DOE/ET-53088-64 IFSR #64

LARGE SCALE PARTICLE SIMULATIONS IN A VIRTUAL MEMORY COMPUTER

P.C. Gray
University of Alaska
Fairbanks, Alaska 99701

J.S. Wagner and T. Tajima
University of Texas
Austin, Texas 78712

R. Million
University of Alaska
Fairbands, Alaska 99701

August 1982

LARGE SCALE PARTICLE SIMULATIONS IN A VIRTUAL MEMORY COMPUTER

P. C. Gray
Geophysical Institute
University of Alaska
Fairbanks, Alaska 99701

J. S. Wagner and T. Tajima
Institute for Fusion Studies
The University of Texas at Austin
Austin, Texas 78712

R. Million
Geophysical Institute
University of Alaska
Fairbanks, Alaska 99701

Abstract

Virtual memory computers are capable of executing large-scale particle simulations even when the memory requirements exceed the computer core size. The required address space is automatically mapped onto slow disc memory by the operating system. When the simulation size is very large, frequent random accesses to slow memory occur during the charge accumulation and particle pushing processes. Accesses to slow memory significantly reduce the execution rate of the simulation. We demonstrate in this paper that with the proper choice of sorting algorithm, a nominal amount of sorting to keep physically adjacent particles near particles with neighboring array indices can reduce random access to slow memory, increase the efficiency of the I/O system, and hence, reduce the required computing time.

I. INTRODUCTION

The computer simulation technique is useful in understanding nonlinear plasma dynamics. In particular, the study of microscopic processes in plasmas can be aided by the technique of particle simulation.[1] In many plasma physics problems the interrelationship between large spatial scale effects and microscopic processes becomes important, as well as the interplay between long-time scale effects and short-time scale phenomena. Typically the relevant time scales vary from 10^{-12} to 1 second, while the relevant spatial scales encompass from 10^{-5} to 1 meter for laboratory plasmas. Because of computer core size limitations, it is not possible to realistically represent plasma by a particle simulation, even on the world's most advanced computers. We envision three paths that lead to more realistic simulations. One method is to improve the algorithm of advancing the plasma dynamic equations in time so that a large time-step Δt can be taken without compromising relevant effects due to short time scale phenomena.[2],[3],[4] The second approach is to increase the effective physical length described by the spatial grid by using more advanced interpolation techniques.[5] The third method is to develop techniques that enable us to carry out very large size simulations on computers with limited memory sizes. This last problem is addressed in the present paper.

Modern high-performance scientific computers like the CRAY-1 have fast memories that can hold 1-2 million words. In order to run codes that require more memory than this, it is necessary to map a part of the code or code-generated data onto the slower disc memory. In a high-speed scientific computer the input/output transfer rate is much slower than the cpu execution rate. This causes a bottleneck in an I/O flow when mapping of memory is required. Then the extraordinary speed of the CRAY-1 cpu cannot be efficiently utilized.

In some cases it may be more cost-effective and time-effective to run very large codes on computers that have operating systems optimized for efficient transfer of information between fast memory and disc.[6] Virtual memory computers have this potential. Virtual memory computers such as the VAX-11/780 can handle codes larger than can the CRAY-1 with no special modification to codes themselves.

The execution rate of a large-scale simulation on a virtual memory machine is strongly dependent on the efficiency of data transfer between core and peripheral memory, especially if a significant amount of memory must be resident on disc. Randomly accessing memory from disc is most inefficient. Systematically requesting adjacent numbers and transferring them in clusters has been found to be much more efficient. The problem is that particle simulations normally access memory randomly. It is possible, however, with minor modification to the simulation code, to control the random accesses to memory and group them into clusters having the same size as the read/write clusters controlled by the virtual operating system. In this way the simulation code is optimized and can be used on grids 8-16 times larger than can be run on a CRAY-1 (without disc mapping).

The essence of a virtual operating system is that processes are partitioned into "pages", which reside in peripheral memory and are migrated or "paged" into a section of the cpu allocated to the particular process, the "working set", as they are needed for further execution of the program. The major overhead in such a system is the paging to and from the cpu and peripheral storage. A worst case situation for the execution of a process is one in which the execution of any instruction is likely to generate a page fault. An example of such a situation is the random access of elements of a very large array. It is clear that most of the paging in the execution of a particle simulation code takes place due to those computations which associate the particle quantities by position with the corresponding grid quantities (e.g., charge accumulation and particle pushing). This is true because a particle position in the grid is random, or will tend to be so, with sufficient number of time-steps.

The approach we have taken to reduce page faulting is to sort the particles by position before associating them with their corresponding grid elements. This leads to a situation where all of the particles in a particular grid cell and the associated grid elements are present in the cpu at the same time. Hence, when the code associates particle quantities with grid quantities, the grid elements are accessed in more or less the order of linear storage. We find that the time expended in keeping the particles sorted is much less than the time gained by reducing the number of page faults.

II. COMPUTER CONFIGURATION

Although various virtual memory machines have slightly different architectures, they have in common the important feature of using peripheral disc storage to supplement main memory. The details of how memory is mapped onto the combination of main and peripheral memory depends on the details of the computer and its operating system. We summarize in this section a typical configuration, the VAX-11/780 in Fig. 1.

The cpu is divided into a set of pages which are shared by all executing jobs, or processes. Each page contains 512 bytes of information, and each process has associated with it a working set of pages consisting of that part of the process which is executable at any given time. The maximum size of the working set is set by the operating system at the start-up. In general, the maximum working set size is the cpu size (in pages) divided by the maximum number of possible processes.

In addition to the working set an executable process has associated with it a modified page list and a free page list. The modified page list contains those pages removed from the working set which were modified during the execution. The free page list contains those pages from the working set that were not modified. Pages are paged out of the working set based on age, into either the modified page list if execution has altered the contents of the page or into the free page list if execution has not modified the page, and from these either back into the working set or onto disc. For example, in a 16-user system the maximum working set is set to 256 pages, the modified page list is 44 pages, and the free page list is the remainder of memory. Processes share the free page list space with other processes.

When a process needs a page that is on disc or in the free or modified page list, it is paged into the working set, along with neighboring pages that make up a page fault cluster. If the working set is full, since its size is fixed, a page must be removed to make room. The oldest page is then transferred to the modified or free page list. If the modified page list is full, a cluster of pages is written back onto disc. In a 16-user system, the write cluster is 32 pages, and only pages in the modified page list need to be written back to disc, since those pages in the free page list are unaltered.

The time required to page to and from the modified page list or free page list is only a few clock cycles and is negligible compared to that for a disc page fault. The time required to page to and from disc is equal to the average access time added to the transfer rate times the cluster size. For the 16-user system this is roughly 51 msec. Since this time is very slow compared to the access time in the cpu, efficient codes must minimize the number of disc page faults.

III. SIMULATION TECHNIQUES

In particle simulations we attempt to model the complicated behavior of an ionized plasma by following the motion of a modest number of computer particles ($<10^6$) in their self-consistent electric and magnetic fields. The computer particle, or macro-particle, is thought of as a cloud of real particles. No computer is capable of handling a realistic particle number (10^{20}). The reduced number of particles can lead to unrealistic effects such as enhanced collisional effects. Techniques such as the use of finite-size particles have been introduced[7] to reduce the effects of short-range Coulomb collisions in computer plasmas. Since it is too expensive to compute the self-consistent motion of the plasma by computing the force on each particle due to every pair, a spatial grid is introduced on which the charge density and current density are accumulated; the electric and magnetic fields are stored there as well. Time-stepping in the simulation involves three processes; a) accumulating the charge density and current density on the spatial grid from the particle positions and velocities; b) solving Maxwell's equations on the grid to find electrical and magnetic fields E and B; c) interpolating the fields on the grid to the particles and using the interpolated force to push the particles in time.

Numerous algorithms are available for the charge accumulation process, such as NGP (nearest grid point), multiple expansions[8], and polynomial splines.[5] Maxwell's equations can be solved with finite differencing or Fourier techniques. Particle pushing is normally done with the second-order accurate leap-frog technique.

Simulations are classified according to the number of degrees of freedom allowed by the particles. Simulations come in 1, 1.5, 2, 2.5, and 3-dimensional versions, where, for example, a 2.5-D code allows the particles to move in the x-y plane but allows three velocity coordinates, which is useful, for example, when there is a magnetic field in the x-y plane. Electrostatic codes compute only the electrostatic field self-consistently from Poisson's equation. Magnetostatic codes compute both electric and magnetic fields self-consistently, but radiation effects are ignored. Fully electromagnetic simulations compute E and B self-consistently and include radiation.

Our goal is to have the capability to simulate plasmas on nearly realistic temporal and spatial scales. To some extent lack of computer capabilities prevent us from achieving our goal. The best algorithms available to us today do not allow us to describe large spatial scale or to use big enough time-steps, although there are several significant advances in furthering time-step[4] and spatial scale.[5] The reason for the spatial problem is that the grid spacing Δ must be no larger than the smallest spatial wavelength λ we wish to describe in the simulation. In conventional simulations[1,7] the smallest phenomenon of interest must be larger than the electron Debye length λ_{De}, or

$$L_x \gtrsim \lambda \gtrsim \lambda_{De} \approx \Delta \quad .$$

Laboratory-scale plasmas have spatial lengths on the order of 10^6 Debye lengths in one dimension. Since our grid spacing (Δ) must be on the order of the Debye length for a typical simulation using the multiple expansion technique, laboratory scale would require 10^6 grids along any direction. Present day computers restrict us to grids of size $L_x \sim 1024\Delta$ in one dimension, or $L_x \times L_y \sim 128 \times 128$ in two dimensions for electrostatic codes.

In electromagnetic simulations we are further constrained by the collisionless skin depth

$$L_x \gtrsim \lambda \gg c/\omega_{pe} \gg \frac{v_{th}}{\omega_{pe}} = \lambda_{De} \approx \Delta \; .$$

In addition, we have a limit on the maximum time-step we are allowed to use, based on the highest frequency phenomenon we are interested in. In most cases this is the electron plasma frequency ω_{pe}. The maximum time-step allowed is then typically

$$\Delta t \sim 0.1 \omega_{pe}{}^{-1} \quad .$$

This is a very severe limitation for processes which occur on the ion time scale and slow magnetohydrodynamic phenomena. Some progress has been made recently to increase the time-step by using the guiding center approximation and partially implicit techniques and these codes are very useful when they apply.

IV. SORTING TECHNIQUE - Partition Sort

There are two considerations associated with the choice of appropriate sorting scheme. The first consideration is related to the degree of ordering of particles at any point in the simulation. Since the particles do not move more than a grid length for any time-step in the simulation it seems unnecessary to completely sort the particles at each time-step, the ordering of particles having changed little between adjacent time-steps.

