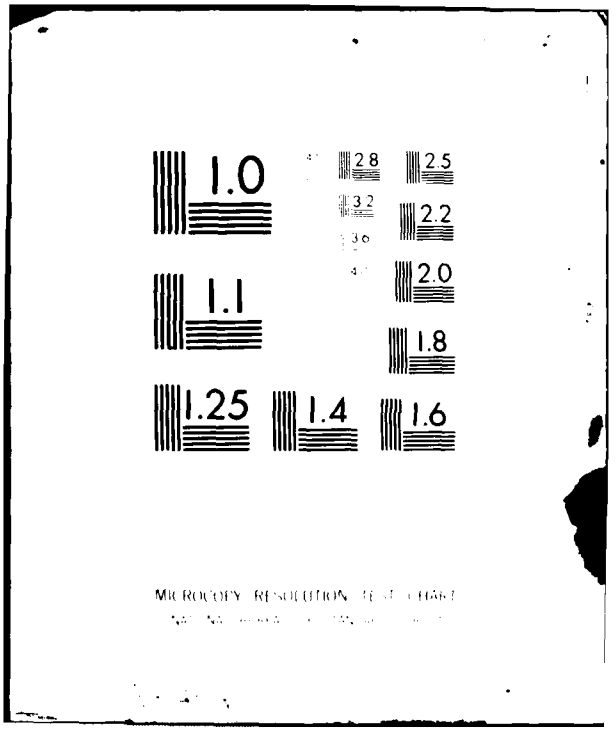
AD-A098 354 ILLINOIS INST OF TECH CHICAGO F/G 9/2
THE COST EFFECTIVENESS OF MINICOMPUTERS VERSUS MAIN FRAMES IN T--ETC(U
MAR 81 A K VON MAYRHAUSER, R T HAFTKA N00014-80-C-0364
UNCLASSIFIED NL


1.0
2.8
2.5
3.2
2.2
3.6
2.0
1.1
1.8
1.25
1.4
1.6

LEVEL II

AD A098354

THE COST EFFECTIVENESS OF MINICOMPUTERS VS. MAIN FRAMES IN THE SOLUTION OF STRUCTURAL ANALYSIS PROBLEMS, INTERIM TECHNICAL REPORT

Illinois Institute of Technology, Chicago, Illinois 60616

by

A.K. Von Mayrhauser

Raphael T. Haftka

and Martha Evens

Illinois Institute of Technology

Chicago, Il. 60616

Sponsored by Office of Naval Research Contract No N00014-80-C-0364

March 1981

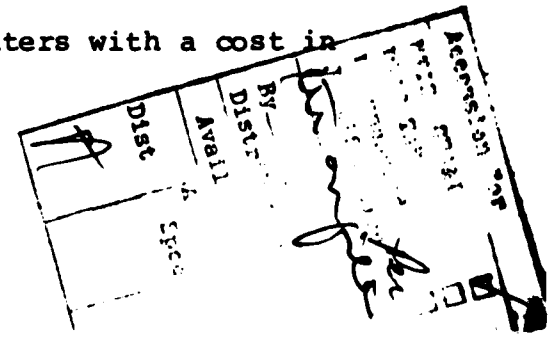
Abstract

A study on the cost effectiveness of mini computer vs. main-frames is being conducted at the Illinois Institute of Technology. The study is intended to compare the performance of several finite element programs on the IIT Prime 400 mini computer and the United Computing Systems UNIVAC 1100/81 main frame. Additional comparisons are to be made with other main frame and minicomputer installations. The present report describes the methodology used in the study and the matrix of computer programs and structural analysis problems. Also, preliminary results for one program and one problem are given and analysed.

I. INTRODUCTION

Minicomputers promise cheaper, more widely available computing facilities, but they pose many problems, particularly to those with large calculations in mind. The smaller main memory means that users must make more use of disk space. Many minicomputers have a smaller word size (8 or 16 bits per word) with a devastating effect on accuracy. (Storaasli and Foster, 1978, report 4 digit accuracy on a PRIME 400 for a medium-sized problem as compared to 8-13 digit accuracy on a CDC CYBER 173.) Longer elapsed times leave more room for machine hardware failures during a run. At Berkeley, Pearson reports some theoretical calculations required twenty-four hours or more, but the (MTBF) mean time between failures was also approximately twenty-four hours. Lack of adequate software libraries, debugging facilities, documentation and operations staff often hampers users of minicomputers.

Several past studies of the performance of minicomputers and main frames have concentrated on the machine cost aspect. Chemists have taken the lead in doing large scale calculations on small machines, perhaps because of their familiarity with the use of minicomputers for laboratory process control. Peter Lykos of the Illinois Institute of Technology organized an American Chemical Society Symposium on Minicomputers and Large Scale Computations in June, 1977. Four of the papers from this meeting are particularly relevant to the present study since they involve large minicomputers with a cost in $100,000 - $200,000 range:



Pearson et. al.,; Norbeck and Certain; Wagner et. al.; and Freuler and Petrie. These papers have shown that based on machine cost alone minicomputers may be two to four times more cost effective than main frames. However, the National Resource for Computation in Chemistry Committee has concluded that "Minicomputers alone cannot meet the needs of the chemistry community" (1978, p.1).

The structural analysis community is also experiencing a shift from main frames to minicomputers (e.g., Swanson, 1979). For example, most of the major finite element packages are now available on the PRIME 400.

The users of structural analysis software on minicomputers have conducted several studies that point to the economic advantages of using such machines (e.g., Storaasli and Foster, 1978, Conaway 1979). These are based on data gathered by running structural analysis programs for various problems. The scope of the comparison was limited by the relatively small amount of data that was generated.

The purpose of the present study is to extend the previous studies in several directions. First, to use a wider mix of programs and problems for the comparison. Second, to produce performance models of the computer programs that will permit prediction of performance for additional problems and different computer environments. Last, the present study also attempts to further assess the human factors involved in such comparison.

Pearson et. al.,; Norbeck and Certain; Wagner et. al.; and Freuler and Petrie. These papers have shown that based on machine cost alone minicomputers may be two to four times more cost effective than main frames. However, the National Resource for Computation in Chemistry Committee has concluded that "Minicomputers alone cannot meet the needs of the chemistry community" (1978, p.1).

The structural analysis community is also experiencing a shift from main frames to minicomputers (e.g., Swanson, 1979). For example, most of the major finite element packages are now available on the PRIME 400.

The users of structural analysis software on minicomputers have conducted several studies that point to the economic advantages of using such machines (e.g., Storaasli and Foster, 1978, Conaway 1979). These are based on data gathered by running structural analysis programs for various problems. The scope of the comparison was limited by the relatively small amount of data that was generated.

The purpose of the present study is to extend the previous studies in several directions. First, to use a wider mix of programs and problems for the comparison. Second, to produce performance models of the computer programs that will permit prediction of performance for additional problems and different computer environments. Last, the present study also attempts to further assess the human factors involved in such comparison.

The present preliminary report describes the computer programs, the selected mix of problems and the evaluation tools developed for measuring performance. Results for one of the test problems are reported and analysed.

II. CHOICE OF STRUCTURAL ANALYSIS PROGRAMS AND PROBLEMS

Since the overwhelming majority of structural analysts use finite element methods, the present study is limited to such programs.

It is impossible to get a very accurate comparison of the performance of the mini and main frame systems without an exhaustive battery of test problems. However, we believe that by judicious choice of the type of problems and the type of structural analyses employed, one can obtain a reasonably reliable comparison of performance. The following are the elements in such a comparison.

A. Type of Analysis

The type of structural analysis to be included in the tests must include the ones which are most commonly in use. This study will include the following:

(i) Linear static solution for displacement and stresses

(ii) Linear eigenvalue analysis - calculation of buckling loads and vibration modes

(iii) Nonlinear response due to large deformations and material nonlinearities

(iv) Transient Dynamic analysis

B. Type of Problem

Three structural problems are used:

1. A simple cantilever beam
2. A plate with a hole
3. A stiffened cylinder

These problems will be solved for the different analysis types. Several models will be used for each problem ranging from a very crude model to a very refined one. The number of degrees of freedom ranging from a few dozen for a crude model to more than a thousand for the refined one.

C. Computer Programs

Three computer programs are used with the objective of using two programs for each problem. These programs are:

1. SAP IV - A general purpose finite element program developed at the University of California at Berkeley is probably the most widely used "free" (it costs $200.) finite element code
2. SPAR - A general purpose finite element program developed by W. D. Whetstone which serves as a prototype of a commercially developed code
3. TWODEL - A special purpose finite element program developed by D. Malkus at IIT for large deformation analysis of 2 dimensional elasticity problems. It is a representative of in house codes.

III. THE COMPUTER PROGRAMS

SPAR

SPAR is a general purpose finite element program developed by W.D. Whetstone for NASA, first at Lockheed and then at EISI. The program has linear static analysis, eigensolutions for vibration and buckling and modal response capabilities. It does not have nonlinear analysis or direct integration capabilities. The program has a public version which is distributed by the government through COSMIC. It also has a proprietary version called EAL (for Engineering Analysis Language). The public versions for the UNIVAC and CDC systems are maintained by the developer while the minicomputer versions (Prime and VAX) are maintained by NASA. The proprietary version is available on all four systems but only in its executable version. Because of the expense of acquiring the proprietary version and the difficulty of instrumenting it without access to the source code, the public version was used for this study.

SPAR is a modular system composed of more than 20 small programs called processors. The processors communicate through a data base system which is also directly accessible to the user. (see Giles and Haftka, 1978, for more information). The list of the SPAR processors and their functions is given in Table 3.1.

The public version of SPAR was installed on the UNIVAC without any trouble. Installation on the Prime was much

Table 3.1 - SPAR PROCESSORS

Processor Name	Function
TAB	Creates data sets containing tables of joint locations, section properties, material constants, etc.
ELD	Defines the finite elements making up the model
E	Generates sets of information for each element, including connected joint numbers, geometrical data, material and section property data
EKS	Adds the stiffness and stress matrices for each element to the set of information produced by the E processor
TOPO	Analyzes element interconnection topology and creates data sets used to assemble and factor the system mass and stiffness matrices
K	Assembles the unconstrained system stiffness matrix in a sparse format
M	Assembles the unconstrained system mass matrix in a sparse format
KG	Assembles the unconstrained system initial-stress (geometric) stiffness matrix in a sparse format
INV	Factors the assembled system matrices
EQNF	Computes equivalent joint loading associated with thermal, dislocational, and pressure loading
SSOL	Computes displacements and reactions due to loading applied at the joints

Table 3.1. - Concluded

Processor Name	Function
GSF	Generates element stresses and internal loads
PSF	Prints the information generated by the GSF processor
EIG	Solves linear vibration and bifurcation buckling eigenproblems
DR	Performs a dynamic response analysis
SYN	Produces mass and stiffness matrices for systems comprised of interconnected substructures
STRP	Computers eigenvalues and eigenvectors of substructured systems
AUS	Performs an array of matrix arithmetic functions and is used in construction, editing, and modification of data sets
DCU	Performs an array of data management functions including display of table of contents, data transfer between libraries, changing data set names, printing data sets, and transferring data between libraries and sequential files
VPRT	Performs editing and printing of data sets which are in the form of vectors on the data libraries
PLTA	Produces data sets containing plot specifications
PLTB	Generates the graphical displays which are specified by the PLTA processor

more troublesome. Difficulties were due to some bugs in the Prime version that had to be corrected and with the virtual memory system of the Prime that did not seem to work well for very large arrays. Additionally, the Prime company issues new releases of the operating system quite often. Many times the older version of SPAR did not work with the newer operating system and the programs had to be recompiled and reloaded (a non trivial effort because SPAR is composed of so many individual programs).

SAPIV

SAPIV is also a general purpose finite element program that has static, vibration and dynamic analysis capabilities. It has been developed by Wilson at Berkeley and is available at nominal cost ($200.) to the public. It is probably the most widely used "free" finite element program. There are more advanced versions of SAP denoted as SAPV, SAPVI, etc. which are available at considerable cost ($9,000) from the University of Southern California. In the present study SAPIV is used.

The program was originally written for a CDC system. However, it has been converted to other systems. On the UCS UNIVAC 1100/81, there are three versions of SAP IV. However only one of these is working. The program can be compiled using UNIVAC's FORTRAN V compiler. An attempt to use the more efficient ASCII FORTRAN compiler was unsuccessful. A substantial change in the program and I/O format statements may be required to make SAP IV suitable for this compiler.

There also is an absolute version of SAP IV which is supported by UCS and is available for problem solution

The Prime version of SAP IV was generated by Feeser of RPI. The first version of SAP IV which we received from the PRIME users' library was highly mutilated. Another tape was then obtained from RPI, Troy, after several months wait. This tape had about 60 lines of code missing towards the end of the STRETR SUBROUTINE. Two following routines were also destroyed. Luckily this particular piece of code was correct on the earlier tape, and we were able to patch the code to mak it work. Both times we received 800 BPI tapes. The PRIME installation at IIT has only a 1600 tape drive, however. The conversion was another nontrivial task.

TWODEL

TWODEL is a special purpose finite element program for two dimensional finite elasticity developed by Malkus at IIT. It is a relatively small program and was developed simultaneously for the UNIVAC and PRIME systems. The problem description is built into the program so that refining the mesh necessitates changes to program patches.

IV. MACHINE ASPECTS

1. PRIME 400

This section describes some of the aspects of the PRIME operating system PRIMOS (Rev. 17.2). It deals with memory management, page fault handling, and process scheduling. Collecting this information was much more cumbersome than anyone had anticipated, mainly because the kind of information we wanted is not needed in day-to-day operation. Therefore, it was not available either from the IIT Academic Computing office or from PRIME's Oakbrook headquarters. The problem was aggravated due to unavailability of proper documentation. Thus, many times information received from these sources was either incomplete or incorrect or both. This eventually forced us to investigate the PRIME Macro Assembler code of PRIMOS.

a) Memory Management

PRIME has a segmented, paged virtual memory system. The page size is 1024 (16 bit) words. The segment size is 0 to 65536 (64K) words in units of 1024 words (page size). There are 4096 segments to a virtual space (2**28 words). The segments are in four groups of 1024 segments each. There are four descriptor table address registers which point to tables containing page map entries. These in turn point to physical pages of memory. Thus, a 28-bit virtual address contains 2 bits of descriptor table selection, 10 bits of segment selection, and 16 bits of word selection. The process of physical address translation is given in Figure 4.1.

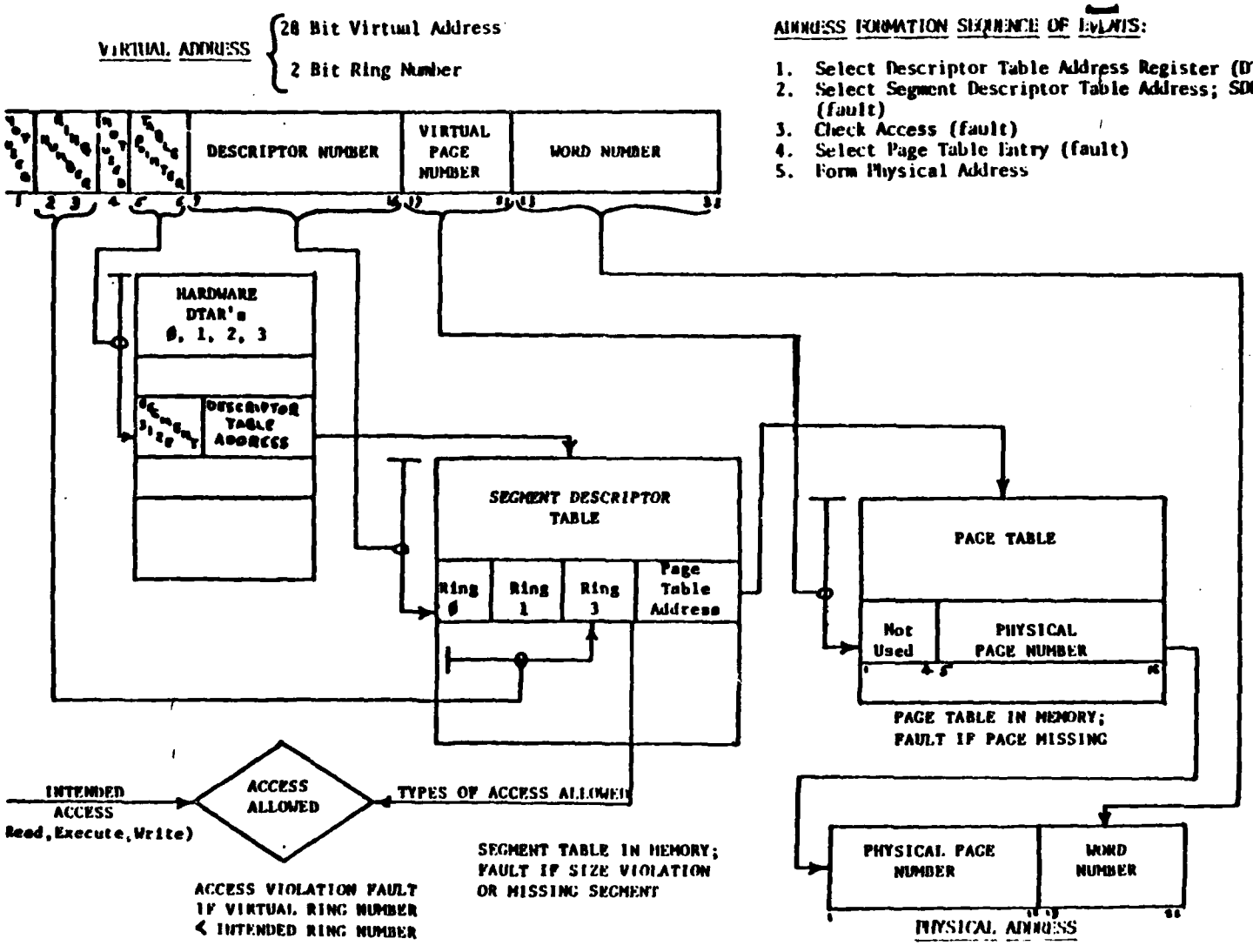


Figure 4.1 Physical Address Formation

It should be noted that the hardware implemented automatic process-exchange mechanism does not affect the descriptor table address registers 0 and 1, therefore, all processes share the same first 2K segments of virtual address space and have the second 2K segments as private space. Finally, the presence of both paging and segmentation permits the separation of physical memory management from user address space management. Table 4.1 shows the formats of descriptor table address registers, segment descriptor words, and page map entries.

A descriptor table has from 0 to 1K entries, must begin on an even word, and must not cross a segment boundary. A page table always has 64 entries and must not cross a 64K boundary. Pages must begin on a 1K word boundary. To facilitate memory management, primos maintains three kinds of map-tables. These are the page table, the disk table and the memory map table. The first two are 64 words long and are maintained separately for each segment. The memory map table is a system table with one word per physical page. At our system, it is 1K words long (we have 1M bytes - 512K words of memory).

Page faults are handled in microcode and regular assembly/FORTRAN software. Faults are detected and an entry is made into the fault table. If it is a page fault, a branch is made to the page fault catcher which saves registers and finally a routine is invoked which turns the page. After the page has been brought in, the system wide page fault

Table 4.1 Virtual Memory formats

DESCRIPTOR TABLE ADDRESS REGISTER FORMAT
(32 bits)

SSSSSSSSSSDDDDDD
-DDDDDDDDDDDDDD

1-10: 1024 minus descriptor table size (SSS...S).
11-16, 18-32: High-order 21 bits of 22-bit physical address descriptor table origin, low bit taken as zero (DDD...D).
17: Not used.

SEGMENT DESCRIPTOR WORD FORMAT
(32 bits)

PPPPPPPPPP------
FAAABBBCCCPPPPPP

17: Fault if 1 (F).
18-20: Access allowed from ring 1 (AAA).
000: No access.
001: Gate (for procedure call).
010: Read.
011: Read and write.
100, 101: Reserved.
110: Read and execute.
111: Read, write, and execute.
21-23: Reserved for future expansion (BBB).
24-26: Access allowed from ring 3, same code as above (CCC).
27-32, 1-10: High-order 16 bits of the 22-bit physical address of the page table origin (PPP...P).
11-16: Reserved, must be zero.

PAGE MAP ENTRY
(16 bits)

VRUSAAAAAAAAAAAA

1: Valid: page resident if 1, fault if 0 (V).
2: Referenced: set by hardware when page is referenced (R).
3: Unmodified: reset by hardware when page is modified (U).
4: Shared (inhibit usage of cache buffer): set by software when memory page is shared among processors (S).
5-16: High-order 12 bits of physical page address, low-order 10 bits are taken as zero (AAA...A).

counter is incremented and a return from the fault handling routine is made.

Primos uses an approximation of the global least recently used (global LRU) algorithm which is commonly called a clock algorithm with prepaging. This algorithm will swap out three pages whenever its free page pool is exhausted and a page fault occurs, in order to create some space for future page faults. It has been widely used in industry, for example in the CP/67 and VS/370 operating systems. Because of all the possible interactions with other jobs and the resulting workload dependency, it is difficult to analyze.

Process scheduling on the PRIME is priority based, i.e., the highest priority active process is always dispatched. There are 5 priority levels. 0 is the lowest and 4 is the highest (system priority). A user normally is at level 1. The time slice for priority levels 1 to 4 is 300 msecs, for priority 0 it is 100 msecs. One would guess that the time slice of a process is renewed whenever a process is dispatched but this is not so. The time slice is renewed only when the time slice has expired and not every time a process is dispatched after CPU deallocation.

2. UNIVAC

The UNIVAC 1100/81 is a batch oriented machine. Attempts to run interactively on the UNIVAC yield results less than favorable. This is because interactive user processes must run at virtually real-time priority to maintain a reasonable

response time. This places a heavy burden on the system, as its batch oriented management techniques cannot be used.

The memory management scheme for EXEC-VIII, the UNIVAC 1100 series operating system, will fill available memory with jobs from the eligible run queue. The jobs are taken highest priority first, real-time over batch, according to sub-category priority. The sub-priorities are established at the beginning of the run and remain constant for the duration of the run. When a user task requests memory, the executive shuffles the batch tasks to form a free area above the requesting task. Thus, by simply extending the address limits of the task, enough memory is made available to satisfy the task's requirements. When there is insufficent memory available, the executive will swap lower priority tasks out to secondary storage in order to free enough memory. If there are no lower priority tasks, the executive will suspend the requesting task until sufficient memory becomes available. The UNIVAC memory management forces the entire user task to be resident, if it is active. The user task has no control over this other than to release some previously allocated core to decrease the task size.

V. MEASUREMENT TOOLS

In order to measure the performance of SPAR, SAPIV, and TWODEL measurement tools to determine the CPU and I/O time, and the amount of memory used by the program are required. In the present study, only software measurement tools are used on both PRIME and UNIVAC because of the hardware tools. Measurement tools can be applied in two ways, in the form of independent monitors and by using them as subroutines called by an instrumented version of the program under study. Measurement tools can create both time and space interference. First, they need time to execute (important for instrumentation). Second, they also create data which need to be stored in core or in buffers and subsequently written onto secondary storage devices. Also, software measurement tools use memory space themselves - they are program modules - and the space of an instrumented version of a program will increase as well as its execution time. Clearly, this is undesirable, because we would like to measure the original program's behavior, not the instrumented version's. There are several requirements which when met can alleviate this problem.

1. The space and time interference should be as small as possible. This is also important for the resolution of the measurement tool. The faster a measurement can be taken, the more often it is possible to measure. Naturally, the measurement data cannot be more accurate than the meters which

collect them. On the other hand, the accuracy required depends on how the data are being used and how detailed an analysis is to be performed.

2. It should be possible to separate the resources (e.g. CPU or I/O time) consumed by the meter from the resources consumed by the program. For an instrumentation tool this can be done in the following way: Compute the time interference. For CPU time, this is the time to execute the measurement routine once, multiplied by the number of calls to the routine. For I/O time, it depends on how data is written to secondary devices. If measurement data are written onto a device (e.g. a tape) which the program itself does not utilize, then the amount of time for access to this device can be seen directly, since it means usage of a different resource. If this is not the case, then the number of data items divided by the number of items which fit into the meter's I/O buffer will give the number of transfers. If this is not an integer, it has to be rounded to the next higher integer value. The number of I/O transfers multiplied by the average time to do such a transfer yields the resources used. This applies only if buffers are written out when the buffer is full or at program termination. If buffers are flushed out for other reasons, this technique will merely provide a lower bound. Also, if the disk seek-time is added to the I/O time for charges, the I/O time may vary considerably depending on how full the disk is or which algorithm is employed for the search.

