to run it. The user is not even aware of this choice. Finally, there is a single, system-wide file system. The files in a single directory may be located on different machines possibly in different countries. There is no concept of file transfer, uploading or downloading from servers, or mounting remote file systems. The fact that a file is remote is not visible to the user at all.

The remainder of this paper will give a short overview of Amoeba, a description of the hardware and some speed measurements of an application running on a multiprocessor.

2. AMOEBA SYSTEM ARCHITECTURE

The Amoeba architecture consists of four principal components, as shown in Fig. 1. First are the workstations, one per user, on which users can carry out editing and other tasks that require fast interactive response. The workstations are all diskless, and are primarily used as intelligent terminals that do window management, rather than as computers for running complex user programs. We are currently using SUN-3s and VAXstations as workstations. In the next generation of hardware we may use X-terminals.

Second are the pool processors, a group of CPUs that can be dynamically allocated as needed, used, and then returned to the pool. For example, the *make* command might need to do six compilations, so six processors could be taken out of the pool for the time necessary to do the compilation and then returned. Alternatively, with a five-pass compiler, 5 times 6 = 30 processors could be allocated for the six compilations, gaining even more speedup. Many applications, such as heuristic search in AI applications (e.g., playing chess), use large numbers of pool processors to do their computing. At the Vrije Universiteit we currently have 42 single board VME-based computers using the MC68020 and MC68030 CPUs.

Third are the specialized servers, such as directory servers, file servers, data base servers, boot servers, and various other servers with specialized functions. Each server is dedicated to performing a specific function. In some cases, there are multiple servers that provide the same function, for example, as part of the replicated file system.

Fourth are the gateways, which are used to link Amoeba systems at different sites and different countries into a single, uniform system. The gateways isolate Amoeba from the peculiarities of the protocols that must be used over the wide-area networks.

B: box wid 1.4i ht 2.2i "Processor Pool" at last box.n above L1: line right 1i with .start at last box.nw + (0.2i, -0.5i) line up 0.3i with .start at 1/11 <L1.start, L1.end> line up 0.3i with .start at 2/11 <L1.start, L1.end> line up 0.3i with .start at 3/11 <L1.start, L1.end> line up 0.3i with .start at 4/11 <L1.start, L1.end> line up 0.3i with .start at 5/11 <L1.start, L1.end> line up 0.3i with with .start at 8/11 <L1.start, L1.end> line up 0.3i with .start at 9/11 <L1.start, L1.end> line up 0.3i with .start at 10/11 <L1.start, L1.end> L2: line right 1i with .start at L1.start - (0, 0.5i) line up 0.3i with .start at 1/11 <L2.start, L2.end> line up 0.3i with .start at 2/11 <L2.start, L2.end> line up 0.3i with .start at 3/11 <L2.start, L2.end> line up 0.3i with .start at 4/11 <L2.start, L2.end> line up 0.3i with .start at 5/11 <L2.start, L2.end> line up 0.3i with .start at 6/11 <L2.start, L2.end> line up 0.3i with .start at 7/11 <L2.start, L2.end> line up 0.3i with .start at 8/11 <L2.start, L2.end> line up 0.3i with .start at 9/11 <L2.start, L2.end> line up 0.3i with .start at 10/11 <L2.start, L2.end> L3: line right 1i with .start at L2.start - (0, 0.5i) line up 0.3i with with .start at 3/11 <L3.start, L3.end> line up 0.3i with .start at 4/11 <L3.start, L3.end> line up 0.3i with .start at 5/11 <L3.start, L3.end> line up 0.3i with .start at 6/11 <L3.start, L3.end> line up 0.3i with .start at 7/11 <L3.start, L3.end> line up 0.3i with .start at 8/11 <L3.start, L3.end> line up 0.3i with .start at 9/11 <L3.start, L3.end> line up 0.3i with .start at 10/11 <L3.start, L3.end> L4: line right 1i with .start at L3.start - (0, 0.5i) line up 0.3i with .start at 1/11 <L4.start, L4.end> line up 0.3i with .start at 2/11 <L4.start, L4.end> line up 0.3i with .start at 3/11 <L4.start, L4.end> line up 0.3i with .start at 4/11 <L4.start, L4.end> line up 0.3i with .start at 5/11 <L4.start, L4.end> line up 0.3i with .start at 6/11 <L4.start, L4.end> line up 0.3i with with .start at 9/11 <L4.start, L4.end> line up 0.3i with .start at 10/11 <L4.start, L4.end> line right 0.3i at B.e move up 0.5i line down 1i arc to last line.end + (0.15i, -0.15i) rad 0.15i BOTTOM: line right 1.5i arc invis to last line.end + (0.15i, 0.15i) rad 0.15i line invis up 0.35i line up 0.15i RIGHT: line up 0.5i arc to last line.end - (0.15i, -0.15i) rad 0.15i TOP: line left 1.5i arc to last line.end - (0.15i, 0.15i) rad 0.15i define workstation X line up 0.15i line right 0.15i line up 0.30i line left 0.15i line down 0.20i line left 0.15i line down 0.10i line right 0.15i X move to TOP.end + (0.15i, 0) workstation move to TOP.center workstation move to TOP.start - (0.15i, 0) workstation "Workstations" at TOP.center + (0, 0.45i) above oldwid=boxwid oldht=boxht boxwid = 0.3i boxht = 0.3i line down 0.15i at BOTTOM.start + (0.15i, 0) box with .n at last line.end line down 0.15i at BOTTOM.center box with .n at last line.end line down 0.15i at BOTTOM.end - (0.15i, 0) X: box with .n at last line.end box invis "Specialized servers" "(file, data base, etc)" with .w at X.e + (0.6i, 0) line right 0.3i at RIGHT.start box arrow right 0.5i " WAN" ljust "Gateway" at last box.n above boxwid=oldwid boxht=oldht

