JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM
SFB: 366-91-1
July 1, 1991

TO: Time Warp Group

FROM: Steve Bellenot

SUBJECT: The state skipping paper

The enclosed paper "State skipping performance with the Time Warp Operating System" is being submitted to January 1991 PADS conference in Newport Beach. (Only slightly late due to the recent earthquake.)

State Skipping Performance with the Time Warp Operating System

(The State Skipping Paper)

Steve Bellenot

ABSTRACT

Optimistic methods of synchronizing parallel discrete event simulations have a state saving overhead that neither sequential, nor conservative methods require. In an optimistic method, a simulation object can rollback to an earlier simulation time and hence copies of the state variables at earlier simulation times are needed. However, the state variables need not be saved after every event, since missing copies of these states can be recomputed. State skipping is the number of states which you don't save between states that are saved.

Performance results for a number of benchmarks suggest that there are increases in speed of execution with state skipping when running on a small number of processors, but these benefits decrease or can become liabilities as the number of processors increase. These results are roughly in agreement with the theoretical predictions of Lin and Lazowska.

Introduction

The Time Warp Operating System (TWOS) is the Jet Propulsion Laboratory's (JPL's) operating system implementation of Time Warp [J et. al.]. Time Warp is a method of optimistically synchronizing parallel discrete event simulations [J]. For TWOS, a simulation is divided into "objects." Objects have a "state" which contains all of its state variables. Optimistic synchronization requires the archiving of old copies of this state. This allows objects to "rollback" to earlier simulation times and resume execution. Thus if a straggler event (message) arrives, then the object, which is (perhaps incorrectly) executing in the future, will be "rollback" to execute the straggler.

It is not necessary to save a copy of an objects state after every event. Missing states can be reconstructed as long as there is a state with an earlier simulation time and copies of all events (messages) between the two states. By not copying a state after an event, one can save the storage and the execution time needed to allocate the

storage and to do the actually copying. On the other hand, by not copying a state, one risks losing execution time by having to reconstruct the state after a rollback.

State skipping is the number of states which you don't save between states that are saved. Current versions of TWOS have a state skipping number of zero. They always save a copy of the state of an object after every event. (But see the historical note below.) We modified version 2.4.1 of TWOS to test the effects of varying the state skipping number on the run time of four benchmarks. A term related to the state skipping number is the checkpoint interval. The checkpoint interval is the number of events between save states. Thus the checkpoint interval is always one more then the state skipping number.

The motivation for this work was to resolve an apparent contradiction. Old studys on TWOS showed that saving a copy of the state after every event produced the fastest speedups. While Preiss, MacIntyre and Loucks [PML] have shown that state skipping of one or more can significantly improve the execution time of a simulation. The number of processors seems to be the key to this mystery. As one increases the number of processors used to run a particular simulation, the value of state skipping decreases.

Copying Overhead in Optimistic Simulation Synchronizations

The overhead associated with state copying in Time Warp has been mentioned several times in the literature. State copying (and also the message copying) are (both space and time) overheads to optimistic methods but are totally unneeded in either sequential methods or conservative methods. Both software and hardware solutions to minimizing this overhead have been offered. For software, Lin and Lazowska [LL] have, under simplifying assumptions, theoretical results on how often to save a state. (Their results are outlined below.) Preiss, MacIntyre and Loucks [PML] have empirical results which show that this software solution can save both space and time. For hardware, Fujimoto, Tsai and Gopalakrishnan [FTG] have designed the "Rollback Chip" which would off load all the state saving into hardware. Integrated Parallel Technology is currently in the process of producing rollback chips [BRF]. Felderman and Kleinrock [FK] have theoretical results on state saving in the two node case. Even the original design of TWOS worried about state saving overheads (see historial note below).

Extreme Case 1: Time Warp on One Node.

When running Time Warp on one node (processor) several optimizations are possible. Since there are no rollbacks in a one node Time Warp (see below), the is no need to archive any state copies. Thus all of the state copying overall could be removed by making the state skipping number infinity. (Lin and Lazowska also predict infinity for the one node case.) By analogy, one would not be too surprized that if a simulation run on a few nodes had few rollbacks, then relatively high state skipping numbers would produce the fastest run times.

In practice there are two problems with infinite state skipping. The first is that one runs out of memory because old fossil messages are never garbage collected. Indeed, all the messages since the last saved state are needed to recreate the missing states. If the state skipping number is too high, one can run out of memory storing all of these messages. (The bank benchmark below shows that this can happen in practice.) The second problem is that rollback can occur on one node, if the simulation has events for "now."

Extreme Case 2: Time Warp with Idle Time.

It is possible to spread the objects of a simulation over enough nodes that there isn't enough useful work to keep all of the nodes busy. Since this idle time is free, it would cost nothing to use this time saving states. (Well it does cost storage.) That is some of the state saving overhead is using CPU time that would be otherwise unused. Thus as one increases the number of nodes, one would expect the idle time to increase and the value of state skipping to decrease.

The amount of idle time is related to the message density. Message density can be measured on a per node or per object basis. The object message density is the number of active messages divided by the number of objects. The node message density is measured on a pure node basis. If there is no idle time, then the node message density must be large enough to keep all the nodes busy.

Of course bottleneck and runaway objects won't have idle time. One hopes that a well designed decomposition of the simulation into objects would not produce bottlenecks. Runaway objects, objects which will are always ready to execute, often incorrectly, often way in the future of the rest of the simulation, will use any otherwise idle time to do what is likely wrong computation. Again using the time to save a state benefits the simulation as a whole as it slows the

runaway object. (Runaway objects are often self-propelled by always scheduling a future event for itself.)

We will see below (Figure 1) that the JPL standard benchmarks all have idle time on large number of nodes. On the other hand, the benchmarks of Preiss, MacIntyre and Loucks [PML] all have a per processor message density of at least 32 with one exception which had a density of 8 messages per processor. Assuming random placement of 64 messages over their 8 processors (message density 8), the probability there is a processor with no message is less than 0.0016, and with 256 messages (message density 32) the probability decreases by a factor of a trillion.

Increasing the number of nodes increases the chances for idleness in two ways. Even if the message density per node remains constant as the number of nodes increase there is a better chance for some node being idle as Table 1 shows. More often, the number of messages remain constant as the number of nodes increase and idleness increases even faster (see Table 2). A graph of node zero idleness versus number of nodes for a fixed number of messages grows exponentially. Since the curves in Figure 1 are roughly linear these combinatorics do not explain all that is happening.

Message Density per Node	8 nodes	64 nodes
1	0.9976	0.99999998
4	0.11	0.70
8	0.0016	0.0200

Table 1: Probability of Some node being idle.

Total Messages	8 nodes	64 nodes
8	0.34	0.88
32	0.014	0.60
64	0.0002	0.36
256	1.4E-15	0.018

Table 2: Probability of Node Zero being Idle

The Middle Ground:

There are two modes of Time Warp behavior which have been called saturated and unsaturated. A saturated simulation always (mostly) has good simulation work to do, where as an unsaturated simulation can (often) run out of correct work. Perhaps what we are witnessing is a transition from one behavior to the other as we change the number of nodes.

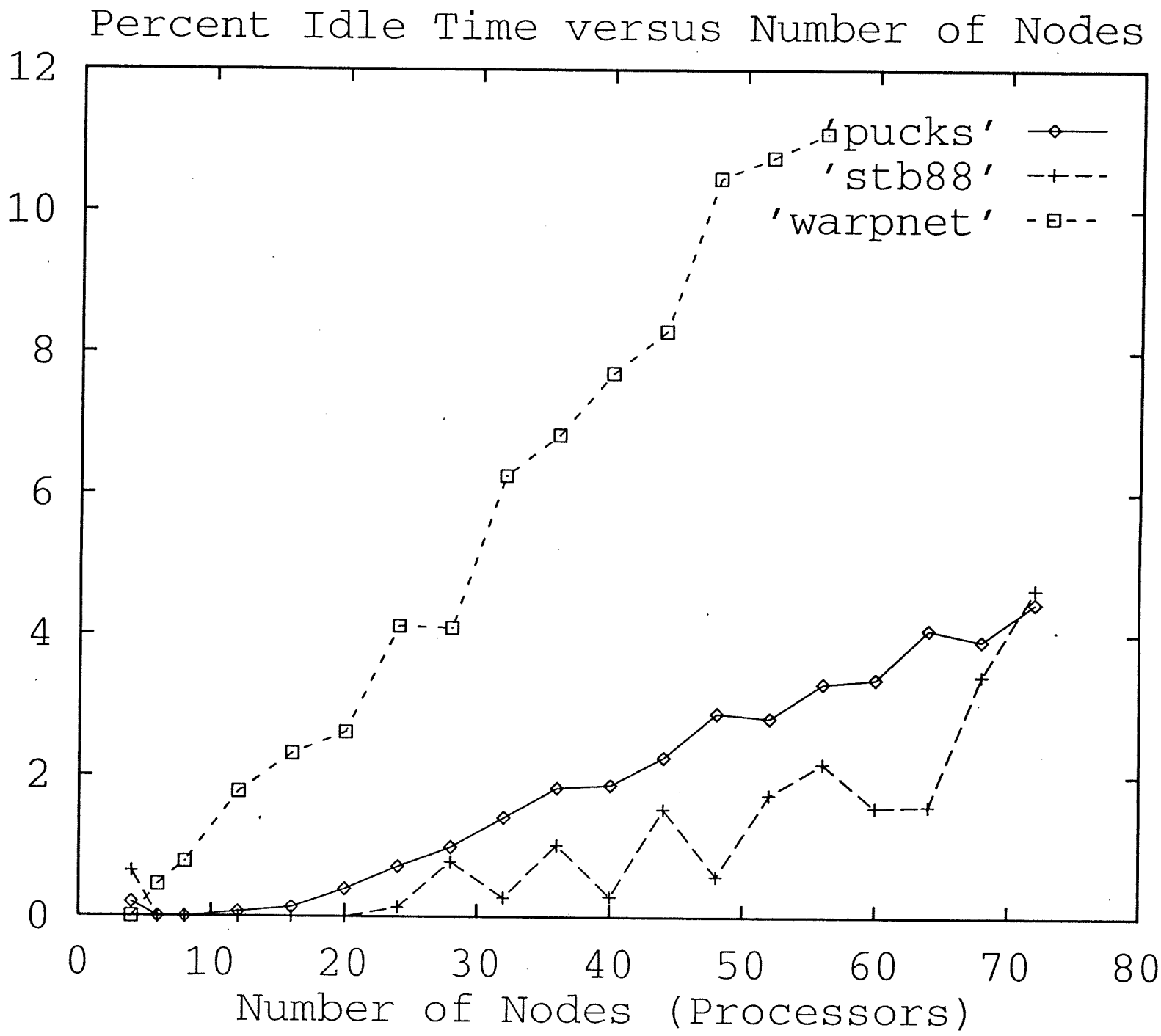
Percent Idle Time versus Number of Nodes
'pucks'
'stb88'
'warpnet'
12
10
8
6
4
2
0
0
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 1.

