1990

Application of the E/T Performance Modeling Methodology to a Computation on a 128 Processor NCUBE

Dan C. Marinescu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
90-998

APPLICATION OF THE E/T PERFORMANCE MODELING METHODOLOGY TO COMPUTATION ON A 128 PROCESSOR NCUBE

Dan C. Marinescu
John R. Rice

CSD-TR-998
July 1990

Application of the E/T Performance Modeling Methodology to a Computation on a 128 Processor NCUBE

Dan C. Marinescu
and
John R. Rice

Computer Sciences Department
Purdue University
Technical Report CSD-TR-998
July, 1990

1. **BACKGROUND.** The E/T model involves the use of *Event Traces* on *Threads of Control* to study and diagnose the performance of parallel computations. This model is described in detail in [Marinescu, Rice and Vavalis, 1990]. Roughly speaking, a thread of control is a process or the program running on a particular processor. An event is typically an interruption of the computation in the thread of control for communication purposes (a read or write) or awaiting data needed for the computations to proceed. The E/T model data is relatively straightforeward to obtain and its general behavior can indicate important characteristics of the behavior of the computation.

2. **THE APPLICATION.** A linear system of equations is created by a PDE solving system with the equations distributed in the nodes of a 128 processor NCUBE 1. Subcubes of various dimensions can be used. The distribution of the equations is highly, but not perfectly, uniform. An iteration method, called JACOBI SI, is used to solve the linear system iteratively. The basic idea is that each processor improves the values for its equations then sends a few of its new values (the boundary or coupling values) to other processors and receives a few new values from other processors. This procedure is iterated a large number of times. The exchange of values is highly "local" in nature so that if the sets of equations are appropriately assigned the processors, then the exchanges takes place between groups of a few near neighbors in the hypercube.

   This computation should be very well suited for parallel computers and yet the performance observed is poor. The E/T methodology was applied in order to determine why and to suggest possible remedies.

3. **THE RESULTS.** Our first result explained the source of the inefficiency observed. One of the simplest properties of the E/T model is the relationship between the number of events and the number of threads of control. Our analysis, reported in detail in [Marinescu, Rice, and Vavalis 1990], showed that this relationship is quadratic, the number of events grows with the square of the number of

threads of control (processors). Such a relationship indicates that either the computation is not well suited for highly parallel machines or that it has been implemented improperly. The latter is the case here, some communication utilities of the NCUBE do not perform as expected or advertized. The remedy is to either reimplement these utilities better or to reprogram the application so that the communication is explicitly controlled by the program, relying only on the system to send messages from one NCUBE processor to a nearest neighbor on the hypercube. Since the NCUBE 2 has a new communication system, we are waiting to measure the performance in this new environment.

Our second result, which is the main subject of this report, is that there is a serious problem with algorithmic synchronization delays in this computation. It is noteworthy that this effect is clearly revealed by the E/T methodology even though it is an order of magnitude smaller in effect than the poorly implemented communication primitives. This inefficiency will not be alleviated by the new communication system in the NCUBE 2. It is directly affected by the ratio of communication to computation speeds, but these are not likely to improve soon as the evolution of CPU's seems to be faster than that of communication systems. Other (partial) remedies that might be considered are: (1) Use less synchronization in the iteration. The numerical consequences of this are hard to analyze. (2) Use a non-uniform distribution of the equations to keep as many processors computing productively during the synchronization. This complicates the algorithm substantially and gives only a limited improvement. (3) Use fewer processors so that the synchronization cost is truly small (negligible) compared to the computation cost.

4. **THE PERFORMANCE DATA.** Among the data available from the E/T model data is the size (length of time) of the *blocking read events*. These events are where the thread of control is idle waiting on data to arrive. Their sizes do not include the times to actually process the information in input buffers, etc. We study these data for a particular computation using 128 processors with 1089 equations, about 9 equations per processor. Figures 1 through 7 are composite plots for groups of processors of the blocking time versus the event number. Thus, for each processor (thread of control) we have recorded the blocked time measured in clock ticks on the NCUBE 1 (one tick = 0.167 milliseconds). The processors are grouped as follows:

Figure	1	2	3	4	5	6	7
Processor IDs	0,1	2,3	4-7	8-15	16-31	32-63	64-127
Vertical scale	150	400	900	1000	1200	1500	1500

There is an artificial smoothness to these plots because the discrete data has been fit with a cubic spline interpolant in order to create an actual curve. The "negative" blocking times of these curves are due to the interpolants, and are not in the data.

If one examines the data of blocking times for a particular processor, one sees a narrow bell shaped distribution of small values plus one or two much smaller outlying distributions of much larger values. It is only after these times are plotted as a function of event number and then grouped properly that one sees the striking similarities that exist. Thus, in Figure 7 when we plot 64 curves together they are almost as one. Similar results occur when one plots the 32 curves from processors 32 to 63 (Figure 6), the 16 curves from processors 16 to 31 (Figure 5), and so on until the 2 curves from processors 2 and 3 (Figure 2). Such coincidences cannot be accidental and we believe they arise from the nature of the synchronization mechanism and the fact that the computations are almost perfectly uniform even though the communication is quite nonuniform. Note all these processors in the groups described above are at the same level in the broadcast – collapse tree. In the framework of the E/T methodology we need to analyze only one typical thread from each group.

Figure 9 shows a sampling of data taken from a problem with 2500 equations run using 64 processors, about 40 equations per processor. This sampling appears similar in behavior to the data analyzed here in more detail and further study has confirmed this similarity.

Figure 1. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 0 and 1.

Figure 2. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 2 and 3.

Figure 3. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 4 to 7.

Figure 4. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 8 to 15.

Figure 5. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 16 to 31.

Figure 6. Plot of blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 32 to 63.

Figure 7. Plot of Blocking time in ticks (0.167 milliseconds) versus event number for processors (threads of control) 64 to 127.