Fig. 1. The Amoeba architecture.

All the Amoeba machines run the same kernel, which primarily provides multithreaded processes, communication services, and little else. The basic idea behind the kernel was to keep it small, to enhance its reliability, and to allow as much as possible of the operating system to run as user processes, providing for flexibility and experimentation.

3. A CLOSER LOOK AT THE PROCESSOR POOL

Communication between CPUs is the basic hardware requirement for the Amoeba processor pool.

3.1. LAN coupled CPU boards

The simplest setup from a hardware point of view is using stand alone processors connected by a Local Area Network (LAN). We have an experimental setup with 16 MC68020 boards with 2Mbyte RAM each connected to a 10Mbit/sec Ethernet. We call this a *loosely coupled* system, because the processors only communicate over Ethernet; there is no shared memory. The advantage is the great flexibility: you just tie more CPUs to the LAN to expand your computing capacity. You can do this while Amoeba is alive and running on the other systems. The drawback is that the bandwidth of the LAN poses a severe limit on your communication speed. When the LAN is being used for other purposes, even less bandwidth is available.

3.2. Multiprocessor

For applications requiring a lot of data communication, one can use a higher bandwidth medium. This results in a *tightly coupled* system in which the processors share the same backplane and bus.

We chose the VME bus as a universal board level bus for communication. Currently we are using two multiprocessor systems based on the VME bus. One is built with 10 MC68020 CPU boards. Another one contains 17 MC68030 boards. We will give a more detailed description of the latter. It consists of 16 CPU30ZA boards and a CPU30ZAE with Ethernet interface to talk to the outside world. In this paper we will call the board with Ethernet interface the *master*. The other boards are the *slaves*. This has nothing to do with the VME concept of master and slave. An ASCU-2 bus controller and a VME RAM board are also part of the system. The CPU boards are manufactured by Force and contain a MC68030 CPU with 4Mbyte dual ported RAM. Furthermore the on-board FGA002 programmable gate array gives a good support for multiprocessing.

For the system to function we had to solve the following problems.

1) All boards should be able to access the bus even when it is heavily loaded.
2) Each board should be able to communicate with all the others.

3.2.1. Arbitration.

The VME bus has four levels for bus arbitration. The prioritised arbitration scheme would be a bad choice, because the higher-level boards could take all the bus bandwidth leaving nothing over for the lower level boards. Moreover, our multiprocessor concept is based on the idea that all processors have equal access to the bus as all CPUs have equal access to the LAN in the loosely coupled system. The VME specification also offers a round robin arbitration, which is better, but within a level we keep a prioritised daisy chain for possible bus masters with decreasing priority along the chain. The FGA002 solves this problem by introducing a fair arbitration scheme within a certain arbitration level. This means when a board gets the bus due to the round robin arbiter giving the level for arbitration it will not take the bus again if there is a lower level board in the daisy chain asking for the bus. This results in fair bus access for the lower boards in the daisy chain even in a heavy loaded system.

To demonstrate the effect of fair arbitration we wrote a test program. The master downloads the slaves and generates a broadcast message (Force Message Broadcast, FMB) to start the test. During the test the slaves try to increment a counter in global (VME) memory. A reset will stop the process and the counter values can be read.