The Lin and Lazowska Estimates

For full details see [LL], we start with four measured variables for a simulation:

G The granularity, the average real time it takes an object to execute one event.

C The average real time it takes to save an object's state.

E The number of events executed in the simulation when states are saved after every event.

R The number of rollbacks in the simulation when states are saved after every event.

From the above variables we derive two ratios. The "space/time" ratio of an object, T = C/G. Note that T is independent of the number of nodes which the simulation is executed on. The second ratio is the expected number of events between rollbacks, N = E/R. In general, N varies as the simulation is executed on different number of nodes. (In particular, N is infinity when the number of nodes is one.) Define U = (2N+1)T and L = (N-1)T, then the estimated optimal state skipping number SS satisfies

squareroot(L) - 1 <= SS < squareroot(U).

Note that the results of [LL] are stated in terms of a checkpoint interval, which is one more than than the state skipping number. (I.e. a checkpoint interval of one -- saving state after every event--is the same as zero state skipping--skip saving zero states between two saved states).

Measuring R and E:

Events are not atomic in TWOS. An object can rollback while in the middle of an event. TWOS does not count the number of rollbacks. Thus we are stuck estimating both R and E. For E we used the number of events completed. For R we used the number of events rollback over which is events completed minus committed events. So N is (Committed Events + R)/R. The use of events rollback over for R assumes that the average rollback rolls over one completed event.

TWOS also counts the number of events started which is the number of times the function objhead is called. This function is called before any event is started, but the object can rollback before any object code is actually executed. There are more events started than events completed. So the average of one completed event per rollback has a chance of not being totally arbitrary..

However, TWOS does use rollback for things other than synchronization. In particular, memory allocation failures can cause both a rollback and a reverse message which will cause a rollback on the message's sending node. Perhaps this is why TWOS does not count rollbacks. In any case, our N is only a rough estimate which we hope at least has the correct per node behavior.

The Benchmarks

We used four benchmarks. Three were standard benchmarks from the JPL TWOS benchmark suite [R2]. Pucks [H et. al.] is a colliding pool balls simulation. STB88 [W et. al.] is a theater level combat simulation. Warpnet [P et. al.] is a computer network simulation. The fourth, "bank", is an artificial simulation designed to gauge the cost of state saving for the upcoming rollback chip. Both the size of a bank's state and the message density per object are easily varied with bank.

All executions were made on JPL's BBN butterfly, a GP-1000 with over 80 nodes. The execution times used TWOS 2.4.1 modified to support state skipping. The idle time measurements used TWOS 2.4.2 modified to estimate idle time. Our estimates of the Lin Lazowska estimates used data from the TWOS 2.5 benchmark run [R2] (without dynamic load management) and [R1] for state saving costs. For comparison with [PML], TWOS was run with lazy cancellation and what they call MVT scheduling. (In MVT scheduling a object which is recreating a state at time A to execute an event at time B is scheduled with respect to A and not B.)

Figure 1 shows idle time as a function of node for the three standard benchmarks. This is a conservative estimate of idle time. While idle in the sense of Figure 1, a node has nothing to do. The non-zero idle time for STB88 and Pucks at 4 nodes is due to memory problems. TWOS can stop executing objects when it runs out of memory, it will attempt to run again when a message arrives.

Figures 2 and 3 show our estimate of N and the estimates of Lin and Lazowska for Pucks, Stb88 and Warpnet. These are all driven by the number of events rollback over. For these three benchmarks, events rollback over is, at least to a first approximation, linear in the

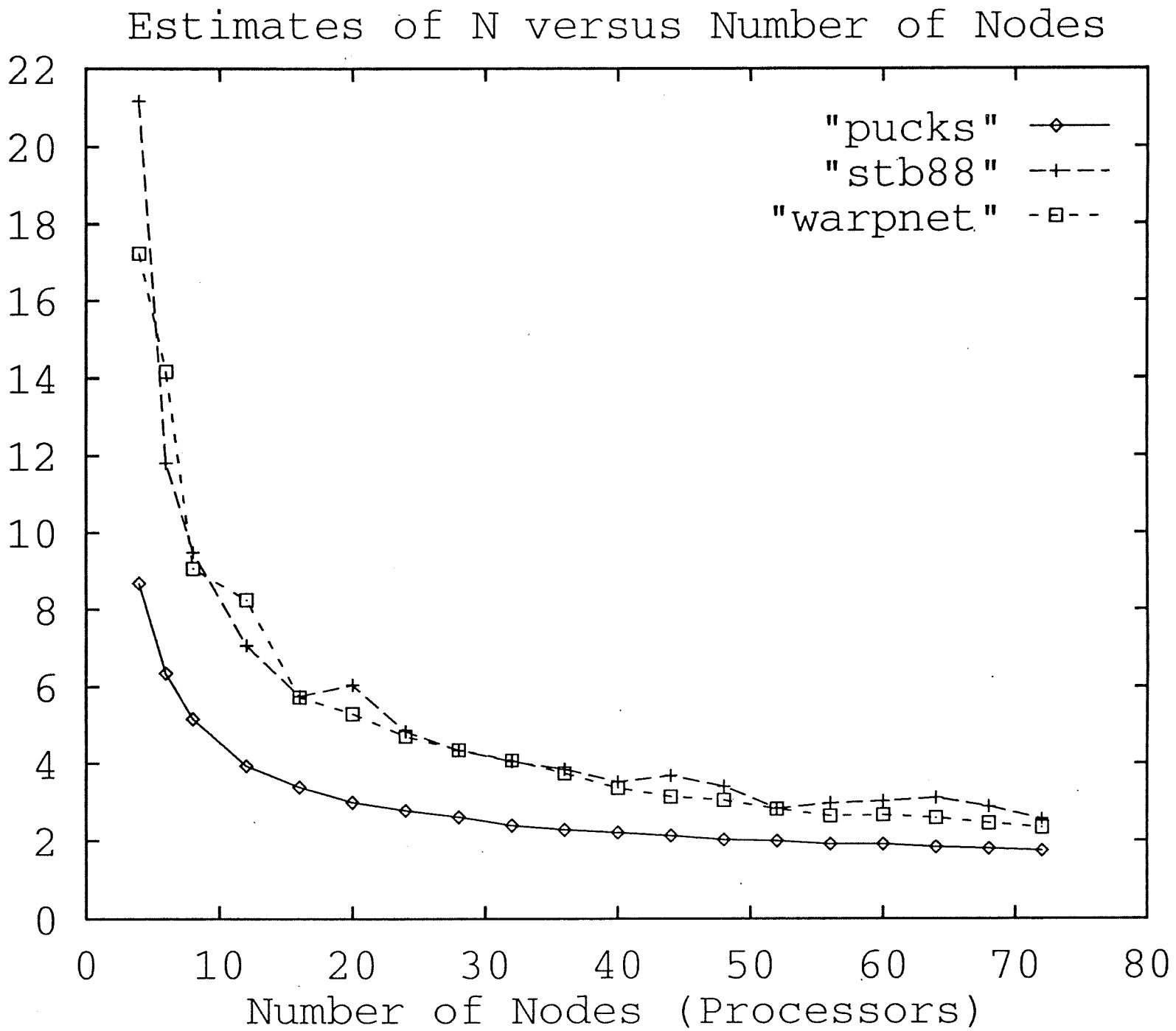
Estimates of N versus Number of Nodes
"pucks"
"stb88"
"warpnet"
22
20
18
16
14
12
10
8
6
4
2
0
0
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 2