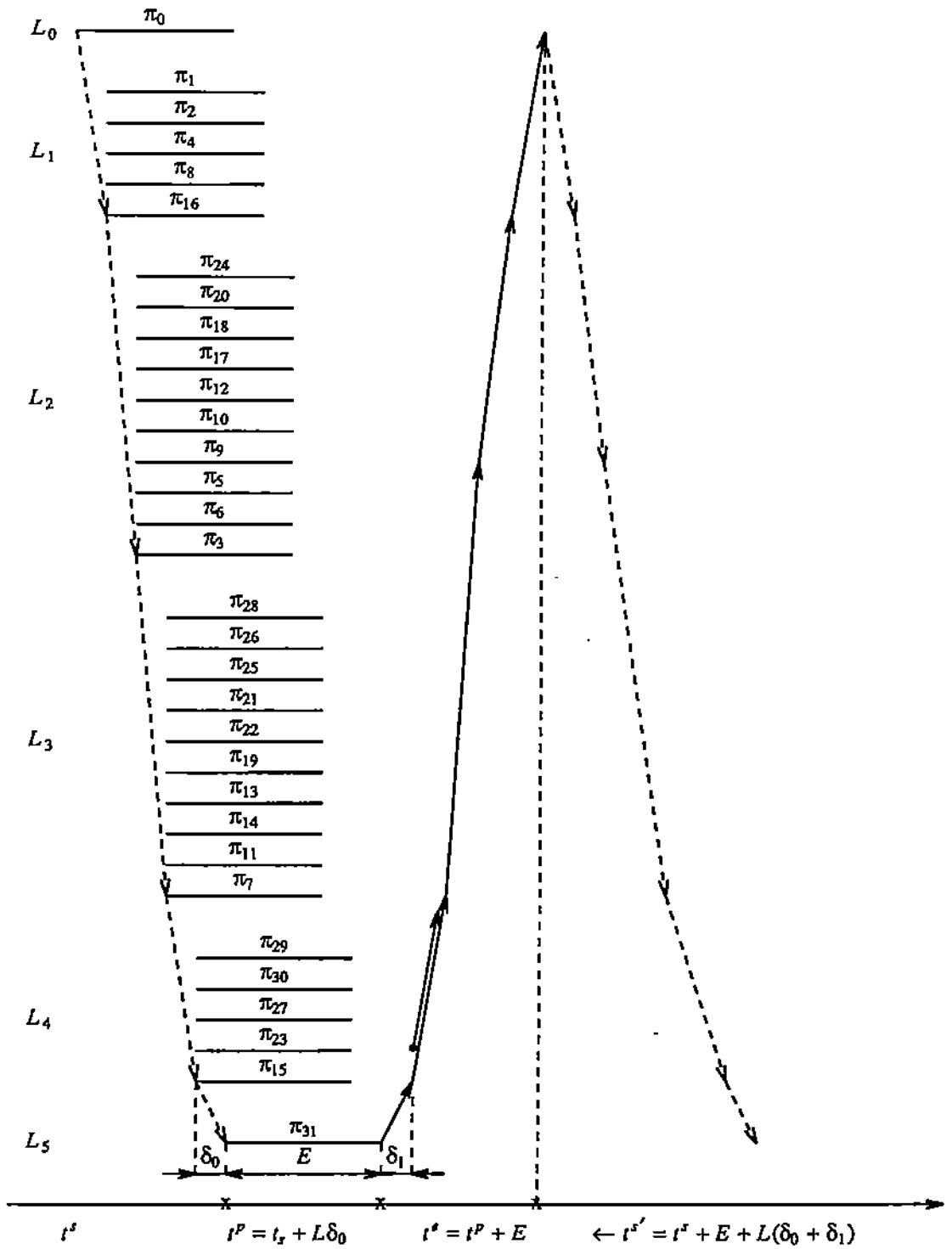


Figure 8. Schematic to illustrate explanation of the performance data of Figures 1 to 7. The levels refer to the groups of Figures 1 to 7 (only 6 levels are shown here). The shaded area represents the computations of the iteration. The global synchronization delay is the width (about 1500 ticks or 250 milliseconds) of the base of the gap at the right. The schematic is not exactly to scale.

Figure 9. Plots of blocking time in ticks (0.167 milliseconds) versus event number for selected processor groups (threads of control) from a second data set.

5. **EXPLANATION OF OBSERVED DATA.** It is important to realize that these data indicate very poor efficiency because the computation of the iteration are about 15 ticks here or 1% of the blocking times seen.

   1. *Initialization of an iteration.* Processor 0 starts the computation by sending a message to Processor 1 and then to Processor 2 telling them to start. Processor 1 sends a message to Processor 3 telling it to start, thus Processors 2 and 3 start at almost the same time. This continues until all 128 processors have been started. This cascading of start times is indicated by the stairsteps on the left of Figure 8 showing that different groups of processors become active ''together''.

      Note that the similarities in Figures 1 to 7 occur only because the horizontal variable is event number. One does not see such similarities if time is used as the horizontal variable.

      A lower bound on the time for initialization is $2s*\ln P$ where s is the time to send one word to a nearest neighbor, P is the total number of processors ($\ln P$ is the cube dimension) and the 2 comes from the fact that every processor must send two start up messages (except for the bottom group). For one 128 processor case, this lower bound is perhaps 20 msecs or 120 ticks.

   2. *Synchronization between iterations.* When a processor finishes an iteration, it sends a message to processor 0 saying it is finished. The group of processors 64-127 finish last since they start so late. These 64 messages reverse the stair step of processor levels as they go back to processor 0. Eventually processor 0 has all the messages and then initiates the next iteration.

   3. *Congestion of events at lower levels.* The number of blocking read events for processors 64 to 127 is about 10 per iteration. One can easily see that the events should be about this number. There are events to start and stop the iteration and to exchange values. If one sends and receives values from four other processors, this gives 10 events per iteration. The excess number of events per iteration for the other levels are given below

Level	6	5	4	3	2	1	0
Number of excess events	2	5	9	9	14	12	12

      Two plausible, but completely conjectural, reasons for these excess events are: (a) the assignment of equation to processors is less efficient at the lower levels so more exchanges of values are required, (b) the synchronization message passing causes more events near the root of the ''decision'' tree.

6. **POTENTIAL PROGRAM IMPROVEMENTS.** The E/T performance analysis methodology has identified an important efficiency problem in this iteration algorithm. Since this is a typical example of a broad class of iterations that arise in many computations, it is appropriate to consider steps to alleviate this difficulty. We present observation of four types.

   A. Use Better or Less Synchronization. We can look for less costly synchronization schemes. Since the simple one used is the best possible for true synchronization, we must sacrifice something. For example:

   A-1. *Synchronize only every 5 or 10 iterations.* This has an unknown effect on the numerical properties of the iteration but it is certainly something to try.

   A-2. *Use adaptive sychronization.* Have every processor continue computing while the synchronization messages and analysis is taking place. Processor 0 initiates a true synchronization only when the timing of the messages indicates that the iterations have gotten too far from synchronized.

   A-3. *Local group synchronization.* Have small subgroups, changing over time, synchronize themselves. The numerical effects are unknown but this is probably pretty safe.

   B. Special Load Balancing. Figure 8 shows that processors at levels 1-6 spend a long time waiting on level 7 to finish. This fact could be taken into account when the equations are distributed to processors and thus the utilization could be improved. This probably would have a useful but not large effect on overall efficiency.

   One could take another approach to detect an unbalanced load, what we might call the *local waiting time balance test.* Each processor measures how long it is blocked compared to its computation time. When it is blocked for too long it signals that the load balancing is bad and someone takes remedial action.

   C. Use Faster Communication Hardware. The synchronization problem identified here is algorithmic in nature, the convergence theory of the iteration only applies if the iteration is synchronized. However, one can reasonably hope that exact sychronization is not really required for good convergence. If we had a perfectly balanced computation, it would be synchronized without explicit action. On the other hand, if the communication hardware were fast enough, we could synchronize without undue cost.

   It is unclear whether communication hardware will keep up with speed improvements in arithmetic processors. The current use of network-like protocols for communication seems to make it impossible to have communication speeds comparable to arithmetic speeds. But then, such difficulties

motivate people to devise better ways. As an indication of how things are going one can compare the speeds of the NCUBE 1 and NCUBE 2 as below (the values given are approximate, all in microseconds)