We have found two sorting algorithms which are particularly well suited to the particle sorting problem we have just described. For a general discussion of techniques, including those used in this paper, we refer to D. E. Knuth.[9] The two types of sorting routines we have applied are appropriate for sorting records into bins of predetermined size, they are the radix sort and a variant of the partition exchange sort, or quicksort.

The first of these two algorithms, the radix sort, is similar to a technique used by card sorters, and can be described as follows. One defines a set of bins, for example in a 2.5-D system a bin will contain all of the particles with y-position between the (i) and (i+1) grid levels. The number of particles which fall in each bin is then counted, in order to determine the needed temporary storage per grid cell. Particle quantities are then placed in the proper bin and finally the arrays are reordered.

The partition exchange sort is an entirely different approach than that of the radix sort. If we think of the array to be sorted as increasing from the left to the right in its proper order, we first choose some median test value, then starting from the left, we find the first value greater than our test value but occuring before it in the array. Then starting from the right we find the first value less than the test value but occuring after it in the array. The two elements are then switched, and the process is continued until the two elements we are looking for are less than some optimal partition size. At this point the array is divided into three sections, a central segment, and both ends. We repeat the process on both end segments independently until we are left with a set of ordered segments no larger than the chosen partition size. The choice of the partition size is important to the efficiency of the sorting for the application of this technique to our problem, since it defines the degree of order in the particle arrays. This implies that the amount of paging is proportional to the degree of ordering of the particles at any time-step. If there are N elements in a page for the particle position array, then it would seem that at any time-step it is only necessary that every particle be within N elements of its literal order by position to reduce paging. There are two essentially equivalent methods for achieving this; if one knows the mean-time required for a particle in the simulation to move the length of a grid is M steps, then one could sort the particles to literal position every M time-steps.

Alternatively, one could sort the particles to within N elements of their literal position every time-step. The second of these two approaches is more easily implemented, since the mean time for a particle to move the length of a grid cell is a function of the problem being modeled in the simulation.

The second consideration is the trade-off between sorting the particles and the time taken up by page faulting. To a reasonable extent this is related to the configuration of the computer being used and how efficiently the operating system handles paging. At a certain size simulation the amount of paging per time-step requires more time than would be required to sort the particles. One reaches this point sooner the less efficient the operating system, and the more efficient the choice of sorting routine.

For our study we have implemented the sorting of particles for 1-D electrostatic and 2.5-D electrostatic and electromagnetic codes.[10,11] With the 1-D system particles are to be sorted to within a page of their final position in literal order before charge accumulation and the pushing of the equations of motion. The implementation is somewhat less obvious with the 2.5-D case than with the 1D case. We note that the linear storage of a two-dimensional array (i.e., any grid quantity) is by columns. It is, therefore, only necessary to sort particles by their y-position to reduce paging.

There are advantages to both of the sorting techniques described above. The partition exchange sort is fast, with speed going like $N\log_2 N$, where N is the number of elements to be sorted. However, it is inappropriate for application to "parallel" machines. Furthermore, while we may force every element to be within a chosen distance of the position it would occupy, if the array were sorted to literal order by choosing that distance as the partition size, there will not be, in general, the distinct separation between the elements in the (i) and (i+1) grid level that the radix sort produces. As for the radix sort, it is the appropriate choice for a parallel machine. However, while its speed on moderately large systems matches that of the partition sort, it is noticeably slower on very large systems in a virtual memory environment. For this reason we have chosen to use the partition sort in our numerical tests of this technique.

V. RESULTS AND DISCUSSION

A. Benchmark Timings with No Sorting

In the first part of this section we examine the relationship between execution rate and working set size without sortings. From this information it is possible to estimate the execution rates for a variety of related simulations.

We have tested the sorting technique described above on a variety of particle codes, both one and 2.5 dimensional, electrostatic and electromagnetic. Our particular machine is configured with four mega-bytes of random access memory, hardware floating point operations, and sufficient disc storage. The operating system is also DEC's VMS (2.3), the operating system parameters for the test were those suggested by DEC for a 16-user configuration, with the exception that the virtual page count, which designates the maximum number of pages any one process is allowed, was increased to allow the linking of very large grid systems (e.g., 128×128 and 256×256 grids). In Fig. 2 we show results for 64×64 two-dimensional electrostatic simulation with 128×128×2 particles. We separately timed different sections of the code, namely charge accumulation, the Poisson solution (using the fast Fourier transforms) and the particle push. For the largest working set we tried, 700 pages, one time-step took about seven seconds. Two-thirds of this time was spent pushing the particles. The remainder of the time was shared by the Poisson solution and charge accumulation processes. A sharp decrease in time is observed between working sets of 100 and 60, and then slowly decreases after that for larger working sets. This example shows how the computer becomes I/O bound when a significant fraction of the code is resident on disc as the working set size becomes small or the code size becomes large.

Figure 3 shows an even larger system, 128×128 grid and 256×256×2 particles. With a working set of 700 pages, the code requires about 30 seconds per step. The execution rate would be smaller for larger working sets. Another example with the same grid but fewer particles is shown in Fig. 4. These results indicate the importance of fitting as much of the code into core as possible, particularly the grid quantities; yet when required the computer can process codes which exceed the allocated core space. The largest 2-D electrostatic code we tried was a 256×256 grid, 512×512×2 particles and a working set size of 1500 pages. The Poisson solution took 12 seconds, the charge accumulation took 13 seconds, and the particle push took 49 seconds for one time-step.

B. Test Results with Sorting

In this second part of the section we examine the execution rates of various codes with sorting. For the following two reasons we give our results in terms of page faults and in terms of a partition size normalized to the effective page size, the write cluster. First, the VMS operating system uses the unused portion of the main memory as a "cache" for pages which are paged out of the processor's working sets. For this reason, the average cost of a page fault is very much a function of the work load on the machine and the machine configuration, in particular, the amount of main memory. Second, in order to facilitate the transfer of pages to and from disc, one of the operating system parameters, the "cluster factor", is set so that more than one page is moved into the working set when a page fault occurs; this parameter defines what is an effective page size. The results given may be interpreted in terms of processor time by determining the average page fault cost for the given machine and work load.

The results from our tests are shown in Figs. 6-17. From 1-D codes the number of particles per cell was 20 (i.e., 20 ions and 20 electrons), while for the 2.5-D codes it was four. The sort partition measures the size of one unsorted datum. The write cluster is the cluster of words transmitted between the disc and core. (See Fig. 1). Therefore, larger the ratio of the sort partition to the write cluster, the less sorting is done.

Two features are evident from these figures. First, the cpu time required for execution increases sharply as the number of page faults increases beyond some certain value. This is due to the excessive random access memory when the sorting is done only sparsely. The number of page faults is the sum of the page faults to disc and the page faults to the modified and free page lists. Hence, the less random-access memory the machine has available for the modified and free page lists, the more representative of execution time is the number of page faults. The second noticeable feature is that there is a break-even point for the cpu time (depending on the size of the simulation) below which it requires more time to sort than is saved in reduction of page faults. This effect is again strongly dependent on the machine configuration. This break-even point is about where there is enough space in the working set and the modified page list for the entire simulation. In every case maximum benefit is gained by sorting until the sort partition roughly equals the cluster size.

When the sorting algorithm is actually applied in a production run it is not necessary to sort every time-step. It need only be applied after the number of time-steps it takes for the particles to randomize themselves into partitions larger than the cluster size. Hence, in production runs the sorting time spent is much less than time gained by running the sorted particles through a number of time-steps.

VI. CONCLUSIONS

We have examined the feasibilities of using a virtual memory computer such as the VAX-11/780 on very large-scale particle simulation. We have run 1-D electrostatic codes up to 32,000 grids, 2-D electrostatic codes up to 256×256 grids and 2-D electromagnetic codes up to 128×128 grids. Larger runs are possible in principle. We find that execution rates for very large simulation can be significantly reduced if the particles are kept sorted so that physically adjacent particles have neighboring array indices.

It is not necessary, however, to keep particles completely sorted. Instead, the sort partition must be on the order of the read/write cluster size of information that is transferred between slow and fast memories. Since the particles do not have to be sorted every time-step, the time spent sorting is small compared to the time saved by executing the code with sorted particles.

The present technique should be also applicable to other virtual memory computers. It would be particularly interesting to apply this technique to future computers such as the CRAY-1XMP with very fast solid-state peripheral memory. In such an environment the present technique may become crucial to run a very large code.

Acknowledgements

Two of the authors (J.S.W. and T.T.) thank the National Science Foundation for its support through the grant NSF-ATM-81-10539. The other two authors were supported by the Department of Energy grant DE-AT06-76ER7005.
Also, the grant of the Department of Energy DE-FG05-80-ET-53088 is appreciated for the support of the present work.

The encouragement by Professor S. I. Akasofu and the help by Drs. A. Macmahon and J.N. Leboeuf are appreciated.

References

1. R. Hockney and J. W. Eastwood, Computer Simulation Using Particles, (1981, McGraw Hill, New York).
2. J. Denavit, J. Comput. Phys. 42, 337 (1981).
3. R. Mason, J. Comput. Phys. 41, 233 (1981).
4. T. Tajima and J. N. Leboeuf, Bull. Amer. Phys. Soc. 26, 986 (1981). See also D.C. Barnes and T. Kamimura, IPPJ Report #570 (1981).
5. C. Z. Cheng and H. Okuda, J. Comput. Phys. 25, 33 (1977).
6. A VAX-11 is about two orders of magnitude more inexpensive and also two orders of magnitude slower than CRAY-1.
7. C. K. Birsdall, A. B. Langdon, and H. Okuda, Methods in Computational Physics, Vol. 9, p. 241, (1970).
8. W. Kruer, J. M. Dawson, and B. Rosen, J. Comput. Phys. 13, 114 (1973).
9. D. E. Knuth, Fundamental Algorithms, (1968, Addison-Wesley, Reading, Mass.).
10. A. T. Lin, J. M. Dawson, and H. Okuda, Phys. Fluids 17, 1995 (1974).
11. A. B. Langdon and Lasinski, Methods in Computational Physics, Vol. 16, p. 327 (1976).

Figure Captions

Fig. 1 Schematic diagram of the VAX-11/780, showing how the paging process transfers information between fast and slow memory.

Fig. 2 Comparison of cpu time as a function of working set size for a 2.5-D electrostatic simulation with 64×64 grid and 128×128 particles.

Fig. 3 The cpu time as a function of working set size for a 2.5-D electrostatic simulation with 128×128 grids and 256×256 particles.

Fig. 4 The cpu time as a function of working set size for a 2.5-D electrostatic simultion with 128×128 grids and 128×128 particles.

Fig. 5 The execution rate for a 1-D electrostatic code as a function of sort partition/write cluster. The time is minimum when the sort partition equals the write cluster. The sorted code executes in 95 sec. (sum of 234). The unsorted code executes in 480 sec. (sum of 234). The simulation had 32×10^3 grids with 20 particles per cell.