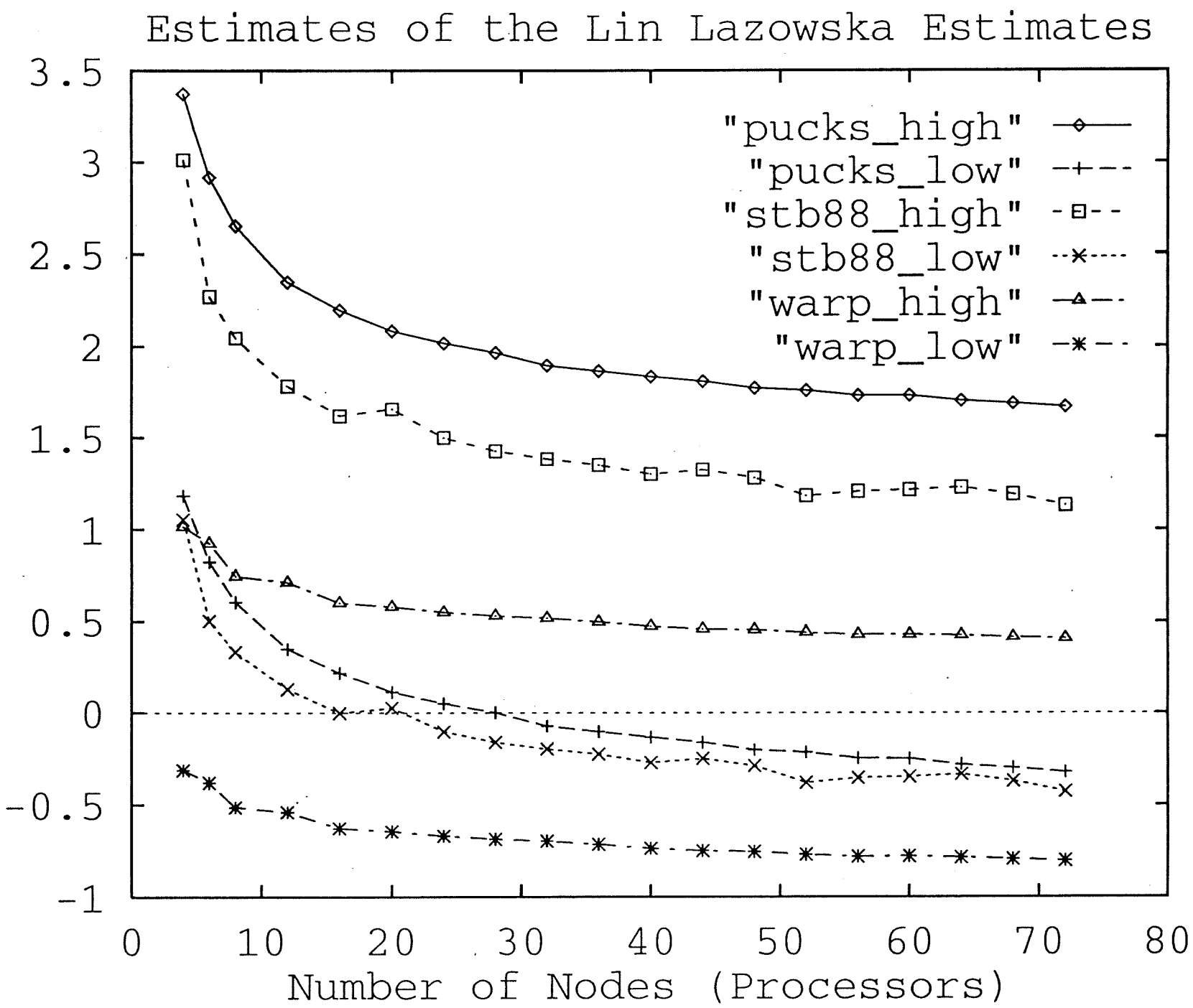
Estimates of the Lin Lazowska Estimates
"pucks_high"
"pucks_low"
"stb88_high"
"stb88_low"
"warp_high"
"warp_low"
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 3

number of nodes. We define the "rollback factor" to be the slope of this line divided by the number of committed events.

Pucks:

Pucks has 304 objects, 362824 committed events, 405802 committed event messages and has a speedup near 14. Of the three JPL benchmarks, Pucks has the smallest granularity with a G = 2.7 milliseconds. The size of an objects state ranges from over 4K to under 8K, making C between 1.3 and 2.0 milliseconds. For Figures 2 and 3 we used a middle value so that T is about 5/8, by far the largest of the three standard benchmarks. Also Pucks rollback factor was the largest, roughly 1.8% or each node added roughly an additional .018 * 362824 events rolled over.

Figure 4 shows the results for Pucks. Figure 4a shows the run times for state skipping of zero, one, two and three for 16 and more nodes and Figure 4b shows the same data in percent change from the zero state skipping case. Of the three JPL benchmarks, Pucks is the only one which state skipping speeds up the fastest case. Even here the improvement is less than one percent.

Not shown is the improvement on 12 or fewer nodes. On 12 nodes, state skipping of one wins with a 4.8% increase over the zero state skipping case. For 6 and 8 nodes the state skipping of two wins with 8.0% and 6.8% increases over the zero state skipping case. On 4 nodes state skipping of three is 121% faster than the zero state skipping case.

For Pucks state skipping is a win. A big win a small number of nodes and a modest winner at the high number of nodes. All of the best state skipping numbers are within the ranges given by Figure 3.

STB88:

STB88 has 380 objects, 389382 committed events, 603472 committed event messages and has a speedup above 26. The granularity of STB88 is in the middle with a G = 8.1 milliseconds. The size of an objects state has a wilder range than Pucks, making C between 0.9 and 2.5 milliseconds. For Figures 2 and 3 we used a middle value so that T is about 1/5, the middle of the three standard benchmarks. However STB88's rollback factor was in the smallest, roughly 0.8% or each node added roughly an additional .008 * 389382 events rolled over. Figure 5 shows a "best linear fit" of STB88's events rollback over versus number of nodes.

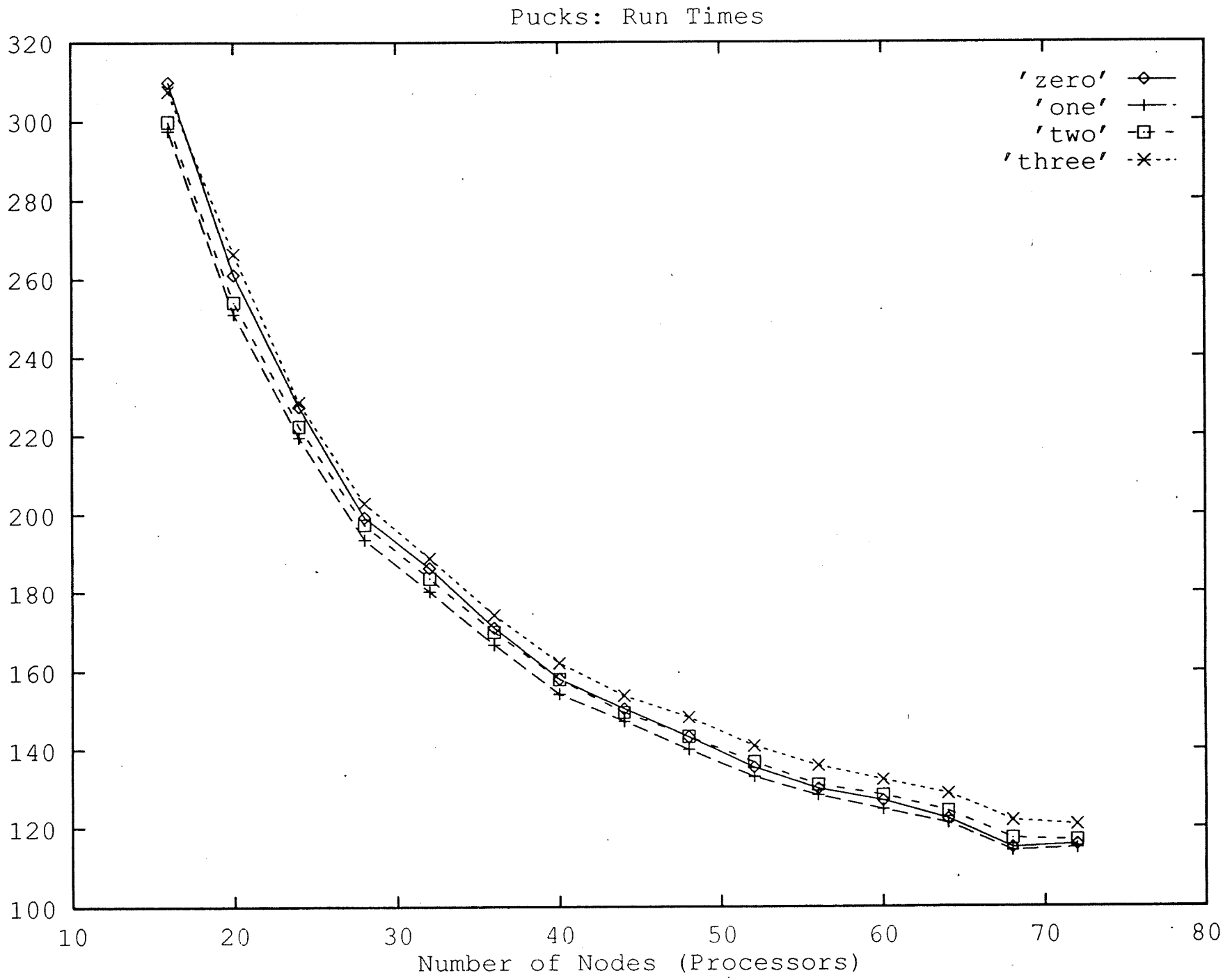
Pucks: Run Times
320
300
280
260
240
220
200
180
160
140
120
100
10
20
30
40
50
60
70
80
Number of Nodes (Processors)
'zero'
'one'
'two'
'three'