Measurement Tools on the PRIME

Since the PRIME is a paged system, any program's CPU and I/O behavior will have two aspects, the essential CPU and I/O time and the overhead due to paging. This means that we are interested in a measurement tool which can measure

- CPU time
- I/O time
- Number of page faults
- Average time (CPU, I/O) to handle a page fault.

The following measurement tools are available:

TIMDAT

This is a user callable subroutine which returns, among other things, wall clock time, CPU time and disk I/O time since login. The resolution is 1 tick. Each tick is approximately equal to 3 msecs, i.e., 330 ticks per second. CPU and disk time include both the resources explicitly consumed by the user program and the resources consumed by the system, e.g. paging overhead.

CTIM$A & DTIM$A

These two routines provide CPU and disk time elapsed since login in seconds and centiseconds.

USAGE

Usage is a system metering facility which reports on the system performance, e.g., differential CPU time, differential disk time, page faults, percentage utilization, etc. for each reporting period. This period can be specified by the user.

USAGE also reports on the resource consumption by each user. Though USAGE is a powerful and important tool, it is of limited use to us because it reports most performance data on a system wide basis. This means that it collects unnecessary statistics for us. This also influences the reporting interval which may not be smaller than 30 seconds which is too large for our purpose. Table 5.1 shows an example of USAGE's output.

In addition to the limitations of TIMDAT and other routines mentioned above, there are certain other factors which make meaningful measurements on any multiprogramming system difficult. These problems arise due to the need to overlap I/O operations. The system maintains various buffers for each device user request. Whenever a user issues an I/O operation, the contents are written to buffers or brought into buffers from disk, if they are not already present in the buffers due to an earlier request for input. For write operations, buffers are written out on disk either when they are full or for other reasons (for example, the user has not accessed his buffers lately and therefore it is assumed he is not likely to reference them in the near future). Other difficulties arise from the overhead introduced through the calls to the timer routine within the program. If the timing could be done by an independent parallel process which communicates with the program under observation, the overhead can be re reduced.

Table 5.1

```
r usage -freq 30 -times 1


 04/03/80 19:29: 7.13  DTIME=   30.01  CPTOT=11956.25  IOTOT=11993.59

DCPTOT=   3.949  %CP=  13.161  DPFCN=    50  PF/SEC=   1.666
DIOTOT=   4.043  %IO=  13.473  DIOCN=   142  IO/SEC=   4.732  %OVLAP=   0.000
DLOCNT=   179 LO/SEC=   5.97 DLOFCT=    92 DLOSCT=     0 DLOUCT=     0
DLOCCT=    87 LN/SEC=   2.90 %MISS=  48.60   %XCP= 2.39
%CLK= 2.00 %AML= 2.22 %MPC= 0.00 %IPC= 0.00 %FAR= 0.00 %SLC= 0.00  %BAK=79.96
%DSK= 0.27

USR LOGNAM    MEM   CPTIME     DCP     %CP     IOTIME     DIO     %IO

  1 SYSTEM    225    58.522   0.023   0.077   275.939   0.597   1.990
  6 CS         41    11.319   0.275   0.916    15.603   0.076   0.253
  7 CS         15    14.436   0.357   1.190     9.809   0.136   0.453
  9 SYSRUG     49   900.911   1.659   5.529   274.318   1.394   4.646
 12 EE316       8     2.640   0.017   0.057     7.721   0.000   0.000
 18 EE341      38    59.773   0.526   1.753    96.764   0.421   1.403
 19 EE331      21     4.318   0.070   0.233     3.745   0.054   0.180
 20 MISC.      39     3.521   0.474   1.580     9.891   1.752   5.839
 21 EE441      43    69.823   1.087   3.623    36.597   0.151   0.503
 23 CS431      11     9.556   0.305   1.016     6.855   0.097   0.323
 28 MMAE31     10     0.307   0.272   0.906     0.661   0.064   0.213
```

For our studies, the measurement tools which are provided by the system are not sufficient. Therefore, we decided to modify the existing ones to suit our needs better. For PRIME, a possible solution to our measurement problem is an external software monitor which can monitor the activities of a particular program or user. This led to the development of a modified version of USAGE called SUSAGE. The difference between the two is that SUSAGE only takes the measurements we are interested in and thus provides us with the information needed without the additional overhead of USAGE. This improves resolution and reduces interference. Less information is processed, thus reducing both time and space required. We will measure the performance of SAP4, SPAR and TWODEL by instrumentation and by monitoring program execution in order to be able to contrast both methods.

Measurement Tools on the UNIVAC

There are various executive routines available for time measurements, such as wall clock time, CPU and I/O time, etc. The most interesting for us are

TIME$

This executive routine returns the wall clock time. It can be used to measure turnaround times. Since this information can be found on every printout, we do not have a need for measuring how waiting and progress intervals are distributed for a given job. We felt no need for this, because the UNIVAC does not charge the user for wait times,

i.e., when the job is temporarily swapped out because there is not enough memory or because a higher priority job is allowed to run first.

INFO$

This is an alternative executive request which returns the same form of units as PCT$, but forces the values to be updated before returning them, because it immediately terminates the time quantum. A FORTRAN callable subroutine was written which returns these CPU and I/O times in single integer form. The values returned are in microseconds with a resolution of 200 microseconds. A source listing with sample usage is included in Figure 5.1.

A program has been developed to enable an almost automated instrumentation of SAP and SPAR on both machines. The instrumented versions call timing routines at appropriate places. They take CPU and I/O time measurements of the program. On the UNIVAC both CPU time and time for executive requests are treated as CPU time for charging purposes. Therefore, it was decided to treat them both as CPU time. Calibration runs were done to find out what the appropriate blocksize for writing measurement data should be so that the overhead is low, and also to quantify the overhead due to instrumentation.

CPU and I/O overhead depend on the kind and the size of the problem. For the 72 node plate problem, CPU overhead is 16.5 percent, I/O overhead 12.3 percent. If the size of the plate problem is increased to 143 nodes, the respective over-

Figure 5.1

```
@ELT,IL  A.SUPTIM
  .
  .
  .     SUPTIM:   FORTRAN CALLABLE SUBROUTINE TO GET CURRENT CPU
  .               AND I/O SUP TIMES FROM THE EXEC.
  .
  .               NORM BARTEK  23-JUL-80
  .
  .               TO USE THIS ROUTINE, MERELY CALL IT WITH TWO SINGLE
  .               PRECISION INTEGERS WHICH WILL RETURN THE CURRENT
  .               CPU AND I/O TIMES RESPECTIVELY.
  .
  .               CALL  SUPTIM(ICPU,IO)
  .
  .               THE VALUES RETURNED WILL BE IN MICROSECONDS WITH
  .               A 200 MICROSECOND BASIC RESOLUTION.
  .
  .
$(1)       AXR$                        . REGISTER EQUATES
SUPTIM*    LA         A0,(7,SUPBUF)    . SET UP BUF ADDR & LENGTH
           SA         A0,TBL+1         . SET UP ACW FOR ER CALL
           LA         A0,(7,0,0)       . SET UP FUNCTION FIELD ALSO
           SA         A0,TBL           .
           LA         A0,(2,TBL)       . SET UP A0 FOR ER CALL
           ER         INFO$            . PERFORM SUP TABLE FETCH
           LA         A0,SUPBUF+3      . GET CPU SUP FROM TABLE
           MSI,XU     A0,200           . ...TIMES 200 MICROSECONDS
           SA         A0,*0,X11        . PUT IT WHERE USER CAN ACCESS IT
           LA         A0,SUPBUF+4      . GET I/O SUP FROM TABLE
           MSI,XU     A0,200           .
           SA         A0,*1,X11        . STORE I/O RESULT
           J          3,X11            . RETURN TO USER ROUTINE
                                       .
                                       .
TBL        RES        2                . ER CALL TABLE
SUPBUF     RES        8                . RETURNED INFO TABLE
           END
@EOF
@EJECT
```

Figure 5.1 Cont'd

```
@ELT,IL  A.TIMER
C
C
C    TIMER/FTN:    CPU & I/O MEASUREMENT TEST PROGRAM.
C                  (PART 1 OF 2 PARTS)
C
C                  AUTHOR:  NORMAN R. BARTEK   JULY, 1980
C
C                  SYSTEM PERFORMANCE EVALUATION PROJECT
C
C
C                  THIS PROGRAM WILL PRINT OUT THE CURRENT AMOUNT OF CPU & I/O
C                  USAGE FOR THE CURRENTLY ACTIVE RUN.
C
C                  THE TIMES WHICH ARE PRINTED ARE IN MICROSECONDS, WITH A
C                  BASIC RESOLUTION OF 200 USEC. THE TIMES RETURNED ARE THOSE
C                  FROM THE 'SUPTIM' SUBROUTINE WHICH ACCESSES THE PCT VIA
C                  AN ER TO INFO$. FOR FURTHER INFORMATION, REFER TO THE
C                  'SUPTIM' SUBROUTINE.
C
C                  TO USE, JUST EXECUTE WITHOUT ANY PARAMETERS...
C
C
      IMPLICIT INTEGER (A-Z)
C
C  PRINT OUT INITIAL HEADINGS...
C
      WRITE(6,20)
      WRITE(6,30)
C
C  NOW PERFORM THE INFO$ EXECUTIVE REQUEST VIA THE 'SUPTIM' SUBROUTINE...
C
      CALL SUPTIM(CPUTIM,IOTIME)
C
C  THE TIMES RETURNED ARE IN MICROSECONDS ALREADY, SO PRINT THEM OUT...
C
      WRITE(6,40)CPUTIM,IOTIME
      CALL EXIT
   10 FORMAT(1H1)
   20 FORMAT(1X,'***  SYSTEM PERFORMANCE EVALUATION PROJECT  ***',
     1'  TIMER ROUTINE  ***',/)
   30 FORMAT(1X,'CURRENT TIMES FOR ACTIVE RUN:',/)
   40 FORMAT(1X,'CPU TIME: ',I10,8X,'I/O TIME: ',I10,//)
      END
@EOF
@ASM,S  A.SUPTIM,A.SUPTIM
@FOR,S  A.TIMER,A.TIMER
@MAP  ,A.TIMER
 IN  A.TIMER
 IN A.SUPTIM
@EOF
```

heads are only 4.6 and 2.2 percent. Despite this small amount, we made an effort to subtract the overhead at the point where it occurs (where the measurements are taken). Since I/O may overlap CPU activity, a possible problem occurs because the point of measurement is where initiation of I/O occurs, not its completion. We found however, that the major portion of I/O cost occurs at initiation time. There is only a fixed amount of I/O time which is added at I/O termination. This quantity depends on the type of device. For a drum, it is 1.6 msecs. The I/O initiation time depends on the block-size and the type of device. In one of our runs, it came out to be 21.2 msecs. This termination charge is about 7% of the total I/O charge. An approximation algorithm is used which also take this termination charge into account. A set of test runs of uninstrumented and instrumented versions of SAPIV showed that estimated and actual overhead are within 1% of each other.

VI. CHARGING ALGORITHMS

The performance indices which are most important to predict are those which influence the charges most. On the UNIVAC there are more charge - relevant resources than on the PRIME. Appendix I shows current rates for the PRIME 400 and UNIVAC 1100/81 at IIT. The most important differences between the charges are how memory requirements and paging or other system activities on behalf of the user are charged, UNIVAC bills for file storage and core blocks used. This can constitute a major expense for programs with large primary and/or secondary memory requirements. If the programs are overlaid to avoid these large core requirements, then I/O time and its charges can increase significantly, if more I/O needs to be done to make communication between the overlays possible. The charging algorithm shows that the user is charged for the amount of memory used, multiplied by the time it is actively used. This implies that programs which are very heavily I/O bound and pay for memory usage during times of I/O activity may add a significant amount of cost to their charges. PRIME does not charge for memory directly. There are very significant indirect 'memory charges' however. One of them is that the user is charged for any paging on his behalf. If a program would require a large amount of memory on the UNIVAC and does not exhibit sufficient locality properties (i.e., only a small portion of the program and data area needs to be in core at any point in time), the number of

page faults and thus the number of page swaps performed will be large and contribute substantially to both I/O and CPU charges. There is no such charge on the UNIVAC. A second factor can make itself felt drastically. It is the seek time for locating records on disk. If a disk is highly utilized and has become fragmented and nearly full, a higher seek time will result than if it were less full. This seek time is part of the time which is charged to the user as I/O time. We can expect to run into such an unfavorable situation on a machine where no file charges exist, because there is no incentive to discard old files. Not only do both paging and disk seek time contribute to charges on the PRIME, but in both cases the situation can be further aggravated when there is a lot of contention among users. If there is a long disk wait time because of an overload situation, a program may have to wait considerably for input to be brought into memory. Although the wait time is not charged, the waiting programs' pages in core become old, because they are not referenced as often as those of other active programs. In the PRIME environment, this means that pages and a considerable number of them, if contention is high, are swapped out while a program is waiting for data to be brought into core. This means that more paging needs to be done on behalf of that program to bring the pages back into memory and that further increases CPU and I/O time charged. This explains why charges for the PRIME can vary so widely, even though the same

program with the same input data is run, whereas the charges on the UNIVAC are constant. This leaves an unpredictability factor for the PRIME which may well go into orders of magnitude. If a potential buyer knows the expected user behavior (workload, contention), he can make an appropriate decision. One should keep in mind however, that workloads change over time (they VERY seldom decrease) and that a sensitivity analysis is necessary to assure that adequate performance at acceptable cost is possible for most of the projected usages of the system. For all these reasons cost/performance comparisons for the two machines cannot and should not be made on an absolute basis, but rather one should compare ranges of cost and performance of PRIME and UNIVAC.

VII COST-EFFECTIVENESS MEASURES

In trying to estimate the cost-effectiveness of a computer system for a given set of tasks, several different measures can be used. In the present study various structural analysis programs were run on the two computer systems and the following five measures of the system were recorded analyzed for all of the examples.

a) Correctness of Result

This is the first and primary performance index, since a program which terminates abnormally or gives incorrect results is useless.

The correctness of the result may depend on various system and user software related factors. For numerical software such as structural analysis programs some of the factors are:

*Precision of the Machine

This depends on word-length and on machine arithmetic. Precision problems are likely to occur when a program is transported from a machine with high precision to one with lower precision. Word length may influence the result, but need not, if system arithmetic compensates for the smaller word size.

*Errors in Software

Most computer programs of significant size contain errors. The problem is aggravated when a program is converted from one machine to another because of imperfect conversion. Mini-computers suffer doubly on that account.

First, being new, there are relatively few programs that have been originally developed on any particular mini. Second, because minicomputers are rarely attended by a large support staff the conversion of programs to the minicomputer is often not done thoroughly and professionally.

b) Resource Consumption

This is mostly chargeable resource consumption. However, since charging algorithms vary, some data on resource consumption were collected even though they were not cost relevant for the machines which were used. Resource consumption may vary considerably on two different machines for the following reasons:

*The two programs although functionally equivalent are different.

The utilities they use are different and may have different resource requirements, e.g. I/O routines may use dissimilar buffer sizes (impact on memory requirements and number of I/O operations, devices and access techniques). The dynamic memory space used could be much larger on a virtual memory machine than on a real memory machine. This usually results in a trade-off of data vs. paging (which may not be charged). Each implementation will likely use a different size for dynamic memory space. The support software varies, e.g. one function minimization routine may use a more sophisticated method than another (this depends on the availability of certain support software), and thus need different amounts of resources. In extreme cases this may

change a CPU bound job into an I/O bound one or vice versa.

*Computational accuracy.

The word-size on a machine may require a program to be run in double precision which can run in single precision on another machine.

*Operating system and architectural differences.

These will usually account for most of the dissimilarities in resource consumption. On the architectural side there are the questions of hardware components and their speeds or size, the configuration of the system. A system with faster components will often use a smaller amount of CPU and I/O time than a slower one. But that does not necessarily mean that runs are much cheaper. Also a comparison of hardware speeds is not always a good indicator of comparative resource consumption on two machines. Much depends on the operating system which coordinates the operation of all the components. An operating system introduces overhead, which reduces the effective speed of a system. The amount of overhead will influence the overall quality of a system. An important aspect is also how resources and competition for them are managed. Does resource contention have an impact on resource consumption? If so, how much? Is resource consumption predictable? Is it possible to request a particular device which is faster than others of a similar kind? Naturally this will have implications on resource consumption. Some systems put files which are over a system defined size automatically on slower I/O devices - this will increase the resource

consumption for I/O and the time main memory is allocated to a program, because the time to transfer information from and to main memory is now longer. The instruction set of each machine is important, does it have floating point arithmetic?. Compilers and their ability to optimize can make a large difference.

Another factor is the impact from other jobs. Clearly, in a multiprogrammed environment, this will affect performance. It can affect resource consumption as well. As an example take a paged system with a global page replacement strategy such as global LRU (Least Recently Used). Every time a page is needed and not in memory, the least recently used page (or pages, if more than one page is replaced each time) is swapped out. This means that it has to be brought back in should a program request it later on - thus increasing the program's CPU and I/O consumption. The more 'passive' or I/O bound a program is, the more its resource consumption will increase with contention in the system. If the user is charged for paging, a difficulty arises for these systems, because the resource consumption of programs is not easily predictable any more. Accurate prediction would require exact knowledge of the behaviour of all other programs run on the system at the same time. This is impracticable. What is important to know for system comparison, however, is what range of resource consumption can be expected for light, medium, or heavy system loads. Although this will not accurately predict actual resource consumption in a particular

case, it will help to compare two systems and to decide which is more favorable, based on its expected and worst case resource consumption.

c) Charges

This includes both job-step and run charges. In many cases it is important to know which part of the program contributes the highest portion to the total charge, or which resource proved to be the most expensive for a given run. Charges are related to resource consumption, but it depends on the particular charging algorithm which resource is the most expensive. Sometimes it is possible to shift to a less expensive resource and save money, at other times it may be more profitable to use a more expensive but considerably faster device for a shorter time. It makes little sense to compare only resource consumption of a program on a machine or only charging algorithms of a machine. What seems expensive in the charging algorithm actually may constitute only a minor part of the charges because only a small amount of the resource is needed. All that has been said previously influences charges and needs to be taken into account. If resource consumption is not predictable and only a range can be given, the resulting impact on the charges has to be analyzed. How much do the variations of resource consumption influence the charges? Again this depends on the actual charging algorithm and may have only marginal influence on the charges, e.g. if a highly varying I/O time is not charged very much compared to a relatively stable CPU time, the resulting

variations of the charges may well be below 10%. For cost comparisons this can be a very important result.

d) Performance

Performance indicates how well a system works. It shows the effectiveness with which the resources of the host computer system are utilized towards meeting the objectives of the user. It can be seen as the technical equivalent of the economical notion of value. In order to do a job, resources are required. The performance of a system then is determined by the following factors:

*how much of each resource is required (resource consumption)

*in which order are the resources expended

*how does this affect the user oriented measures of performance such as response time and turnaround time or system performance measures such as utilization or throughput?

In other words: how well does a computer system do its job? Performance may depend on how much a user is willing to pay, e.g. when different priorities for jobs exist or when different rates are in effect during business hours and at nights or weekends. Performance as the user sees it (turnaround time for batch systems or response time for interactive system) consists of the time spent towards progress of the program and any waiting time during execution. The former is usually charged to the user, including system overhead when it spends time on the user's behalf, whereas

the latter may or may not be. Waiting time can be of two kinds: internal or external.

*Internal waiting time -

this is all the waiting time the program experiences due to contention on the part of other programs which cause waits for CPU, I/O channels, memory etc. It also includes the time a program has to wait for the completion of a read operation from secondary storage or the time to handle a page fault on behalf of a program. Finally, if a program has a specified starting time or if the system is so crowded that the program cannot be executed promptly, another sometimes very considerable wait time may result. This may also happen, if a very low priority was specified and most other jobs are allowed to bypass the program.

*external waiting time

this includes mannual special set-ups (e.g. mounting a tape) and service tasks such as changing paper for a printer, switching a printer off periodically to remove printout, refilling paper, putting printout into mail boxes, and other related tasks which occur periodically and can cause delays. In this case performance may be related to staffing and thus to human factors, and sometimes considerably more time is spent on these than on the actual computing. Finally, uptime/downtime for the system needs to be considered. This really is most important, since the fastest computation time does not help much, if the system tends to go down for prolonged periods of time frequently and no work at all is possible.

VIII. PRELIMINARY RESULTS

The combination of problems and programs that is the basis of the cost effectiveness evaluation is described in Section II. The present section contains the computation and analysis of the preliminary results obtained so far. Each one of the three structural problems - the beam, the plate, and the stiffened cylinder, was run for several choices of mesh refinements. The main reason for doing this was to determine the effects of problem size on the relative performance on the two types of computers. The maximum problem size was limited by budgetary considerations so that no problems were run that would cost more than $100. This permitted the measurements to be taken only for small and medium size problems. However, in each case curves of resource consumption as a function of problem size were generated. These curves often proved to be good enough for extrapolation to larger problem sizes. At the present time only the beam runs are completed, so only these results are reported.

Beam Problem

A cantilever beam was modeled by plane beam elements (E24 elements in SPAR) and the vibration modes and frequencies were calculated using the SPAR program. The number of nodes was varied from 5 to 600 and with three degrees of freedom per node; the maximum number of degrees of freedom is 1800. However, even for 600 nodes the problem is not costly to run because of the very small band width associated with a one dimensional problem.