Fig. 6 Page faults as a function partition/cluster for the same simulation as in Fig. 5. The number of page faults is minimized when the sort partition equals the write cluster. This graph makes it possible to estimate the usefulness of sorting on virtual memory computers other than the VAX-11/780.

Fig. 7 The cpu time as a function of sort partition/write cluster for a 1-D simulation with 8K grids and 20 particles per cell.

Fig. 8 Page faults as functions of sort partition/write cluster for the same simulation as in Fig. 7.

Fig. 9 The cpu time as a function of sort partition/write cluster for a 2.5-D electrostatic simulation with 256×256 grids and 8 particles per cell.

Fig. 10 Page faults as a function of sort partition/write cluster for the same simulation as in Fig. 9.

Fig. 11 The cpu time as a function of partition/cluster size for a 2.5-D electrostatic simulation with 128×128 grids and 8 particles per cell.

Fig. 12 Page faults as a function of partition/cluster for the same simulation as in Fig. 11.

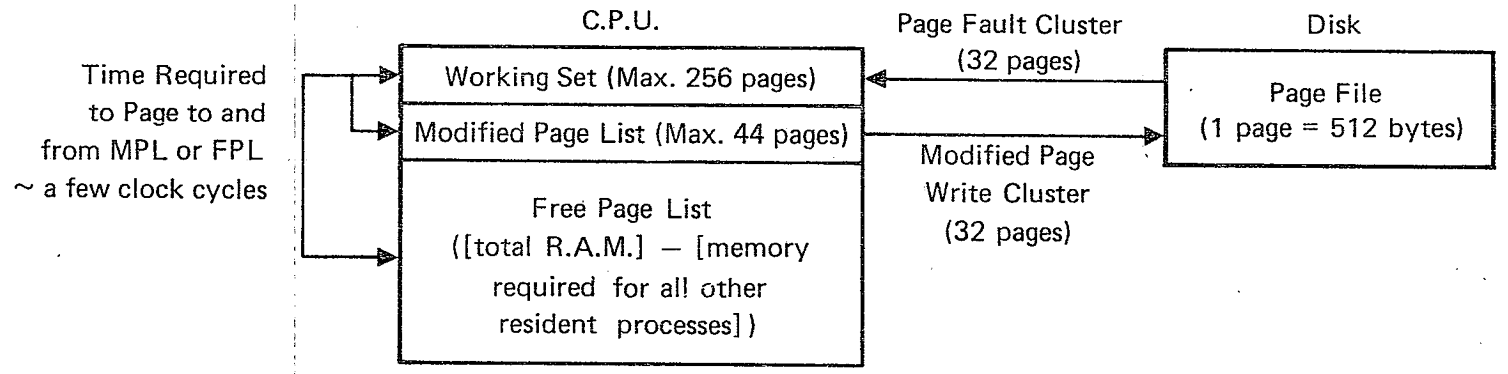
Fig. 13 The cpu time as a function of partition/cluster size for a 2.5-D electromagnetic simulation with 128×128 grid and 8 particles per cell.

Fig. 14 Page faults as a function of partition/cluster for the same simulation as in Fig. 13.

Fig. 15 The cpu time as a function of partition/cluster size for a 2.5-D electromagnetic simulation with 64×64 grids and 8 particles per cell.

Fig. 16 Page faults as a function of partition/cluster for the same simulation as in Fig. 15.

VAX 11/780 Paging*


C.P.U.
Disk
Time Required to Page to and from MPL or FPL ~ a few clock cycles
Working Set (Max. 256 pages)
Modified Page List (Max. 44 pages)
Free Page List ([total R.A.M.] – [memory required for all other resident processes])
Page Fault Cluster (32 pages)
Modified Page Write Cluster (32 pages)
Page File (1 page = 512 bytes)

Modified and Free Page Lists:

Pages are paged out of the Working Set based on *age* into either

1.) Modified Page List (MPL), if execution has altered the contents of the page
2.) Free Page List (FPL), if execution has has not altered the page

Paging to Disk:

1.) only pages in the MPL need be transfered back to disk
2.) pages being transfered to or from disk do so in Clusters of pages

Time required to page to and from disk

$= \text{(Average Access Time)} + \text{(transfer rate)} \times \text{(Cluster)}$

$\cong 38.3 \text{ m sec} + (.4 \text{ m sec/block}) \times (32 \text{ blocks})$

$\cong 51\text{m sec}$

*Numbers based on D.E.C. suggested 16-user system (VMS 2.3) with RM05 Disk drive.

FIG. 1

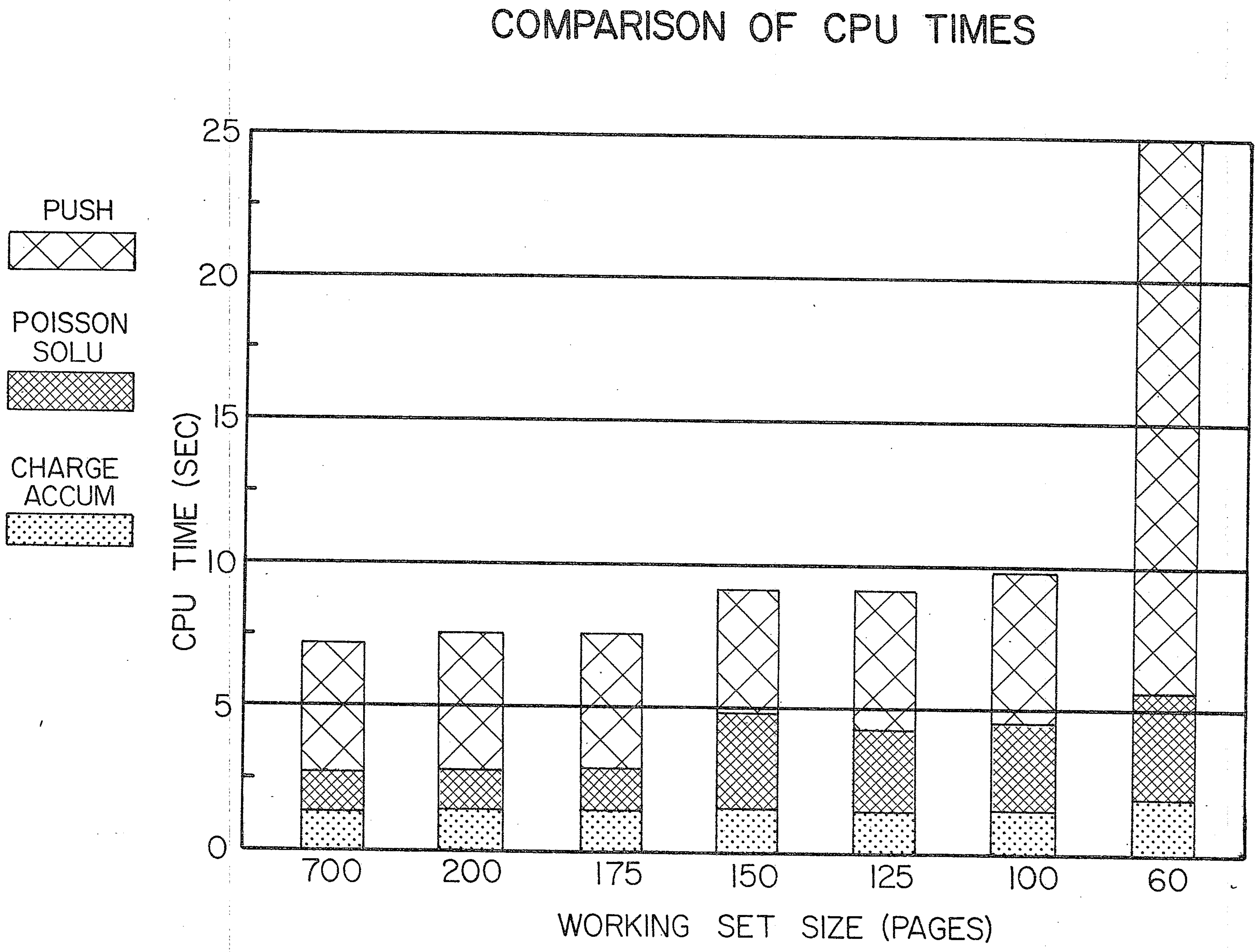

COMPARISON OF CPU TIMES
PUSH
POISSON SOLU
CHARGE ACCUM
CPU TIME (SEC)
0
5
10
15
20
25
700
200
175
150
125
100
60
WORKING SET SIZE (PAGES)

FIG. 2

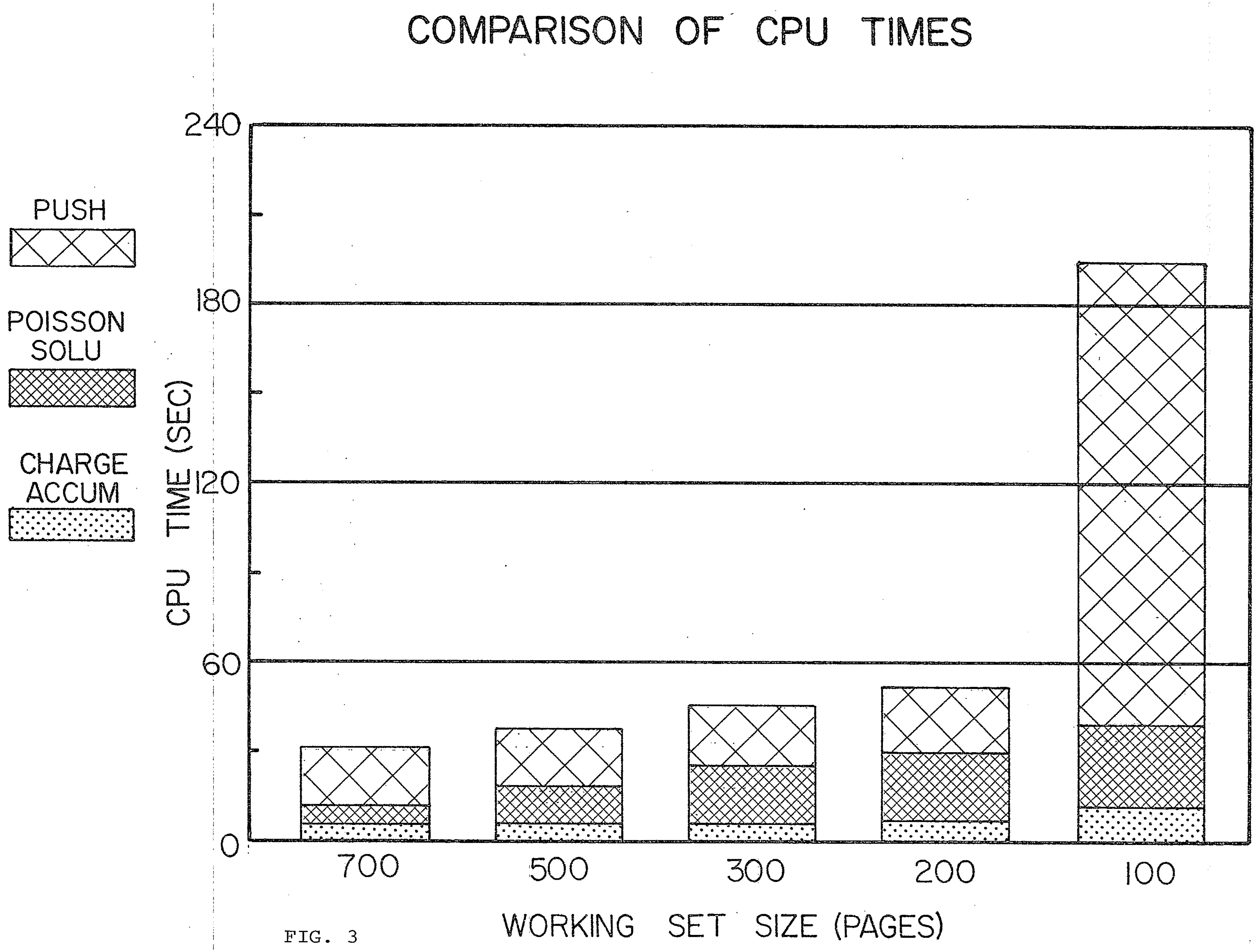

COMPARISON OF CPU TIMES
PUSH
POISSON SOLU
CHARGE ACCUM
CPU TIME (SEC)
240
180
120
60
0
700
500
300
200
100
WORKING SET SIZE (PAGES)