Figure 1a

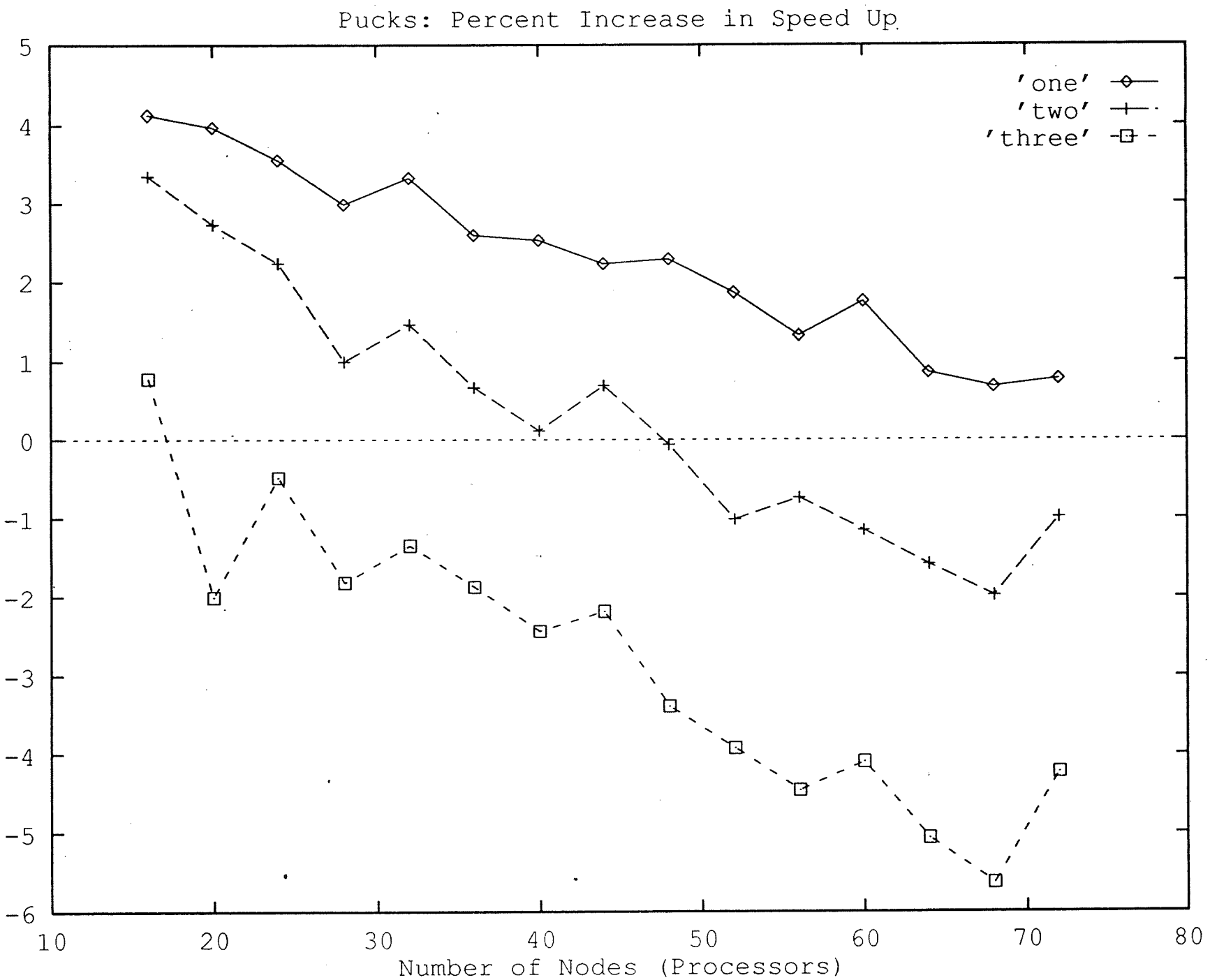
Pucks: Percent Increase in Speed Up
'one'
'two'
'three'
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 4b

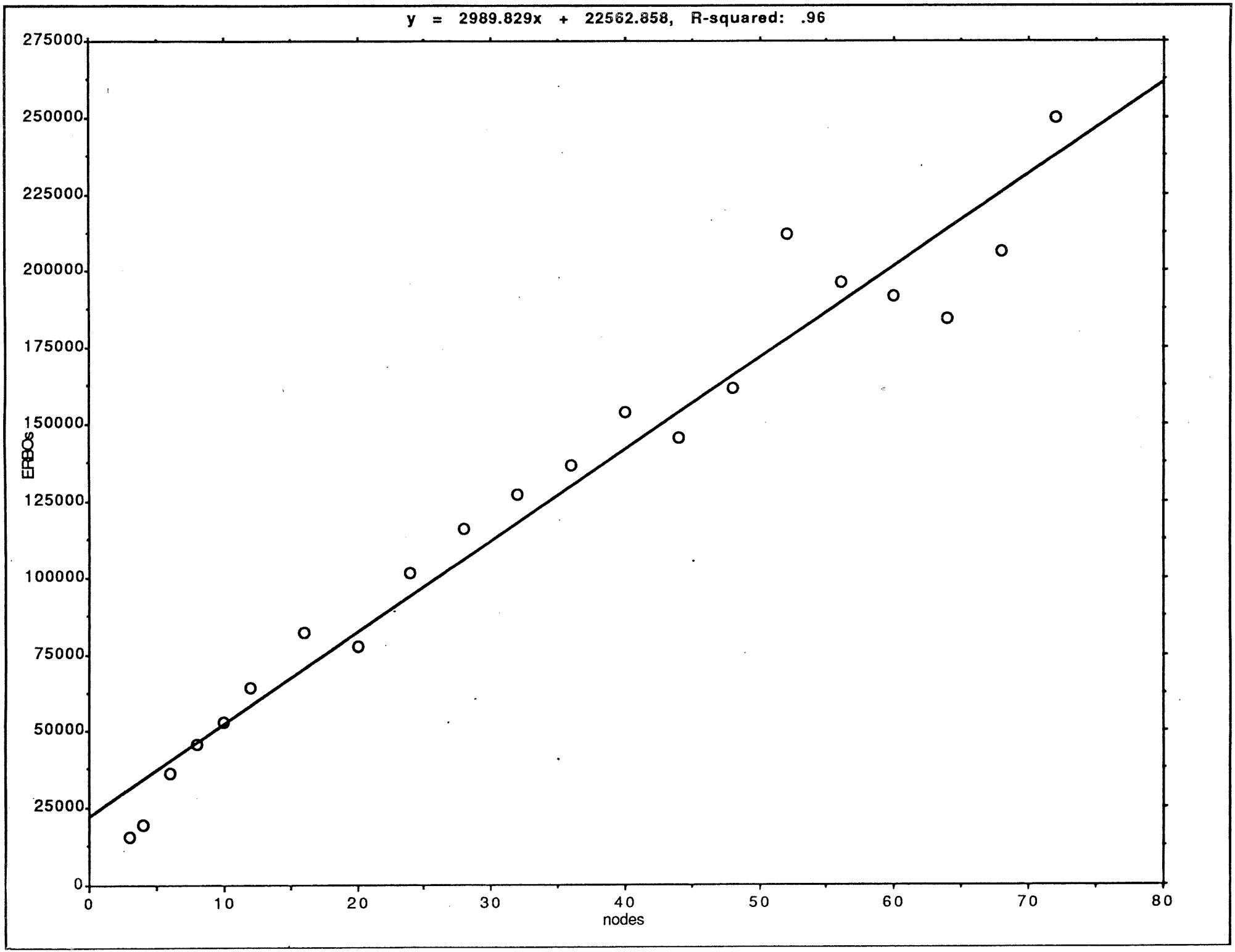
y = 2989.829x + 22562.858, R-squared: .96
275000
250000
225000
200000
175000
150000
125000
100000
75000
50000
25000
0
ERBOs
0
10
20
30
40
50
60
70
80
nodes

Figure 6 shows the results for STB88. Figure 6a shows the run time data while Figure 6b shows the same data in percent change from the zero state skipping case. For STB88 the zero state skipping case is the speed winner. Only at 8 nodes does state skipping of one beat zero state skipping by more than one percent. Again all of the best state skipping numbers are within the ranges given by Figure 3.

Warpnet:

Warpnet has 169 objects, 35027 committed events and 45319 committed event messages and has a speedup of almost 30. The granularity of Warpnet is the biggest with, G = 51.5 milliseconds. Except for some initialization objects all Warpnet objects have the same state size with C = 1.5 milliseconds. This makes Warpnet's T about 0.03, by far the smallest of the three standard benchmarks. Warpnet's rollback factor was roughly 1% which puts it in the middle.

Figure 7 shows the results for Warpnet. Figure 7a shows the run time data while Figure 7b shows the same data in percent change from the zero state skipping case. From 12 nodes out, the zero state skipping case is the fastest. At 6 and 8 nodes state skipping of one is fastest but by less than one percent. At four nodes the state skipping of two is the fastest, but only by slightly more than one percent. All of the best state skipping numbers are within the ranges given by Figure 3 with the exception of the 4, 6 and 8 node cases. Each of these has the optimal state skipping number one higher than the upper estimate.

Bank:

Each of the 64 bank object reacts to a message by generating a message with a random receiver and random exponential arrival time. Thus bank is much like a fully connected server network with zero service time. The execution time of each event is very small. (In TWOS, an object sending a message has at least four context switches included in its timing.) The granularity of bank is G = 0.8 milliseconds. The size of the state is varied by including a large integer array whose length is a compile time constant. We ran experiments with a state size of roughly 1 KB (C = 0.5 milliseconds, which yields a T = 5/8, the same as Pucks) and with roughly 12 KB (C = 3.3 milliseconds, which yields a much larger T of about 4). The (per object) message density was 5 for both state sizes. There were 31747 committed events and 31987 committed messages for the message