For small problem sizes (up to 25) the problem can be run in single precision. For larger number of nodes the limited double precision option in SPAR (double precision used only for assembling the stiffness matrix) must be employed.

a) Correctness

This simple problem exposed a bug in the Prime version of SPAR. The program could not calculate more than two vibration modes and frequencies. The problem occurred in the subspace iteration method and seemed to indicate that the initial vectors generated by a random number routine were not linearly independent. On the UNIVAC there was no problem to get the required three lowest frequencies.

b) Resource Consumption

(i) UNIVAC

Resource consumption was measured by run and by processor. The following resources were measured as a function of the problem size (i.e. no. of nodes) for each run,

*I/O time

*CPU + ERC (executive request call) time

The total resource consumption is shown in Figures 8.1 and 8.2 and individual processor results in Figures 8.2 to 8.20. Linear and quadratic polynomials were used to curve-fit the results, Figs. 8.1 and 8.2 both show only slightly quadratic tendencies. Likewise, most of the processors show resource curves which are linear or almost linear. Only

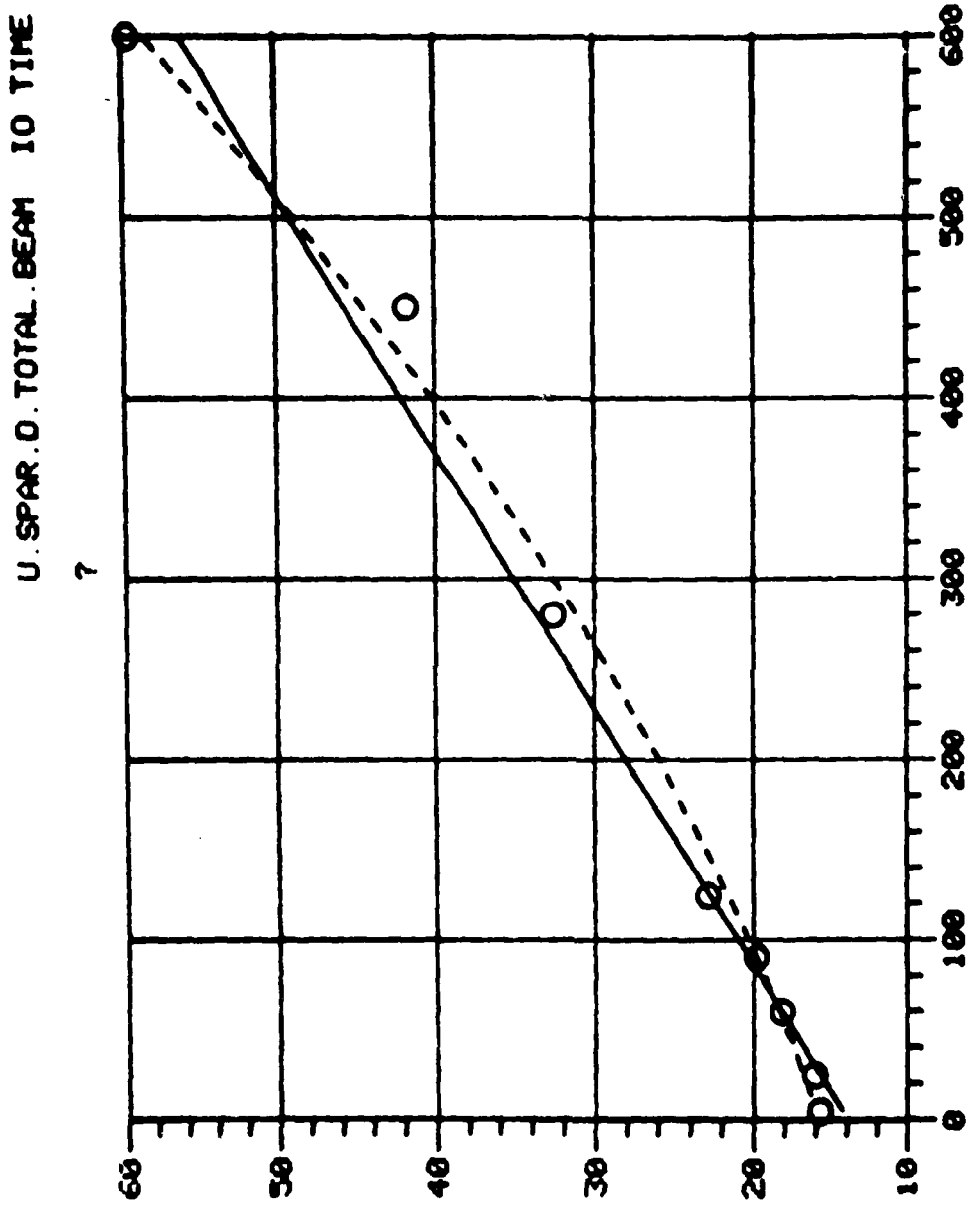


Figure 8.1: Total I/O time (sec) vs. Number of Nodes for Beam Problem on UNIVAC

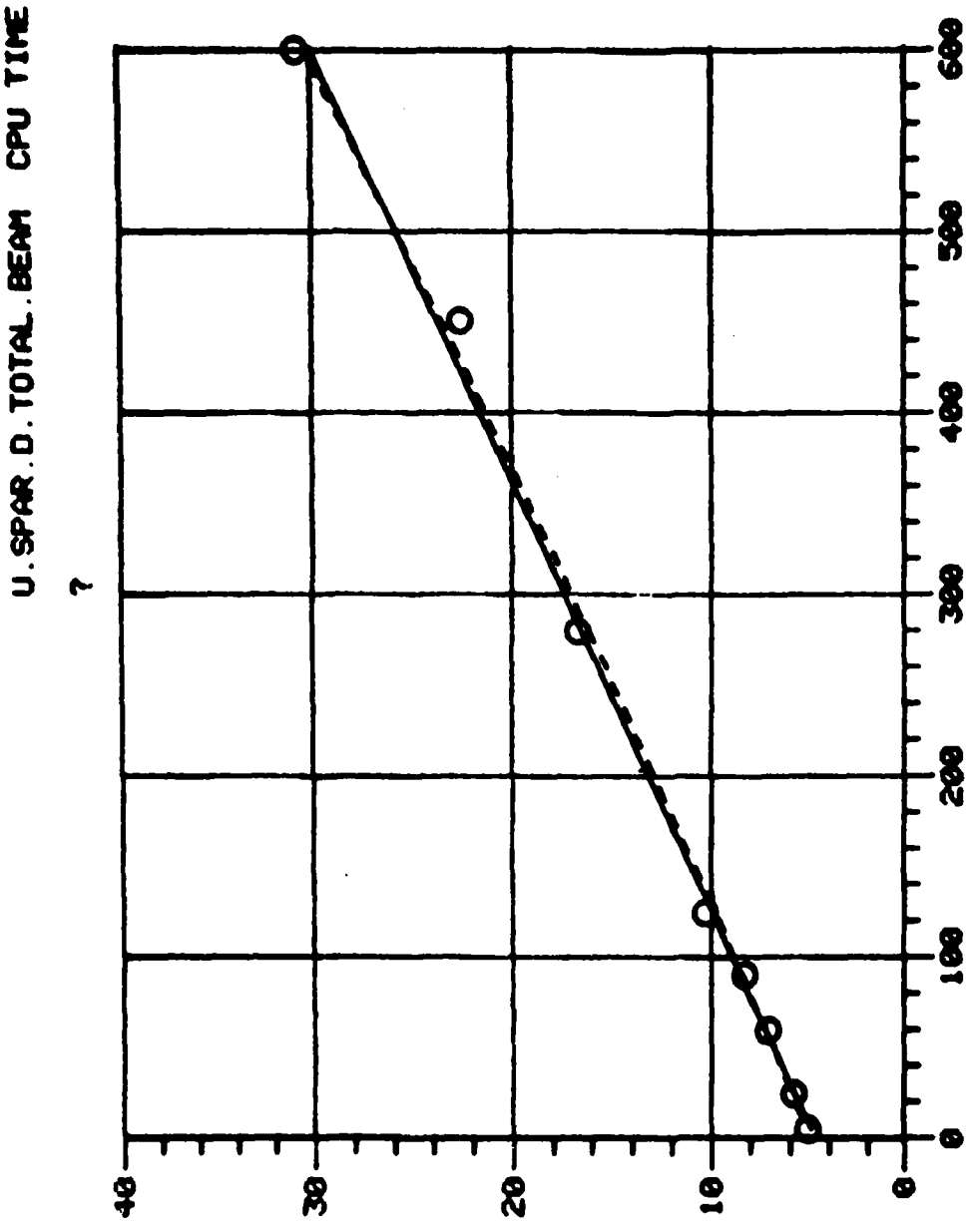


Figure 8.2: Total CPU + ERC Time(sec) vs. Number of Nodes for Beam Problem on UNIVAC

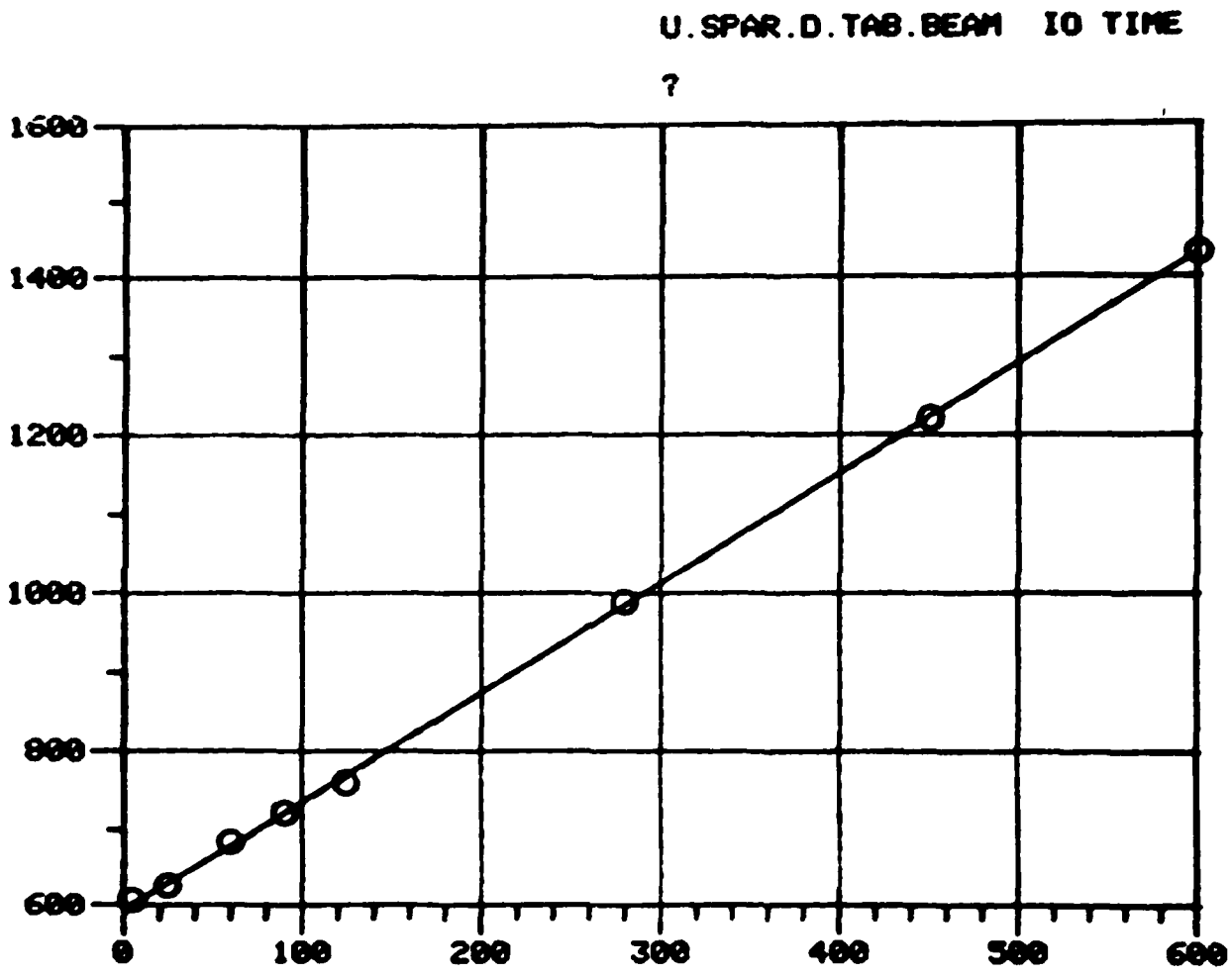


Figure 8.3: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor TAB

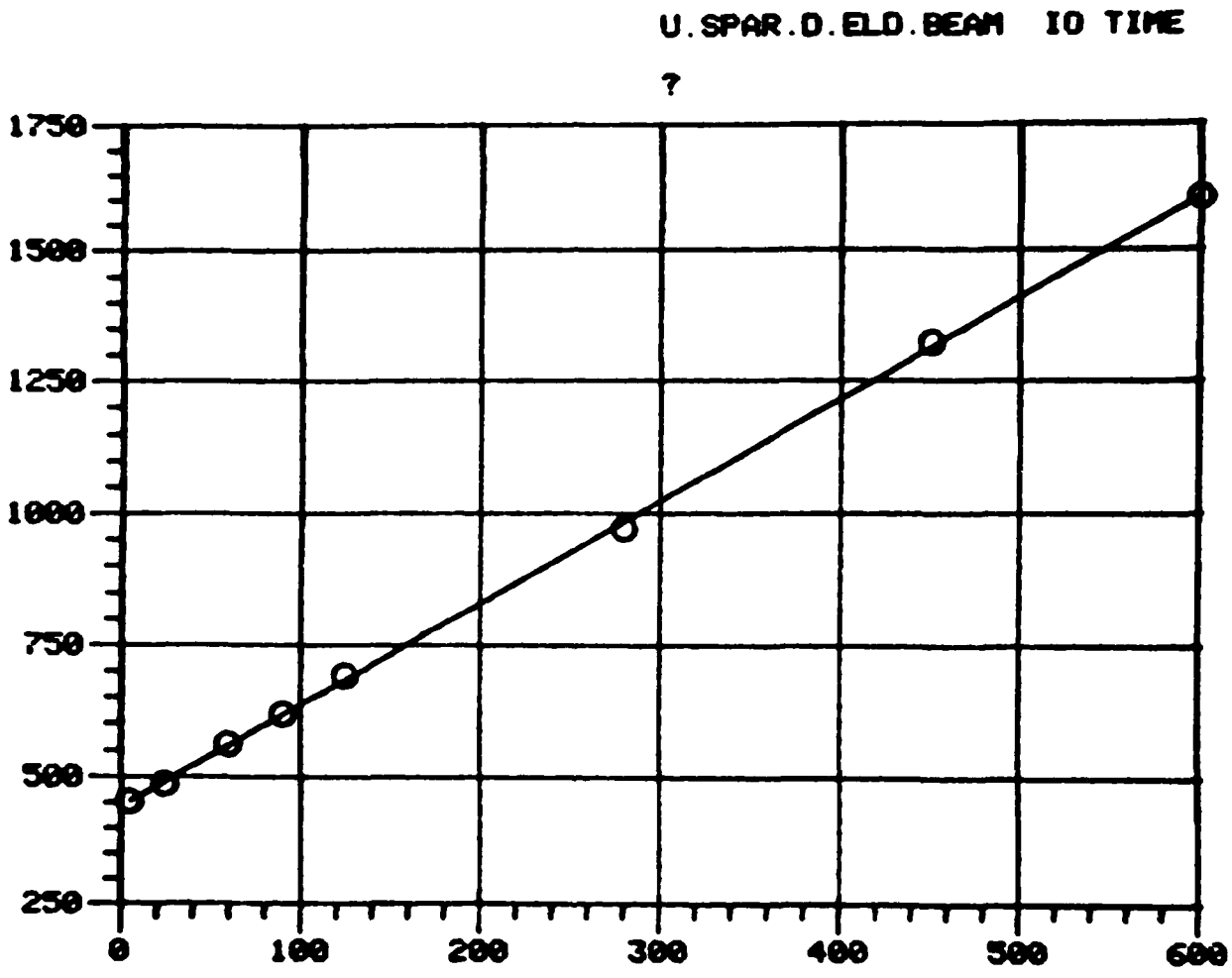


Figure 8.4: I/O Time(sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor ELD

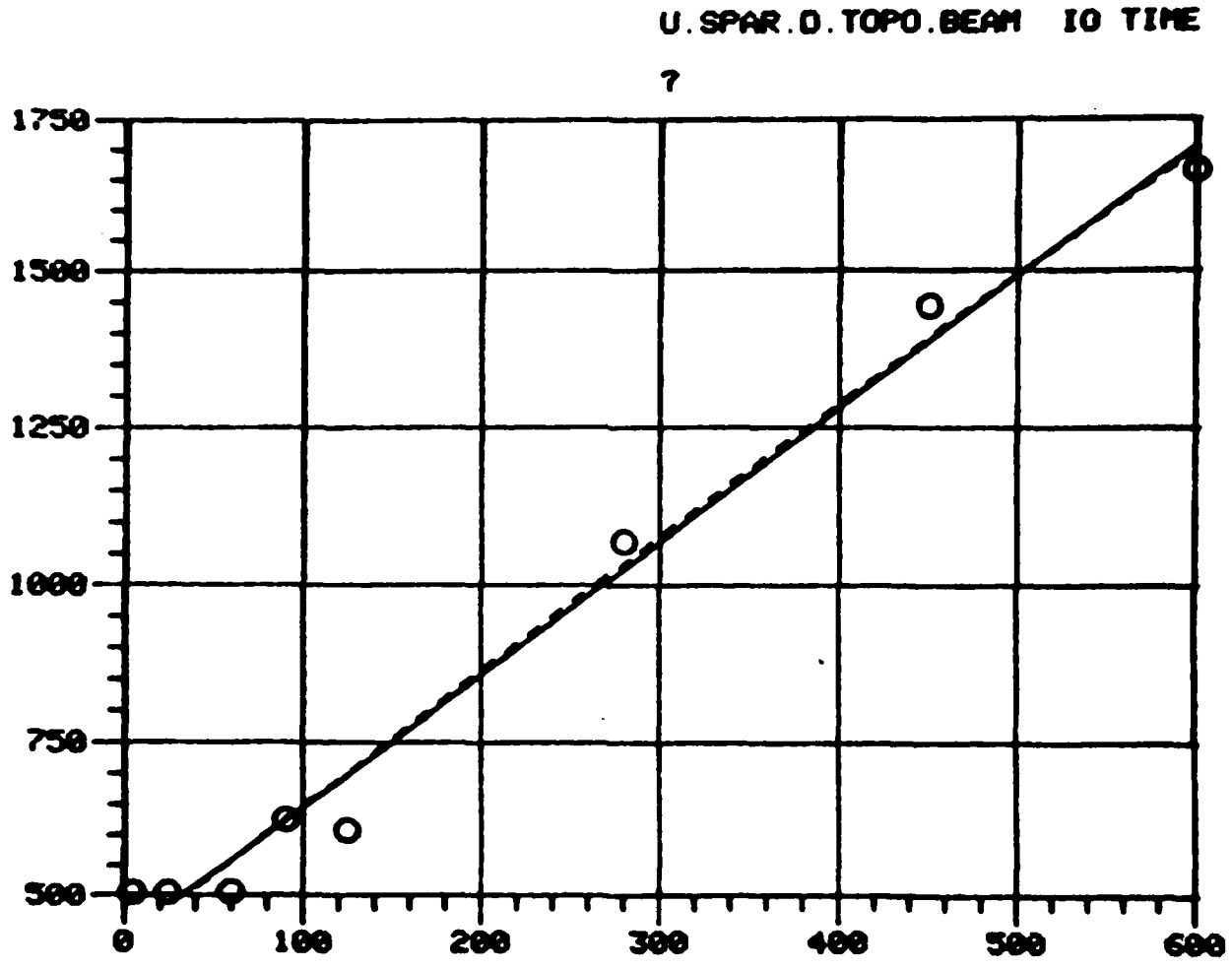


Figure 8.5: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor TOPO

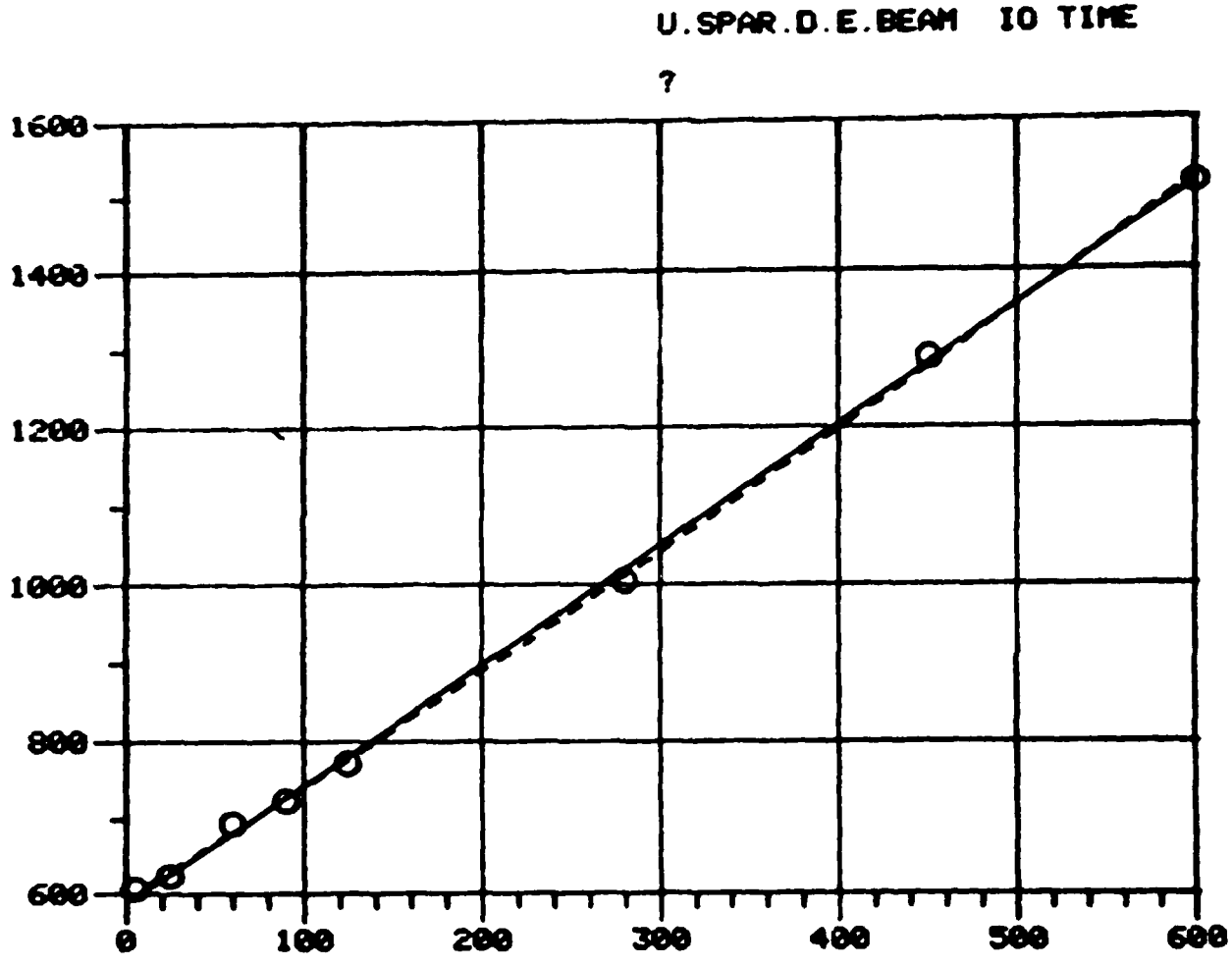


Figure 8.6: I/O Time (sec) vs. Number of Nodes for Bear Problem on UNIVAC: Processor E