FIG. 3

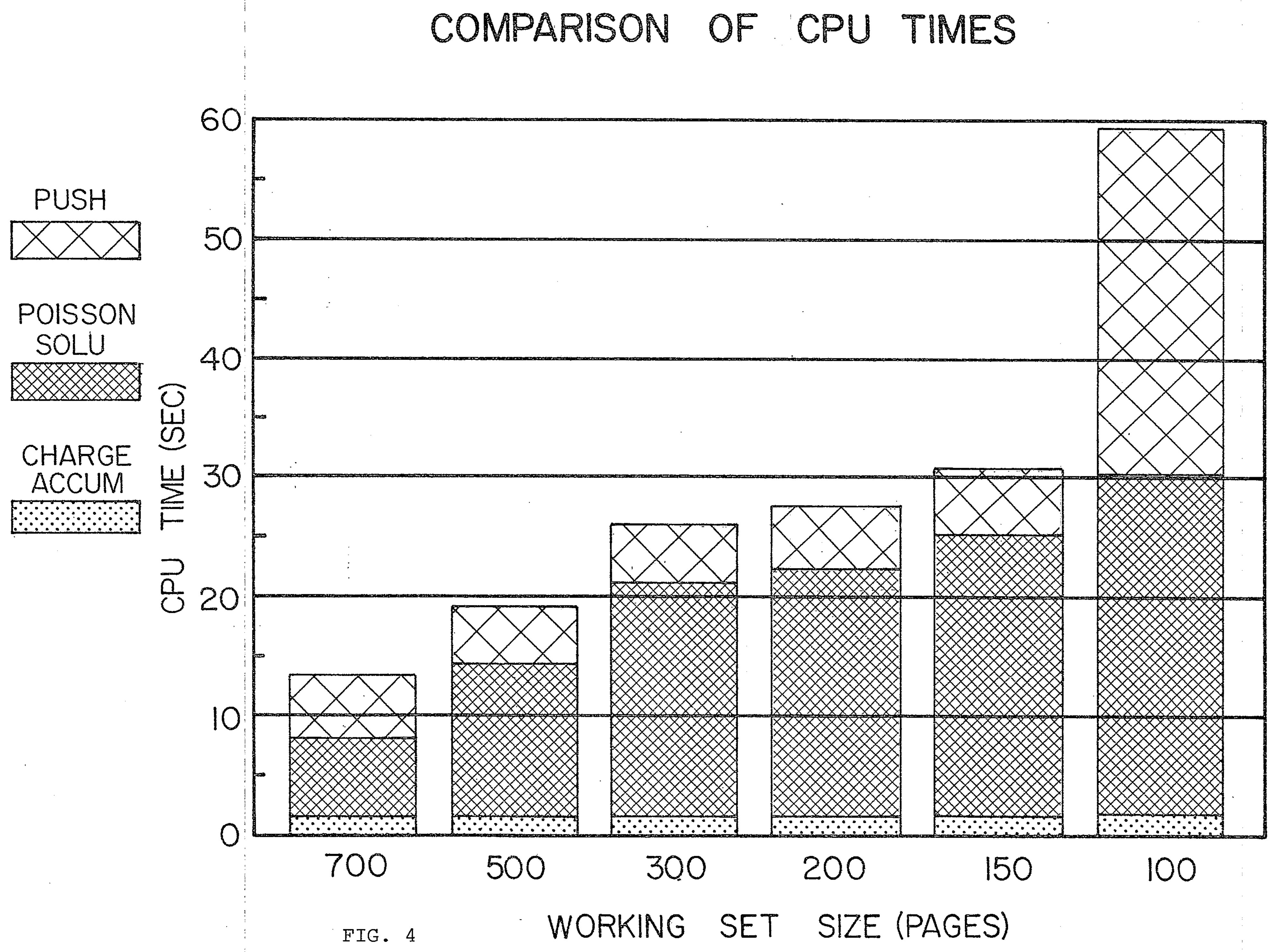

COMPARISON OF CPU TIMES
PUSH
POISSON SOLU
CHARGE ACCUM
CPU TIME (SEC)
0
10
20
30
40
50
60
700
500
300
200
150
100
WORKING SET SIZE (PAGES)

FIG. 4

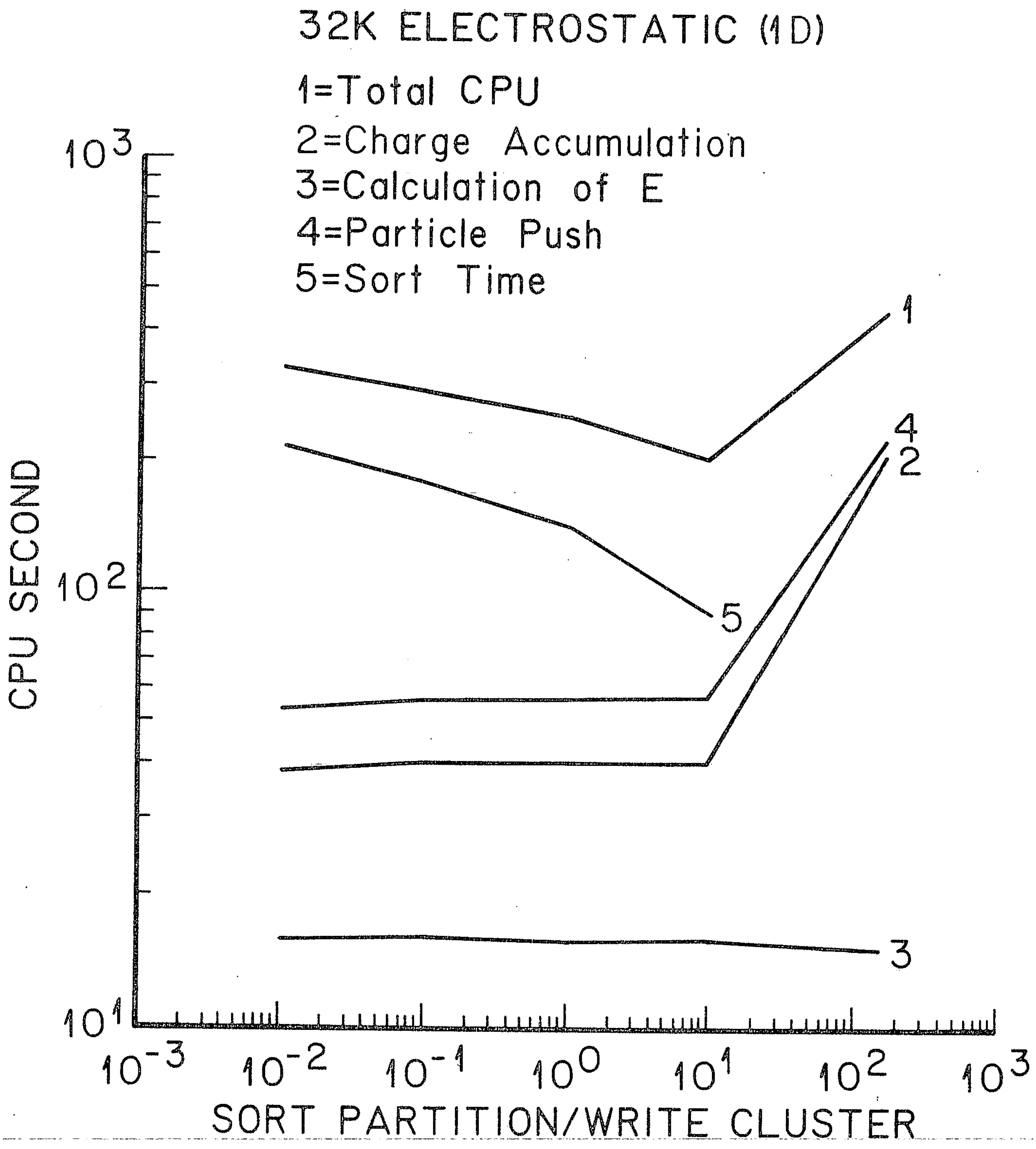
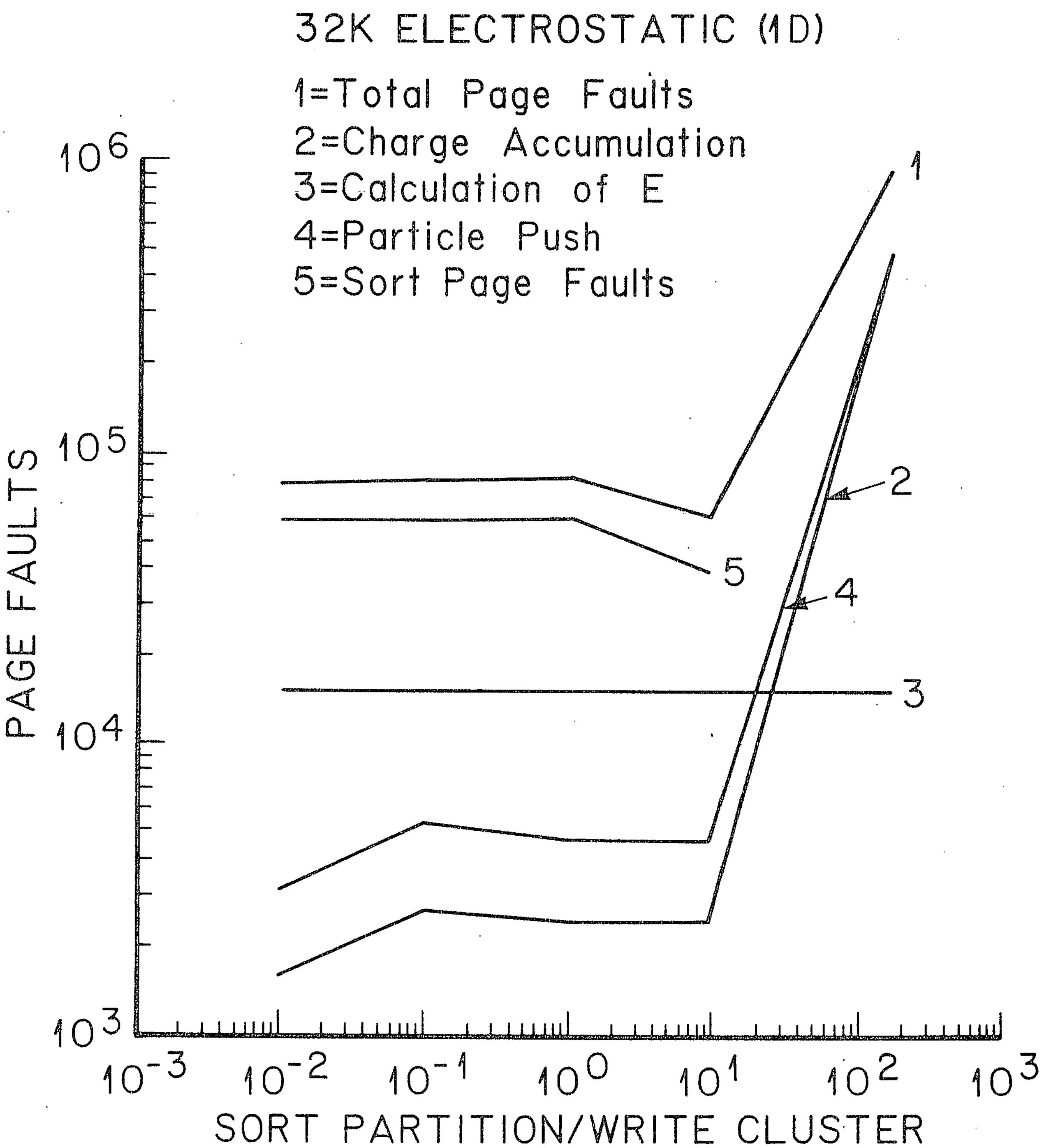