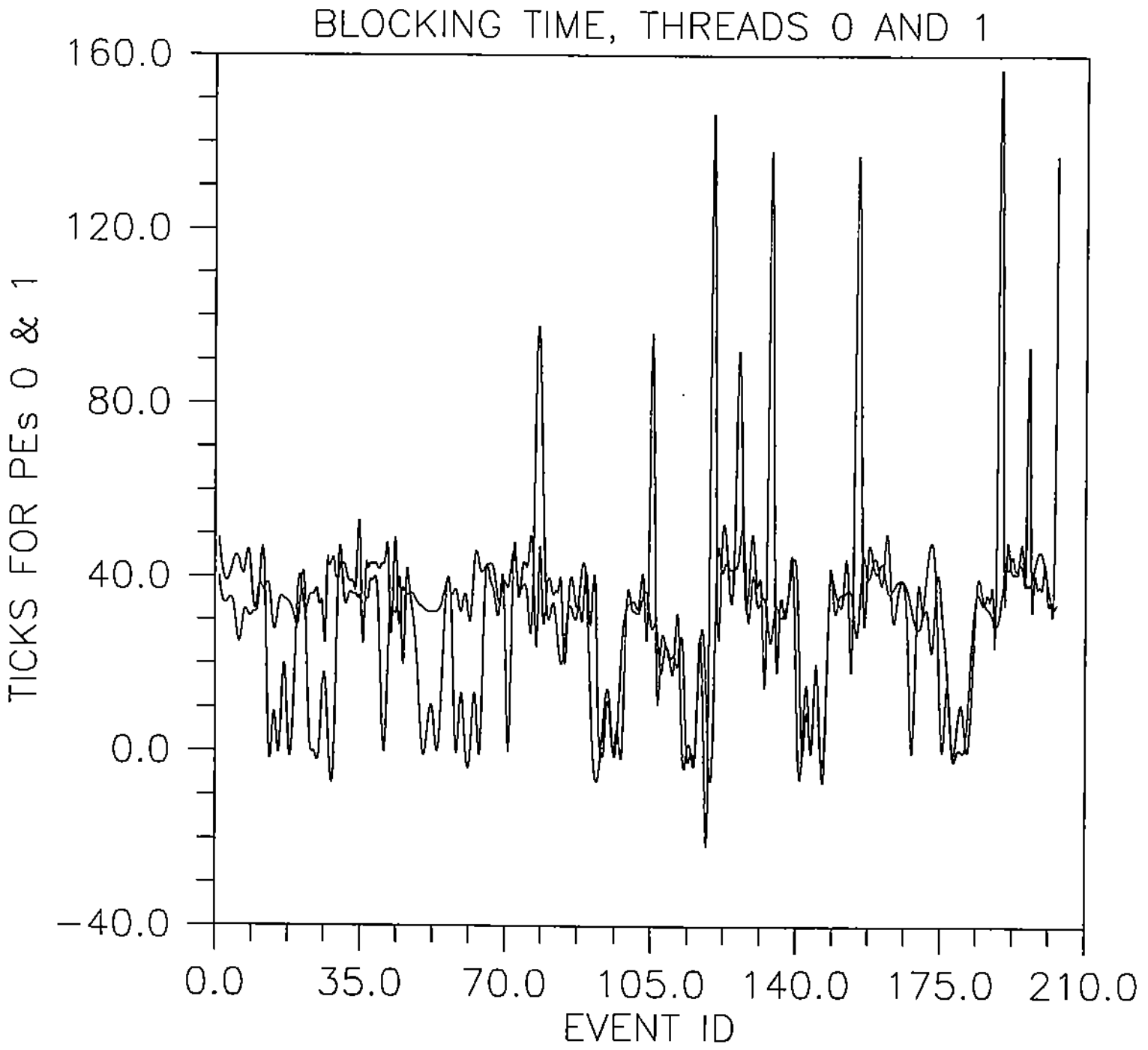
	NCUBE 1	NCUBE 2
CPU cycle time	.14	.05
Add time	3.0	.35
Send 1 word to neighbor	450	140
Send 1 word across 128 cube	25000	152
Send 1000 words to neighbor	9000	5000
Send 1000 words across 128 cube	6×10^5	5000

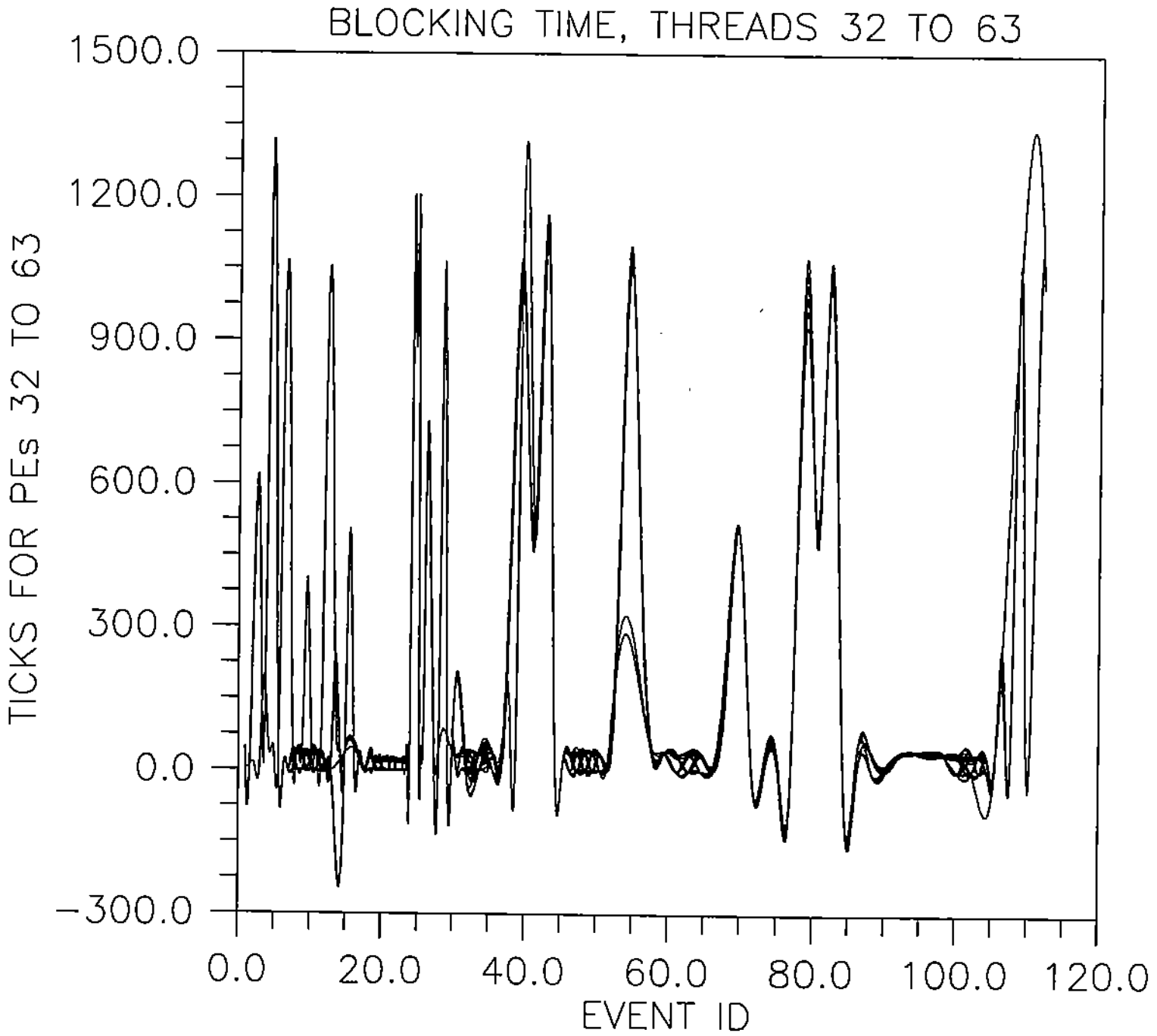
D. Increase Memory per Node. Existing hypercube machines tend to have too little memory per node. Even iterative methods, which tend to have low memory requirements, are limited by the lack of memory. Consider an NCUBE 1 with 128 processors running this program. The NCUBE 1 has 512 Kbytes of memory per node, or 128 Kwords. It is optimistic to expect that 60 Kwords are available for the linear system. In a reasonably compact sparse form one could hope to have about 5,000 equations per node (650,000 total). One iteration on 5,000 equations requires about 30,000 floating point operations. With 0.3 megaflops processors, the NCUBE 1 takes about 100 msecs (or 600 ticks) to do the iteration. Then it takes 1200 ticks to synchronize! The best utilization one could hope for is about 33%.