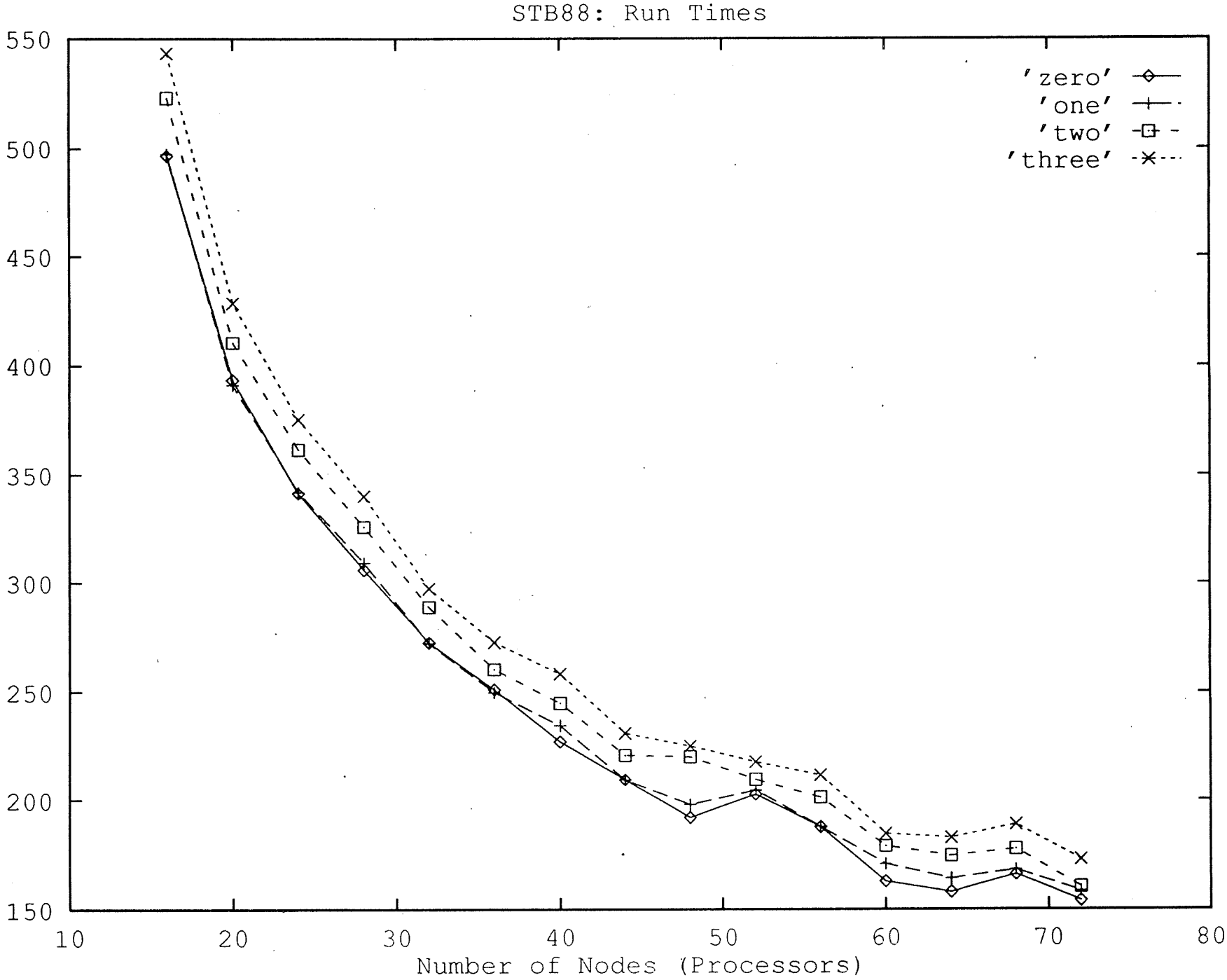
STB88: Run Times
550
500
450
400
350
300
250
200
150
10
20
30
40
50
60
70
80
Number of Nodes (Processors)
'zero'
'one'
'two'
'three'

Figure 6d

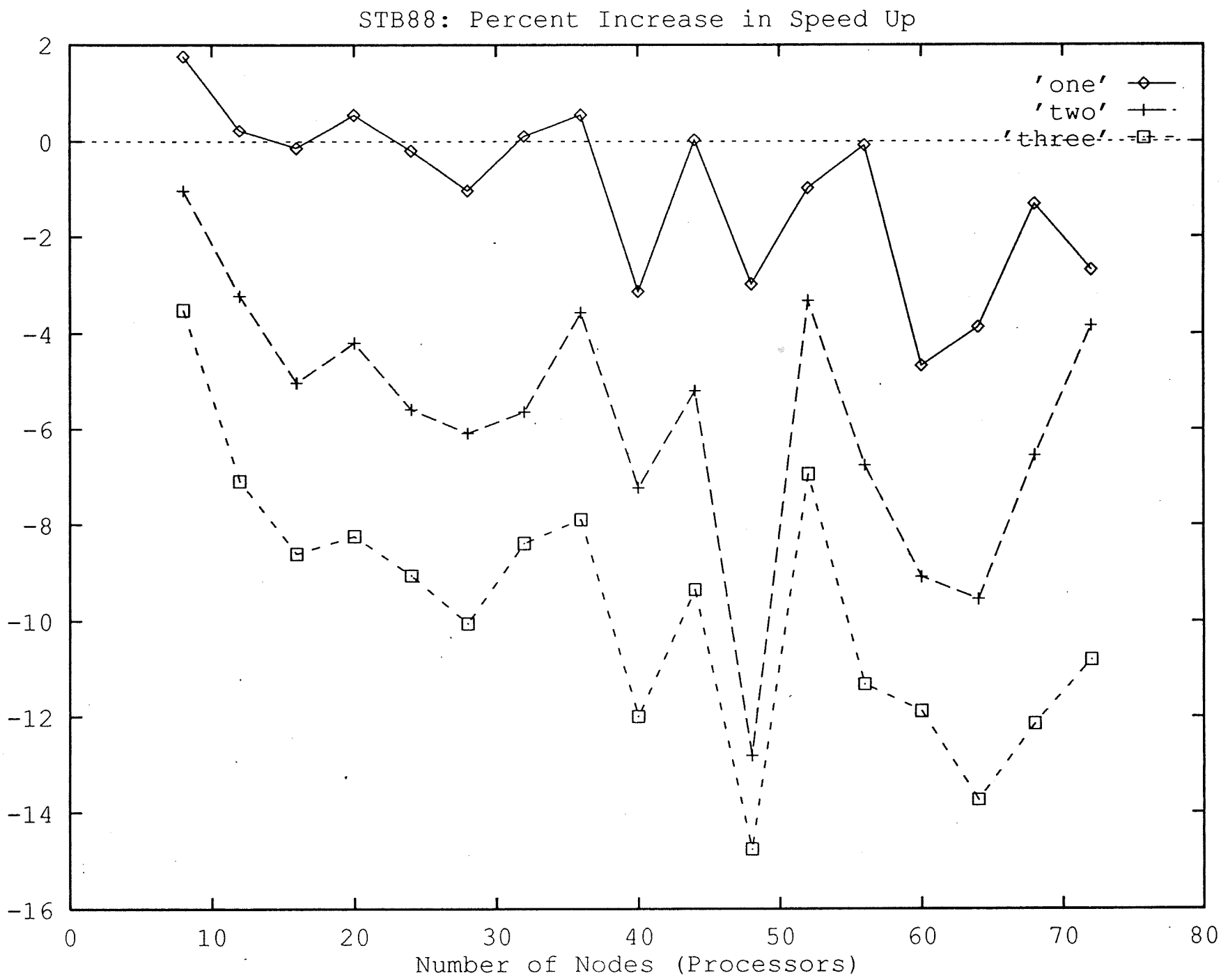
STB88: Percent Increase in Speed Up
'one'
'two'
'three'
2
0
-2
-4
-6
-8
-10
-12
-14
-16
0
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 6b

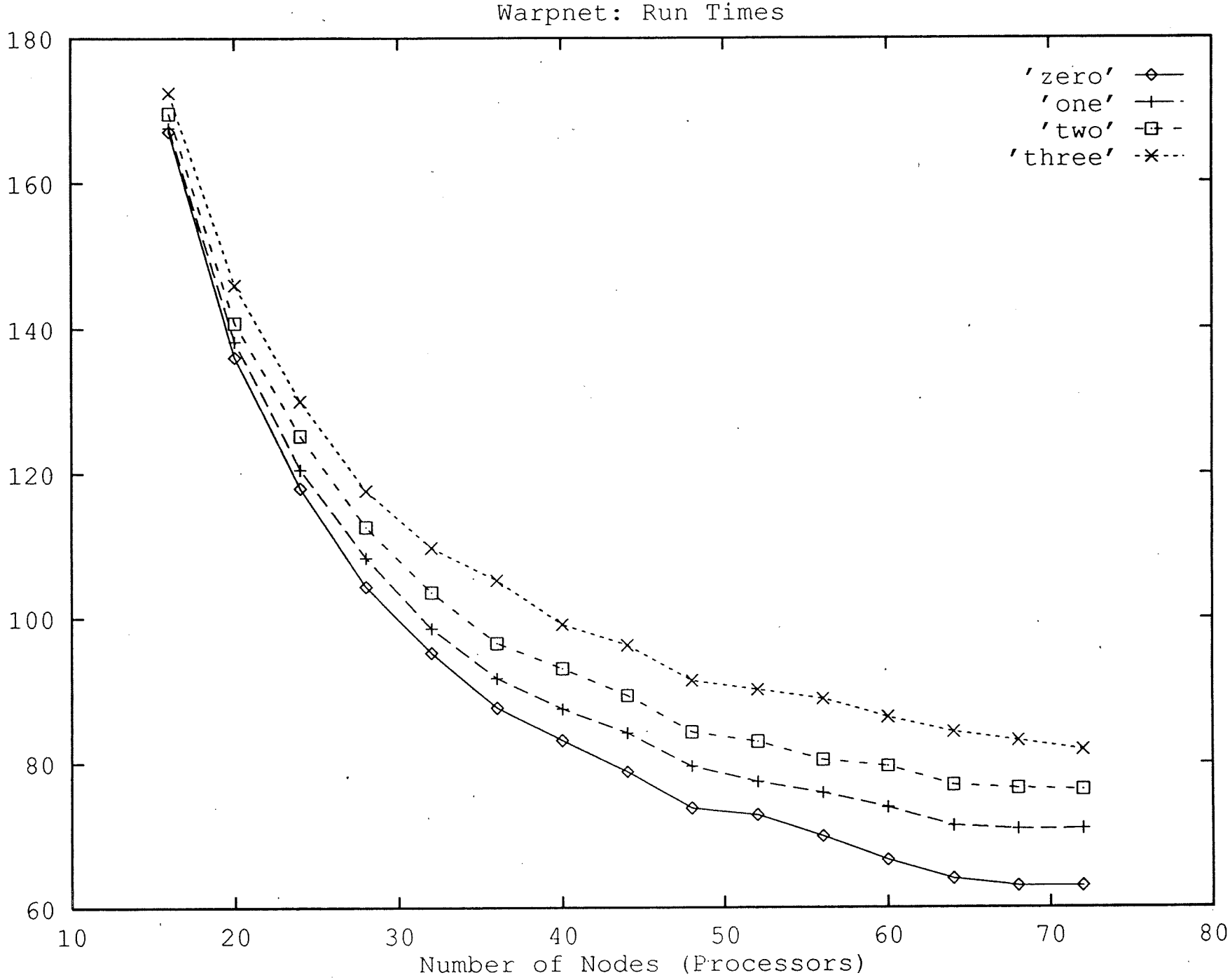
Warpnet: Run Times
180
160
140
120
100
80
60
10
20
30
40
50
60
70
80
Number of Nodes (Processors)
'zero'
'one'
'two'
'three'