32K ELECTROSTATIC (1D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Time
CPU SECOND
10^3
10^2
10^1
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
10^3
SORT PARTITION/WRITE CLUSTER
1
4
2
5
3

FIG. 5



32K ELECTROSTATIC (1D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Page Faults
PAGE FAULTS
SORT PARTITION/WRITE CLUSTER
10^3
10^4
10^5
10^6
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
1
2
3
4
5

FIG. 6

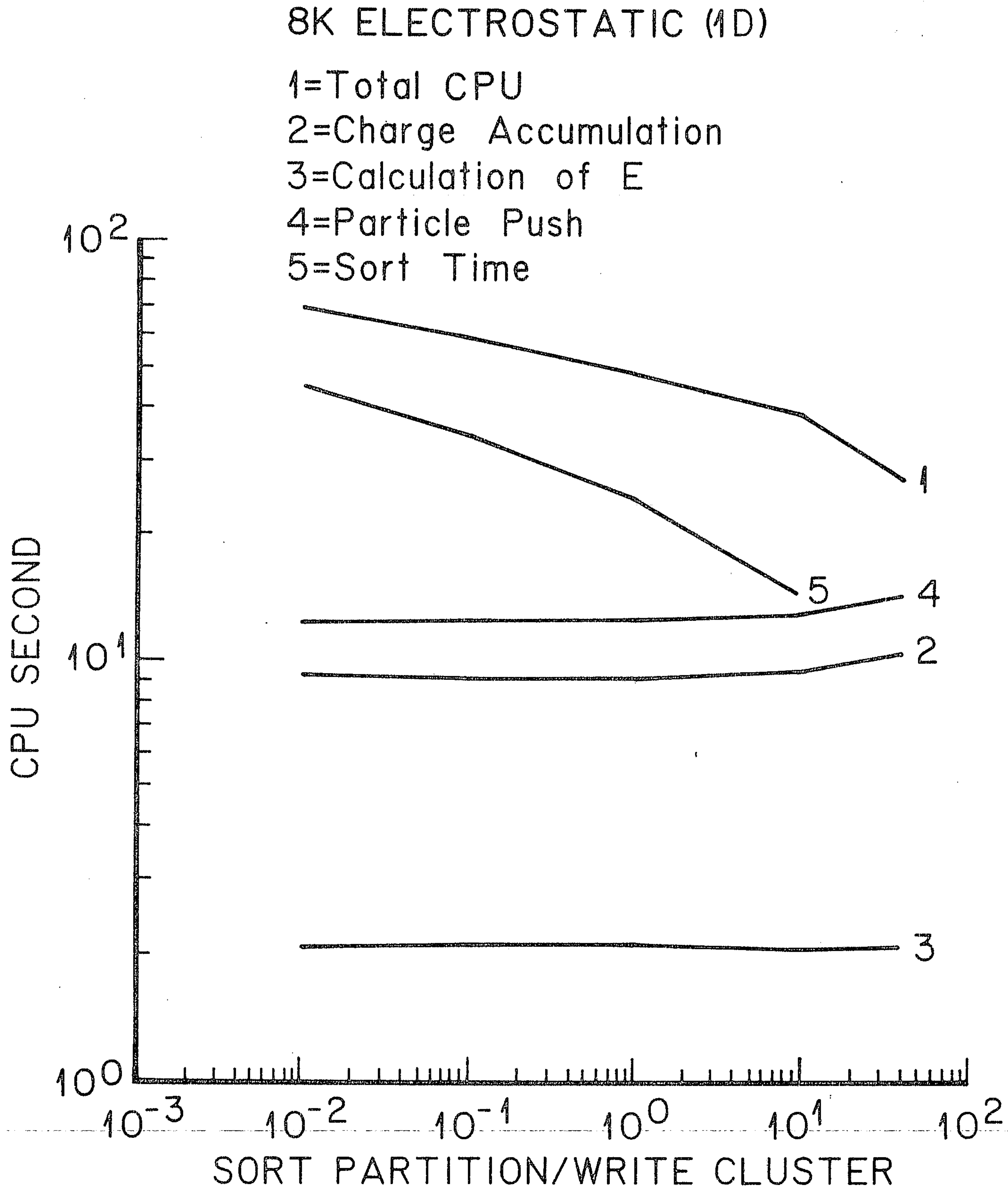

8K ELECTROSTATIC (1D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Time
CPU SECOND
10^2
10^1
10^0
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
2
3
4
5

FIG. 7

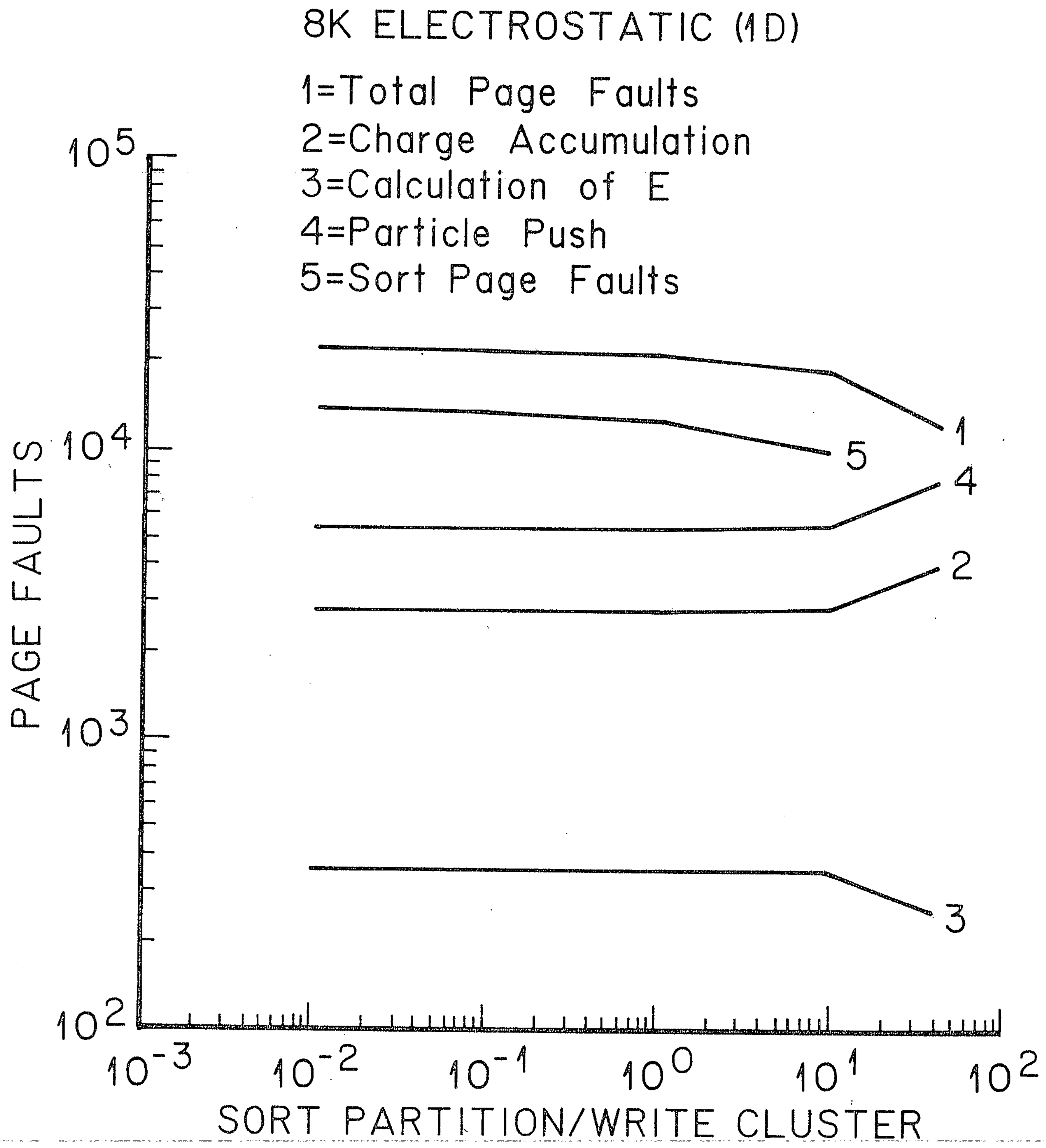

8K ELECTROSTATIC (1D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Page Faults
PAGE FAULTS
10^5
10^4
10^3
10^2
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
2
3
4
5