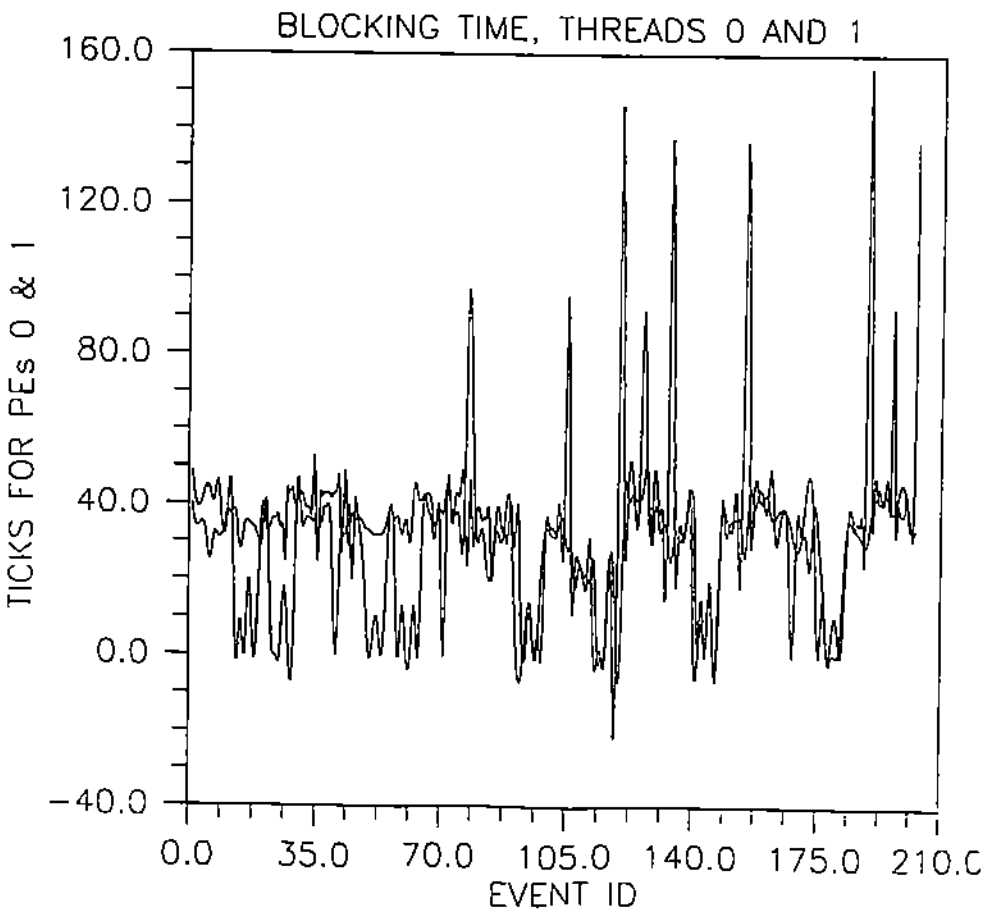
For the NCUBE 2 the speeds are increased by 10 for megaflops, 5 for communication and the memory is increased by 8 (but user memory probably increases by 12 or so). Thus we can have 60,000 equations in one processor's memory and an iteration takes about 160 msecs (or 1000 ticks). The communication time is decreased by a factor of 5 to give about 240 ticks for synchronization. Then the best computation/communication ratio changes from 600/1200 = 0.5 to 10000/240 = 4, a large but not overwhelming improvement. The best utilization one could hope for is thus about 80%. For computations which do not use all of a processor's memory, the ratios are smaller and the performance worse.

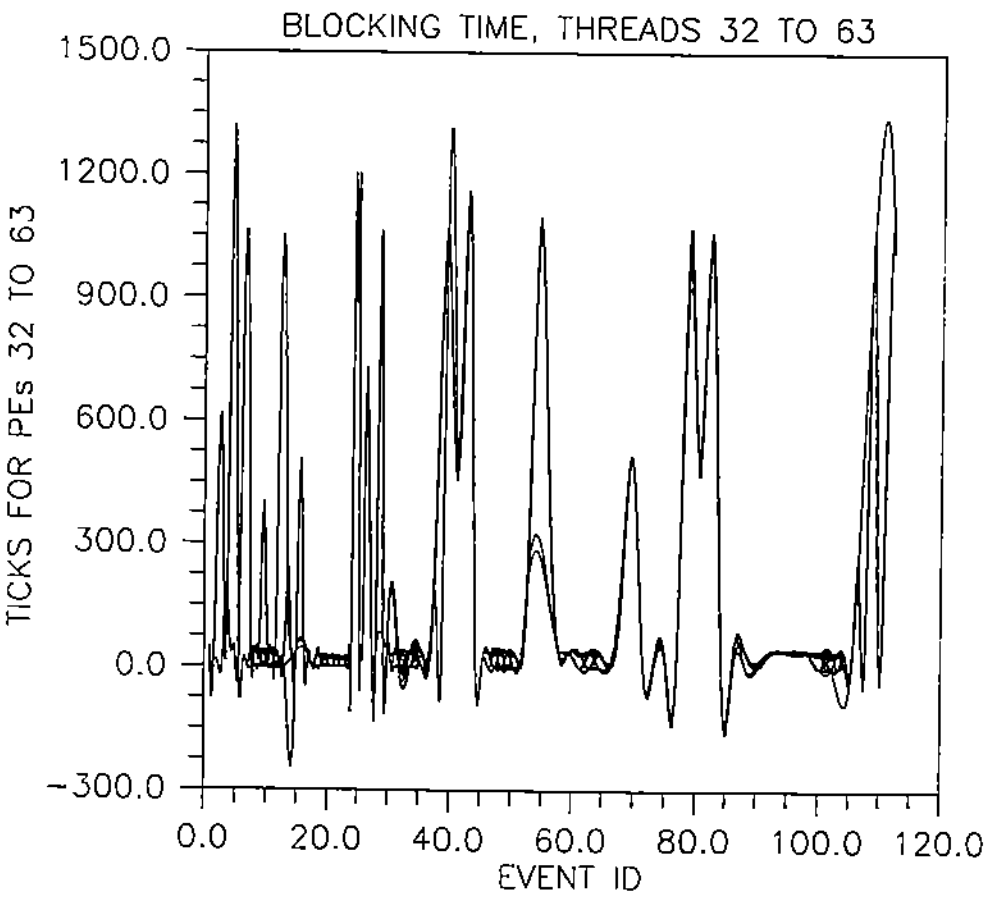
REFERENCES

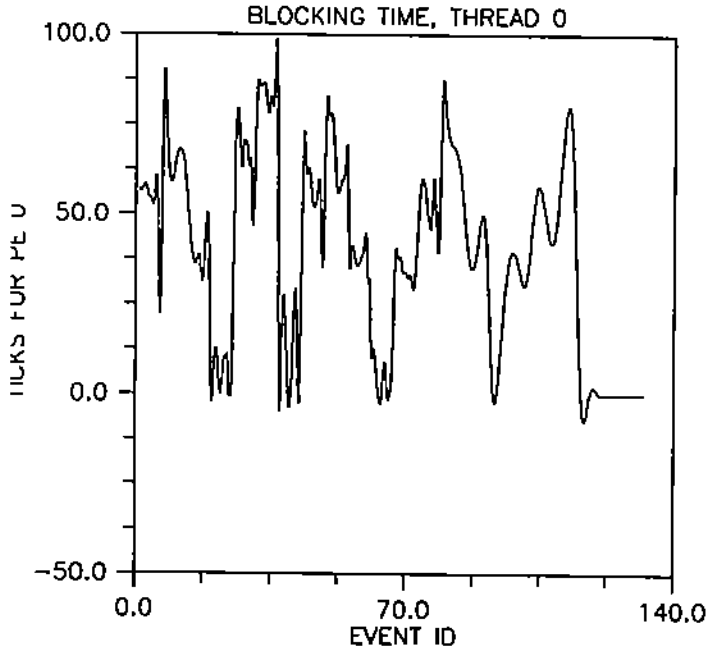
1. D.C. Marinescu, J.R. Rice and E.A. Vavalis, "Communication and Control in SPMD parallel Numerical Computations", CSD-TR-981, Computer Sciences, Purdue University, May, 1990.
2. D.C. Marinescu and J.R. Rice, "Synchronization and Load Balance Effects in Distributed Memory Multiprocessor Systems", CSD-TR-1000, Computer Sciences, Purdue University, July, 1990.