We ran this test with 16 boards (16 slaves) Each bus request level had 4 boards in a daisy chain. Processor 1, 2, 3 and 4 were on BREQ 3; processors 5, 6, 7, and 8 were on BREQ 2 and so on. A FMB interrupt set a flag to TRUE to start the counting. Interrupts were off during counting. Fair arbitration was on in one set of tests and off in another. Between two increment instruictions the CPUs entered a small delay-loop.

Figure 2 and 3 show the result for a delay of 10 arbitrary units. In these figures we see that in the normal arbitration scheme, number four in the daisy chain is completely unable to access the bus. In fact this CPU encountered a bus timeout error so it gave simply up. When we just skip the delay between the increments the results are even more dramatic as figure 4 and 5 show.

Fig. 2. Fair arbitration off

Fig. 3. Fair arbitration on

Fig. 4. Fair arbitration off

Fig. 5. Fair arbitration on

3.2.2. Interboard Communication Using Location Monitors.

The way the CPUs communicate is by writing to a certain buffer in the destination processor's dual ported RAM and then generating a mailbox interrupt. Each processor 'owns' a buffer in the dual ported RAM of the other processor boards. The simplest way to implement a location monitor or mailbox is by generating an interrupt when there is an access to a certain address. This implementation has a drawback. After generating the interrupt, the interrupted CPU has to handle it, which takes a certain amount of time. If another board also decides to generate a mailbox interrupt simultaneously, one of the interrupts will be lost, depending on the software implementation of the interrupt handling. Of course one can solve this problem, called a *race condition*, in software, but it would be nice if the hardware helped.

The implementation of the location monitors on the CPU30 is more like a semaphore. To generate an interrupt, the interrupting CPU reads a byte from a location monitor address. If the read returns a zero the mailbox was free and an interrupt is generated on the CPU accessed. Until the interrupted CPU does a write to the same mailbox, all other boards that try to use this mailbox will get a non-zero result on a read, meaning that the mailbox is not empty and no interrupt is being generated. They thus 'know' they did not get through and have to try again or use another mailbox. The interrupted CPU will eventually free the mailbox by writing to it (in the interrupt service routine).

3.2.3. Dual Ported RAM

The on-board RAM is accessable by both the CPU and the VME bus. This means that all CPUs are able to access each other's on board RAM. With the help of the FGA002 it's possible to protect some parts of this RAM so one board with corrupted code cannot destroy the programs residing on the other boards. Only the communication buffers are accessable.

This brings us to another topic. What if one board crashes? The master board discovers that it cannot communicate anymore with the crashed board. Again the gate array offers a solution. By writing to a certain register (accessable from the VME bus) an on board RESET is generated. So the failing board can be rebooted without disturbing the whole system.

4. LOW-LEVEL BOOT SOFTWARE.

After initialising the gate array. A small program in ROM is started on the slave boards. This program waits for a mailbox interrupt. When it occurs, a certain RAM location is checked for a magic number and a jump to that location is executed.

The master board containing the Ethernet interface runs a small Amoeba kernel in ROM so it can communicate with the outside world. The master is downloaded with the Amoeba kernel for the slaves as well with its own runtime system. It examines the VME bus and copies the slave kernel to all the slaves it discovers. The slaves are then triggered by a mailbox interrupt and start executing their code.

5. APPLICATIONS

Amoeba has been used to program a variety of applications. To mention a few: UNIX† emulation, parallel *make*, traveling salesman, and alpha-beta search.

5.1. Parallel Applications and Orca

Although Amoeba was originally conceived as a system for *distributed* computing, the existence of the multiprocessor with 16 MC68030 CPUs close together has made it quite suitable for *parallel* computing as well. That is, we have become much more interested in using the multiprocessor to achieve large speedups on a single problem. To program these parallel applications, we have designed and implemented a language called Orca (Bal and Tanenbaum, 1988).

Orca is based on the concept of globally shared objects. Programmers can define operations on shared objects, and the compiler and run time system take care of all the details of making sure they are carried out correctly. This scheme gives the programmer to ability to atomically read and write shared objects that are physically distributed among a collection of machines without having to deal with any of the complexity of the physical distribution. All the details of the physical distribution are completely hidden from the programmer. Initial results indicate that almost linear speedup can be achieved on some problems.

5.2. Example of Problem Solving Using Orca

In this section we will describe traveling salesman problem. In the traveling salesman problem, the computer is given a starting city and a list of other cities to be visited. The idea is to find the shortest path that visits each city exactly once, and then returns to the starting place. Using Amoeba we have programmed this application in parallel by having one pool processor act as coordinator, and the rest as slaves.

Suppose, for example, that the starting place is London, and the cities to be visited include New York, Sydney, Nairobi, and Tokyo. The coordinator might tell the first slave to investigate all paths starting with London-New York, the second slave to investigate all paths starting with London-Sydney, the third slave to investigate all paths starting with London-Nairobi, and so on. All of these searches go on in parallel. When a slave is finished, it reports back to the coordinator and gets a new assignment.