FIG. 8

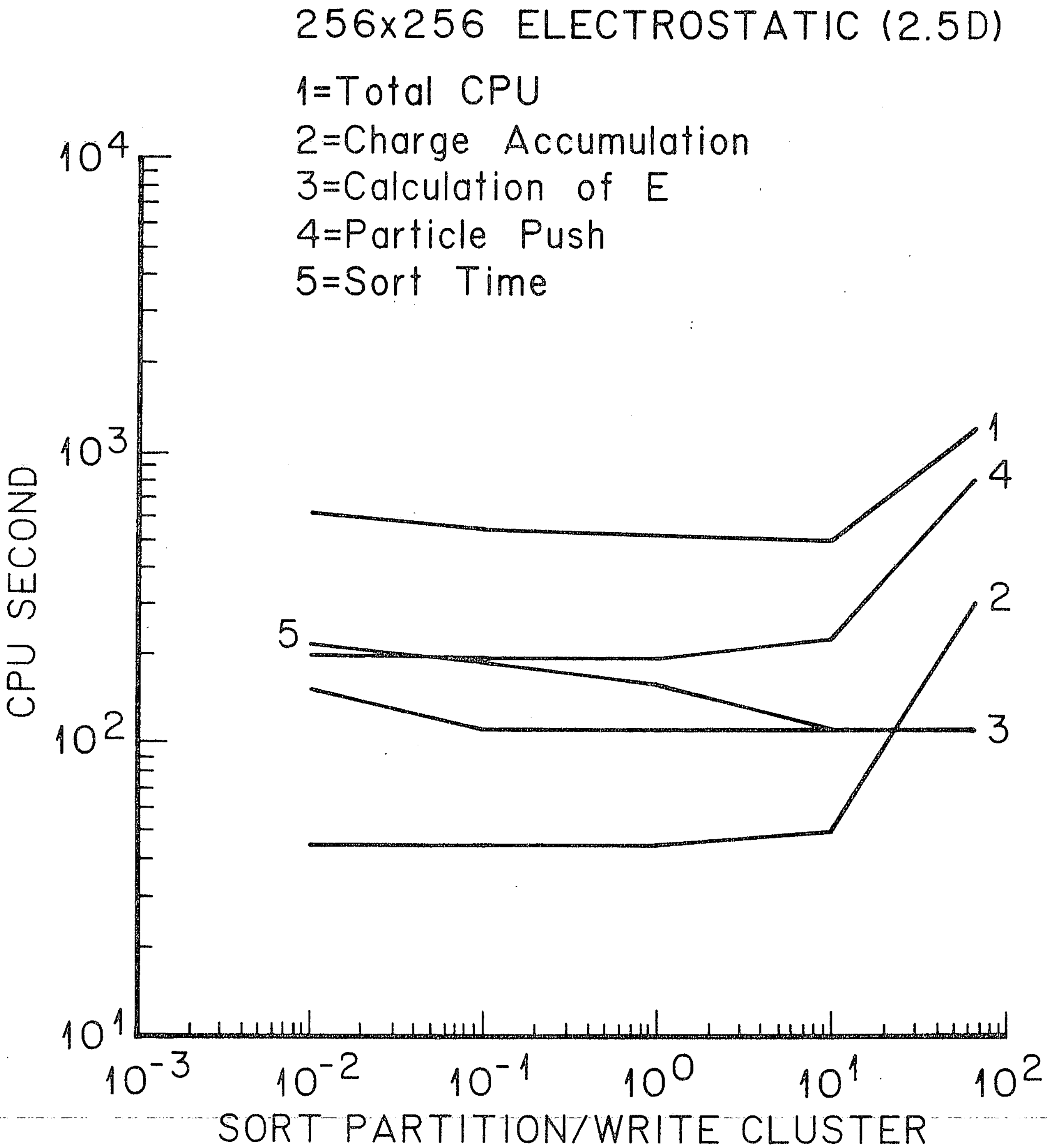

256x256 ELECTROSTATIC (2.5D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Time
CPU SECOND
10^4
10^3
10^2
10^1
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
4
2
5
3

FIG. 9

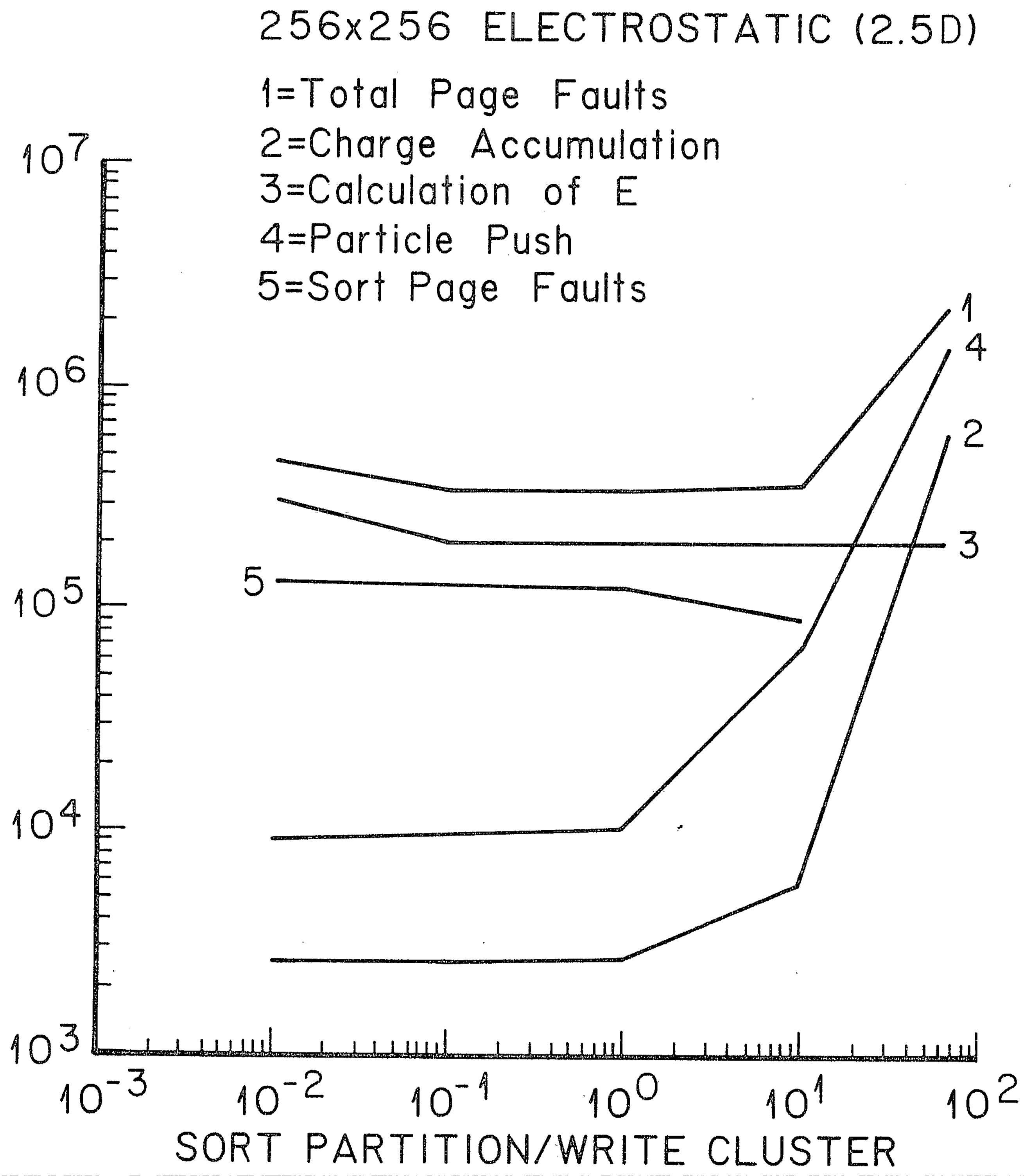

256x256 ELECTROSTATIC (2.5D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Page Faults
PAGE FAULTS
10^7
10^6
10^5
10^4
10^3
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
2
3
4
5

FIG. 10

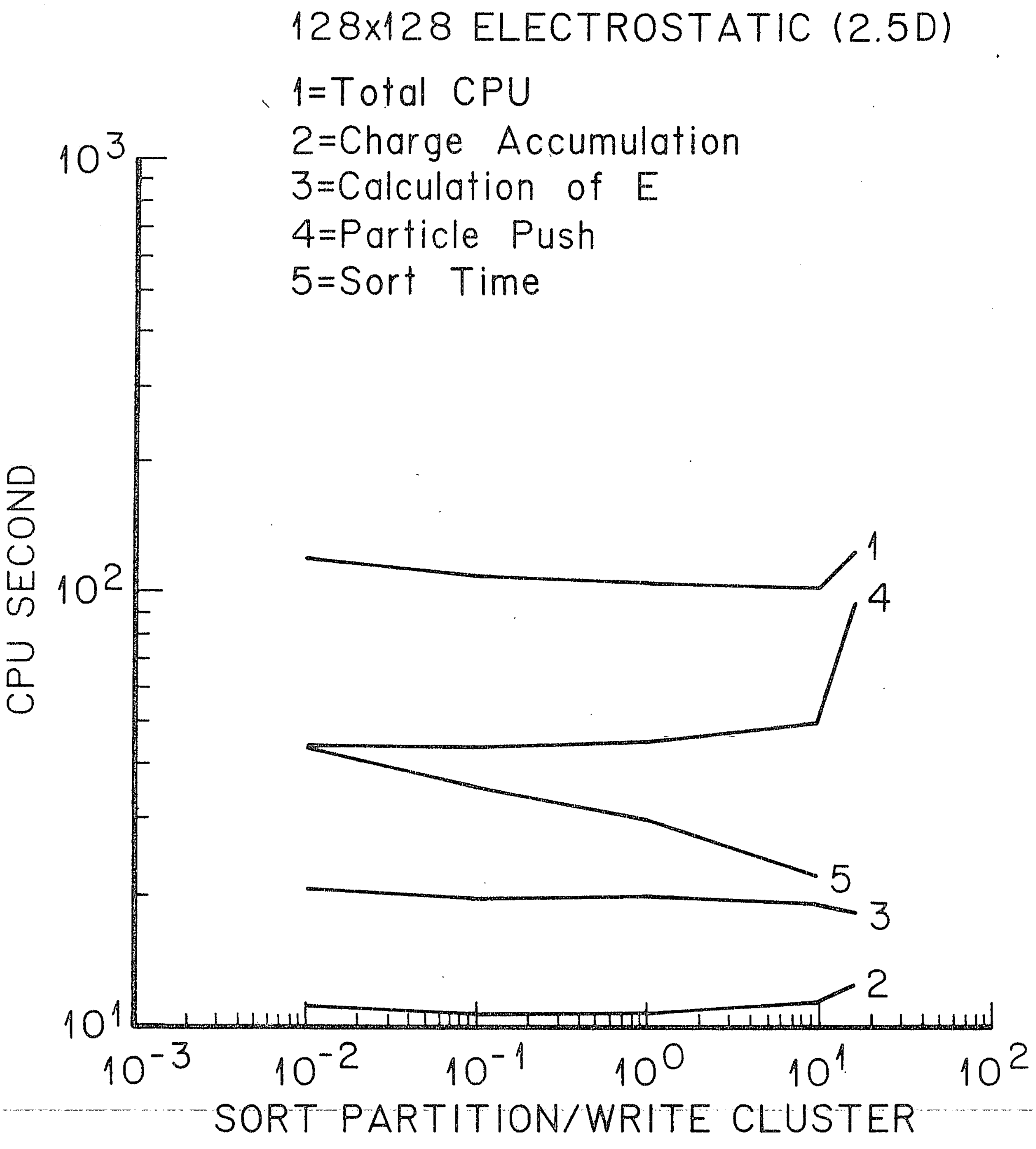

128x128 ELECTROSTATIC (2.5D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Time
CPU SECOND
10^3
10^2
10^1
10^-3
10^-2
10^-1
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
4
5
3
2