Figure 7a

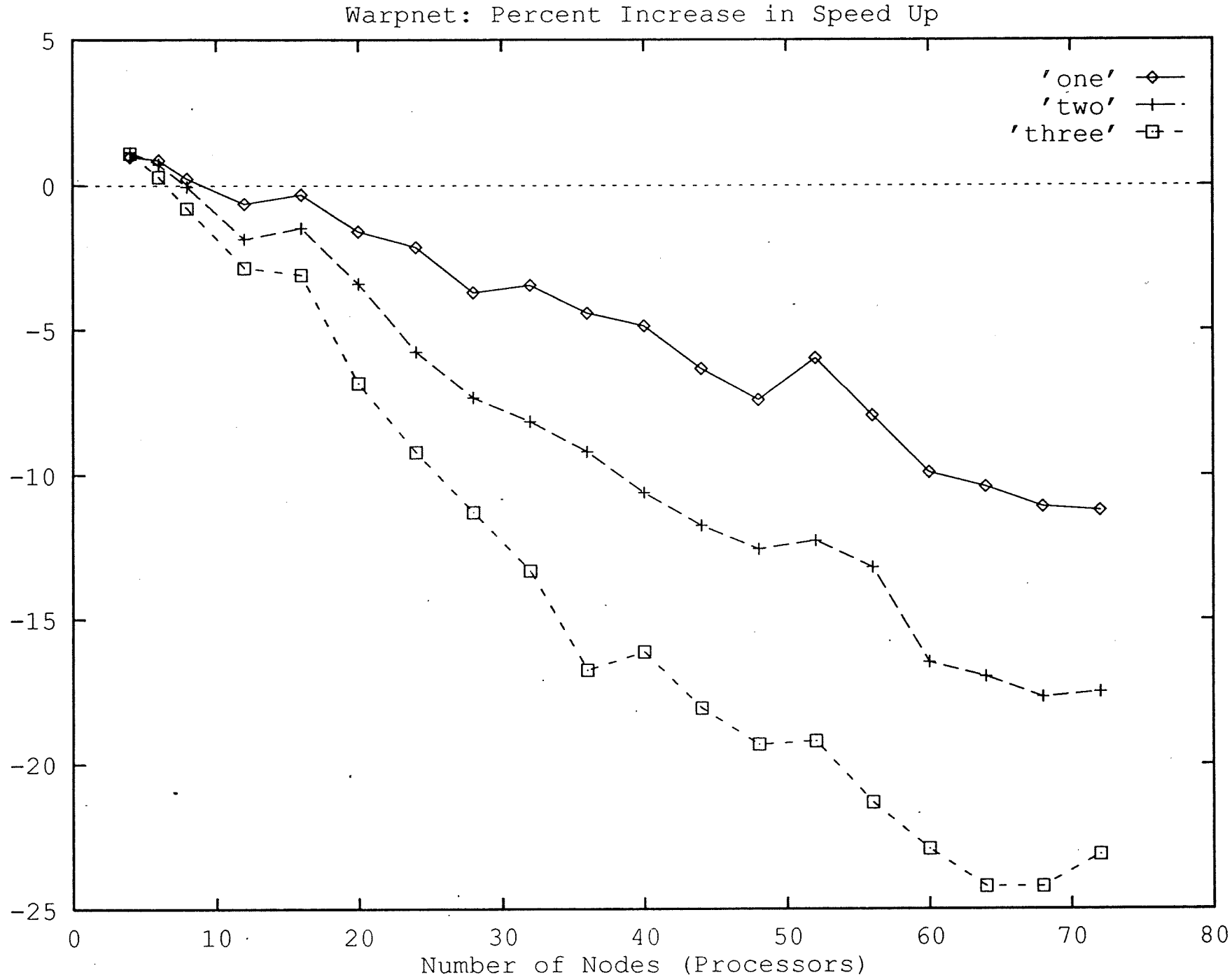
Warpnet: Percent Increase in Speed Up
'one'
'two'
'three'
5
0
-5
-10
-15
-20
-25
0
10
20
30
40
50
60
70
80
Number of Nodes (Processors)

Figure 7b

density 5 simulations. Latter we run with a (per object) message density of 50 in the 12 KB case (317954 committed events and 320354 committed messages).

Figure 8a shows the plot of N for the 1K, 5 message density bank and Figure 8b shows our estimates of the Lin and Lazowska estimates. Figure 9a, 9b and 9c show the results for bank in terms of percent increase in speed up. Runs with state skipping of zero, one, two, four, eight and sixteen are show. Both runs with the 12K state had memory problems in the zero state skipping case. Thus the curves for N in the 12K cases measure something different than what Lin and Lazowska had in mind. (As a first approximation, one could use Figure 8a for the 12K, 5 message density case and multiply the estimates in Figure 8b by about 2.5.)

Figures 9a shows that on one node, higher state skipping isn't always faster (see above). Figure 9c shows a similar behavior at small number of nodes. Here state skipping of 16 is fastest at 6 nodes, but on 4 nodes state skipping of 16 runs out of memory. The state skipping of eight show a similar reversal. The 12K, 50 message density case was run at 64 nodes, the message density was high enough so that the state skipping of four was 19% faster than the zero state skipping case.

At least in the 1K state size, 5 message density case the Figure 8b estimates are slightly lower than best state skipping numbers observed in Figure 9a.

Conclusions:

State skipping can be an effective method of reducing the overhead due to state saving. Our experiments show that the effect of state saving generally decreases as the number of processors increase. It is shown that the effects of state skipping can range from a big win in the saturated case to a lost in the unsaturated case. Our estimates of the Lin and Lazowska estimates on the optimal state skipping number are reasonable fits to the experimental data.

Historical Note

The original design of TWOS [J et. al.] called for periodic state saving. Like state skipping, periodic state saving does not always save a copy of an object's state after every event. Periodic state saving's decision to save a state was based on real CPU execution time. A copy of the state was saved if at least "save period" amount

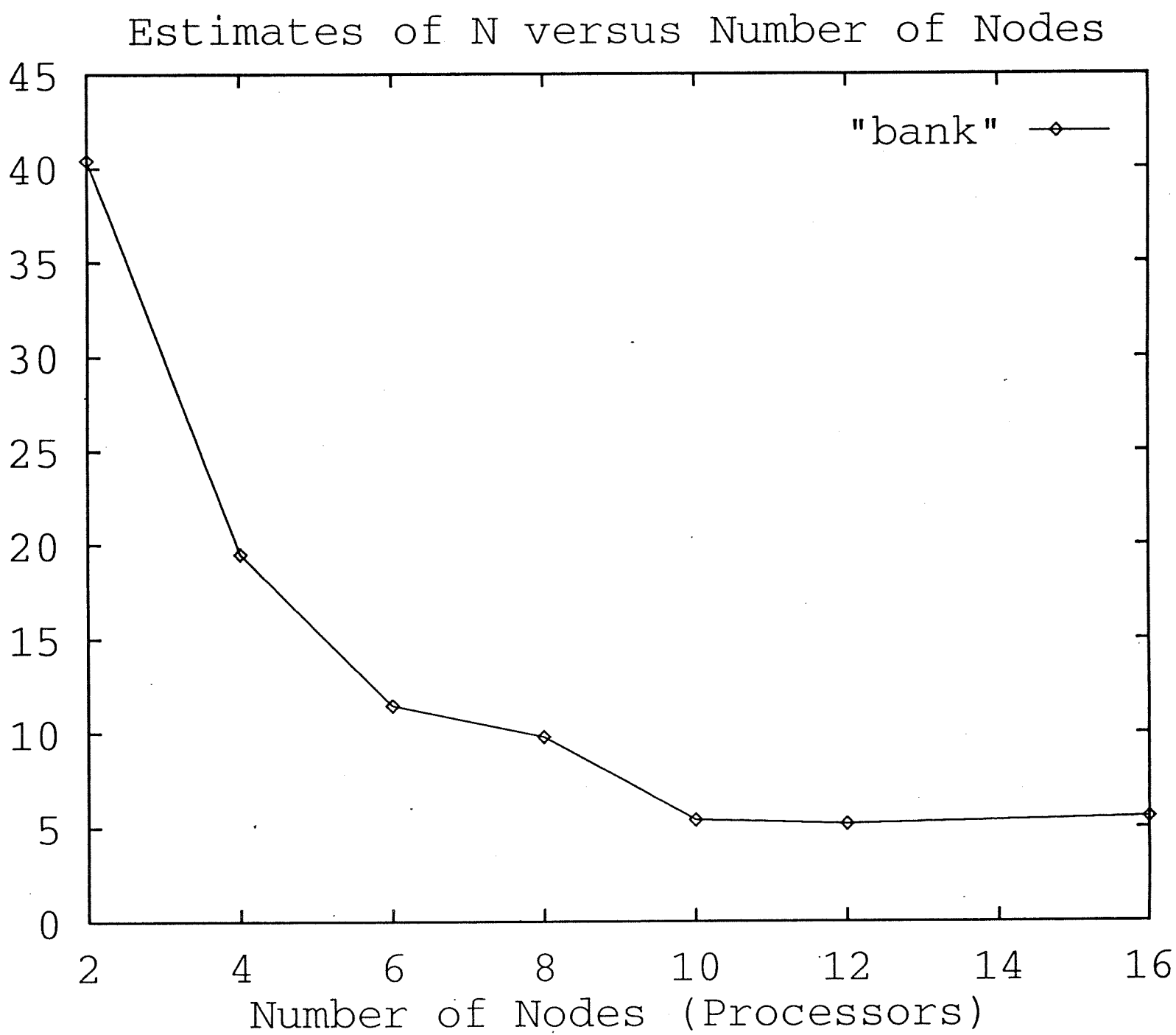
Estimates of N versus Number of Nodes
"bank"
45
40
35
30
25
20
15
10
5
0
2
4
6
8
10
12
14
16
Number of Nodes (Processors)