The algorithm can be applied recursively. For example, the first slave could allocate a processor to investigate paths starting with London-New York-Sydney, another processor to investigate London-New York-Nairobi, and so forth. At some point, of course, a cutoff is needed at which a slave actually does the calculation itself and does not try to farm it out to other processors.

The performance of the algorithm can be greatly improved by keeping track of the best total path found so far. A good initial path can be found by using the ‘‘closest city next’’ heuristic.

† UNIX is a trademark of Bell Laboratories.

Whenever a slave is started up, it is given the length of the best total path so far. If it ever finds itself working on a partial path that is longer than the best-known total path, it immediately stops what it is doing, reports back failure, and asks for more work.

Figure 6 shows the results of a timing measurement of the Traveling salesman problem as a function of the total number of CPUs used to perform the calculation. Graph: [# gg 0 .. 16, 0 .. 16 define xy gg @ (($1)-(0))*.25, (($2)-(0))*.25 @ define x gg @ (($1)-(0))*.25 @ define y gg @ (($1)-(0))*.25 @ frameht = 4 framewid = 4 Frame: box ht frameht wid framewid with .sw at 0,0 textht = .166667 textwid = .6 Label: box invis wid 0 ht 1*textht "Speedup " wid textwid with .e at Frame.w - (0.2,0) textht = .166667 Label: box invis wid 0 ht 1*textht "Number of CPUs" with .n at Frame.s - (0,2 * textht) ticklen = .1 Ticks gg: line left ticklen from (0,y gg(1)) "1 " rjust at last line.end line left ticklen from (0,y gg(2)) "2 " rjust at last line.end line left ticklen from (0,y gg(3)) "3 " rjust at last line.end line left ticklen from (0,y gg(4)) "4 " rjust at last line.end line left ticklen from (0,y gg(5)) "5 " rjust at last line.end line left ticklen from (0,y gg(6)) "6 " rjust at last line.end line left ticklen from (0,y gg(7)) "7 " rjust at last line.end line left ticklen from (0,y gg(8)) "8 " rjust at last line.end line left ticklen from (0,y gg(9)) "9 " rjust at last line.end line left ticklen from (0,y gg(10)) "10 " rjust at last line.end line left ticklen from (0,y gg(11)) "11 " rjust at last line.end line left ticklen from (0,y gg(12)) "12 " rjust at last line.end line left ticklen from (0,y gg(13)) "13 " rjust at last line.end line left ticklen from (0,y gg(14)) "14 " rjust at last line.end line left ticklen from (0,y gg(15)) "15 " rjust at last line.end line left ticklen from (0,y gg(16)) "16 " rjust at last line.end ticklen = .1 Ticks gg: line down ticklen from (x gg(1),0) box invis "1" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(2),0) box invis "2" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(3),0) box invis "3" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(4),0) box invis "4" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(5),0) box invis "5" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(6),0) box invis "6" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(7),0) box invis "7" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(8),0) box invis "8" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(9),0) box invis "9" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(10),0) box invis "10" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(11),0) box invis "11" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(12),0) box invis "12" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(13),0) box invis "13" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(14),0) box invis "14" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(15),0) box invis "15" ht .25 wid 0 with .n at last line.end line down ticklen from (x gg(16),0) box invis "16" ht .25 wid 0 with .n at last line.end Lgg: xy gg(1,1) "□" at xy gg(1,1) line from Lgg to xy gg(2,1.98); Lgg: Here "□" at xy gg(2,1.98) line from Lgg to xy gg(3,3); Lgg: Here "□" at xy gg(3,3) line from