FIG. 11

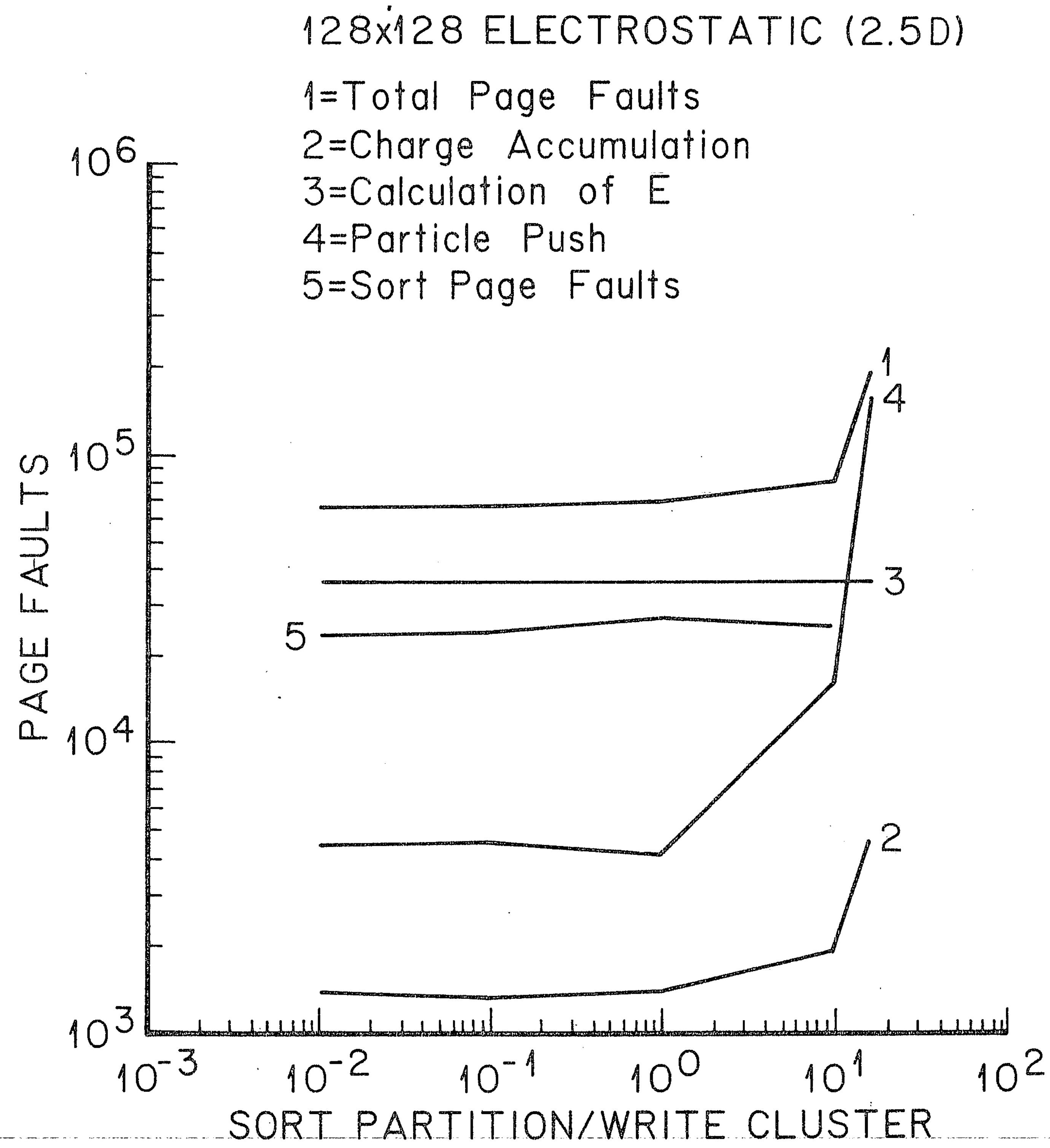

128x128 ELECTROSTATIC (2.5D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E
4=Particle Push
5=Sort Page Faults
PAGE FAULTS
10^6
10^5
10^4
10^3
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
2
3
4
5

FIG. 12

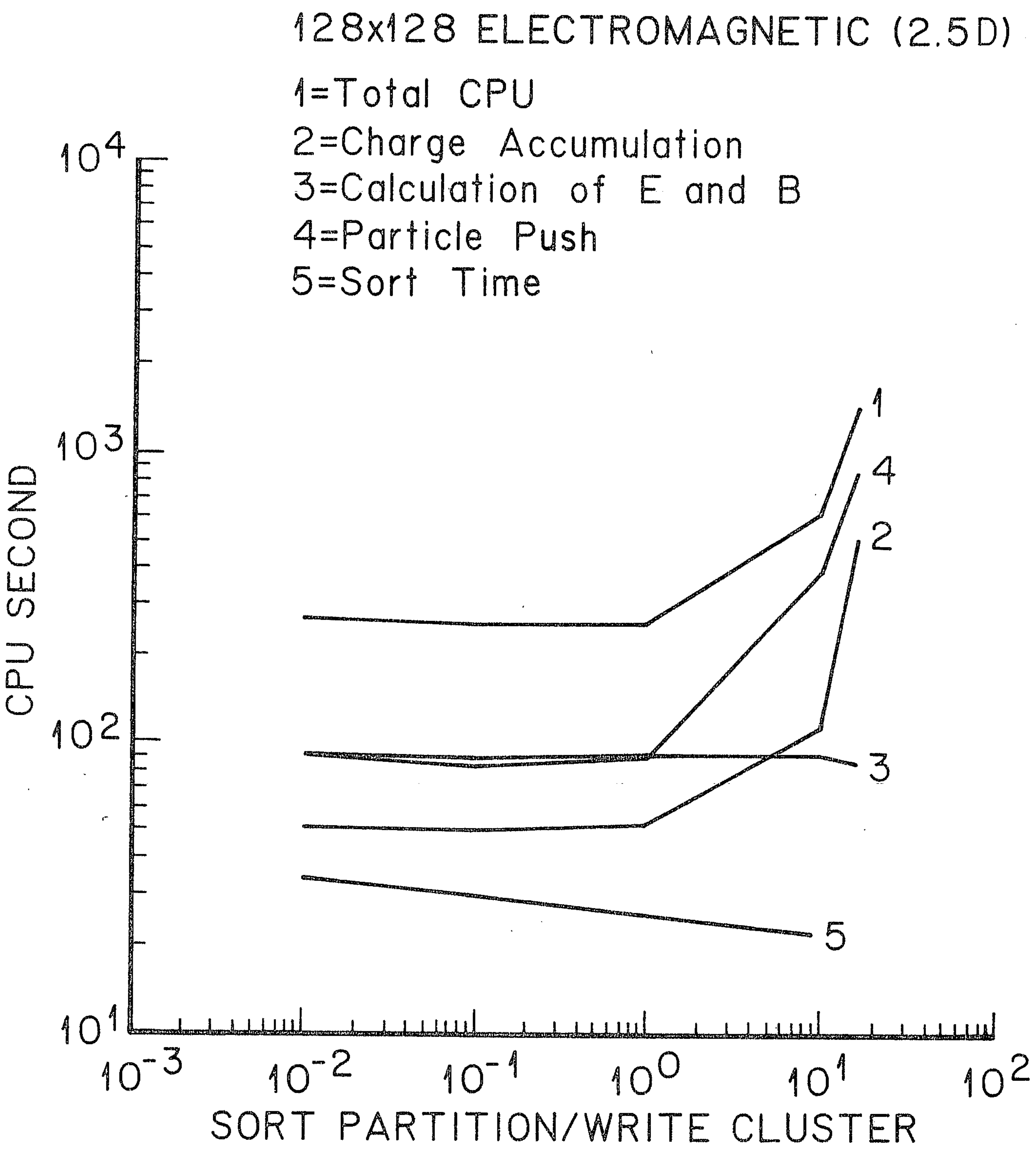

128x128 ELECTROMAGNETIC (2.5D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E and B
4=Particle Push
5=Sort Time
CPU SECOND
10^4
10^3
10^2
10^1
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
4
2
3
5