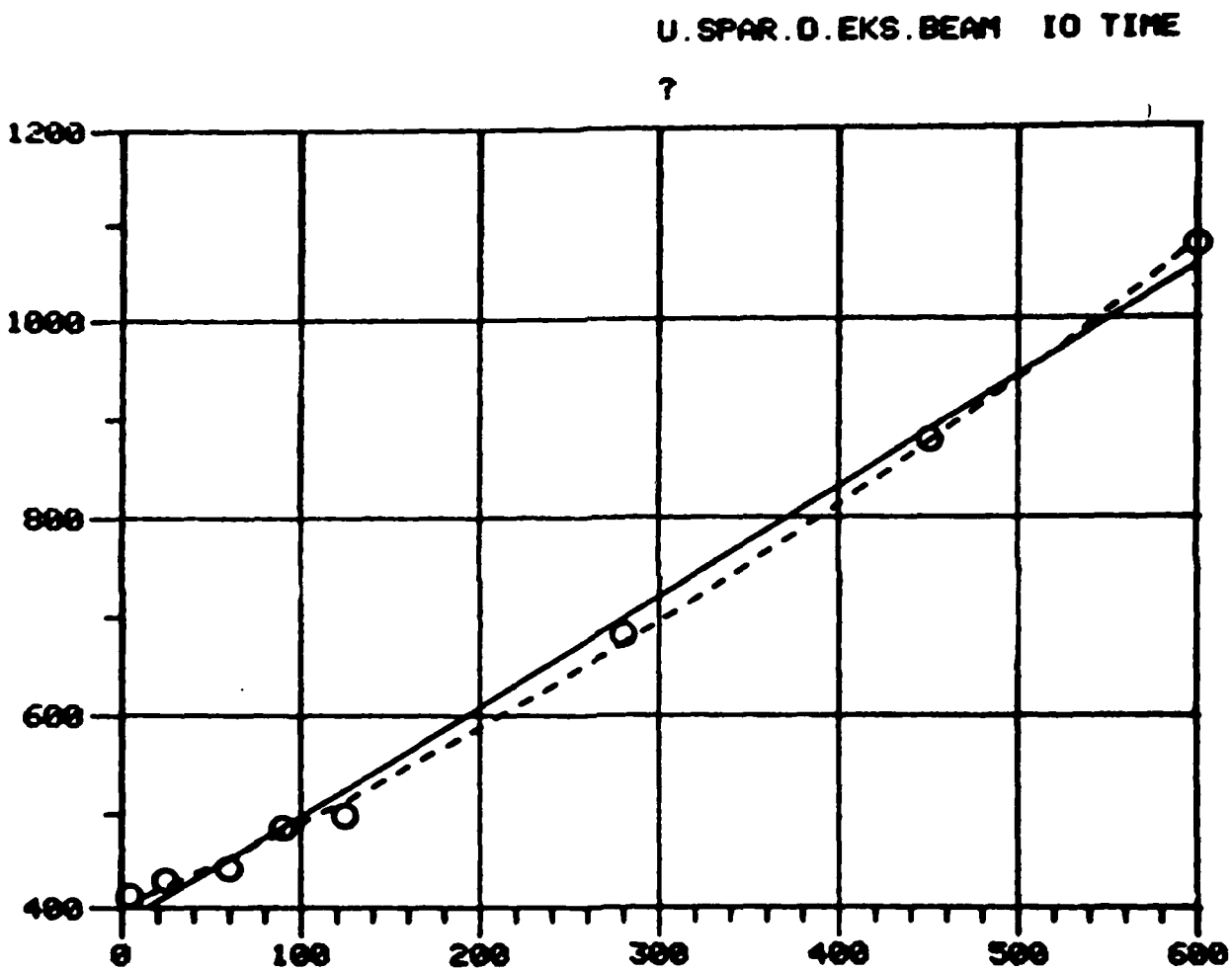


Figure 8.7: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor EKS

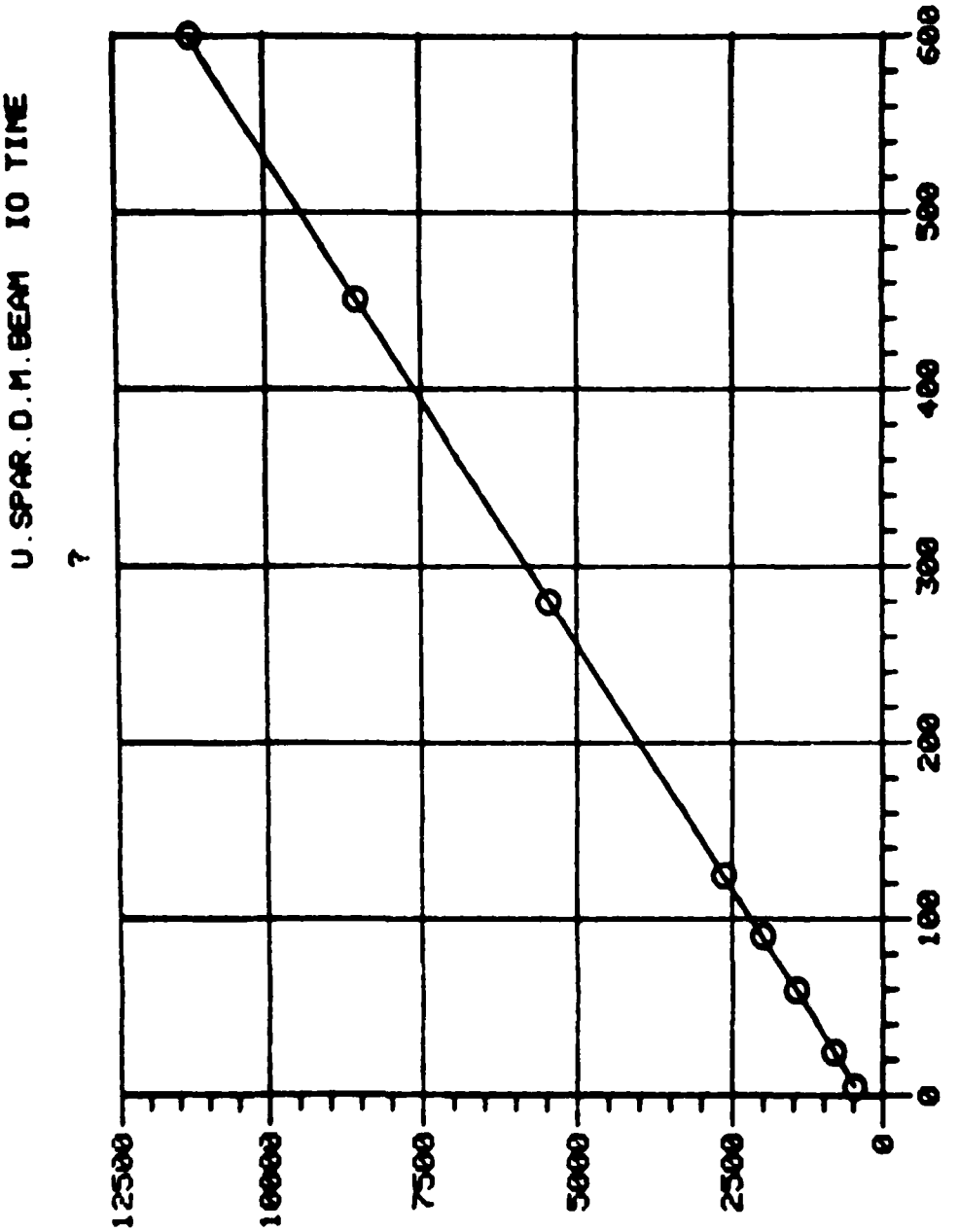


Figure 8.8: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor M

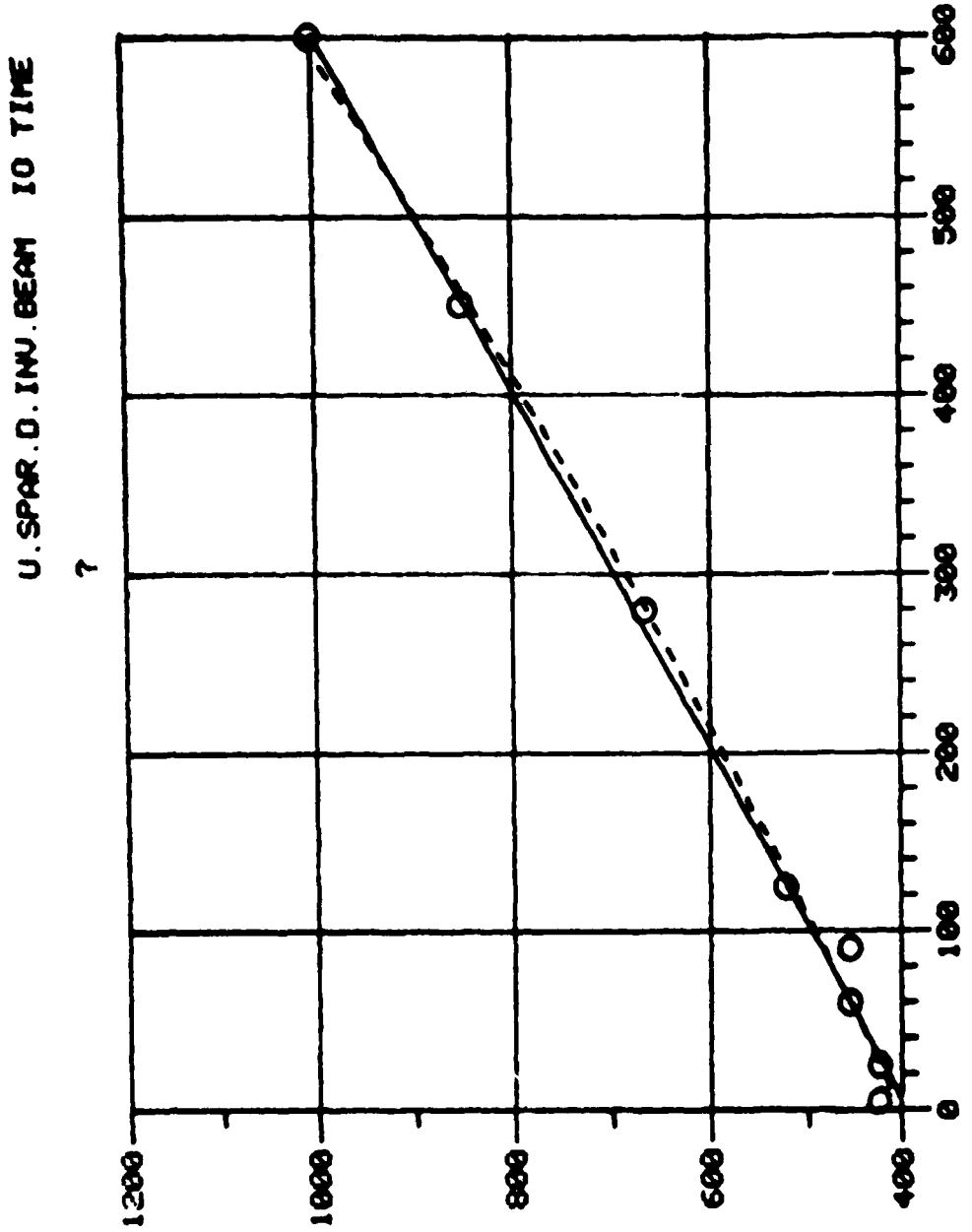


Figure 8.10: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor INV

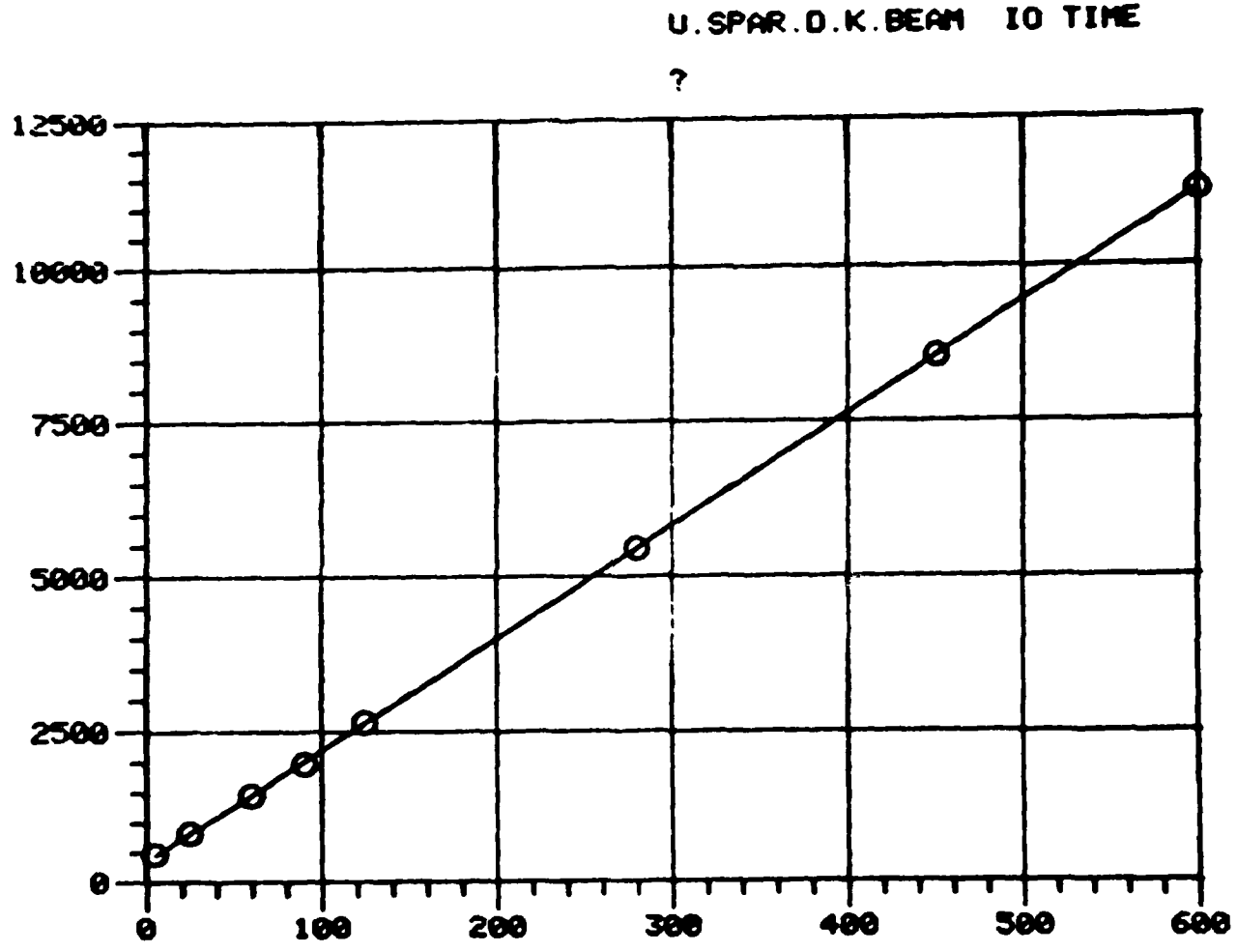


Figure 8.9: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor K

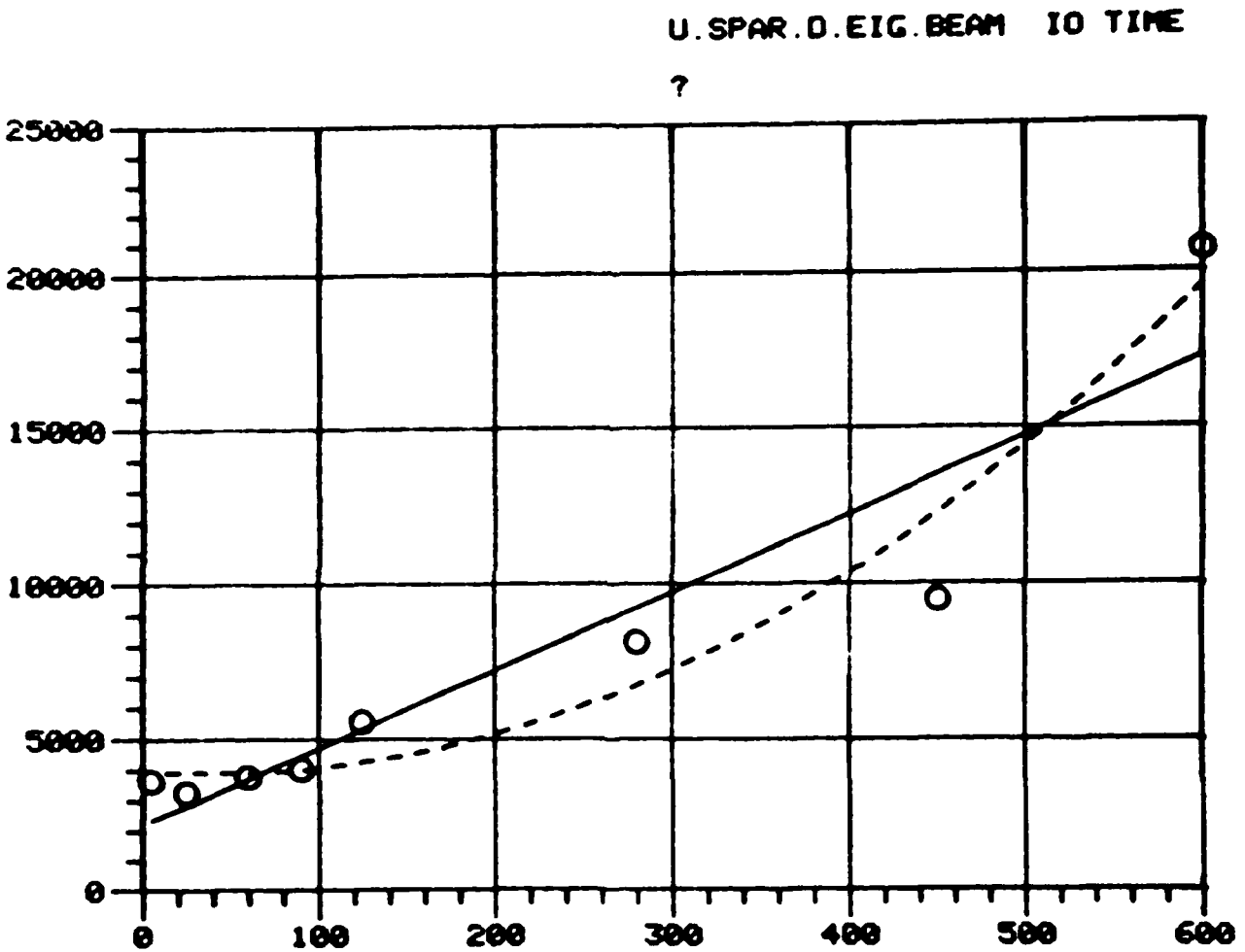


Figure 6.11: I/O Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor EIG

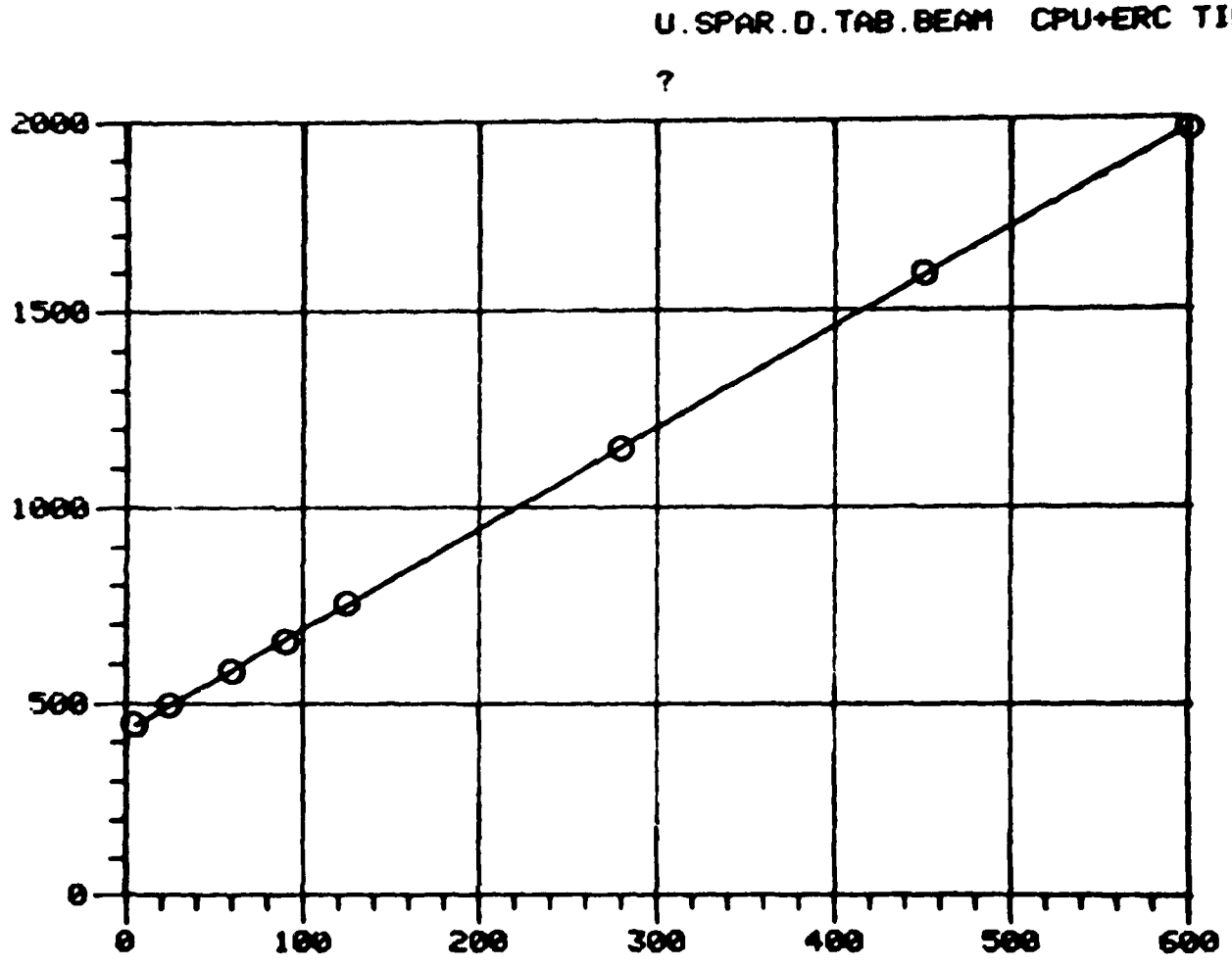


Figure 8.12: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor TAB

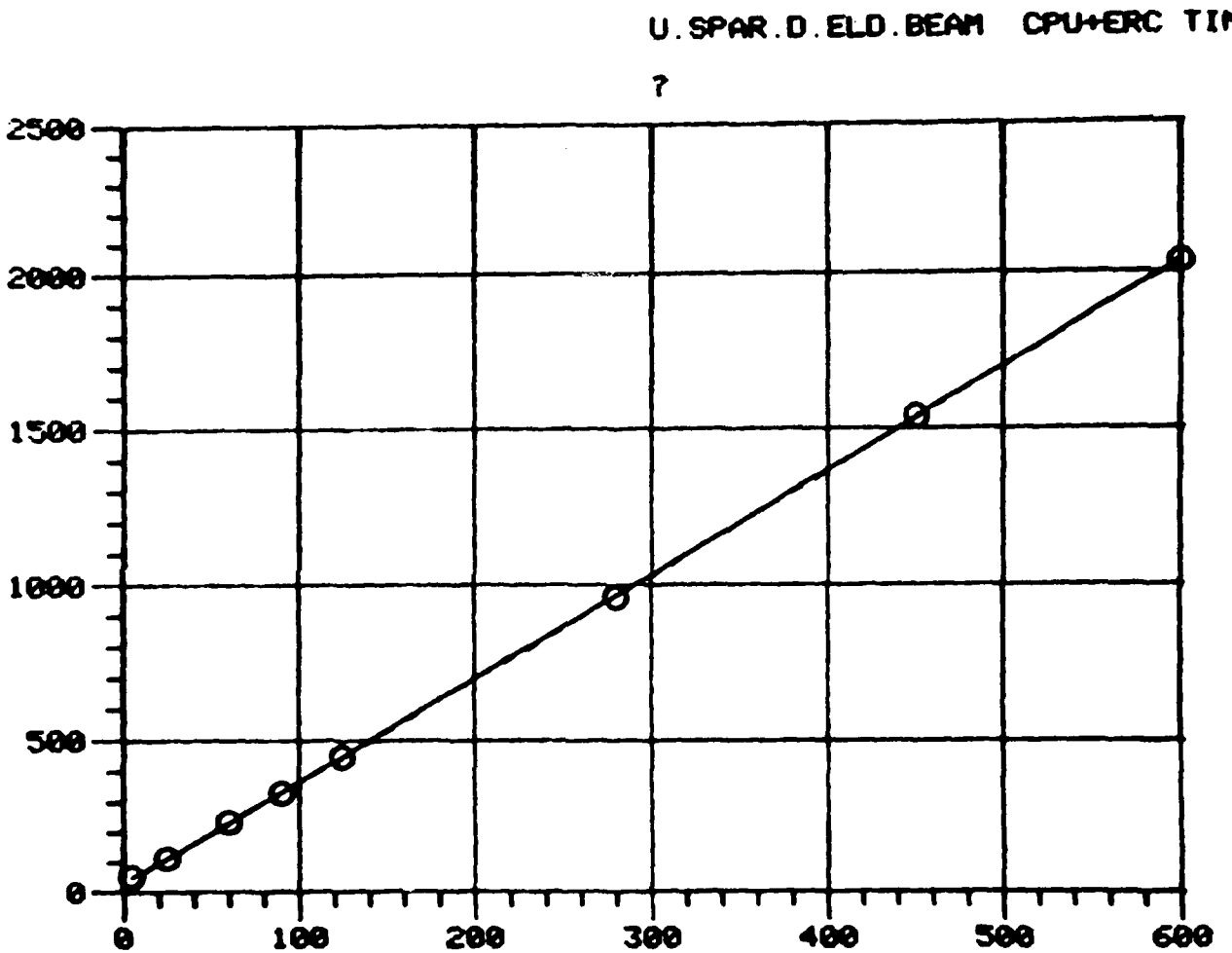


Figure 8.13: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor ELD

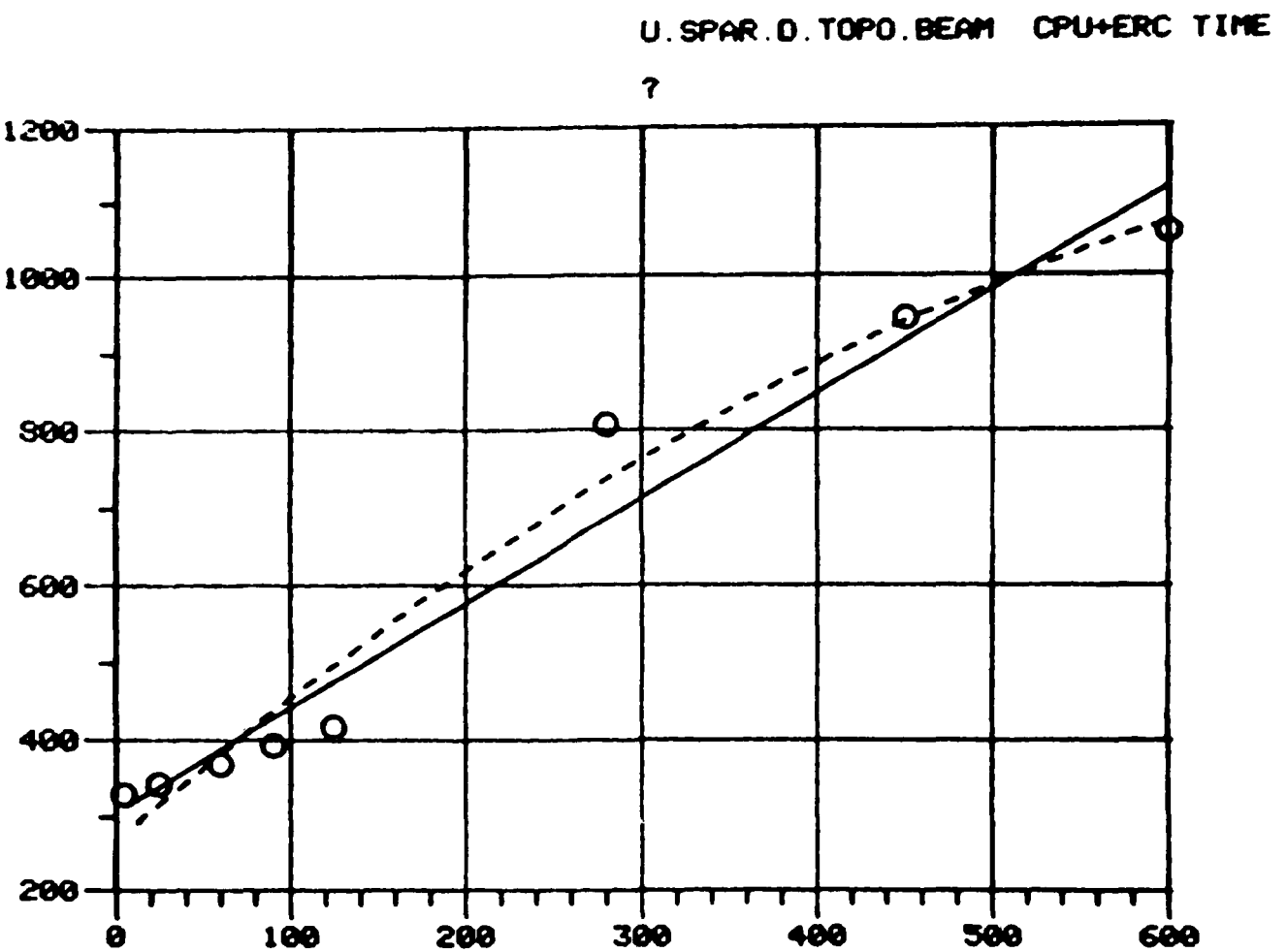


Figure 8.14: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor TOPO