Lgg to xy gg(4,3.96); Lgg: Here "□" at xy gg(4,3.96) line from Lgg to xy gg(5,5.1); Lgg: Here "□" at xy gg(5,5.1) line from Lgg to xy gg(6,5.48); Lgg: Here "□" at xy gg(6,5.48) line from Lgg to xy gg(7,5.98); Lgg: Here "□" at xy gg(7,5.98) line from Lgg to xy gg(8,6.38); Lgg: Here "□" at xy gg(8,6.38) line from Lgg to xy gg(9,6.58); Lgg: Here "□" at xy gg(9,6.58) line from Lgg to xy gg(10,6.64); Lgg: Here "□" at xy gg(10,6.64) line from Lgg to xy gg(11,6.78); Lgg: Here "□" at xy gg(11,6.78) line from Lgg to xy gg(12,6.71); Lgg: Here "□" at xy gg(12,6.71) line from Lgg to xy gg(13,6.78); Lgg: Here "□" at xy gg(13,6.78) line from Lgg to xy gg(14,6.78); Lgg: Here "□" at xy gg(14,6.78) line from Lgg to xy gg(15,6.51); Lgg: Here "□" at xy gg(15,6.51) line from Lgg to xy gg(16,6.39); Lgg: Here "□" at xy gg(16,6.39) line dotted from xy gg(1,1) to xy gg(16,16) line dotted from xy gg(1,15) to xy gg(3,15) box invis wid 0 ht 1*textht "Perfect speedup" ljust at xy gg(3.5,15) Lgg: xy gg(1,14.4) "□" at xy gg(1,14.4) line from Lgg to xy gg(2,14.4); Lgg: Here "□" at xy gg(2,14.4) line from Lgg to xy gg(3,14.4); Lgg: Here "□" at xy gg(3,14.4) box invis wid 0 ht 1*textht "Speedup for Orca" ljust at xy gg(3.5,14.4)

]

Fig. 6. Traveling salesman

This picture shows that we have a problem when too many CPUs try to read the 'best total path found so far' in global VME RAM. This problem can be solved by keeping a copy of this value in local RAM and perform an update whenever a better result is found.

6. REFERENCES

Bal, H.E., Renesse, R. van, and Tanenbaum, A.S. Implementing Distributed Algorithms using Remote Procedure Call, *Proc. National Computer Conference*, AFIPS, 1987. pp. 499-505.

Bal, H.E., and Tanenbaum, A.S. Distributed Programming with Shared Data, *IEEE Conf. on Computer Languages*, IEEE, 1988, pp. 82-91.

Cheriton, D.R. The V Distributed System, *Commun. ACM* 31, (March 1988), pp. 314-333.

Marsland, T.A., and Campbell, M. Parallel Search of Strongly Ordered Game Trees, *Computing Surveys* 14, (Dec. 1982) pp. 533-551.

Mullender, S.J., and Tanenbaum, A.S. The Design of a Capability-Based Distributed Operating System, *Computer Journal* 29, (Aug. 1986), pp. 289-299.

Mullender, S.J., and Tanenbaum, A.S. A Distributed File Service Based on Optimistic Concurrency Control, *Proc. Tenth Symp. Oper. Syst. Prin.*, 1985, pp. 51-62.

Renesse, R. van, Tanenbaum, A.S., and Wilschut, A The Design of a High-Performance File Server *Proc. Ninth Int'l Conf. on Distr. Comp. Systems*, IEEE, 1989a, pp. 22-27.

Renesse, R. van, Staveren, H. van, and Tanenbaum, A.S. Performance of the Amoeba Distributed Operating System, *Software—Practice and Experience* 19, (March 1989b) pp. 223-234.

Renesse, R. van, Staveren, H. van, and Tanenbaum, A.S. Performance of the World's Fastest Distributed Operating System, *Operating Systems Review* 22, (Oct. 1988), pp. 25-34.

Renesse, R. van, and Tanenbaum, A.S. Voting with Ghosts, *Proc. Eighth Int'l Conf. on Distr. Comp. Systems*, IEEE, 1988, pp. 456-461.

Renesse, R. van, Tanenbaum, A.S., Staveren, H. van, and Hall, J. Connecting RPC-Based Distributed Systems Using Wide-Area Networks, *Proc. Seventh Int'l Conf. on Distr. Comp. Systems*, IEEE, 1987, pp. 28-34.

Tanenbaum, A.S.A UNIX Clone with Source Code for Operating Systems Courses, *Operating Syst. Rev.* 21, (Jan. 1987), pp. 20-29.

Tanenbaum, A.S., Mullender, S.J., and Renesse, R., van Using Sparse Capabilities in a Distributed Operating System *Proc. Sixth International Conf. on Distr. Computer Systems*, IEEE, 1986.

Tanenbaum, A.S., and Renesse, R. van A Critique of the Remote Procedure Call Paradigm *Proc. Euteco '88* 1988, pp. 775-783.

Tanenbaum, A.S., and Renesse, R. van Distributed Operating Systems, *Computing Surveys* 17, (Dec. 1985) pp. 419-470.

What follows is a shar archive of postscript files for the figures. These were not done in PIC.