FIG. 13

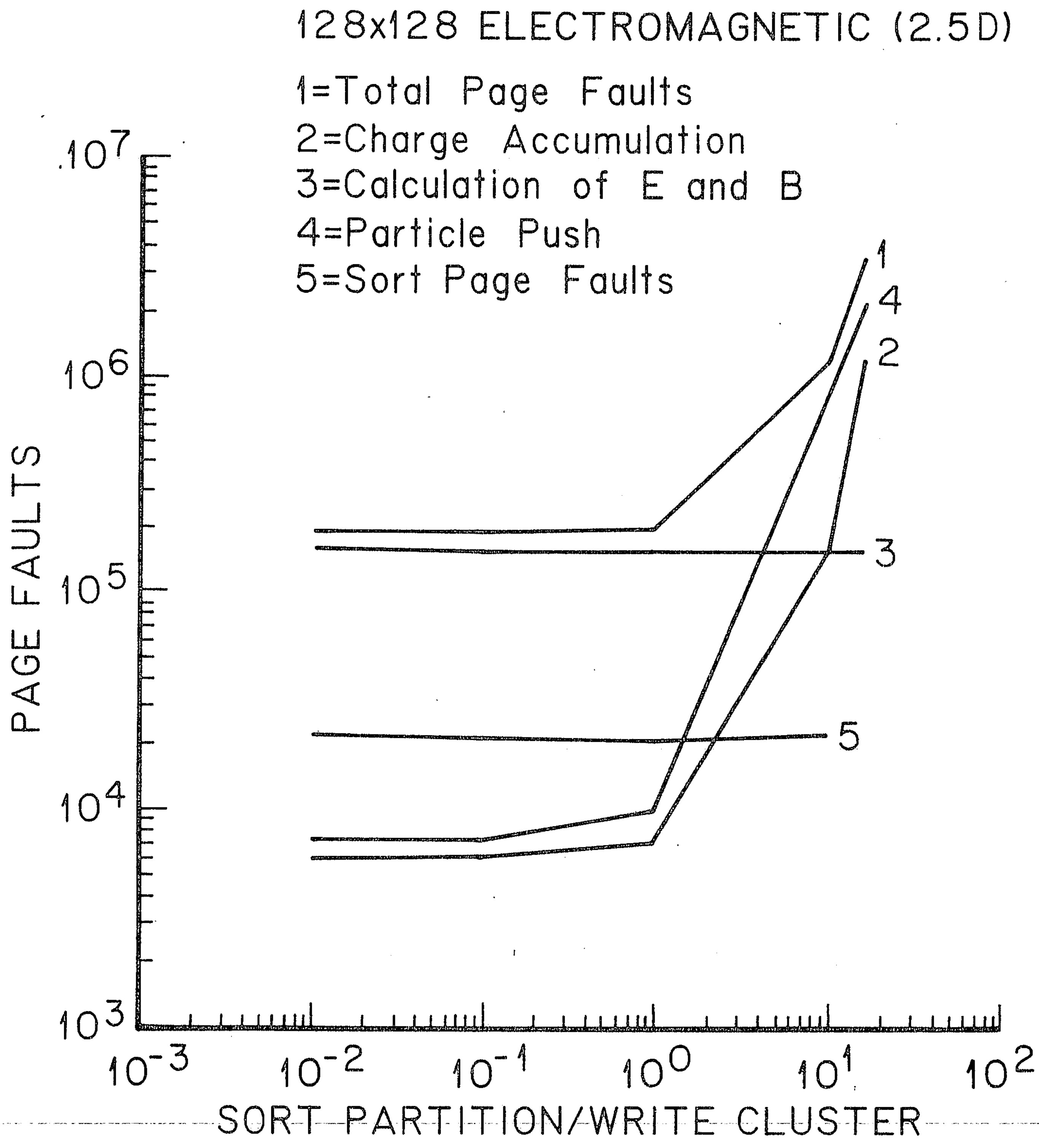

128x128 ELECTROMAGNETIC (2.5D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E and B
4=Particle Push
5=Sort Page Faults
PAGE FAULTS
10^7
10^6
10^5
10^4
10^3
10^-3
10^-2
10^-1
10^0
10^1
10^2
SORT PARTITION/WRITE CLUSTER
1
4
2
3
5

FIG. 14

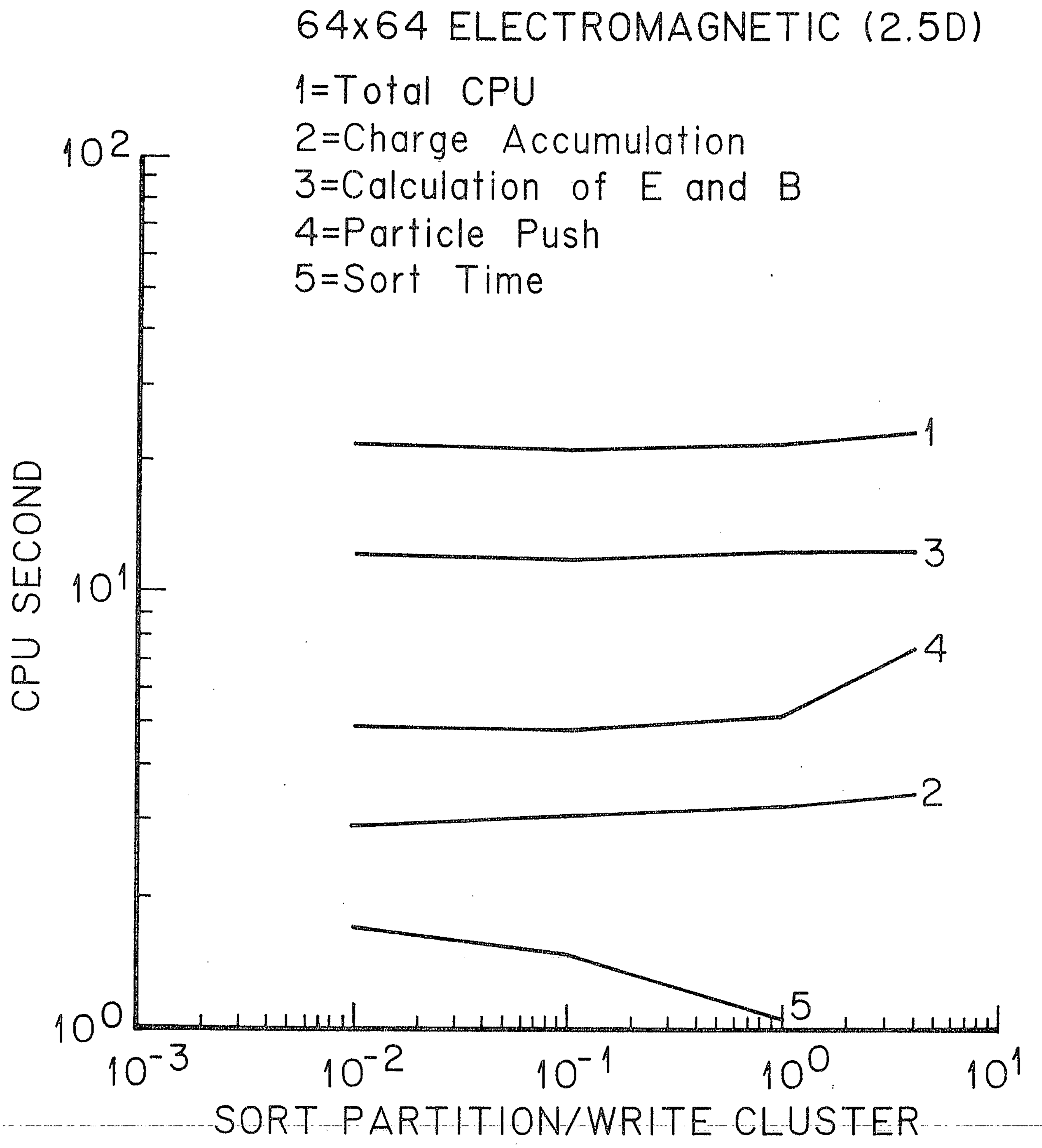

64x64 ELECTROMAGNETIC (2.5D)
1=Total CPU
2=Charge Accumulation
3=Calculation of E and B
4=Particle Push
5=Sort Time
CPU SECOND
10^2
10^1
10^0
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
SORT PARTITION/WRITE CLUSTER
1
3
4
2
5

FIG. 15

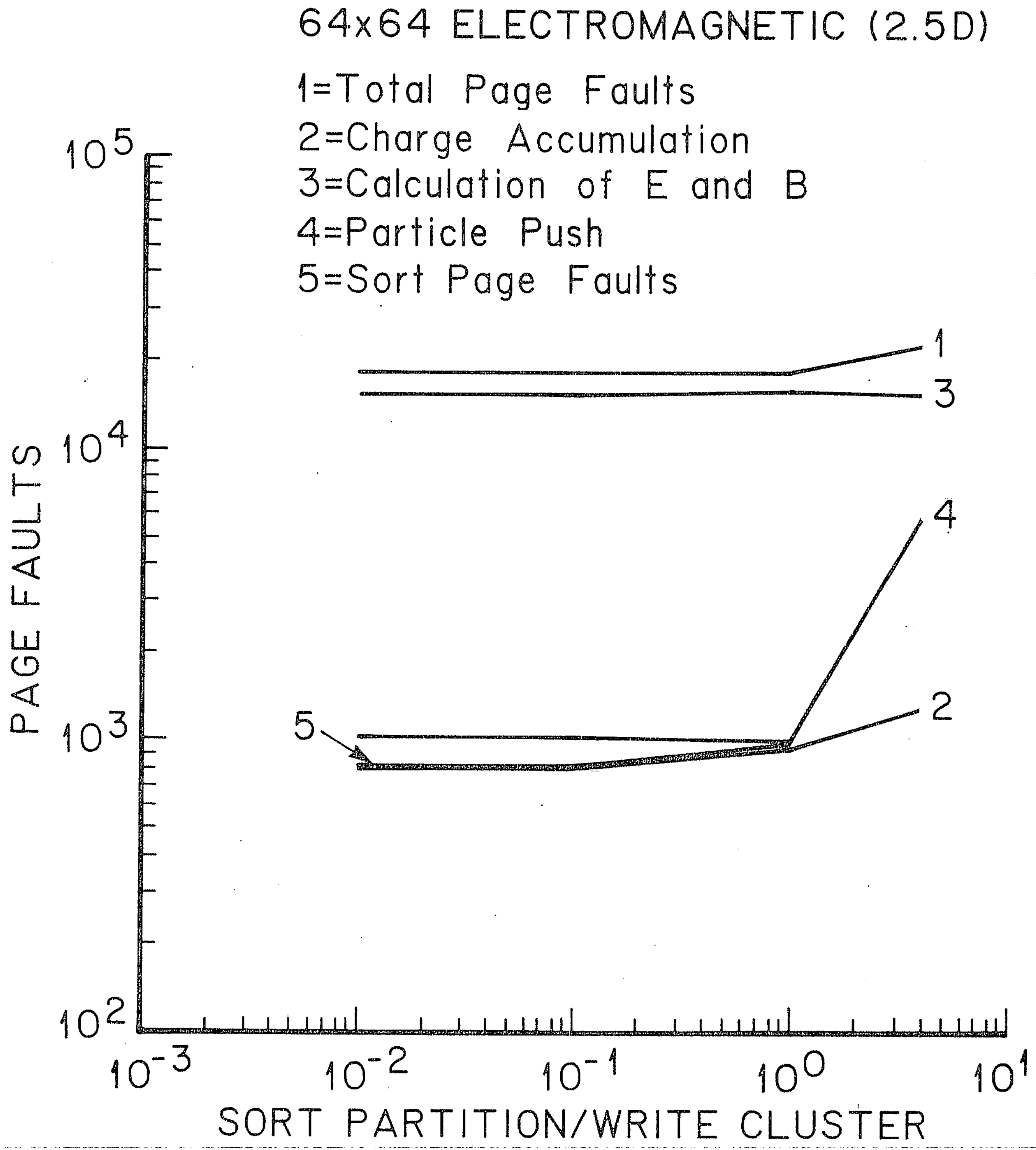

64x64 ELECTROMAGNETIC (2.5D)
1=Total Page Faults
2=Charge Accumulation
3=Calculation of E and B
4=Particle Push
5=Sort Page Faults
1
3
4
2
5
PAGE FAULTS
10^5
10^4
10^3
10^2
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
SORT PARTITION/WRITE CLUSTER

FIG. 16