Figure 8a

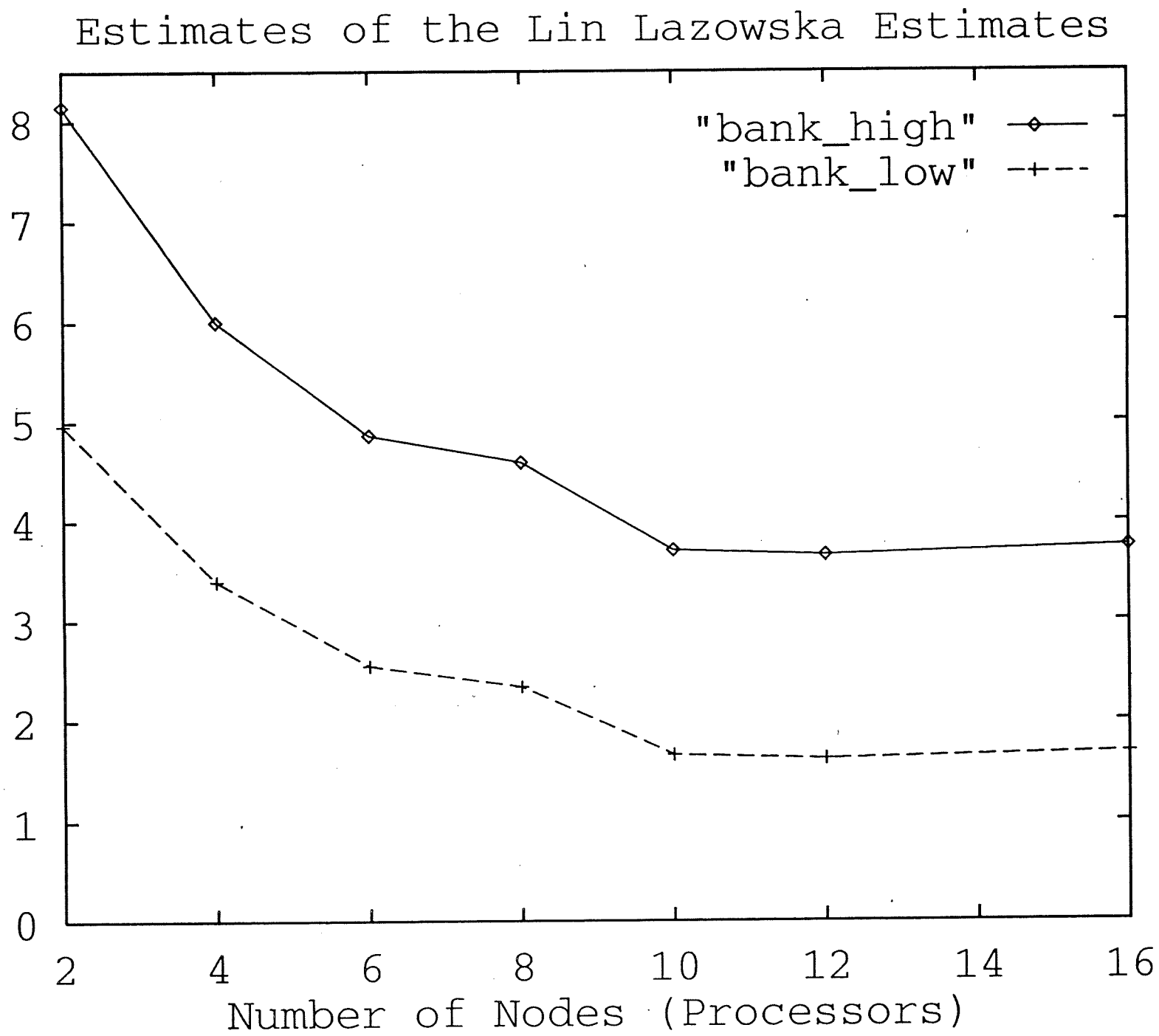
Estimates of the Lin Lazowska Estimates
"bank_high"
"bank_low"
0
1
2
3
4
5
6
7
8
2
4
6
8
10
12
14
16
Number of Nodes (Processors)

Figure 8b

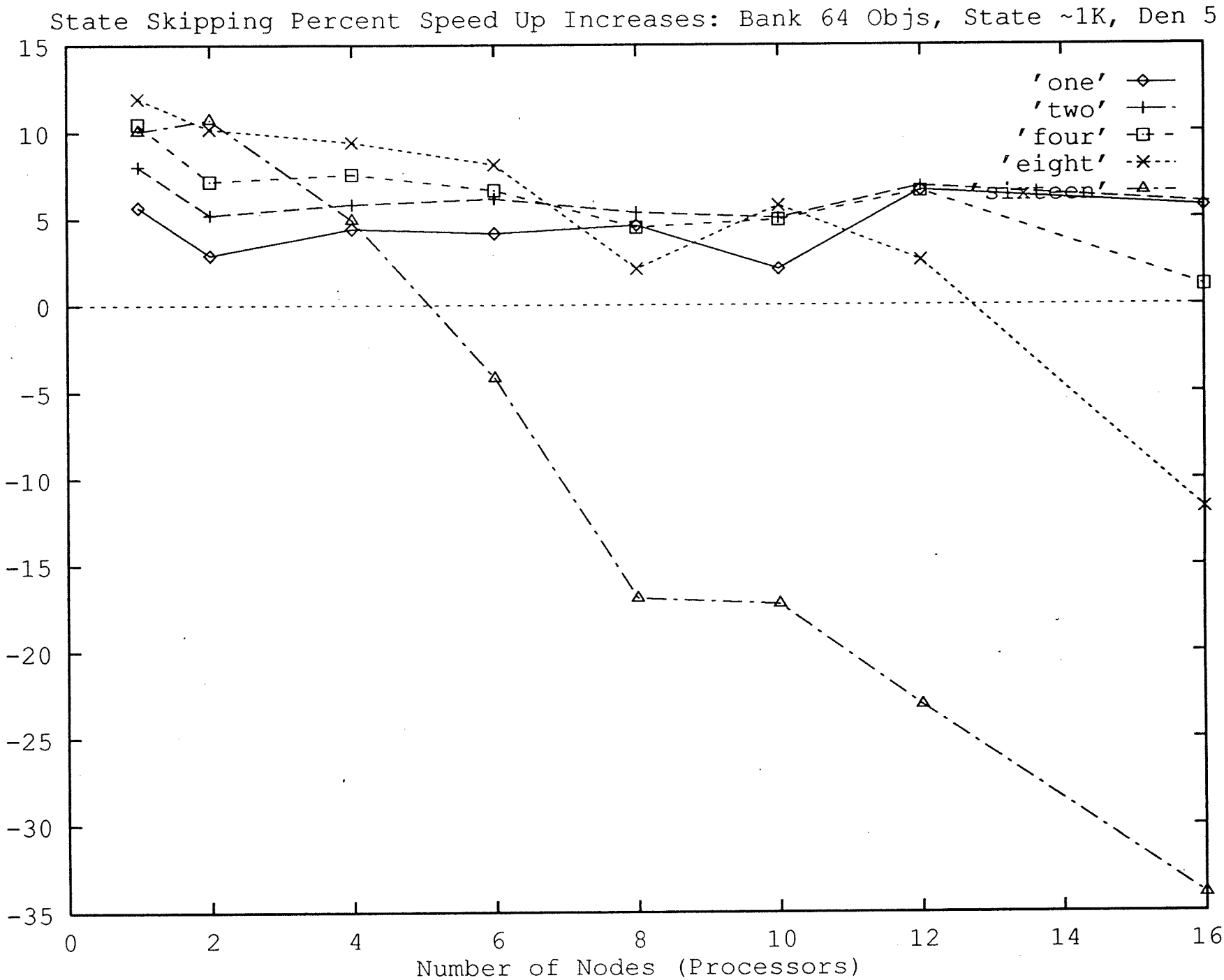
State Skipping Percent Speed Up Increases: Bank 64 Objs, State ~1K, Den 5
'one'
'two'
'four'
'eight'
'sixteen'
15
10
5
0
-5
-10
-15
-20
-25
-30
-35
0
2
4
6
8
10
12
14
16
Number of Nodes (Processors)