: This is a shar archive. Extract with sh, not csh. : This archive ends with exit, so do not worry about trailing junk. : --------------------------- cut here -------------------------- PATH=/bin:/usr/bin:/usr/ucb echo Extracting ’Fig2.ps’ sed ’s/^X//’ > ’Fig2.ps’ << ’+ END-OF-FILE ’’Fig2.ps’ X%! X%%BoundingBox: 36 36 558 400 X%% GKS example with some little PostScript-only games played. X/vsc 10.0 def % vertical scale is 0.1 (horiz is ok) X/llbl [(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)]def X/stxt /Courier findfont .35 scalefont def X/ntxt /Helvetica findfont .30 scalefont def X/axlbl /Helvetica findfont .46 scalefont def X/ltxt /Times-Roman findfont .46 scalefont def X/GKD 50 dict def /xdf {exch def} def X X/bars [1. .998 .500 .000 .99 .99 .500 .000 .99 .99 .500 .000 .99 .99 .500 .000 .000] def X%--------------------- X X% (str) x y cshow X/cshow{GKD begin /ctry xdf /ctrx xdf /cstr xdf X cstr stringwidth pop 2 div ctrx exch sub ctry moveto cstr show end}def X X% draw a big dot at current position X/drawdot{gsave currentpoint newpath moveto currentpoint lineto X lwid 2 mul setlinewidth stroke grestore}def X X% xarray yarray /pointfunc linewidth polyline X/polyline{GKD begin gsave 1 setlinecap 1 setlinejoin X /lwid xdf /ptfunc exch load def /my xdf /mx xdf X newpath 0 1 my length 1 sub {/i xdf X mx i get my i get i 0 eq X {moveto}{lineto}ifelse ptfunc} for lwid setlinewidth stroke X grestore end}def X X% llx lly urx ury lwid filltype fbox X/fbox{GKD begin /tfill xdf X /lw2 xdf /ury xdf /urx xdf X /lly exch lw2 add def /llx xdf X newpath llx lly moveto llx ury lineto urx ury lineto urx lly lineto X closepath gsave clip gsave tfill setgray fill grestore X lw2 2 mul setlinewidth stroke X grestore end X}def X X%--------- X/onegraph { X gsave 20 20 scale X% move origin to (x,y) 4,7 X 4 7 translate X% draw bounding box X -1.2 -1.2 17.5 11.5 0 0.975 fbox X% draw titel in a box X ltxt setfont (Delay = 10, fair = off) 10.5 10.5 cshow X % draw axes X newpath 0 10.5 moveto 0 0 lineto 17 0 lineto .05 setlinewidth stroke X % put Y axis tick-marks X .025 setlinewidth stxt setfont X gsave newpath 11{-.17 0 moveto 0 0 lineto 0 1 translate}repeat X stroke grestore X % put X axis tickmarks X 1 1 16 {dup llbl exch 1 sub get exch -0.3 cshow} for X X % put X axis label X axlbl setfont (Pool Processor) 8.5 -1 cshow X % put Y axis label X gsave 90 rotate (Relative Performance (%)) 4.5 .7 cshow X% put numbers along y-axis X stxt setfont 0 1 10 {dup 10 mul ( ) cvs exch 0.25 cshow}for X grestore X X % draw the boxes X 0 1 15{/i xdf i .65 add 0 i 1.35 add bars i get X vsc mul .02 .76 fbox}for X X grestore X showpage X} def Xonegraph + END-OF-FILE Fig2.ps chmod ’u=rw,g=r,o=r’ ’Fig2.ps’ set ‘wc -c ’Fig2.ps’‘ count=$1 case $count in 2407) :;; *) echo ’Bad character count in ’’Fig2.ps’ >&2 echo ’Count should be 2407’ >&2 esac echo Extracting ’Fig3.ps’ sed ’s/^X//’ > ’Fig3.ps’ << ’+ END-OF-FILE ’’Fig3.ps’ X%! X%%BoundingBox: 36 36 558 400 X%% GKS example with some little PostScript-only games played. X/vsc 10.0 def % vertical scale is 0.1 (horiz is ok) X/llbl [(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)]def X/stxt /Courier findfont .35 scalefont def X/ntxt /Helvetica findfont .30 scalefont def X/axlbl /Helvetica findfont .46 scalefont def X/ltxt /Times-Roman findfont .46 scalefont def X/GKD 50 dict def /xdf {exch def} def X X/bars [1. .930 .820 .750 .99 .93 .820 .770 .99 .92 .81 .76 .99 .90 .8 .75 .000] def X%--------------------- X X% (str) x y cshow X/cshow{GKD begin /ctry xdf /ctrx xdf