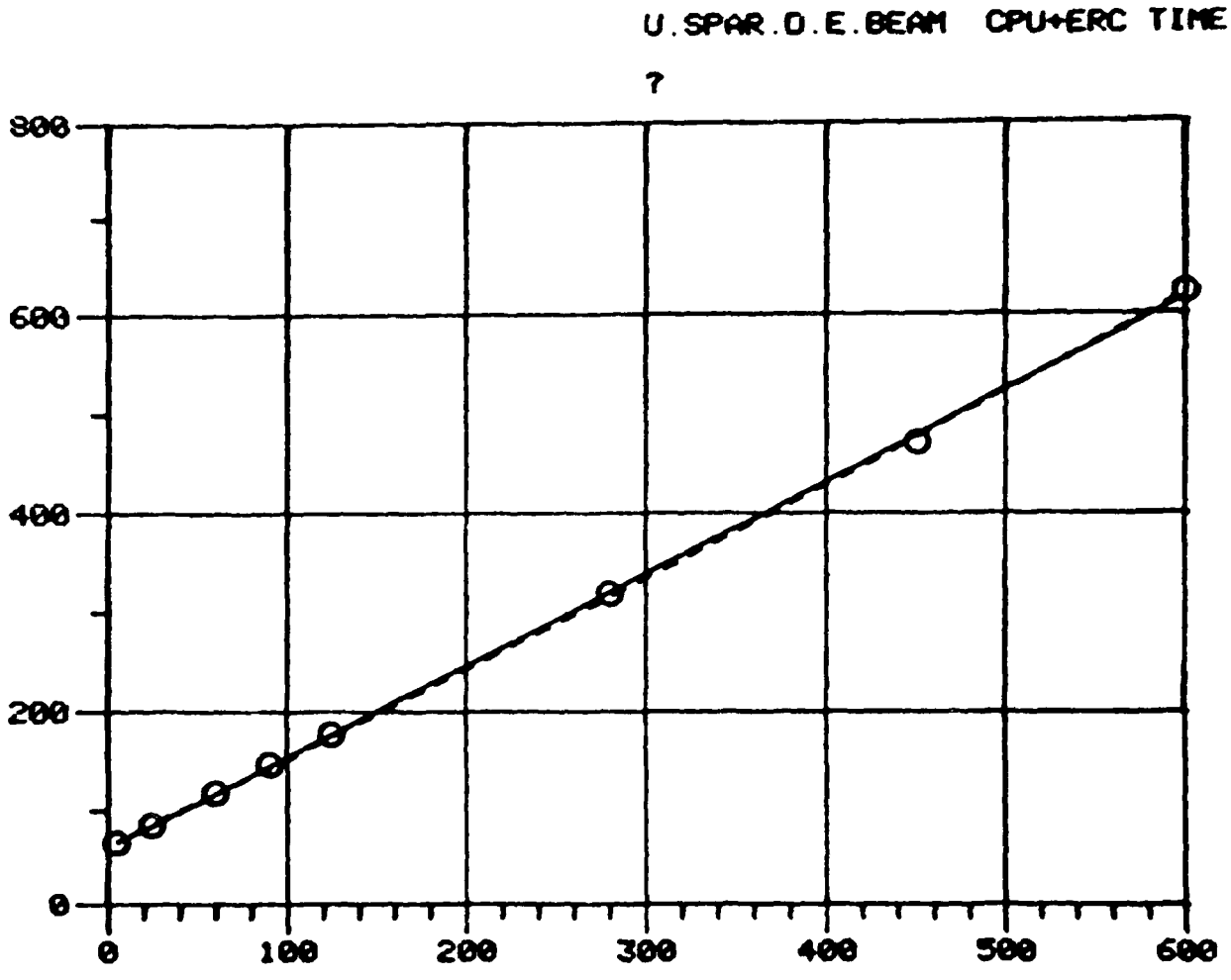


Figure 8.15: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor E

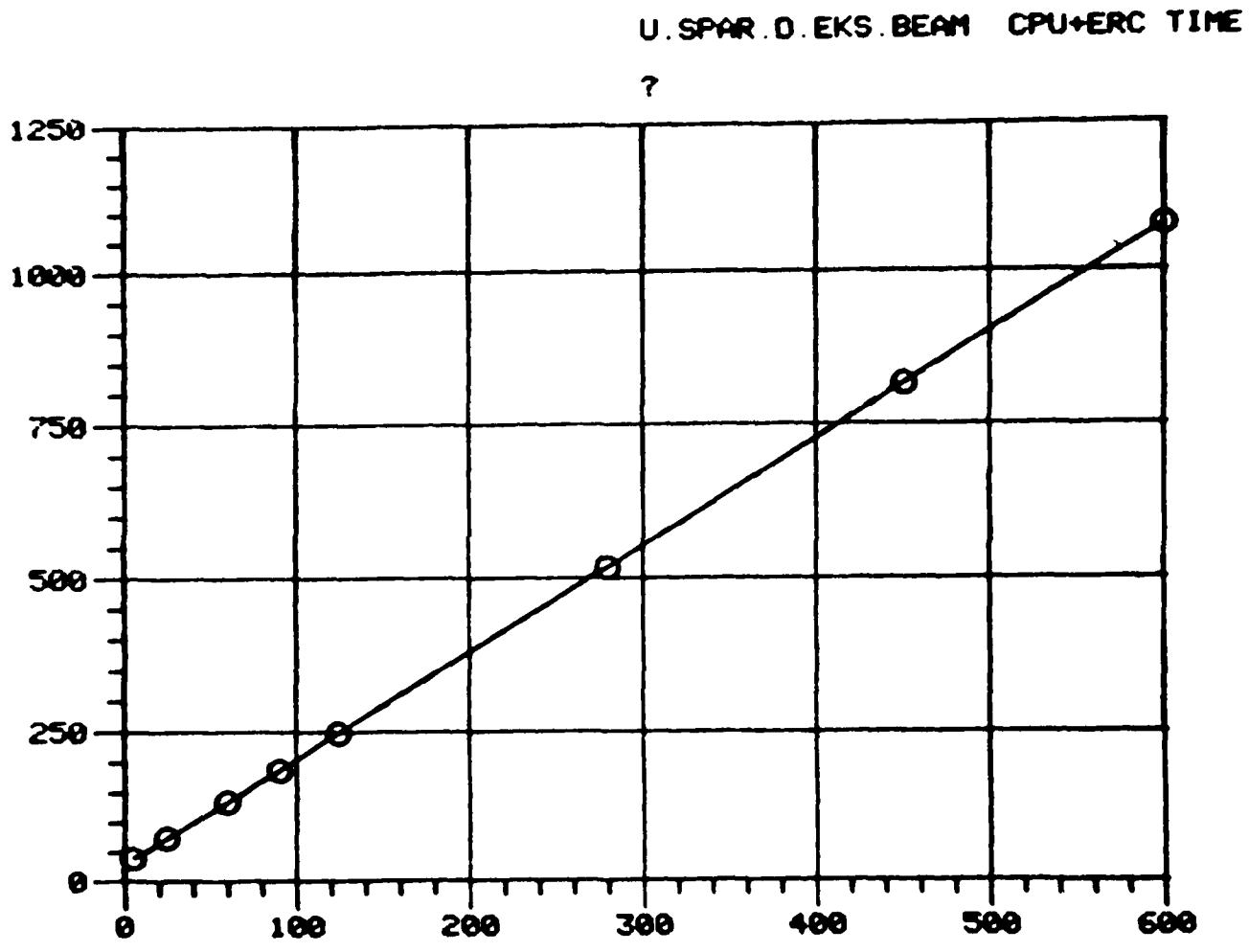


Figure 8.16: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor EKS

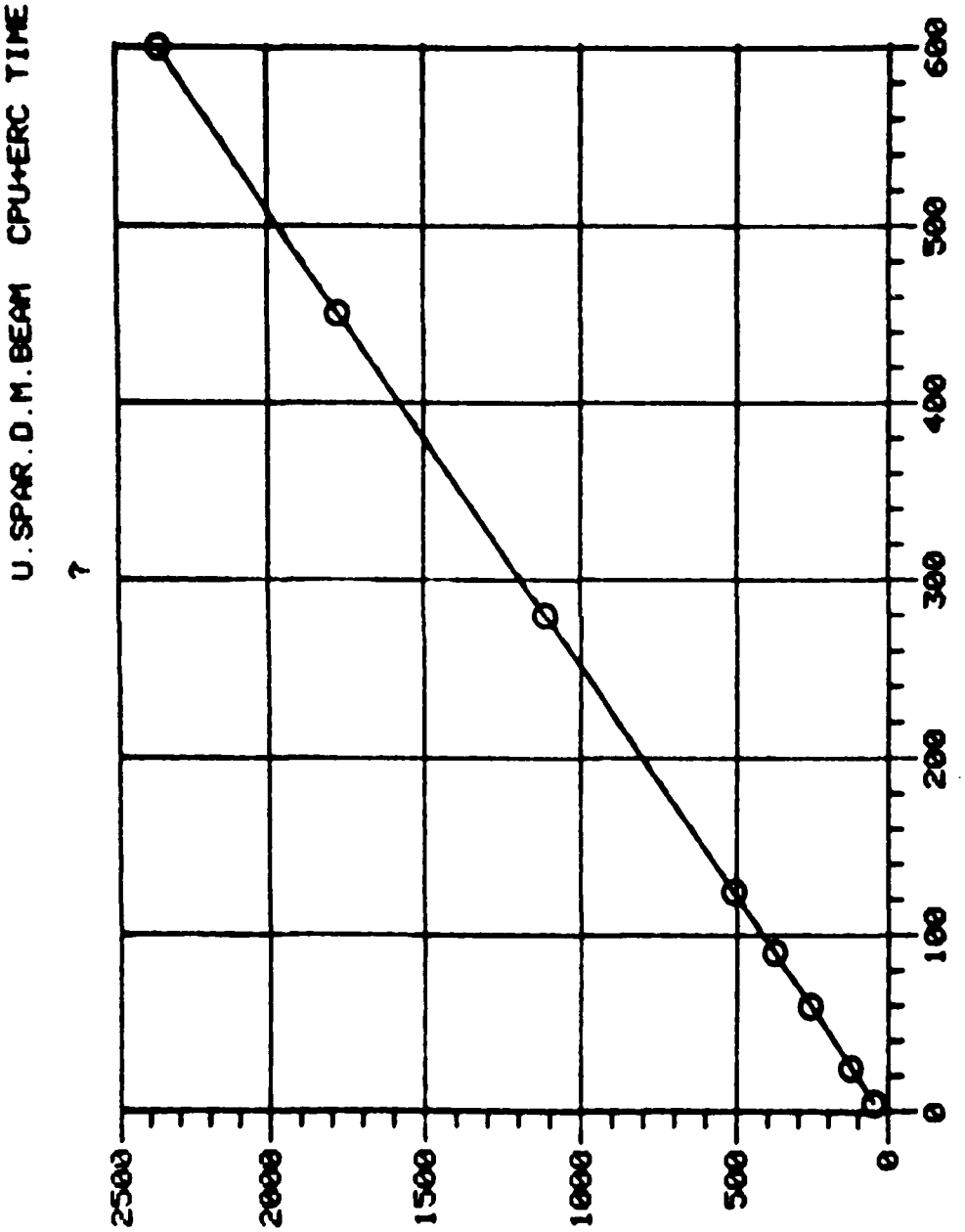


Figure 8.17: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor M

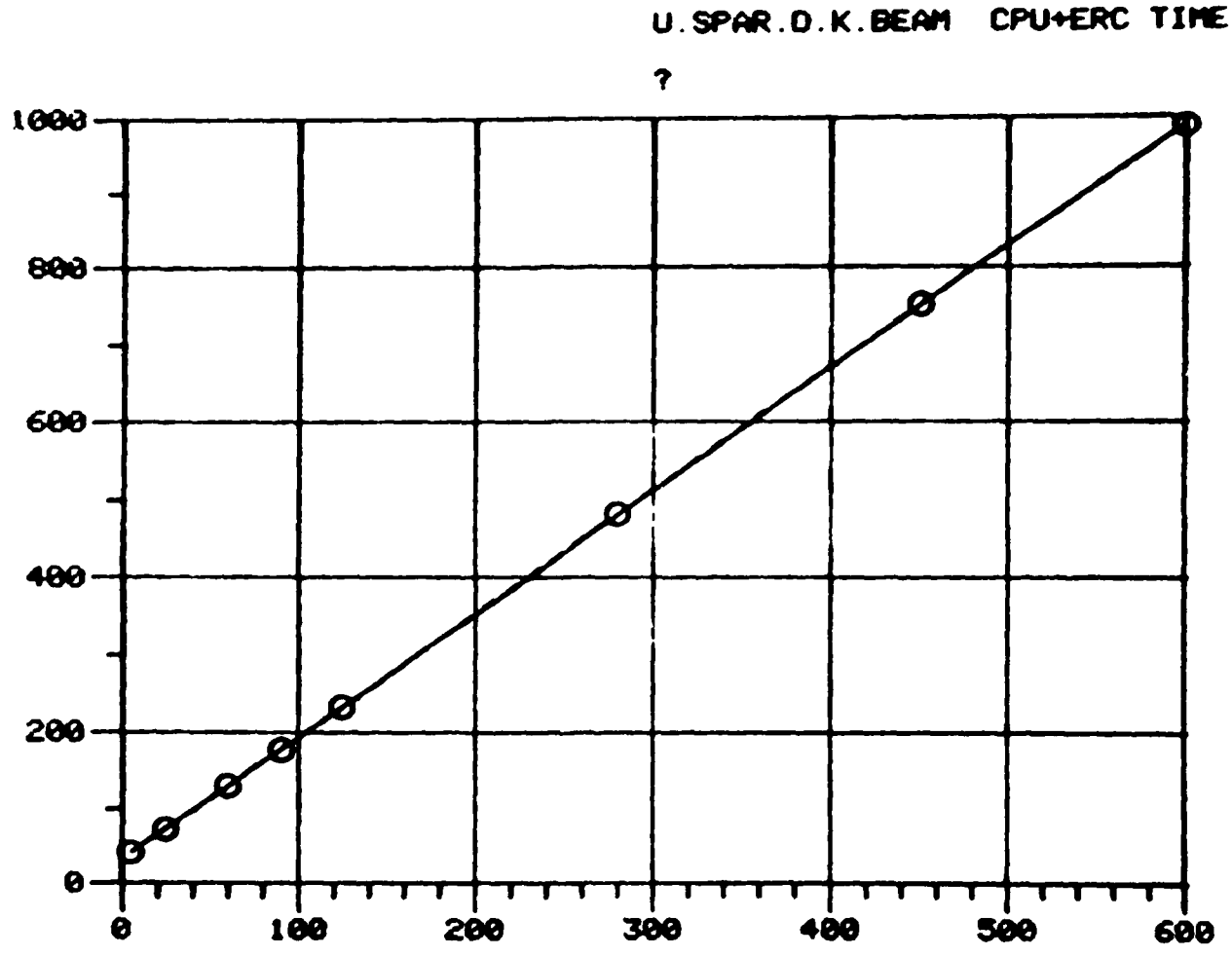


Figure 8.18: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor K

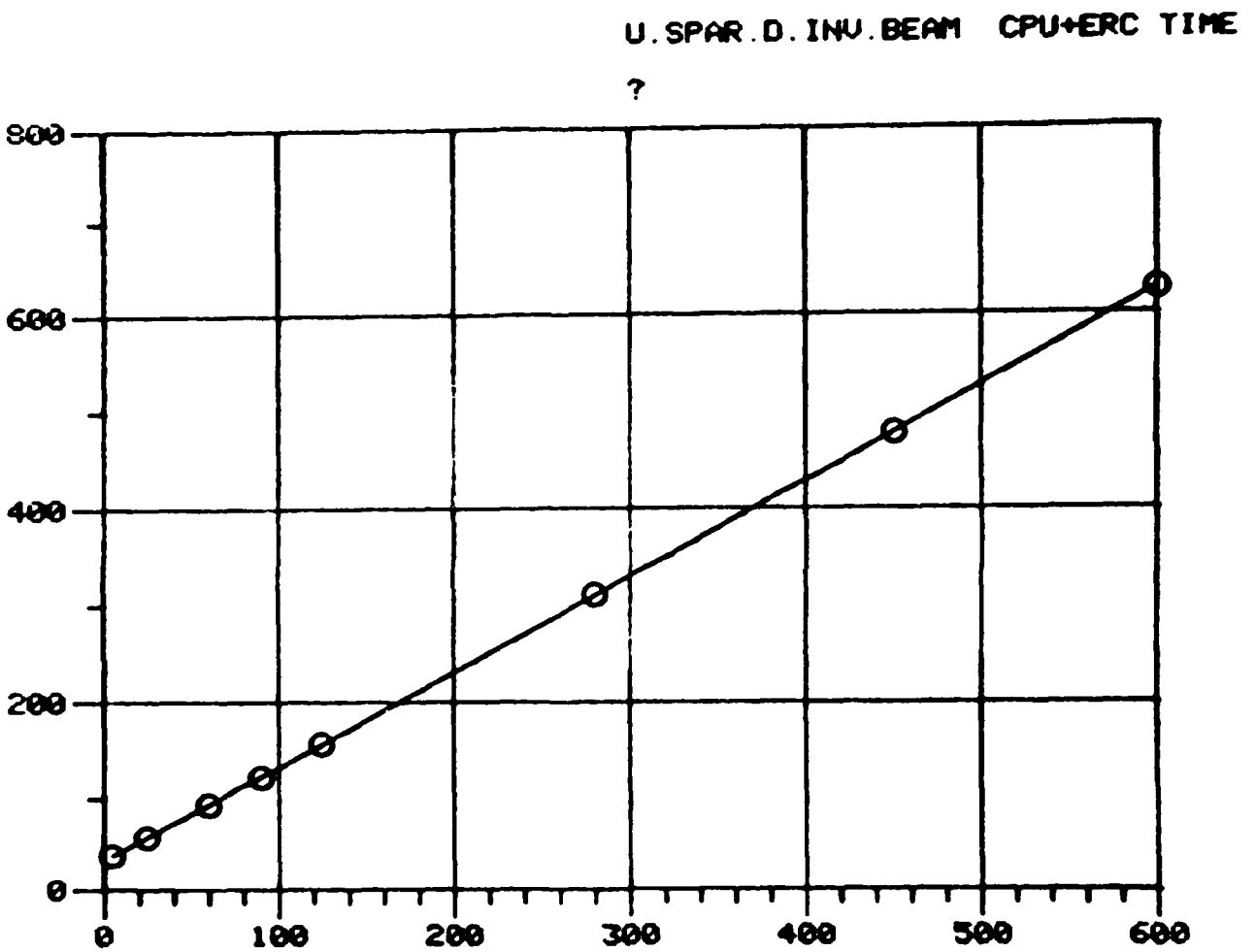


Figure 8.19: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor INV

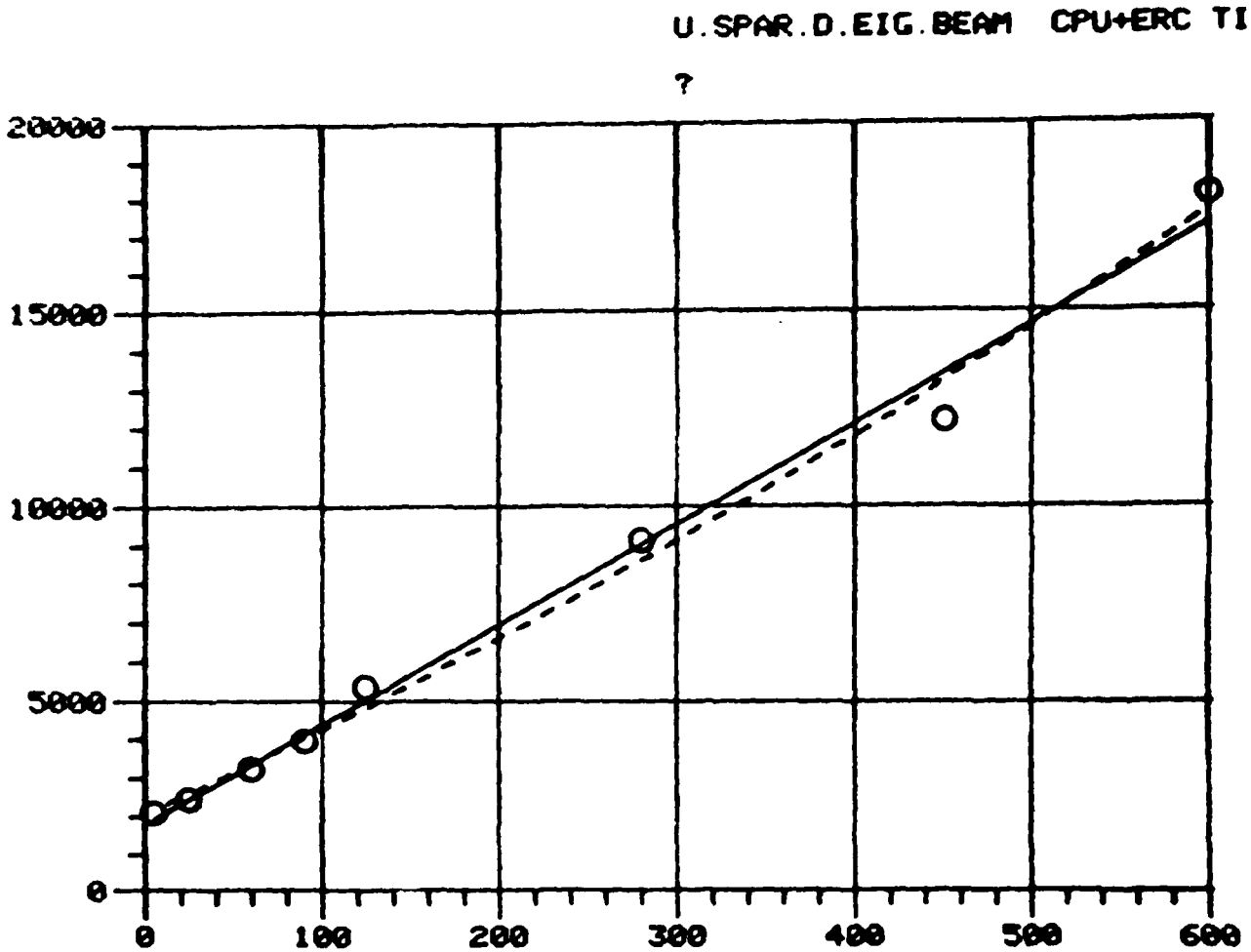


Figure 8.20: CPU+ERC Time (sec) vs. Number of Nodes for Beam Problem on UNIVAC: Processor EIG

E,M and EIG have slight quadratic tendencies. TOPO shows stepwide behavior. Resource consumption on the UNIVAC is predictable and not dependent on other workload or its contention for system resources.