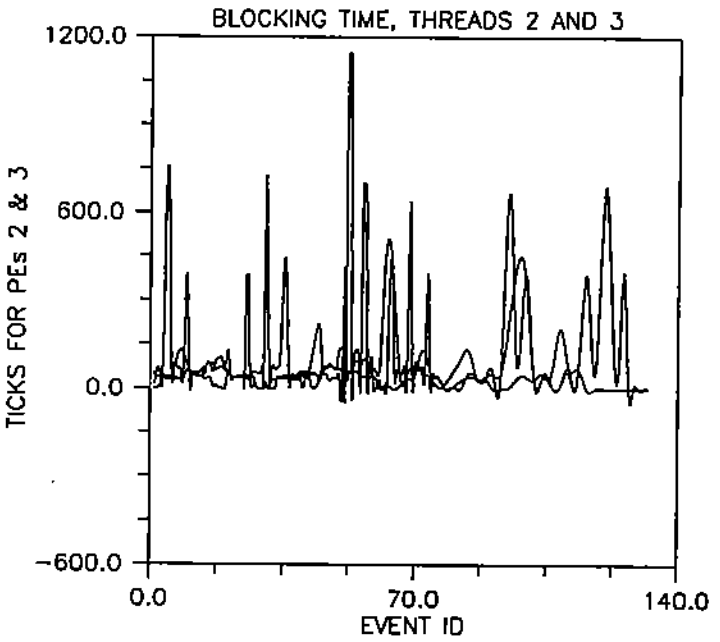

BLOCKING TIME, THREADS 0 AND 1
160.0
120.0
80.0
40.0
0.0
−40.0
TICKS FOR PEs 0 & 1
0.0
35.0
70.0
105.0
140.0
175.0
210.0
EVENT ID

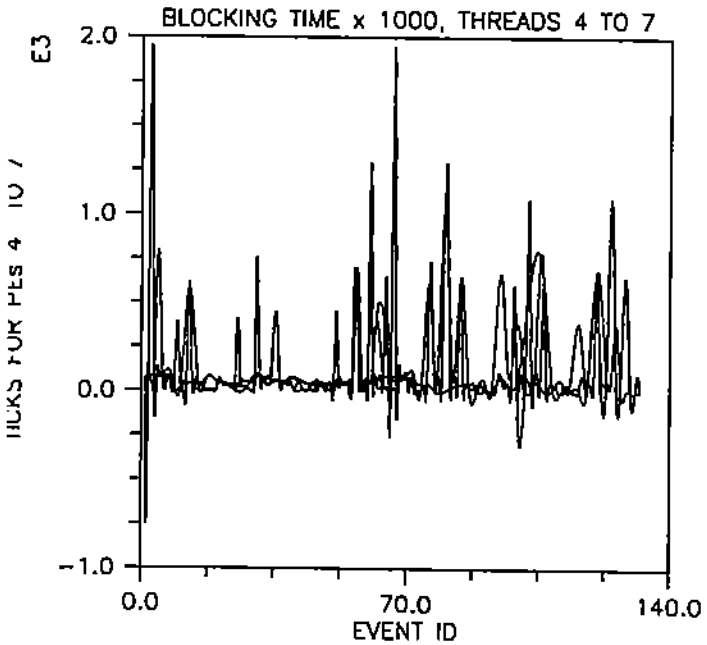

BLOCKING TIME, THREADS 32 TO 63
1500.0
1200.0
900.0
600.0
300.0
0.0
-300.0
TICKS FOR PEs 32 TO 63
0.0
20.0
40.0
60.0
80.0
100.0
120.0
EVENT ID

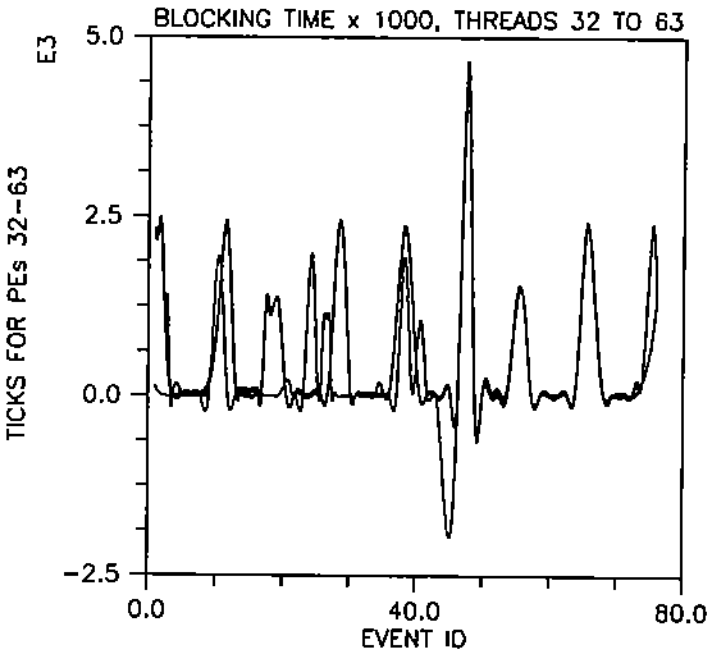

BLOCKING TIME, THREADS 0 AND 1
160.0
120.0
80.0
40.0
0.0
-40.0
TICKS FOR PEs 0 & 1
0.0
35.0
70.0
105.0
140.0
175.0
210.0
EVENT ID

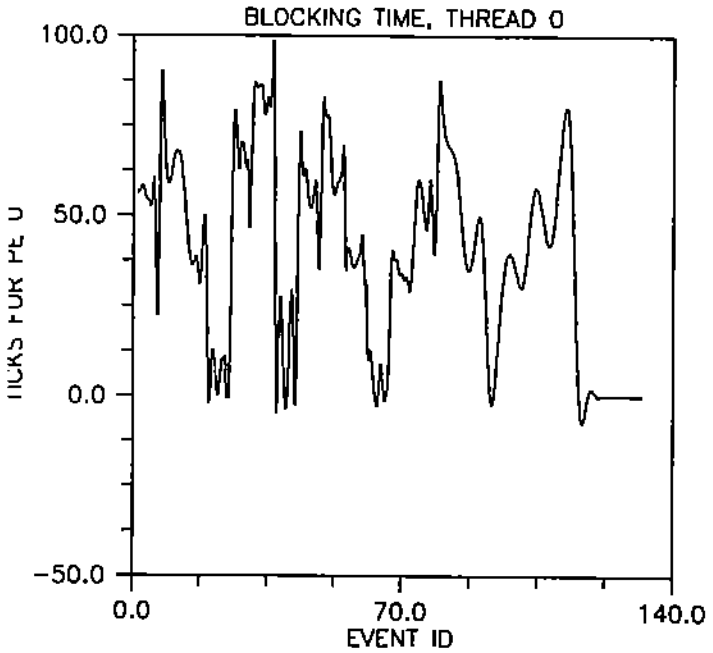

BLOCKING TIME, THREADS 32 TO 63
TICKS FOR PEs 32 TO 63
1500.0
1200.0
900.0
600.0
300.0
0.0
-300.0
0.0
20.0
40.0
60.0
80.0
100.0
120.0
EVENT ID