/cstr xdf X cstr stringwidth pop 2 div ctrx exch sub ctry moveto cstr show end}def X X% draw a big dot at current position X/drawdot{gsave currentpoint newpath moveto currentpoint lineto X lwid 2 mul setlinewidth stroke grestore}def X X% xarray yarray /pointfunc linewidth polyline X/polyline{GKD begin gsave 1 setlinecap 1 setlinejoin X /lwid xdf /ptfunc exch load def /my xdf /mx xdf X newpath 0 1 my length 1 sub {/i xdf X mx i get my i get i 0 eq X {moveto}{lineto}ifelse ptfunc} for lwid setlinewidth stroke X grestore end}def X X% llx lly urx ury lwid filltype fbox X/fbox{GKD begin /tfill xdf X /lw2 xdf /ury xdf /urx xdf X /lly exch lw2 add def /llx xdf X newpath llx lly moveto llx ury lineto urx ury lineto urx lly lineto X closepath gsave clip gsave tfill setgray fill grestore X lw2 2 mul setlinewidth stroke X grestore end X}def X X%--------- X/onegraph { X gsave 20 20 scale X% move origin to (x,y) 4,7 X 4 7 translate X% draw bounding box X -1.2 -1.2 17.5 11.5 0 0.975 fbox X% draw titel in a box X ltxt setfont (Delay = 10, fair = on) 10.5 10.5 cshow X % draw axes X newpath 0 10.5 moveto 0 0 lineto 17 0 lineto .05 setlinewidth stroke X % put Y axis tickmarks X .025 setlinewidth stxt setfont X gsave newpath 11{-.17 0 moveto 0 0 lineto 0 1 translate}repeat X stroke grestore X % put X axis tickmarks X 1 1 16 {dup llbl exch 1 sub get exch -0.3 cshow} for X X % put X axis label X axlbl setfont (Pool Processor) 8.5 -1 cshow X % put Y axis label X gsave 90 rotate (Relative Performance (%)) 4.5 .7 cshow X% put numbers along y-axis X stxt setfont 0 1 10 {dup 10 mul ( ) cvs exch 0.25 cshow}for X grestore X X % draw the boxes X 0 1 15{/i xdf i .65 add 0 i 1.35 add bars i get X vsc mul .02 .76 fbox}for X X grestore X showpage X} def Xonegraph + END-OF-FILE Fig3.ps chmod 'u=rw,g=r,o=r' 'Fig3.ps' set 'wc -c 'Fig3.ps'' count=$1 case $count in 2401) :;; *) echo 'Bad character count in ''Fig3.ps' >&2 echo 'Count should be 2401' >&2 esac echo Extracting 'Fig4.ps' sed 's/^X//' > 'Fig4.ps' << '+ END-OF-FILE ''Fig4.ps' X%! X%%BoundingBox: 36 36 558 400 X%% GKS example with some little PostScript-only games played. X/vsc 10.0 def % vertical scale is 0.1 (horiz is ok) X/llbl [(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)]def X/stxt /Courier findfont .35 scalefont def X/ntxt /Helvetica findfont .30 scalefont def X/axlbl /Helvetica findfont .46 scalefont def X/ltxt /Times-Roman findfont .46 scalefont def X/GKD 50 dict def /xdf {exch def} def X X/bars [1. .000 .000 .000 .999 .000 .000 .000 .999 .000 .000 .000 .999 .000 .000 .000 .000] def X%--------------------- X X% (str) x y cshow X/cshow{GKD begin /ctry xdf /ctrx xdf /cstr xdf X cstr stringwidth pop 2 div ctrx exch sub ctry moveto cstr show end}def X X% draw a big dot at current position X/drawdot{gsave currentpoint newpath moveto currentpoint lineto X lwid 2 mul setlinewidth stroke grestore}def X X% xarray yarray /pointfunc linewidth polyline X/polyline{GKD begin gsave 1 setlinecap 1 setlinejoin X /lwid xdf /ptfunc exch load def /my xdf /mx xdf X newpath 0 1 my length 1 sub {/i xdf X mx i get my i get i 0 eq X {moveto}{lineto}ifelse ptfunc} for lwid setlinewidth stroke X grestore end}def X X% llx lly urx ury lwid filltype fbox X/fbox{GKD begin /tfill xdf X /lw2 xdf /ury xdf /urx xdf X /lly exch lw2 add def /llx xdf X newpath llx lly moveto llx ury lineto urx ury lineto urx lly lineto X closepath gsave clip gsave tfill setgray fill grestore X lw2 2 mul setlinewidth stroke X grestore end X}def X