For a function of the form

$$CPU_u = f(x) = c_o + c_1 x + c_2 x^2 \quad (x \text{ is no. of nodes})$$

the coefficients are given in Table 8.1. Analogous to the CPU-function, an I/O time function can be defined depending on the problem size:

$$IO_u = g(x) = d_o + d_1 x + d_2 x^2 \quad (x \text{ is no. of nodes})$$

The coefficients are given in Table 8.2. TOPO shows again step function behaviour, E,EKS, and INV are slightly quadratic. EIG's I/O needs are between a linear and a stepwise function of the problem size. All other processors resource needs are linear functions of the problem size. Both CPU and I/O times are reproducible on the UNIVAC between different runs. This is not at all true for the PRIME.

(ii) PRIME

On the UNIVAC the relationship between problem size and resources used is very close to linear. As opposed to the UNIVAC where CPU and I/O times are reproducible, on the PRIME only CPU time is fairly stable regardless of the number of other users and their workload. I/O time varies greatly with system load, as much as +350% was observed above single

	C_0	C_1	C_2
TAB	428.923	2.57652	
ELD	30.6118	3.34269	
TØPØ	306.212	1.35297	
E	62.7211	8.84745E-01	7.24682 E-05
EKS	30.7085	1.74234	
M	27.3906	3.86747	2.44555 E-05
K	34.1455	1.59195	
INV	32.2601	9.89308E-01	
EIG	2078.19	2.10312E-01	8.07338E-03
Extra	1864.0		
Entire Run	4925.84	38.109	7.01927 E-03

Table 8.1: Curve Fitting Coefficients for CPU Consumption for Beam Problem on SPAR (UNIVAC)

	d_0	d_1	d_2
TAB	595.215	1.39369	
ELD	441.288	1.93669	
TØPØ	431.941	2.12298	
E	594.906	1.44295	1.648 E-04
EKS	401.836	8.25819 E-01	4.9918 E-04
M	377.765	1.80876E+01	
K	370.656	1 81532E+01	
INV	403.323	8.61281E-01	2.46417E-04
EIG	2202.05	2.50931E+01	
Extra	8460.		
Entire Run	15.745	4.08766E-01	5.07216E-04

Table 8.2: *Curve Fitting Coefficients for I/O Consumption for Beam Problem on SPAR (UNIVAC)*

user environment. The CPU times are slightly affected by large variations in I/O time on the PRIME. This is because setting up, initiating and terminating I/O operations is regarded as CPU time. On PRIME the runs were I/O bound.

As before, resource consumption was measured by run and by processor, this time for different workload conditions. The number of users logged on the PRIME varied between two and twenty-five. The curve fitting results for CPU time and disk time of the entire run as a function of the number of nodes in the Beam are given in Figures 8.21 and 8.22. Selected processor results are shown in Figures 8.23 to 8.30. Analogous to the UNIVAC results we fitted curves of the form

$$CPU_p = f(x) = c_0 + c_1 x + c_2 x^2 \quad (x \text{ is no. of nodes})$$

for the CPU time, and

$$IO_p = g(x) = d_0 = d_1 x + d_2 x^2 \quad (x \text{ is no. of nodes})$$

for the I/O time. Tables 8.3, 8.4 show the curve fitting results for CPU and I/O time as a function of the number of nodes for each processor. As on the UNIVAC most processors exhibit linear behavior for their resource consumption. For CPU time only INV and EIG have quadratic tendencies whereas for the I/O time K also is slightly quadratic.

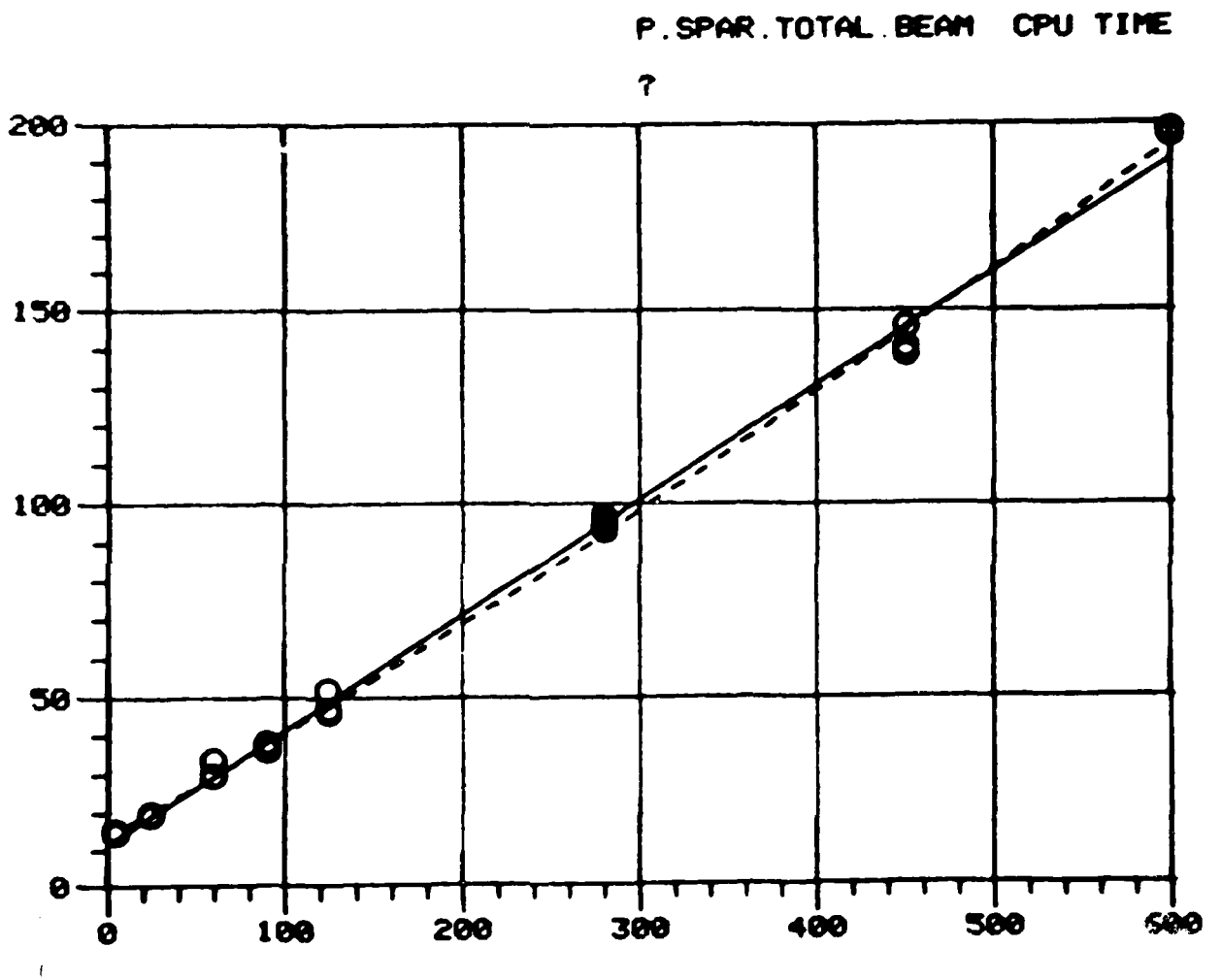


Figure 8.21: Total CPU time (sec) vs. Number of Nodes for Beam Problem on PRIME

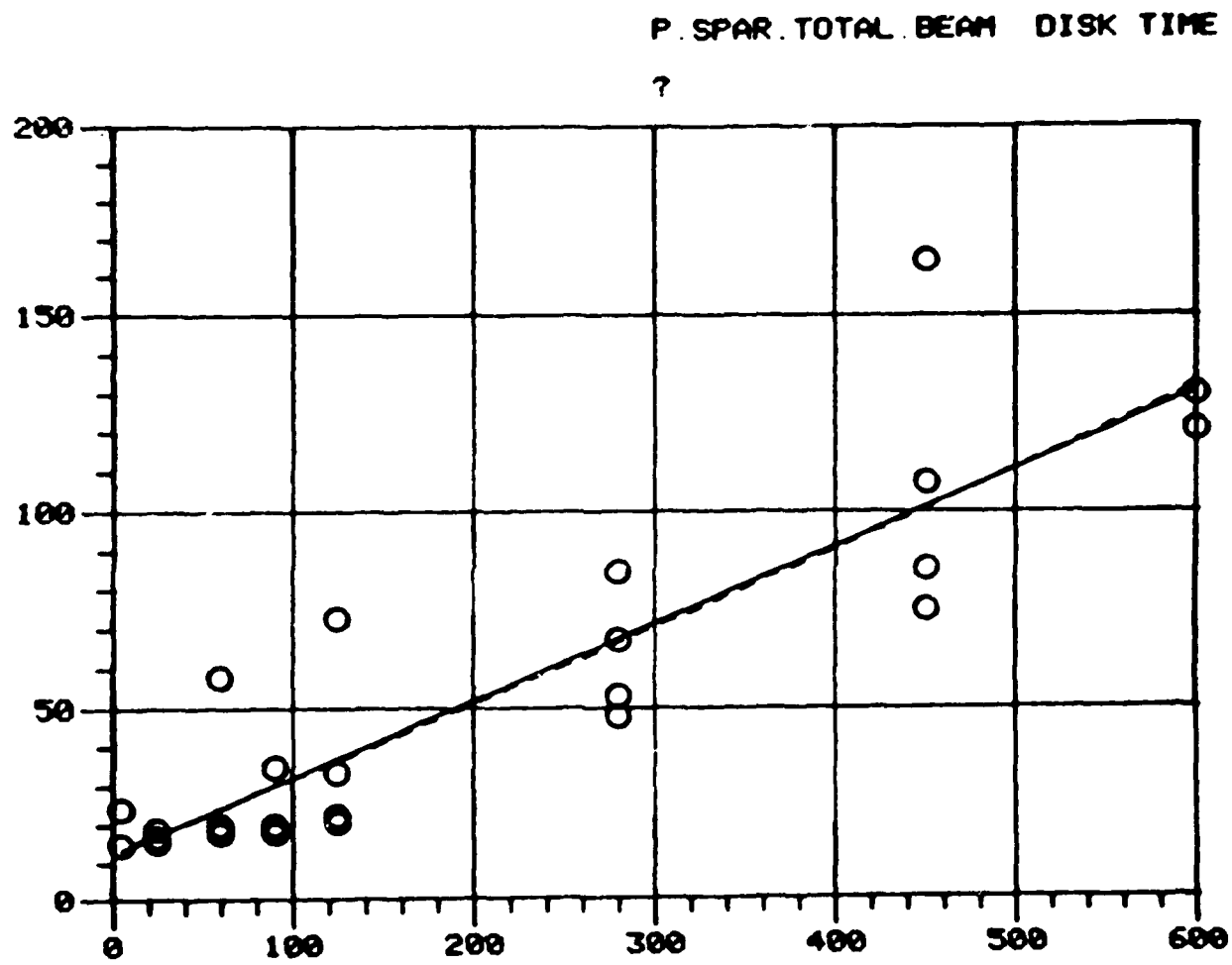


Figure 8.22: Total Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME

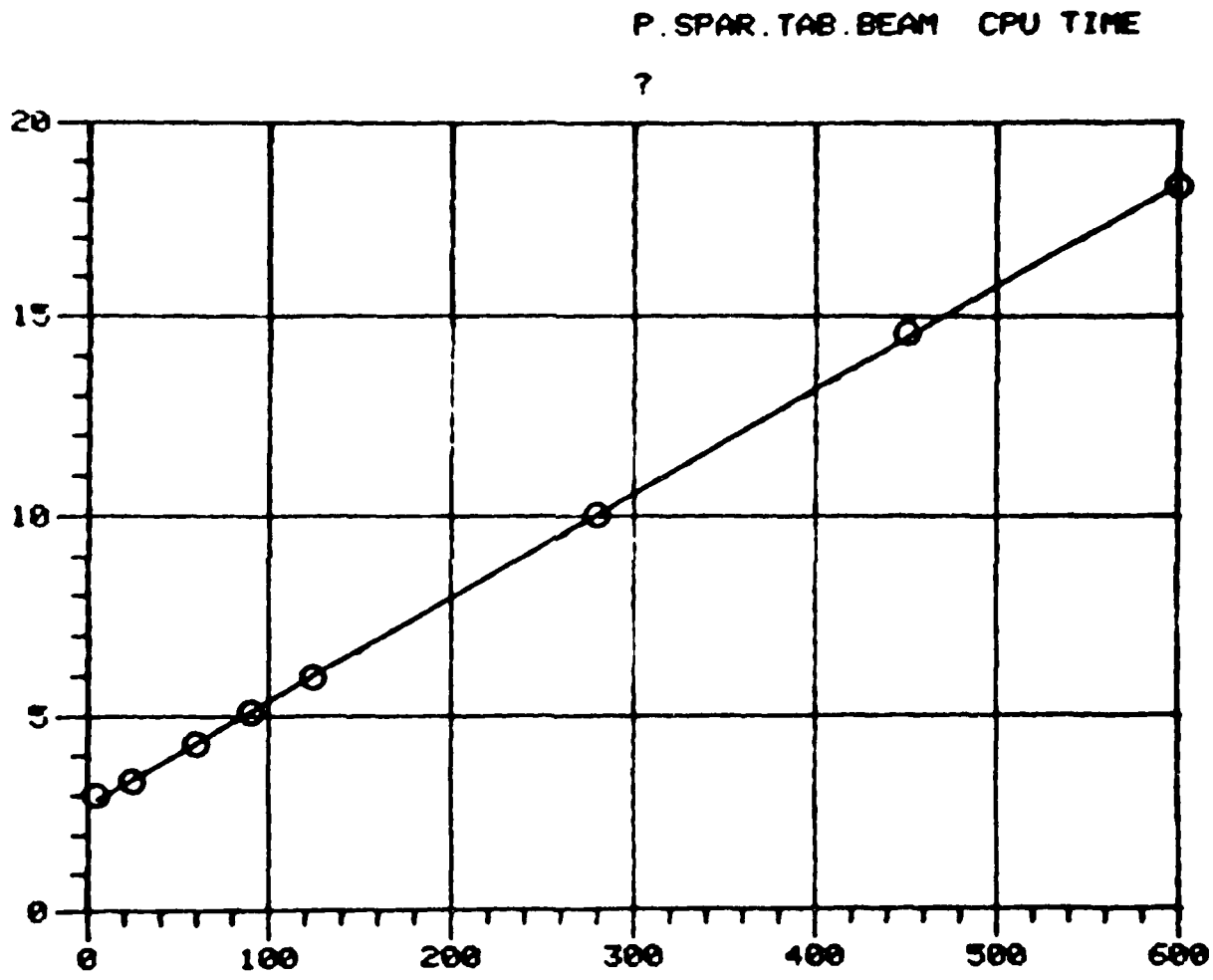


Figure 8.23: Average CPU Time (sec) vs. Number of Nodes for Beam Problem PRIME: Processor TAB

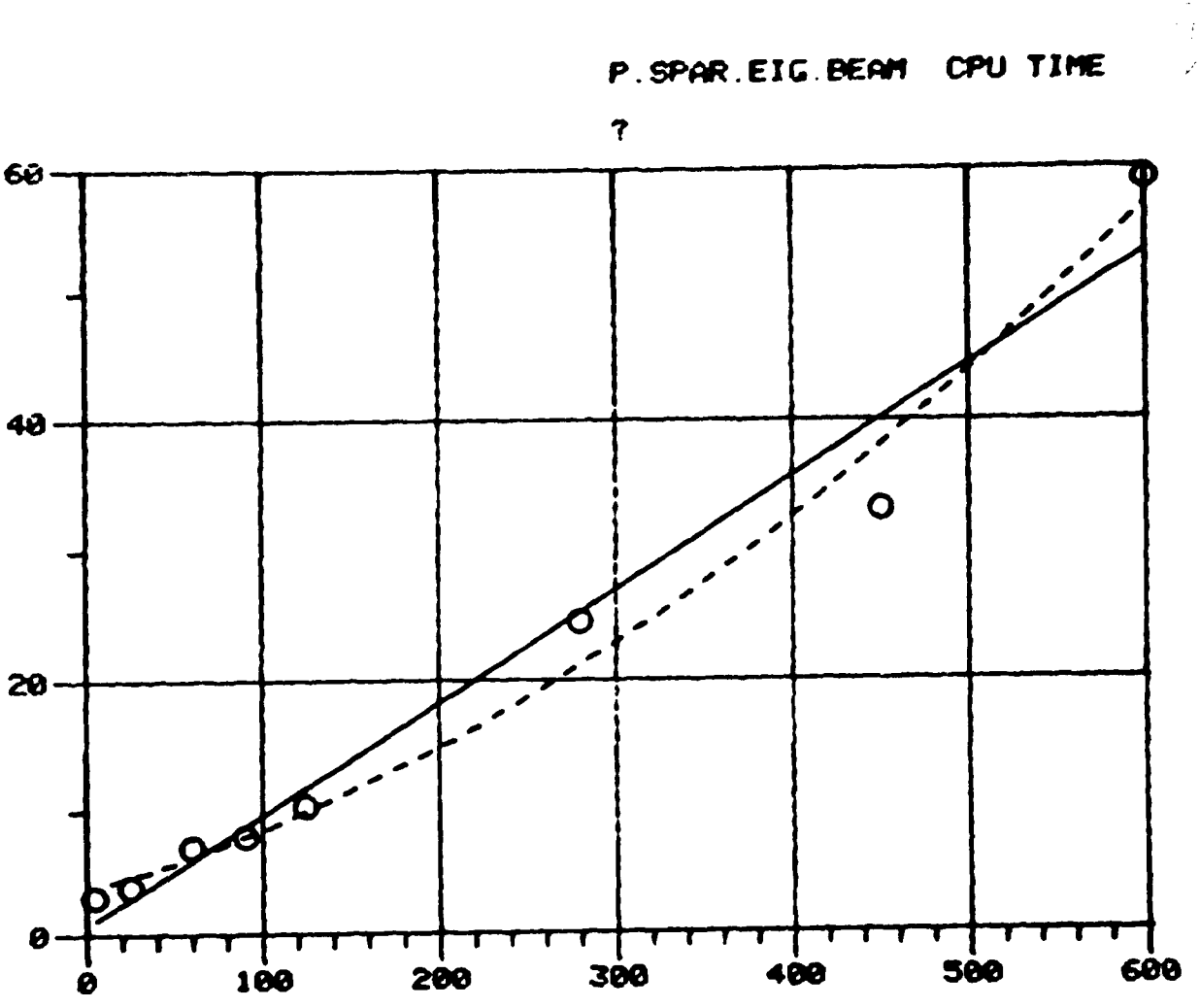


Figure 8.24: Average CPU Time (sec) vs. Number of Nodes for Beam Problem PRIME: Processor EIG