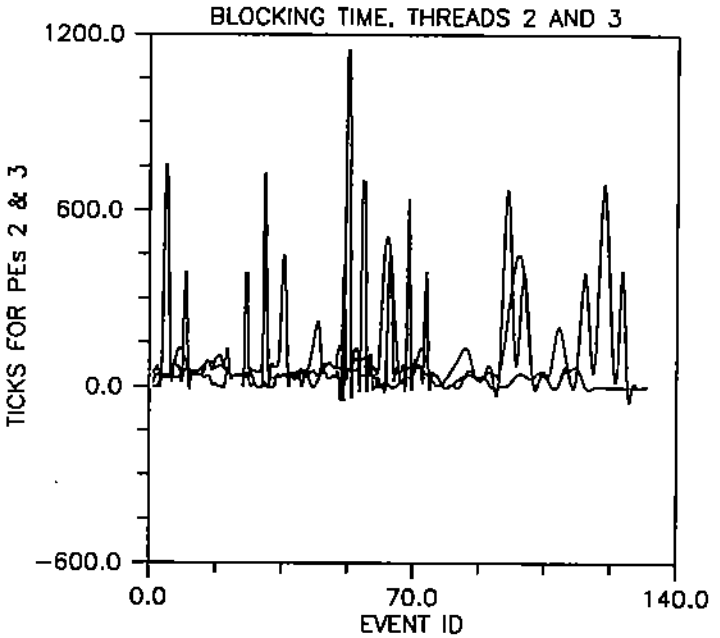

BLOCKING TIME, THREAD 0
100.0
50.0
0.0
-50.0
0.0
70.0
140.0
EVENT ID

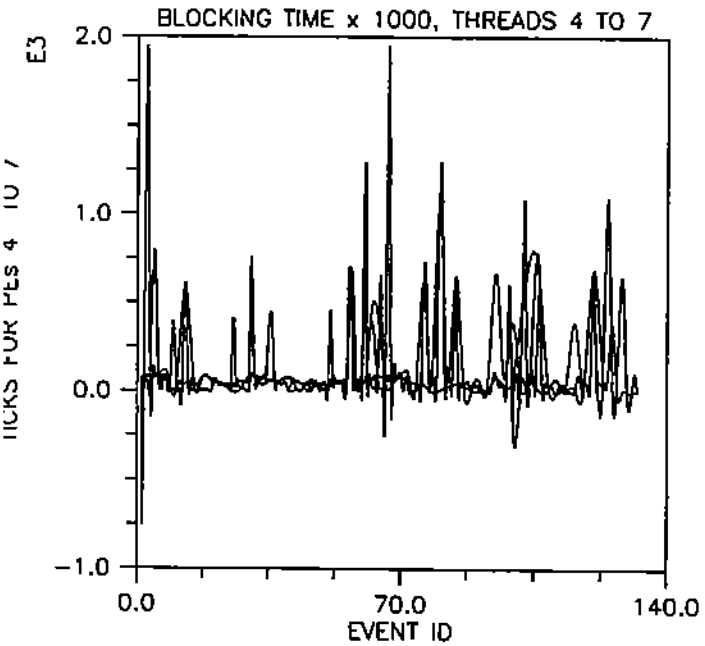

BLOCKING TIME, THREADS 2 AND 3
1200.0
600.0
0.0
-600.0
TICKS FOR PEs 2 & 3
0.0
70.0
140.0
EVENT ID

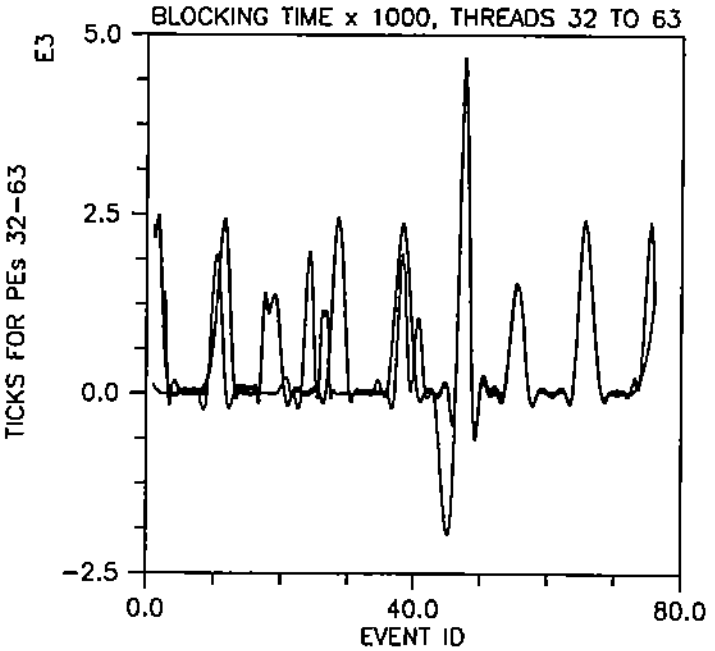

BLOCKING TIME x 1000, THREADS 4 TO 7
E3
2.0
1.0
0.0
-1.0
0.0
70.0
140.0
EVENT ID

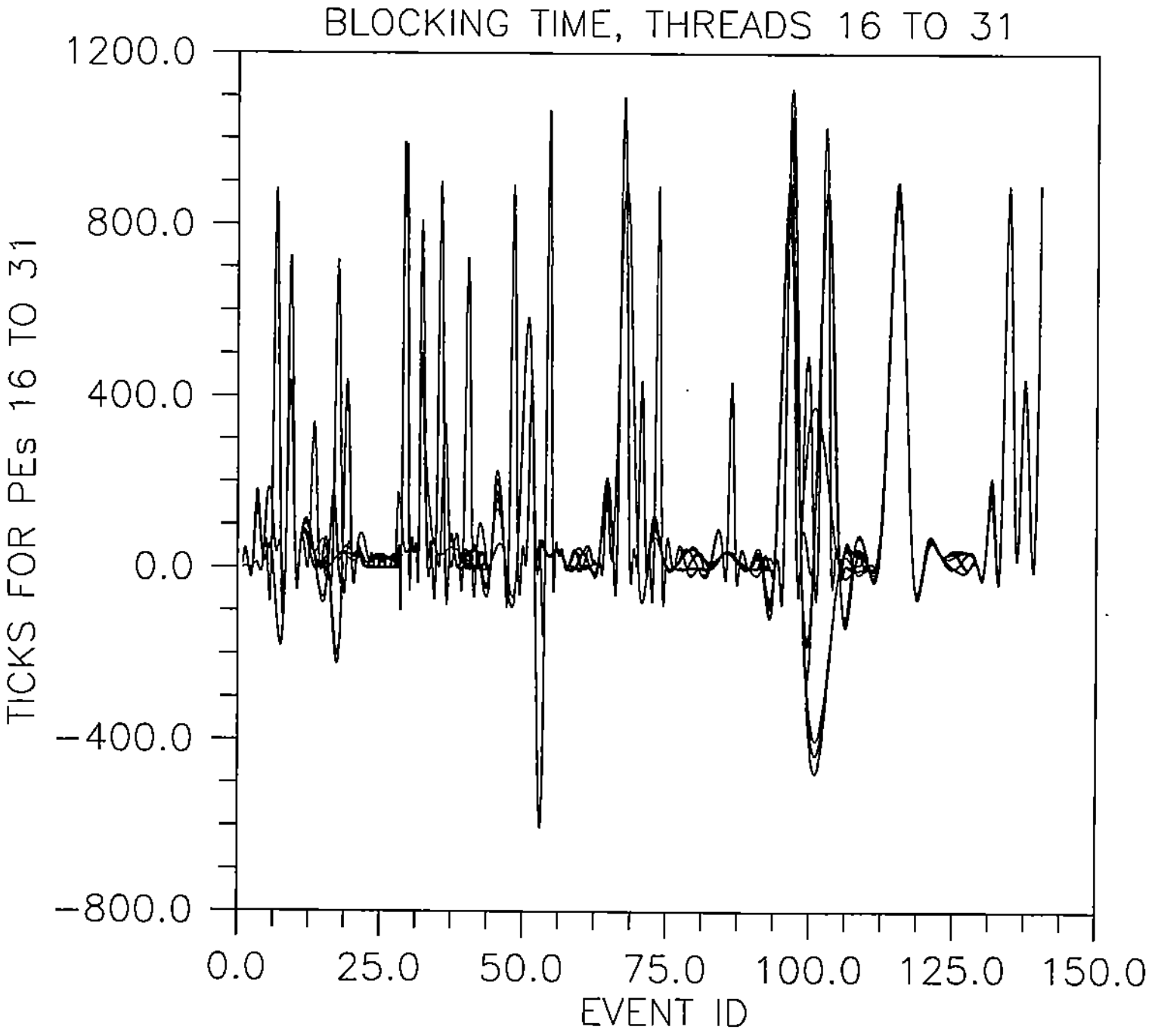

BLOCKING TIME x 1000, THREADS 32 TO 63
E3
5.0
2.5
0.0
-2.5
TICKS FOR PEs 32-63
0.0
40.0
80.0
EVENT ID

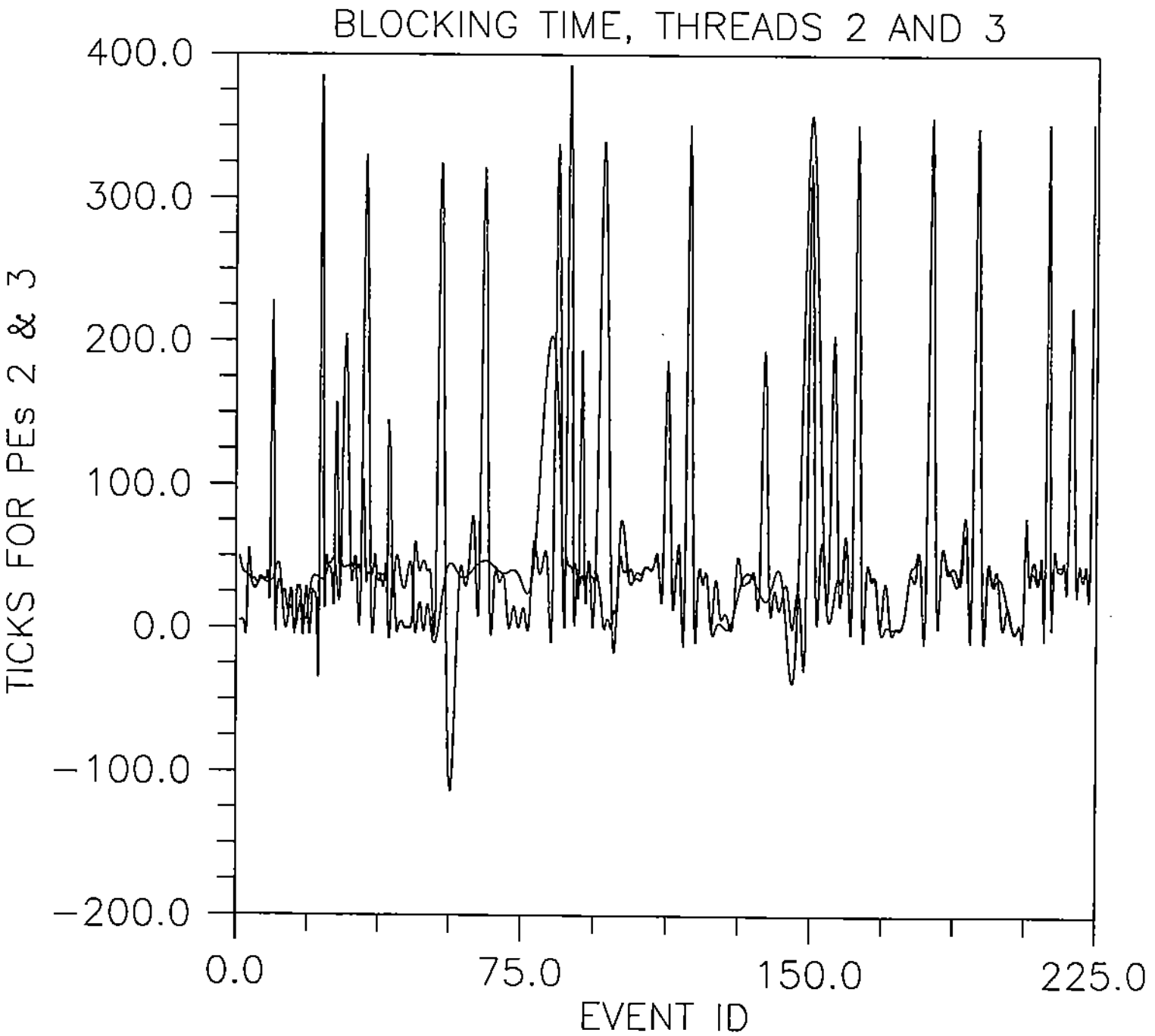