X%--------- X/onegraph { X gsave 20 20 scale X% move origin to (x,y) 4,7 X 4 7 translate X% draw bounding box X -1.2 -1.2 17.5 11.5 0 0.975 fbox X% draw titel in a box X ltxt setfont (Delay = 0, fair = off) 10.5 10.5 cshow X % draw axes X newpath 0 10.5 moveto 0 0 lineto 17 0 lineto .05 setlinewidth stroke X % put Y axis tick-marks X .025 setlinewidth stxt setfont X gsave newpath 11{-.17 0 moveto 0 0 lineto 0 1 translate}repeat X stroke grestore X % put X axis tickmarks X 1 1 16 {dup llbl exch 1 sub get exch -0.3 cshow} for X X % put X axis label X axlbl setfont (Pool Pro-cessor) 8.5 -1 cshow X % put Y axis label X gsave 90 rotate (Relative Performance (%)) 4.5 .7 cshow X% put numbers along y-axis X stxt setfont 0 1 10 {dup 10 mul ( ) cvs exch 0.25 cshow}for X grestore X X % draw the boxes X 0 1 15{/i xdf i .65 add 0 i 1.35 add bars i get X vsc mul .02 .76 fbox}for X X grestore X showpage X} def Xonegraph + END-OF-FILE Fig4.ps chmod 'u=rw,g=r,o=r' 'Fig4.ps' set 'wc -c 'Fig4.ps'' count=$1 case $count in 2412) :;; *) echo 'Bad character count in ''Fig4.ps' >&2 echo 'Count should be 2412' >&2 esac echo Extracting 'Fig5.ps' sed 's/^X//' > 'Fig5.ps' << '+ END-OF-FILE ''Fig5.ps' X%! X%%BoundingBox: 36 36 558 400 X%% GKS example with some little PostScript-only games played. X/vsc 10.0 def % vertical scale is 0.1 (horiz is ok) X/llbl [(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)]def X/stxt /Courier findfont .35 scalefont def X/ntxt /Helvetica findfont .30 scalefont def X/axlbl /Helvetica findfont .46 scalefont def X/ltxt /Times-Roman findfont .46 scalefont def X/GKD 50 dict def /xdf {exch def} def X X/bars [1. .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .999 .000] def X%--------------------- X X% (str) x y cshow X/cshow{GKD begin /ctry xdf /ctrx xdf /cstr xdf X cstr stringwidth pop 2 div ctrx exch sub ctry moveto cstr show end}def X X% draw a big dot at current position X/drawdot{gsave currentpoint newpath moveto currentpoint lineto X lwid 2 mul setlinewidth stroke grestore}def X X% xarray yarray /pointfunc linewidth polyline X/polyline{GKD begin gsave 1 setlinecap 1 setlinejoin X /lwid xdf /ptfunc exch load def /my xdf /mx xdf X newpath 0 1 my length 1 sub {/i xdf X mx i get my i get i 0 eq X {moveto}{lineto}ifelse ptfunc} for lwid set-linewidth stroke X grestore end}def X X% llx lly urx ury lwid filltype fbox X/fbox{GKD begin /tfill xdf X /lw2 xdf /ury xdf /urx xdf X /lly exch lw2 add def /llx xdf X newpath llx lly moveto llx ury lineto urx ury lineto urx lly lineto X closepath gsave clip gsave tfill setgray fill grestore X lw2 2 mul setlinewidth stroke X grestore end X}def X X%--------- X/onegraph { X gsave 20 20 scale X% move origin to (x,y) 4,7 X 4 7 translate X% draw bounding box X -1.2 -1.2 17.5 11.5 0 0.975 fbox X% draw titel in a box X ltxt setfont (Delay = 0, fair = on) 10.5 10.5 cshow X % draw axes X newpath 0 10.5 moveto 0 0 lineto 17 0 lineto .05 setlinewidth stroke X % put Y axis tick-marks X .025 setlinewidth stxt setfont X gsave newpath 11{-.17 0 moveto 0 0 lineto 0 1 translate}repeat X stroke grestore X % put X axis tickmarks X 1 1 16 {dup llbl exch 1 sub get exch -0.3 cshow} for X X % put X axis label X axlbl setfont (Pool Pro-cessor) 8.5 -1 cshow X % put Y axis label X gsave 90 rotate (Relative Performance (%)) 4.5 .7 cshow X% put numbers along y-axis X stxt setfont 0 1 10 {dup 10 mul ( ) cvs exch 0.25 cshow}for X grestore X X % draw the boxes X 0 1 15{/i xdf i .65 add 0 i

1.35 add bars i get X vsc mul .02 .76 fbox}for X X grestore X showpage X} def Xonegraph + END-OF-FILE Fig5.ps chmod ’u=rw,g=r,o=r’ ’Fig5.ps’ set ‘wc -c ’Fig5.ps’‘ count=$1 case $count in 2411) :;; *) echo ’Bad character count in ’’Fig5.ps’ >&2 echo ’Count should be 2411’ >&2 esac exit 0