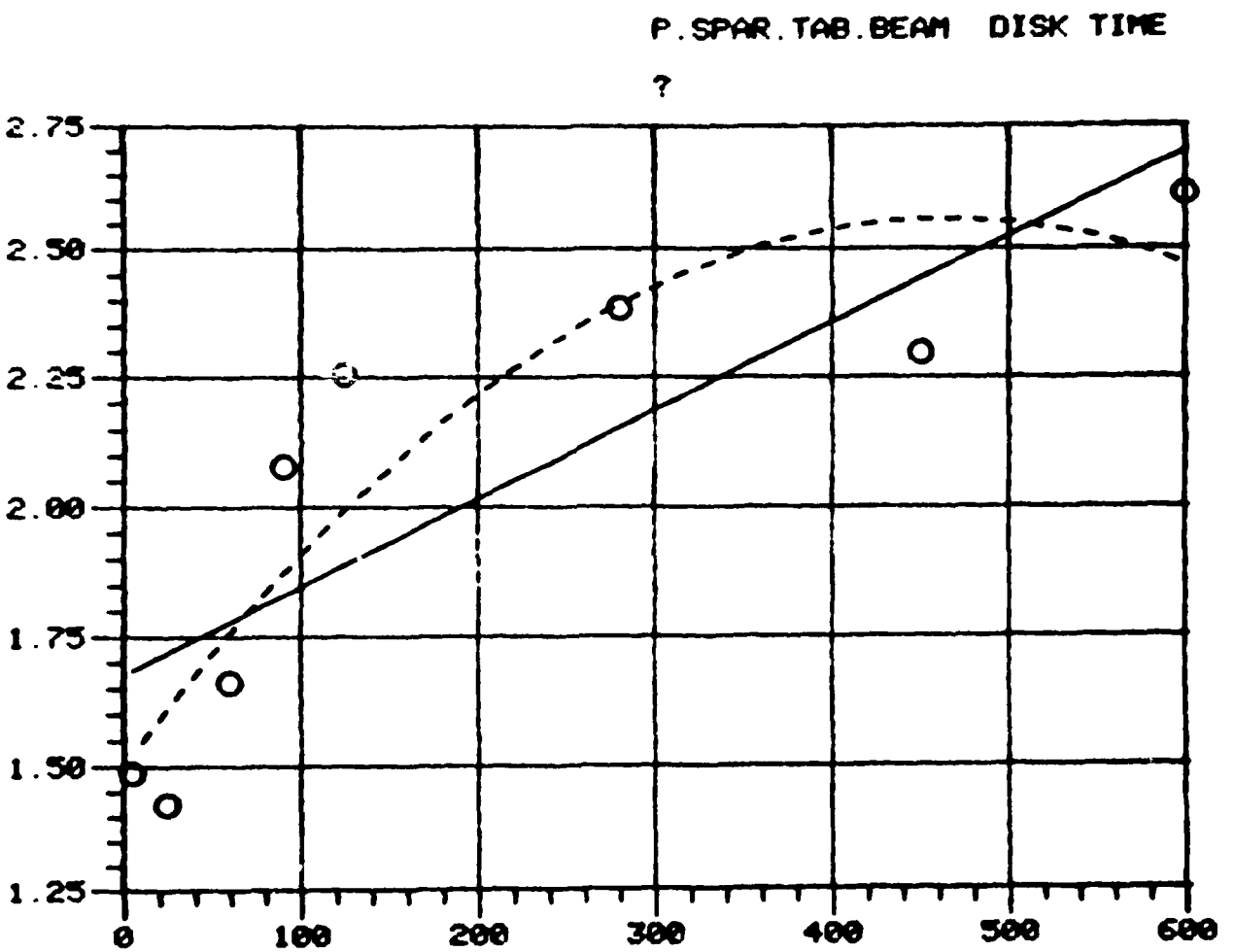


Figure 8.25: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor TAB

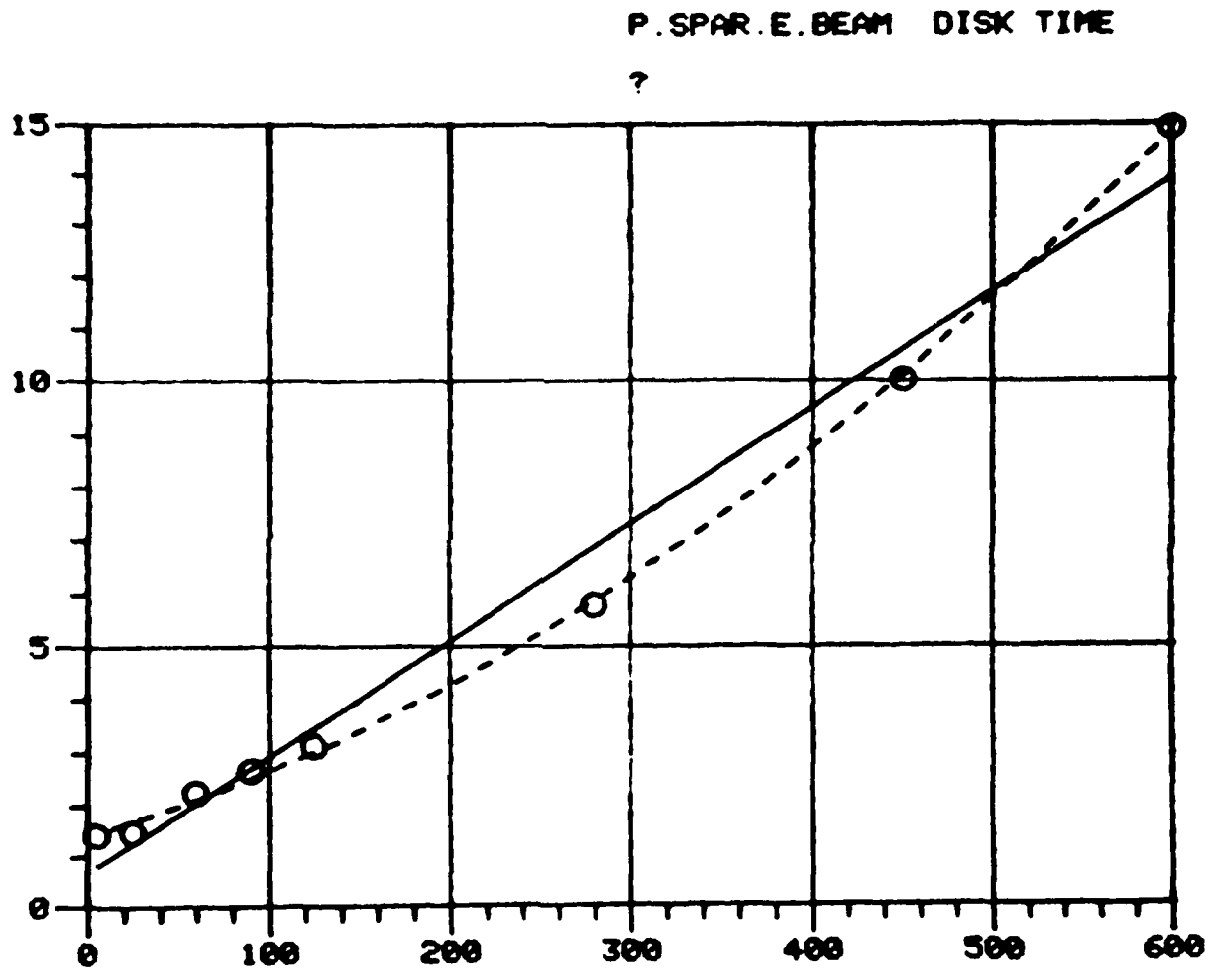


Figure 8.26: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor E

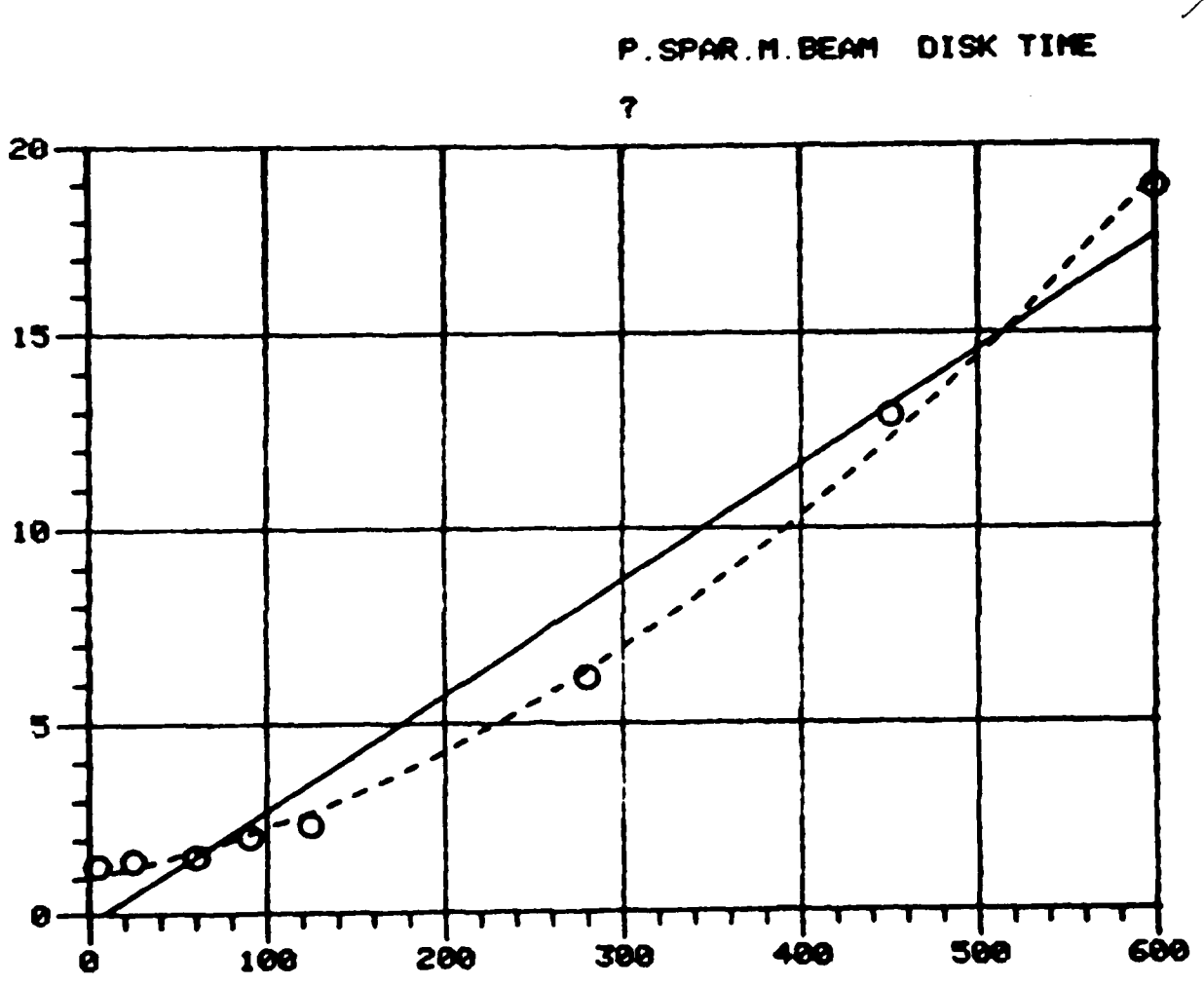


Figure 8.27: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor M

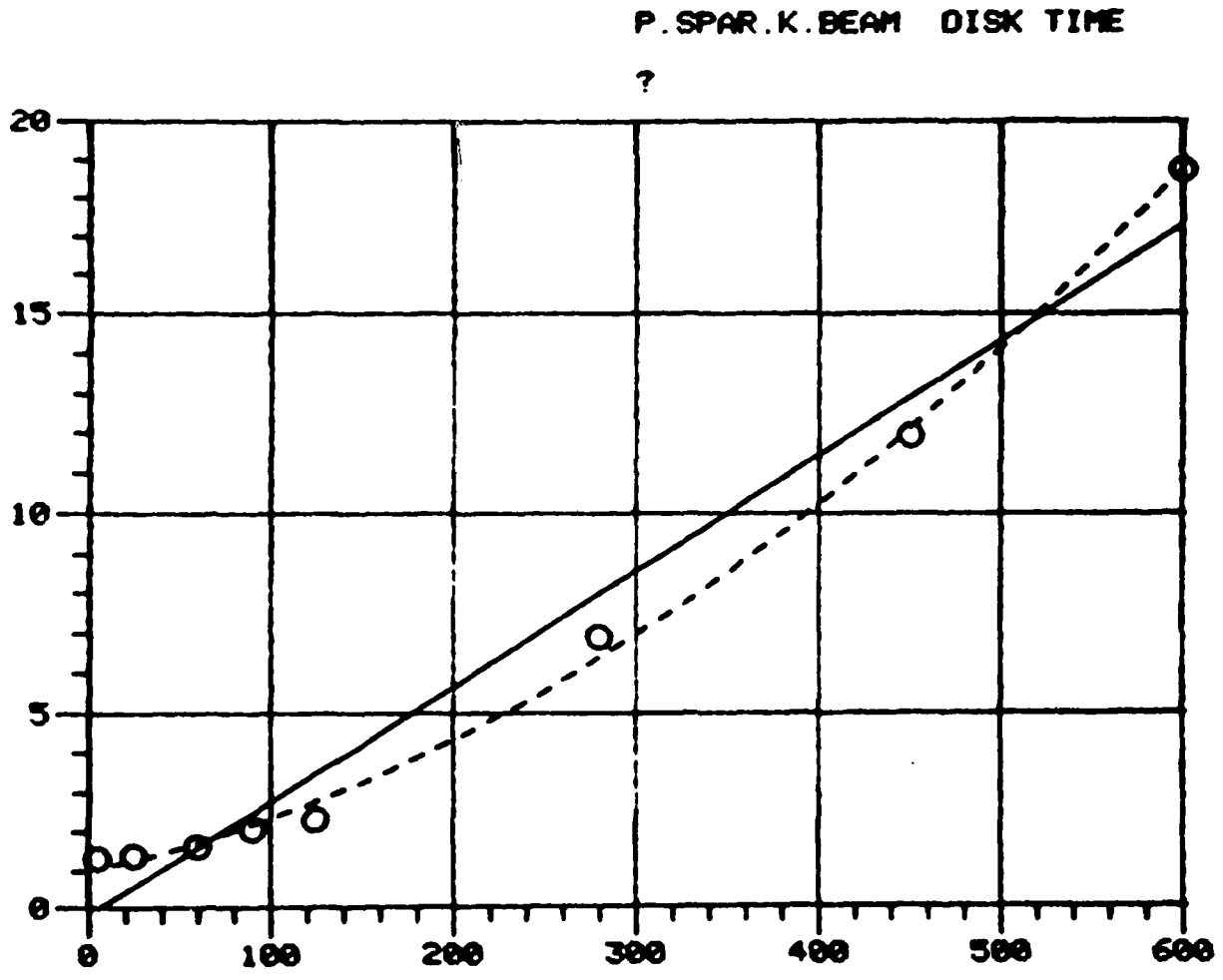


Figure 8.28: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor K

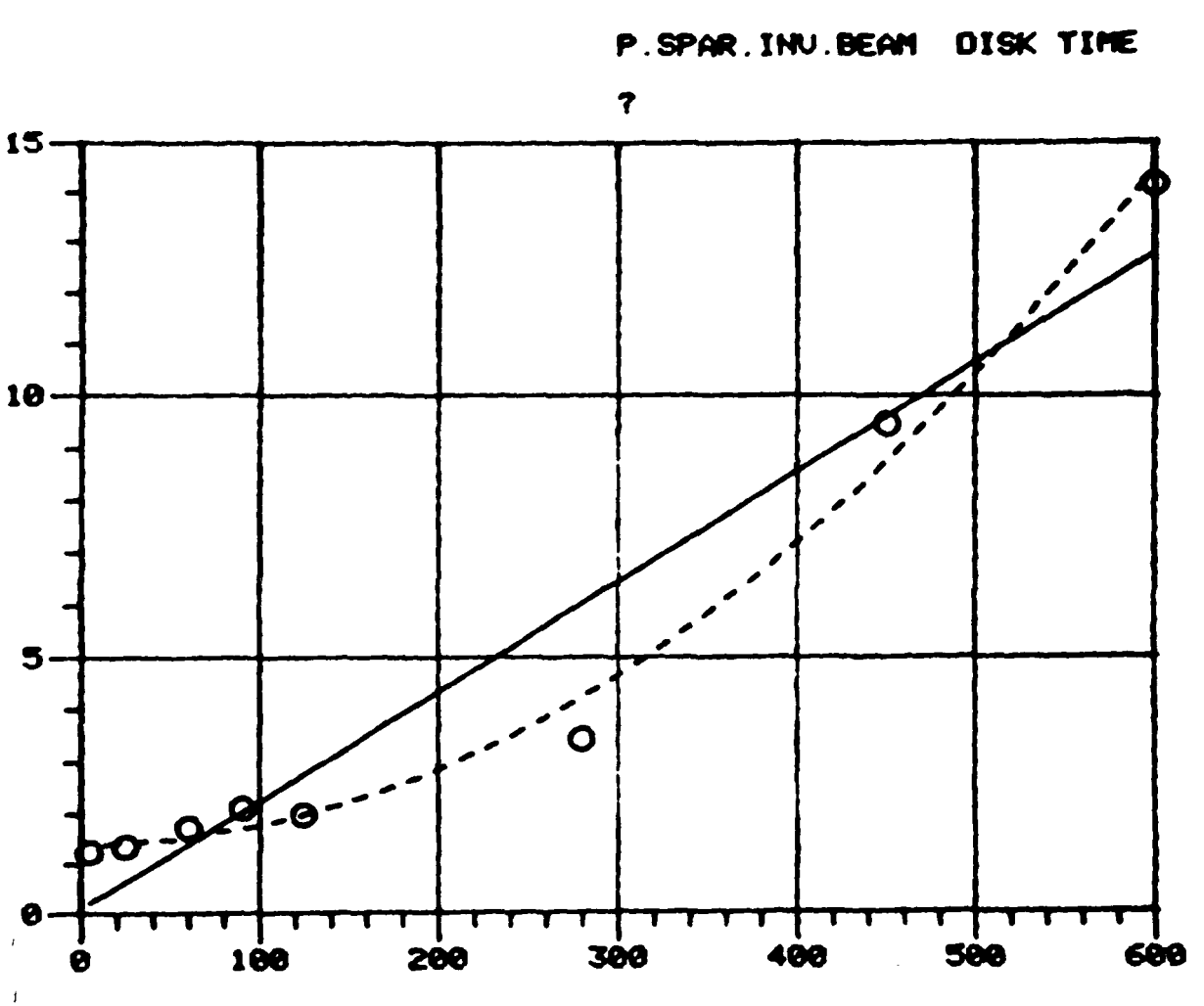


Figure 8.29: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor INV

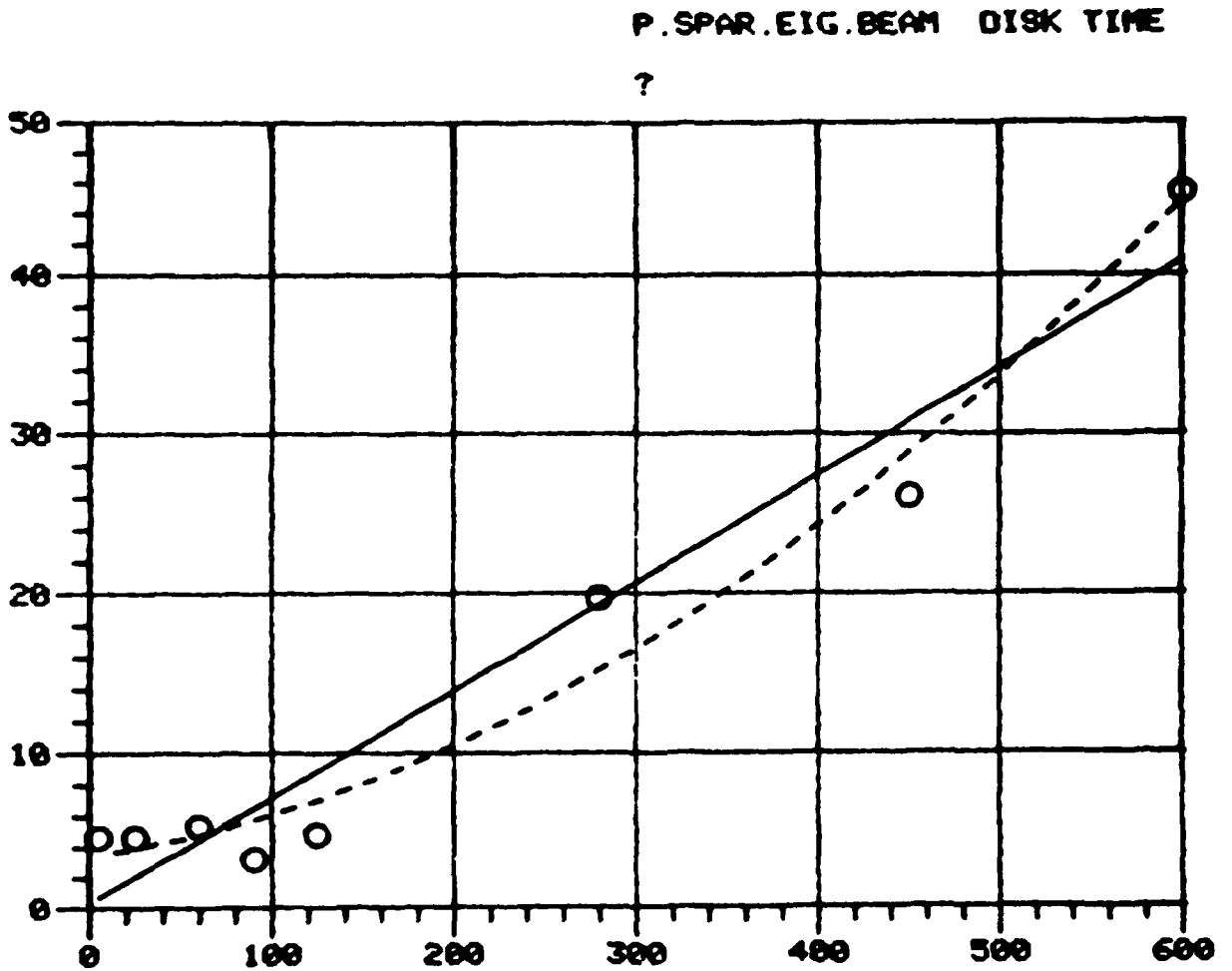


Figure 8.30: Average Disk Time (sec) vs. Number of Nodes for Beam Problem on PRIME: Processor EIG

	c_0	c_1	c_2
TAB	2.82175	2.60476E-02	
ELD	1.22498	2.37648E-02	
TØPØ	1.14709	1.2603E-02	
E	1.12291	1.9357E-02	
EKS	1.06172	3.54174E-02	
M	.96290	5.97628E-02	
K	.945755	2.89618E-02	
INV	.987217	1.09337E-02	3.32624E-06
EIG	3.63656	4.30206E-02	7.12645E-05
Entire Run	13.7082	2.61343E-01	6.63862E-05

Table 8.3: Curve Fitting Coefficients for CPU Consumption for Beam Problem on PRIME

	d_0	d_1	d_2
TAB	2.65327	1.44986E-03	
ELD	1.29594	2.50116E-03	
TØPØ	1.21505	8.56351E-03	
E	1.219	2.6913E-02	
EKS	2.20327	1.53554E-02	
M	.411161	3.0679E-02	
K	.77813	2.36577E-02	1.14976E-05
INV	1.4734	6.3223E-03	2.40503E-05
EIG	10.52890	9.77343E-03	7.14104E-05
TØTAL	12.8567	1.87896E-01	1.64497E-05

Table 8.4: Curve Fitting Coefficients for Disk Time Consumption for Beam Problem on PRIME

Additionally the CPU and I/O functions for the user environment were analysed for all processors. The disk times for EIG, INV,K,M, and E are slighlty quadratic, and so is the CPU time for EIG. The CPU time for TAB is linear, but the disk time for TAB is very erratic and neither linear nor quadratic as a function of the number of nodes in the beam. All other CPU and I/O plots were linear.

c) Charges

From the results on resource consumption, it can be expected that the charges will also be close to linear as a function of the problem size, i.e. cost is directly related to the number of nodes.

(i) UNIVAC

Figure 8.31 - 8.33 show only slightly quadratic behaviour for the total run cost (Fig. 8.31), the core cost (Fig. 8.32) and the CPU cost (Fig. 8.33). When the resource consumption was contrasted to the charges, some inconsistencies were discovered which are not explained by the charging algorithm.

*KCB (i.e. Kilo Core Block) time charges are deferred (up to 50% of the total) until the last job step. There is no explanation for this. Since this makes an analysis of memory charges by job step and hence by processor inaccurate, no such analysis was performed.