BLOCKING TIME, THREAD 0
100.0
50.0
0.0
-50.0
0.0
70.0
140.0
EVENT ID

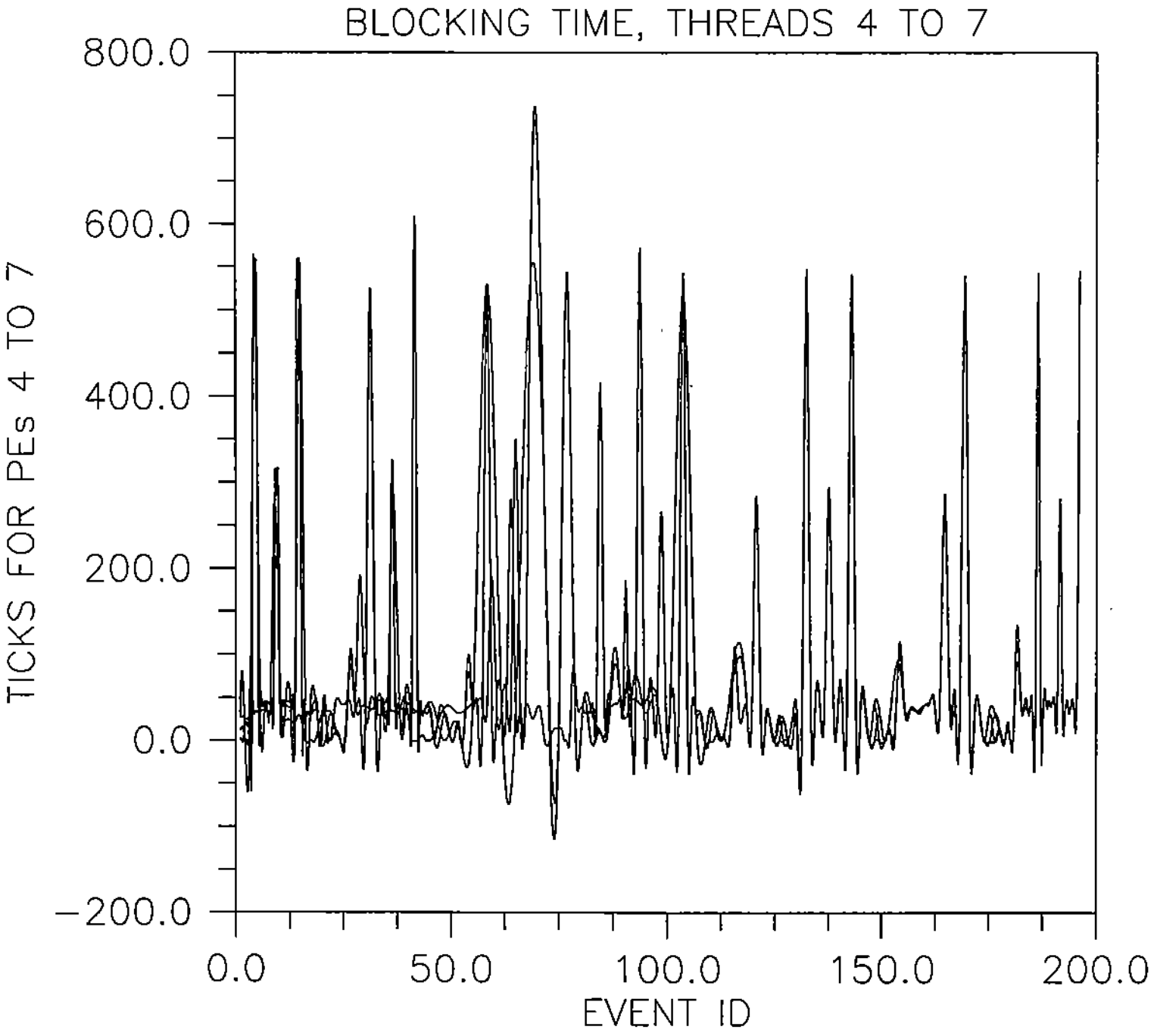

BLOCKING TIME, THREADS 2 AND 3
1200.0
600.0
0.0
-600.0
TICKS FOR PEs 2 & 3
0.0
70.0
140.0
EVENT ID

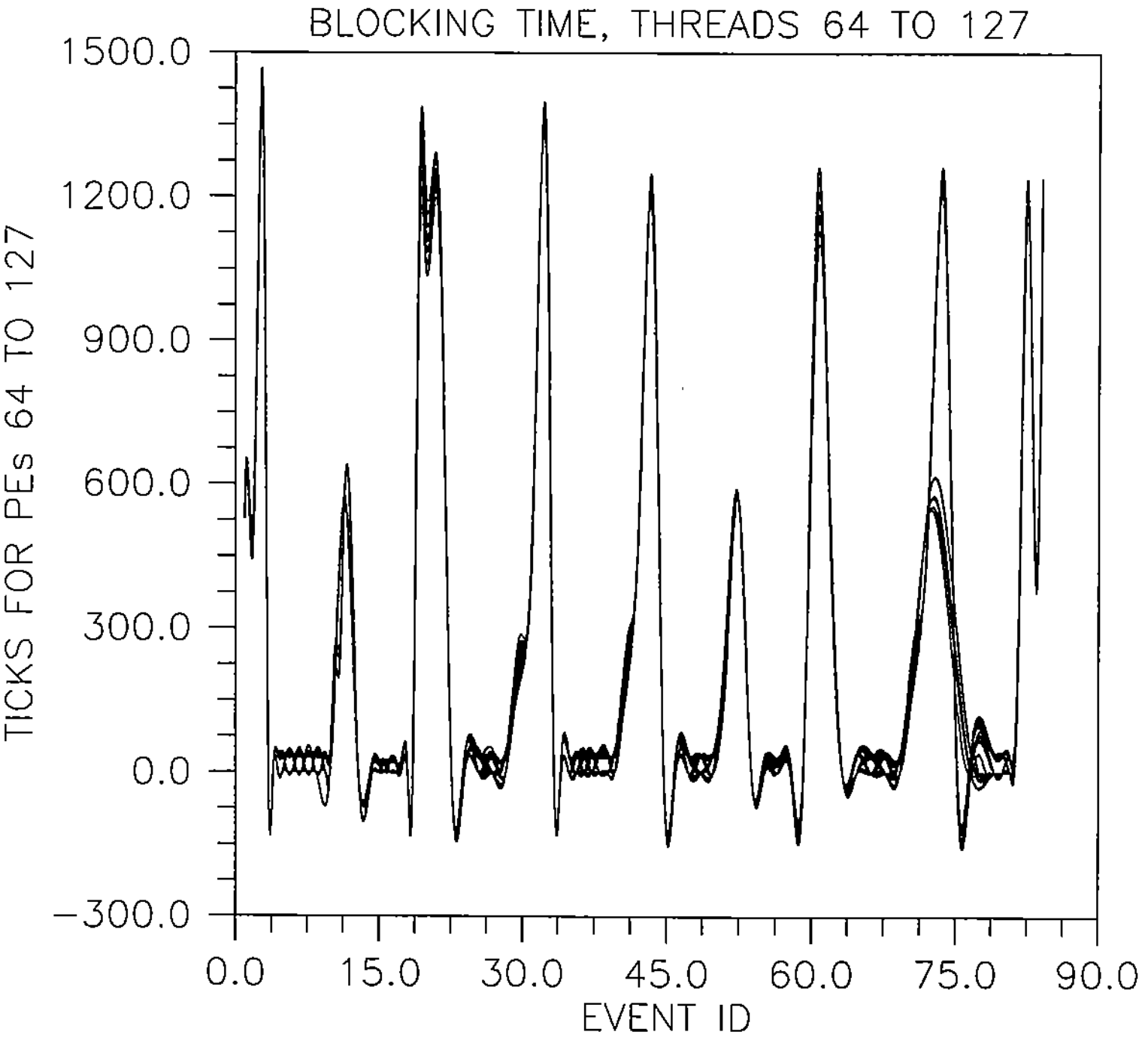

BLOCKING TIME x 1000, THREADS 4 TO 7
E3
2.0
1.0
0.0
-1.0
0.0
70.0
140.0
EVENT ID

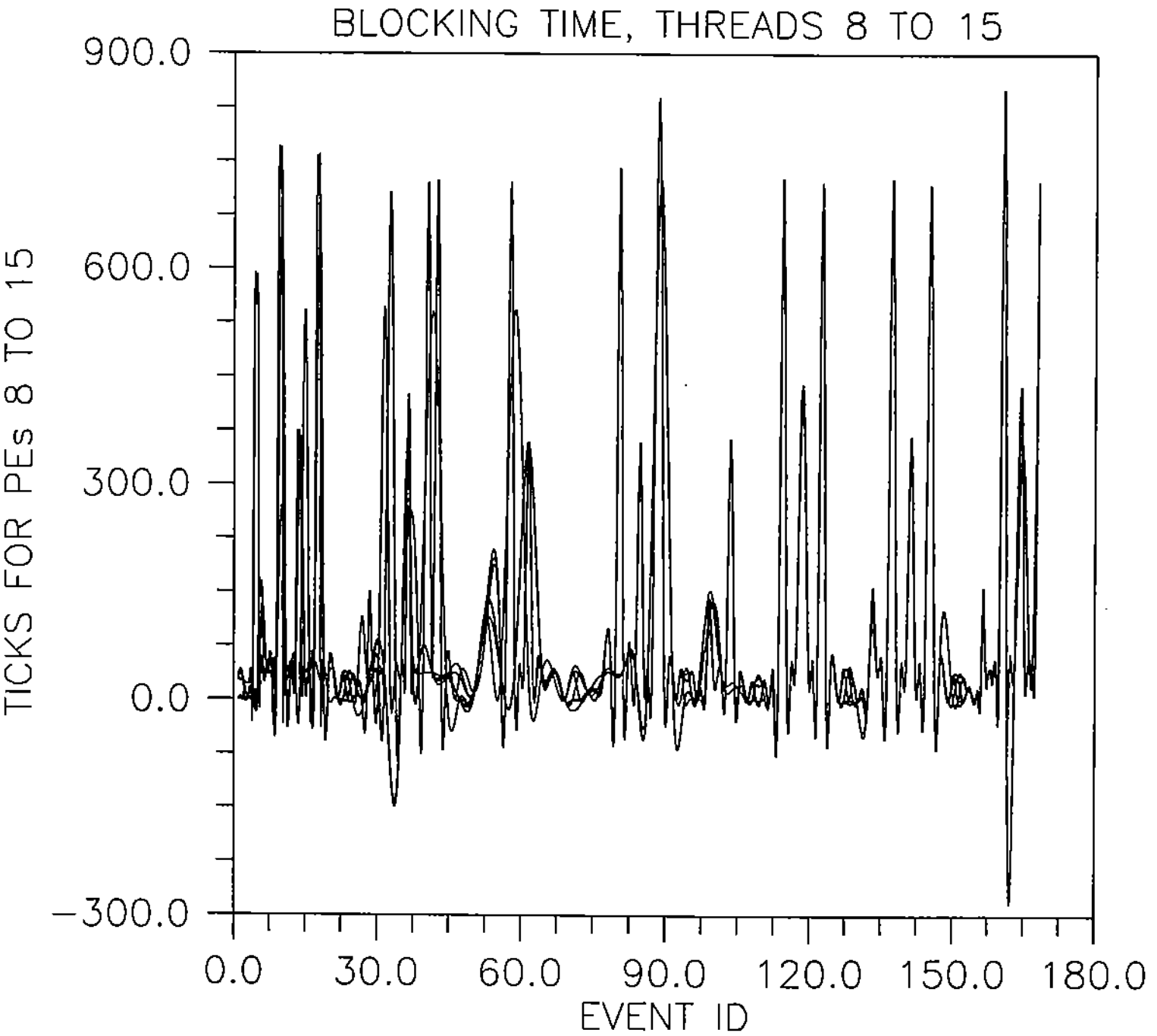

BLOCKING TIME x 1000, THREADS 32 TO 63
E3
5.0
2.5
0.0
-2.5
TICKS FOR PEs 32-63
0.0
40.0
80.0
EVENT ID


BLOCKING TIME, THREADS 16 TO 31
1200.0
800.0
400.0
0.0
-400.0
-800.0
TICKS FOR PEs 16 TO 31
0.0
25.0
50.0
75.0
100.0
125.0
150.0
EVENT ID


BLOCKING TIME, THREADS 2 AND 3
400.0
300.0
200.0
100.0
0.0
−100.0
−200.0
TICKS FOR PEs 2 & 3
0.0
75.0
150.0
225.0
EVENT ID


BLOCKING TIME, THREADS 4 TO 7
800.0
600.0
400.0
200.0
0.0
−200.0
TICKS FOR PEs 4 TO 7
0.0
50.0
100.0
150.0
200.0
EVENT ID


BLOCKING TIME, THREADS 64 TO 127
1500.0
1200.0
900.0
600.0
300.0
0.0
−300.0
TICKS FOR PEs 64 TO 127
0.0
15.0
30.0
45.0
60.0
75.0
90.0
EVENT ID


BLOCKING TIME, THREADS 8 TO 15
900.0
600.0
300.0
0.0
−300.0
TICKS FOR PEs 8 TO 15
0.0
30.0
60.0
90.0
120.0
150.0
180.0
EVENT ID