Figure 9a

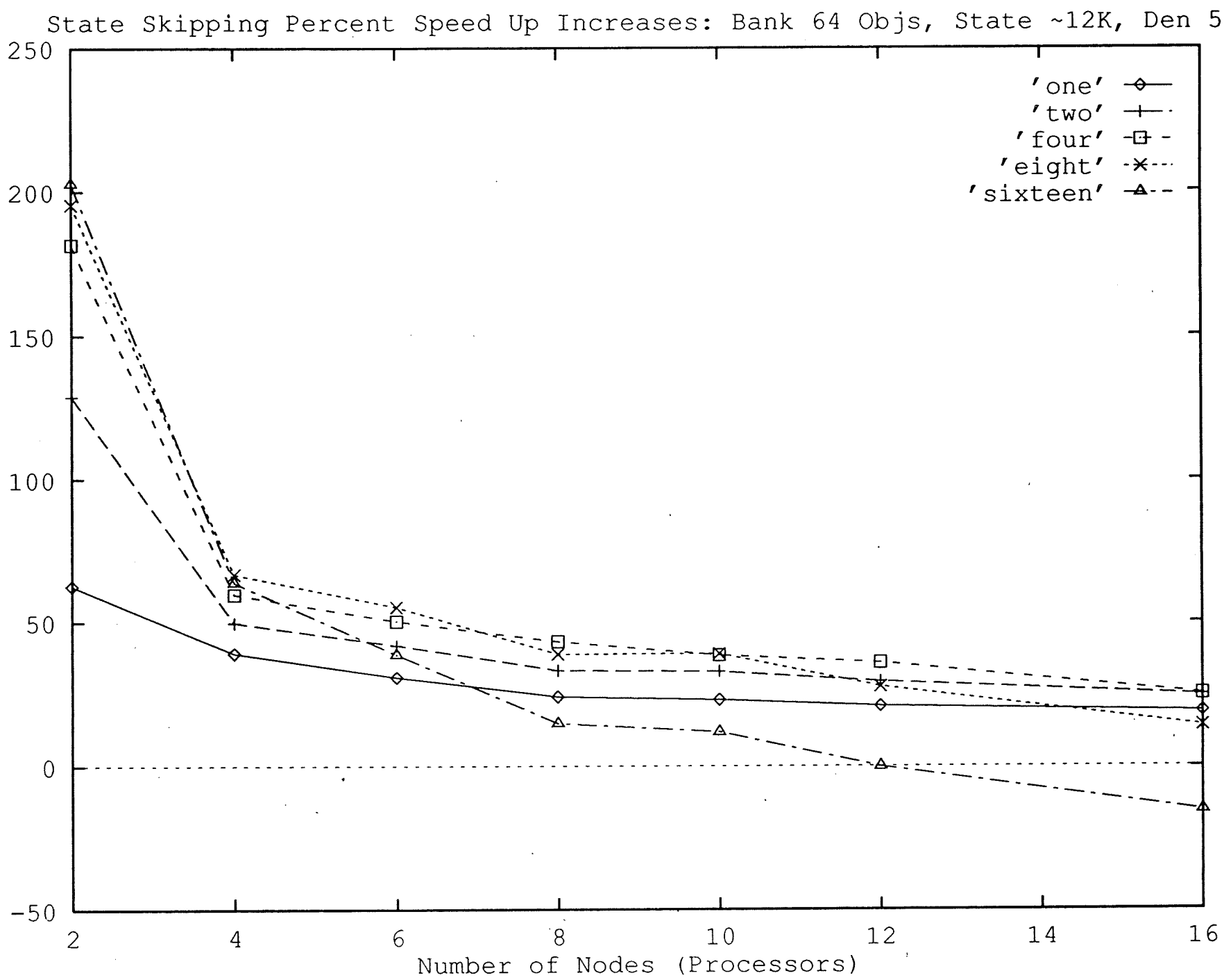
State Skipping Percent Speed Up Increases: Bank 64 Objs, State ~12K, Den 5
250
200
150
100
50
0
-50
2
4
6
8
10
12
14
16
Number of Nodes (Processors)
'one'
'two'
'four'
'eight'
'sixteen'

Figure 9L

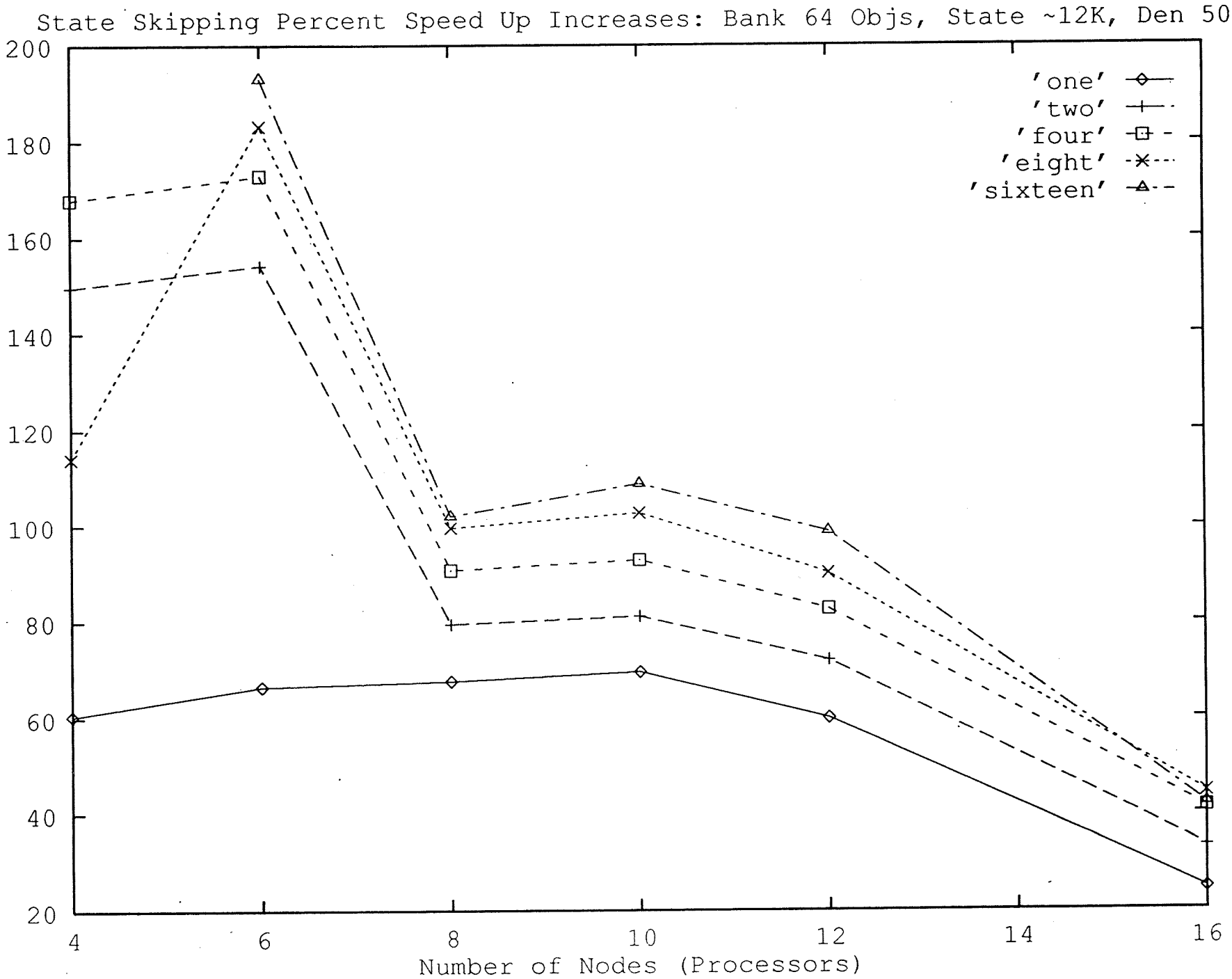
State Skipping Percent Speed Up Increases: Bank 64 Objs, State ~12K, Den 50
200
180
160
140
120
100
80
60
40
20
4
6
8
10
12
14
16
Number of Nodes (Processors)
'one'
'two'
'four'
'eight'
'sixteen'

Figure 1C

of CPU time was used by the object since the last time the object saved a copy of its state.

Periodic state saving was implemented in the first TWOS. Early tests indicated that always saving state (i. e. using a save period of zero) was much faster. The rational for including periodic state saving was more for decreasing memory usage than for increasing speed. The original target machine, the JPL mark II hypercube, had only 256 KB of memory per processor.

Eventually periodic state saving was removed from TWOS. No value of save period was found which would improve a maximum speed up. It any case there was enough idle time on 32 processor Mark III hypercube runs that briefly TWOS did garbage collection of fossils only during idle periods. Although this speeded up 32 processor run times, it was deleted because there was not enough idle time for the 8 processor runs.

References

[BRF] C.A. Buzzell, M.J.Robb and R. M. Fujimoto, " Modular VME Rollback Hardware for Time Warp," Distributed Simulation, SCS Simulation series 22 (1990), 153-156.

[FK] R.E. Felderman and L. Kleinrock, "Two Processor Time Warp Analysis: Some Results on a Unifying Approach," Advances in Parallel and Distributed Simulation, SCS Simulation series 23 (1991), 3-10.

[FTG] R. M. Fujimoto, J.-J. Tsai and G. Gopalakrishnan, "The Rollback Chip: Hardware Support for Distributed Simulations using Time Warp," Distributed Simulation, SCS Simulation Series 19 (1988) 81-86.

[J] D. Jefferson, "Virtual Time," ACM Transactions on Programing Languages and Systems, 7 (1985) 404-425.

[J et. al.] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M Di Loreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J Wedel, H. Younger, and S. Bellenot, "Distributed Simulation and the Time Warp Operating System," Proc. 12th SIGOPS - Symposium of Operating Systems Principles, 1987, 77-93.

[H et. al.] P. Hontalas, B. Beckman M. Di Loreto, L. Blume, P. Reiher, K. Sturdevant, V. Warren, J. Wedel, F. Wieland, D. Jefferson, "Performance of the colliding pucks simulation on the time warp

operating systems (Part 1: Asynchronous behavior and sectoring)," Distributed Simulation, SCS Simulation Series 21 (1989), 3-7.

[LL] Y.-B. Lin and E. D. Lazowska, "Reducing the State Saving Overhead for Time Warp Parallel Simulation," Technical Report 90-02-03, Dept. of Comp. Sci. and Eng., University of Washington, Seattle, Washington, February 1990.

[PML] B. R. Preiss, I. D. MacIntyre and W. M. Loucks, "On the Trade-off between Time and Space in Optimistic Parallel Discrete Event Simulations," preprint 1990.

[P et. al.] M. Presley, M. Ebling, F. Wieland and D. Jefferson, "Benchmarking the Time Warp operating system with a computer network simulation," Distributed Simulation, SCS simulation series 21 (1989), 8-13.

[R1] P. Reiher, "Time Warp State Saving Times," JPL Interoffice Memo 363-88-32, Oct 10, 1988.

[R2] P. Reiher, "Time Warp 2.5 Benchmark Test Results," JPL Interoffice Memo 366-91-14, Jun 21, 1991.

[W et. al.] F. Wieland, L. Hawley, A. Feinberg, M. Di Loreto, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. Jefferson, "Distributed combat simulation and time warp: The model and its performance," Distributed Simulation 21 (1989) 14-20.