*ERC (Executive Request Call) time, which is a part of the CPU time, and the charges for ERC time do not match. It was found that less ERC time is charged than

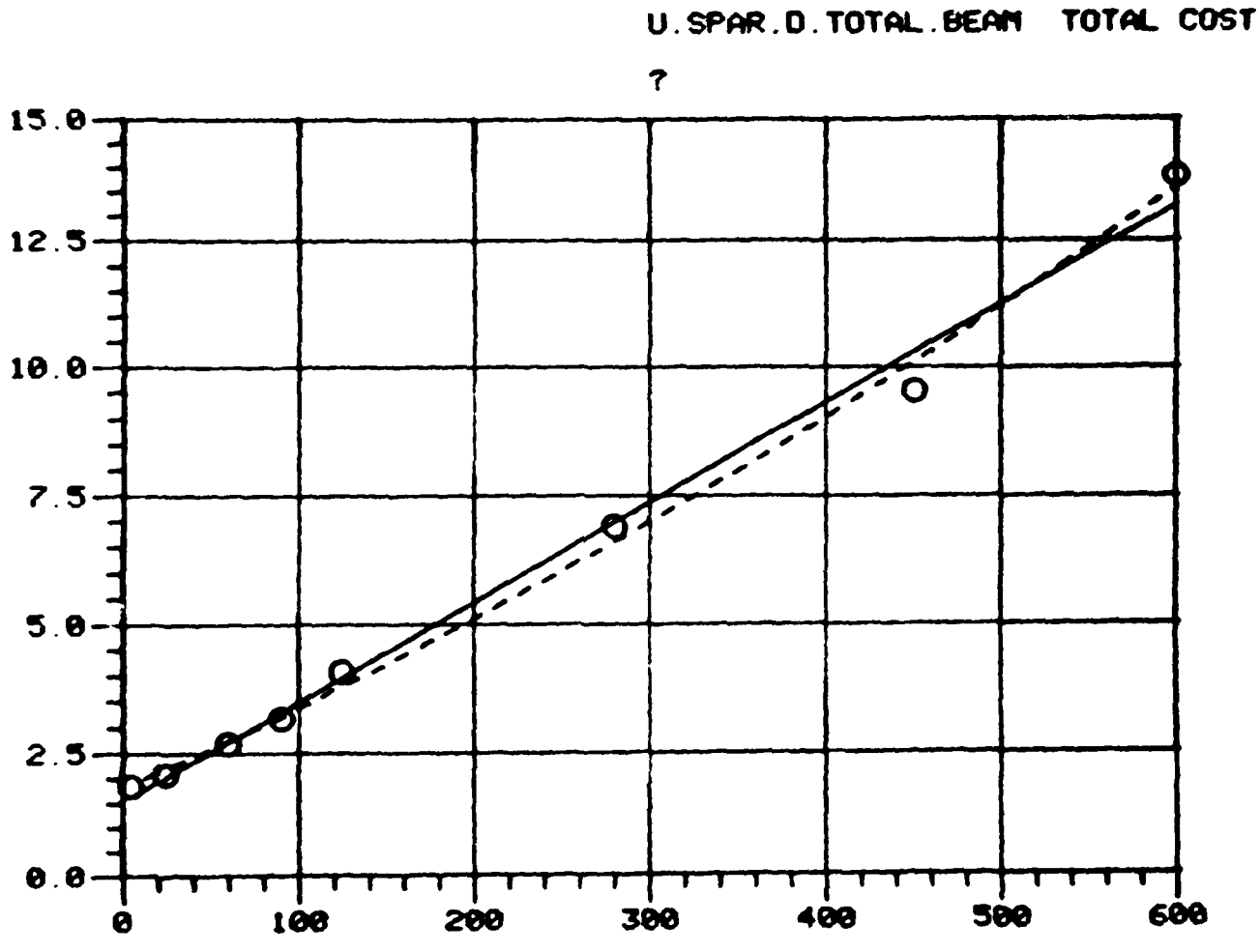


Figure 8.31: Total Cost (dollars) vs. Number of Nodes for Beam Problem on UNIVAC

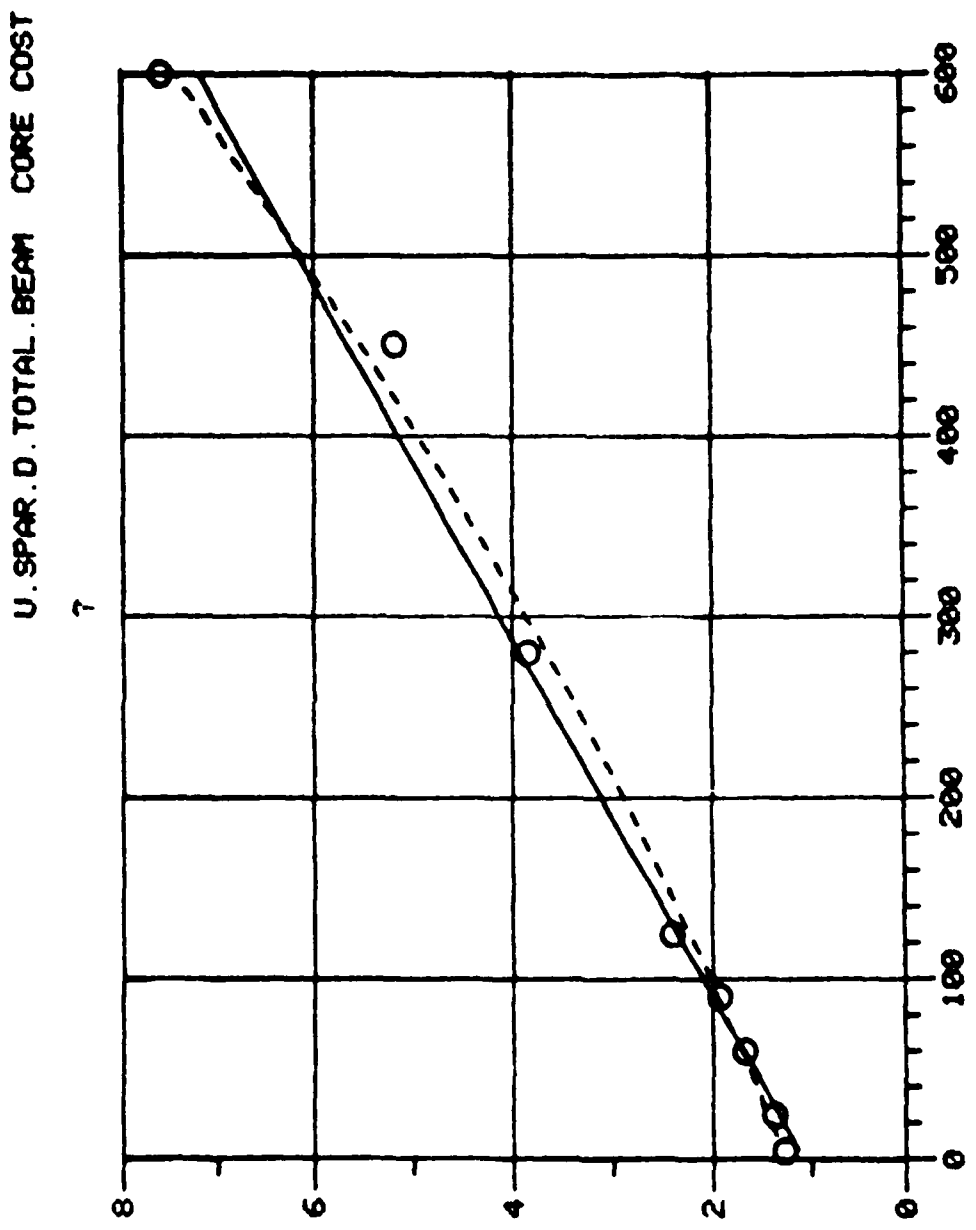


Figure 8.32: Core cost (dollars) vs. Number of Nodes for Beam Problem on UNIVAC

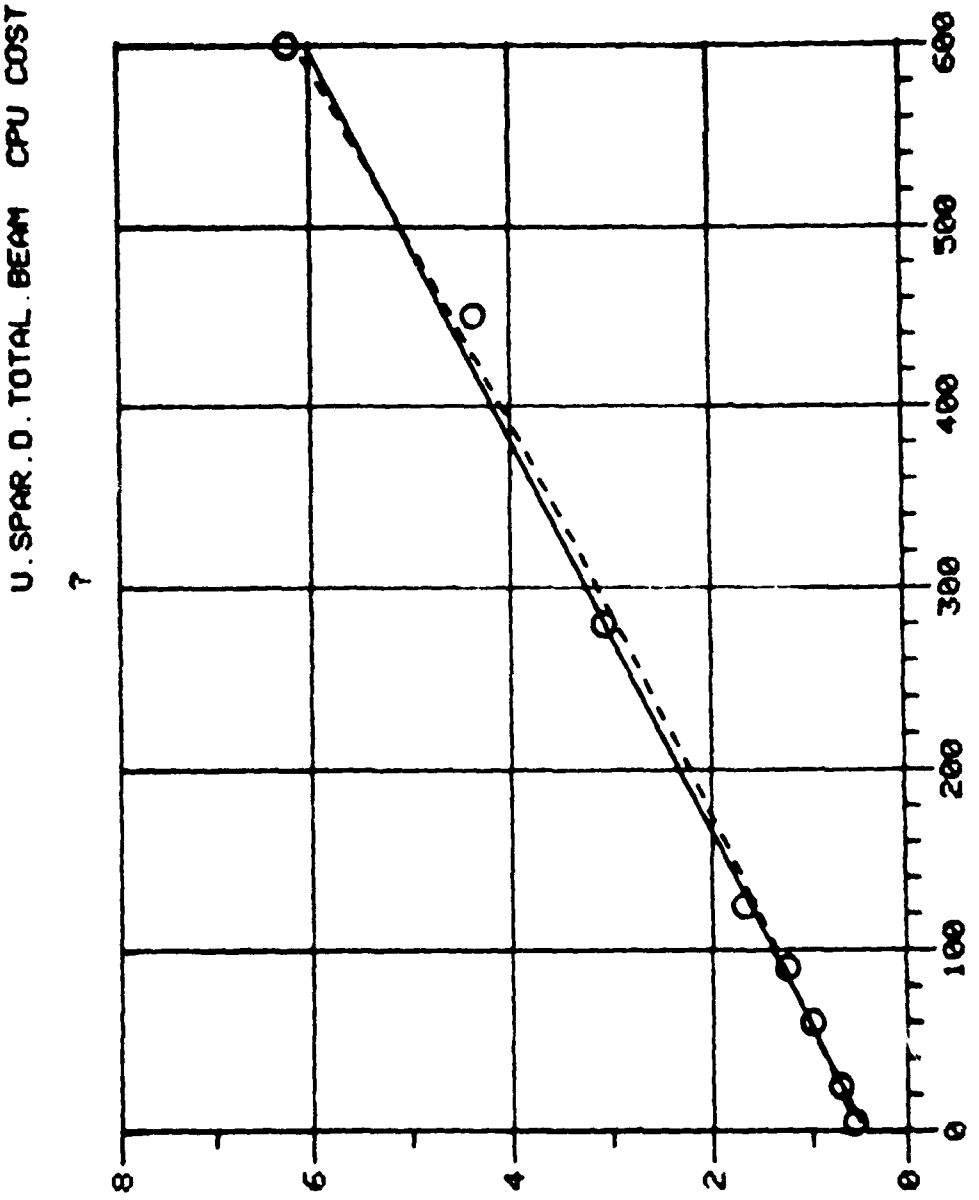


Figure 8.33: CPU Cost (dollars) vs. Number of Nodes for Beam Problem on UNIVAC

consumed, constituting a credit of approximately 3.12 seconds for each job.

*When the core blocks needed and the core blocks charged were compared, it appeared that normally 4 to 5 extra core blocks are charged, above and beyond the actual program size. These blocks are probably used for I/O buffers and task control blocks.

*I/O charges are based on internal, device-dependent operating system constants for transfer rate, seek time, etc. The use of faster devices greatly reduces I/O charges, because the user is charged for the time it takes to transfer data to and from a device regardless of the type. This makes the use of the fastest possible devices highly desirable. To avoid an overload for them, the system imposes size restrictions on the files stored on fast devices and automatically assign slower devices for big files, thus penalizing the use of large files.

(ii) PRIME

Due to the variation in resource consumption, especially as far as disk time is concerned, the charges also vary. The curve fitting in Figure 8.34 shows that the average total cost as a function of the number of nodes in the beam is close to linear as it was for the UNIVAC. Since the charging algorithm for the PRIME is so simple and relates CPU and I/O usage directly to cost, it was not necessary

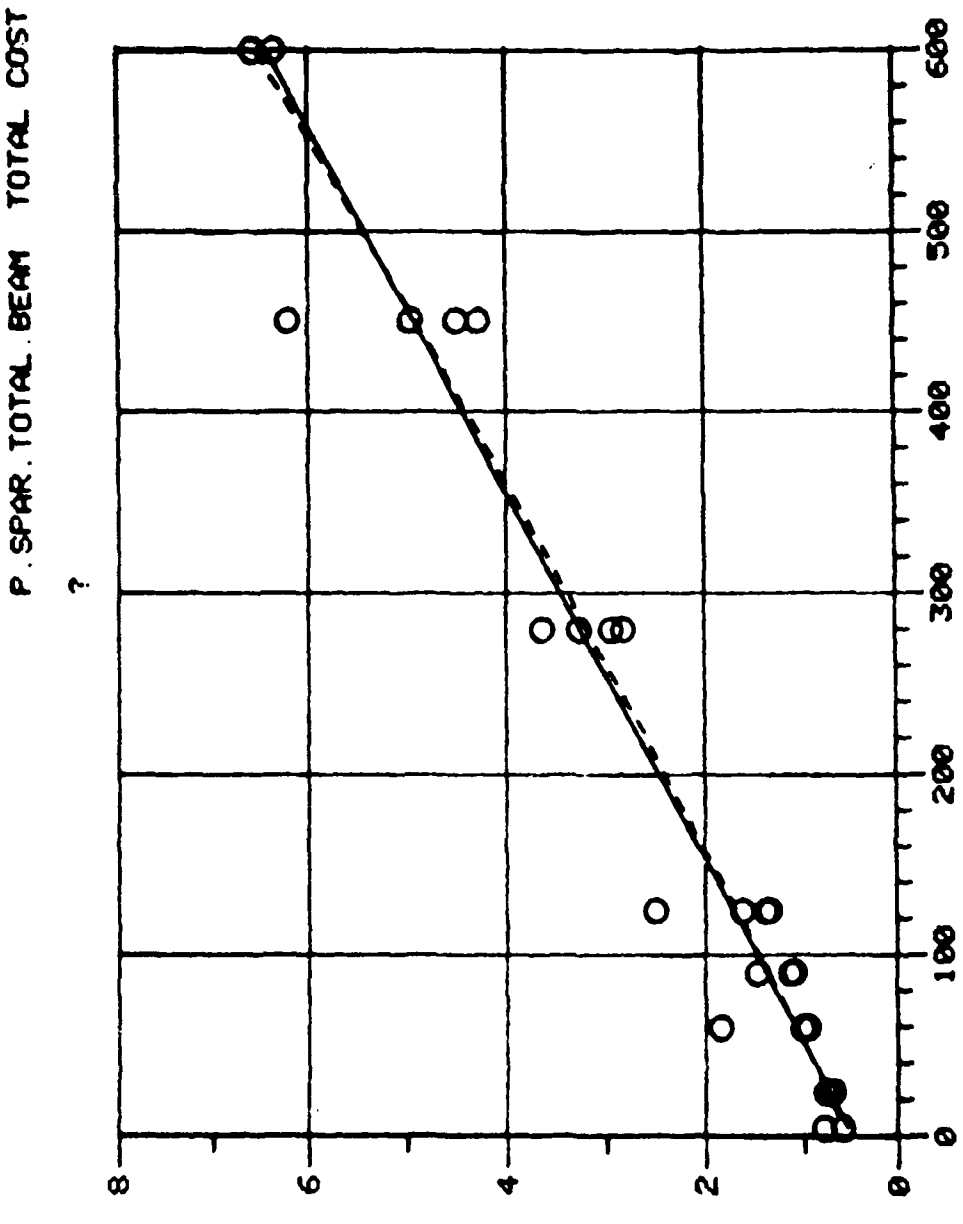


Figure 8.34: Total Cost (dollars) vs. Number of Nodes for Beam Problem on PRIME

to do a separate fitting for CPU and disk time. There are several charge related items:

*executing the first program in a job (in our case the first SPAR processor) incurs the overhead of building segment tables and allocating backing store for memory management.

*writing into an existing file will cause the current user job to be charged for releasing unused space in the file when the file is closed.

Tables 8.5 and 8.6 relate the average PRIME and UNIVAC costs for the beam problem to the different charging rates depending on the time of day when the job is run. These charges are based on CPU and I/O time, not on connect time since we assume that these problems are run as phantoms.

Table 8.5: Cost Variations on PRIME based on Time of Day

time/# of nodes	9 AM-5PM Weekdays (100%)	5PM-10PM Weekdays (75%)	Other (50%)
5	.65	.49	.32
25	.72	.54	.36
60	1.18	.89	.59
90	1.20	.90	.60
125	1.70	1.28	.85
280	3.16	2.37	1.58
450	4.98	3.73	2.49
600	6.45	4.84	3.22

The UNIVAC job costs are consistently much higher, even with "off hour discounts", when charges are compared:

Table 8.6: Cost Variations on UNIVAC Based on Time of Day

# Nodes	Business Hrs. 100%	Weekends/Nights 75%
5	$1.83	$1.37
25	2.07	1.55
60	2.66	2.00
90	3.16	2.37
125	4.07	3.05
280	6.90	5.17
450	9.53	7.15
600	13.80	10.35

They are approximately twice as high as for the PRIME when run in batch mode, up to four times that of the PRIME if they were run interactively (on the UNIVAC there is a 100% surcharge for interactive use)

If it is assumed, however, that a PRIME user does not run the problems as phantoms, but rather is sitting at the terminal, waiting for output, then connect time charges occur in proportion to the response time of the system. In Table 8.7 we added $1.00 per hour connect time for a 300 Bd. connection and $2.00 for a 1200 Bd connection.

Table 8.7: Average PRIME Cost

	Processing Cost	Connect Cost		Total	
		300 Bd	1200Bd	300 Bd	1200 Bd
5	.65	.02	.04	.67	.69
25	.72	.02	.04	.74	.76
60	1.18	.04	.09	1.22	1.27
90	1.20	.03	.07	1.23	1.27
125	1.70	.10	.20	1.80	1.90
280	3.16	.09	.19	3.25	3.35
450	4.98	.18	.37	5.16	5.44
600	6.45	.23	.46	6.68	6.91

Another charging algorithm (NASA, Langley Research Center, PRIME) which does not charge for I/O but only for CPU and connect time bills $15.00/connect hour and $30.00/ CPU hour. The average execution, connect and total charges are given in Table 8.8

Table 8.8: Estimated Cost of Running Beam Problems on NASA Langley Research Center PRIME 400.

Number of Nodes	Processing Cost	Connect* Charges	Total
5	.12	.26	$ .38
25	.16	.26	.42
60	.26	.66	.92
90	.31	.49	.80
125	.40	1.51	1.91
280	.79	1.42	2.21
450	1.18	2.77	1.95
600	1.64	3.44	$ 5.08

*Incurred only if job is run interactively

Compared to the average PRIME cost in the earlier table, it can be seen, that for short connect times the NASA charging algorithm produces much cheaper runs, whereas the large problems incur more equal charges. If the IIT PRIME runs are done during nights or weekends, they may even be considerably cheaper. In general, it can be said that a charging algorithm which mainly charges for predictable resources (as we have seen CPU demands to be) is to be preferred, since the charges also will be much more stable. Both the IIT and the NASA PRIME installation charge for "unstable" resources, i.e.

I/O time and connect time respectively and in both cases this part of the charges may constitute the majority of the expenses.

d) Performance

On the UNIVAC, the turnaround times for the different beam problems range between just below 2 mins. to 6-1/2 mins. for the double precision runs. The system reaction time is between half a minute and two minutes. The exact figures are given in Table 8.9 together with comparisons with Prime response times for the same problems. Not included in these figures are possible delays due to human factors, such as putting a run into a mailbox, etc.

On the PRIME, the average response time (wall times) are very good for the smaller problems (up to 90 nodes)

Table 8.9: Comparison of Response Time for Beam Problem

Turnaround	PRIME Average Response Time (min)*	UNIVAC† Turnaround Time(min)	SYSTEM REACTION TIME††
5	1:03	2:27	0:54
25	1:03	1:53	0:57
60	2:37	5:05	0:59
90	1:57	2:16	0:25
125	6:02	3:25	1:01
280	5:41	6:34	0:43
450	11:04	3:01	0:15
600	13:47	5:44	2:01

*Time from job submission to its completion

†Time between beginning of job execution and job completion

††Time between job submission and beginning of execution

But for the two biggest problems the turnaround time on the UNIVAC becomes about one third that of the average response time on the PRIME. However, the response time for the PRIME problems are subject to a wide variation as shown in Table 8.10. One rather puzzling feature is that the response time can be higher when there are only two users on the system (in our case, the Beam run and a person editing the input file for the next Beam run) than when the system is supporting more than 20 users. An explanation for this behavior are so-called backstop processes which are activated when there is little other work to do.

Table 8.10 shows that the PRIME is much slower than the UNIVAC for a majority of the runs, but it also shows that it worked more efficiently from the user's point of view, when there were between ten and fifteen users active.

Table 8.10: Variations in Response Time with Number of Users on the PRIME for Beam Problem

Number of Nodes	Users	Response Time (Sec)	Average Response Time (Sec)
5	2	85.08	
	13	33.27	62.93
	24	70.44	
25	92	91.54	
	4	59.50	63.15
	13	42.25	
	24	59.31	
60	2	135.33	
	4	90.18	157.38
	13	65.25	
	25	338.78	
90	2	165.46	
	3	108.98	117.35
	13	66.52	
	24	128.45	
125	2	210.05	
	3	146.53	361.62
	13	124.50	
	24	965.42	
280	2	465.88	
	3	336.58	341.25
	15	261.26	
	25	301.28	
450	2	715.41	
	4	629.28	664.09
	12	489.30	
	25	822.37	
600	2	1029.83	
	4	623.33	826.58

IX. CONCLUDING REMARKS

A study on the cost effectiveness of mini-computers vs. main-frames for structural analysis is being conducted at the Illinois Institute of Technology. The study compares the performance of several finite element programs on the IIT PRIME 400 minicomputer and the United Computing Systems UNIVAC 1100/81 main frame. Preliminary results for a planar beam problem with one program were obtained with number of nodes ranging from 5 to 600. These results indicate that based on run cost consideration the PRIME 400 minicomputer is much more attractive than the UNIVAC main frame. Problems of reliability of software and hardware and the quality of the service tend to balance the picture because they are more favorable to the main frame.

APPENDIX I

User Charging Algorithms for The PRIME 400

and The UNIVAC 1100/81.

PRIME 400 and 550 RATES

1. The cost for processing is $0.02 per CPU-second plus $1.00 per connect hour for 300 Bd. connection or $2.00 per connect hour for 1200 Bd. connection.

   The following price differential factor table applies to the above CPU rate.

	Priority*		
	High	Reg.	Low
9 AM - 5 PM, Weekdays	1.5	1.0	.5
5 PM - 10 PM "	1.25	.75	.5
10 PM - 9 PM "	1.0	.5	.25
5 PM - (FRI) - 9 AM (MON)	1.0	.5	.25

   * Users may change their priorty by request at the Academic Computing Center.

2. Permanent file storage presently not billed but will be in the near future.

3. Printing cost $0.015 per page.

4. Magnetic tapes are 5¢ per day rental and 5¢ per day storage.

5. Special services such as keypunching and program development will be billed at negotiated rates.

6. File access and RUN cost limits are inoperative on the PRIMEs.

Univac 1100/80 Service Control Defaults

The Service Controls sections marked with "●" should be filled in by the requester. If the default values indicated below are inappropriate, then please indicate your requirement in the SPECIAL CONTROLS OR COMMENTS SECTION.

25¢	= approx 1 CPU-SEC, 10 pages, no cards
$1	= approx 5 CPU-SEC, 49 pages, 100 cards
$5	= approx 25 CPU-SEC, 99 pages, 500 cards
$100	= approx 10 min, 1000 pages, 5000 cards

UNIVAC 1100/80 RATES AS SET BY UCS

The Cost of computer services provided shall be determined by the following algorithm:

1. UNIVAC 1100/80 PROCESS COST = $.18T_1 + .0011A(T_1 + T_2)$, dollars where: A is the number of 512-word blocks of core being used; T_1 is the time in seconds in active use under run control spent by the central processor processing either the user's code (i.e. a program) or other code directly serving the user; and T_2 is the time in seconds in active use under run control spent by the channel control unit accessing or transferring data at the user's request or on behalf of the operating system.

   Process cost is computed on the task basis; i.e., a run consisting of a compilation, a collection and an execution would consist of three tasks, each task using different A values and T values.

   A surcharge of 25% will be applied to all batch runs which request preemptive scheduling.

   A surcharge of 100% will be applied to all Demand interactive runs initiated.

   A 25% discount will be applied to all Demand and batch computer runs initiated between the hours of 6:00 p.m. through 8:00 a.m. Monday through Friday as well as all operating hours Saturdays and Sundays.

   A 40% discount will be applied to all deferred priority runs.

   Proprietary Application Software is a separately priced service; the cost of which is determined by the supplier of such software.

2. PERMANENT FILE STORAGE COST = 0.0005 x N x T, dollars where: N is the number of tracks used (1 track = 1,792 words); T is the time in hours.

3. CARD READING COST = 0.001 x N, dollars where: N is the number of cards read at the central site.

4. CARD PUNCHING COST = 0.003 x N, dollars where: N is the number of cards punched at the central site.

5. PRINTING COST = 0.0005 x L + 0.010 x P, dollars where: L is the number of lines printed at the cnetral site; P is the number of pages printed at the central site; Special Forms change = $1.00 per occurrence.

6. MAGNETIC TAPE RENTAL AND STORAGE = $1.00 per reel charge for tape inventory set-up, plus 5¢ per day storage charge, plus a 5¢ per day rental charge for UCS-owned tapes.

7. MAGNETIC TAPE = Mounting cost of 50¢ per tape mounted plus $9.80 per hour for each magnetic tape unit utilized.

8. TERMINAL CONNECT TIME COST (synchronous or asynchronous) = $12.00 x T, dollars where:
T is the time in hours the terminal is actually connected to the 1100/80 system through a packet switching network or WATS service. Local access will be at $10.00/hr for 1200 bps and 6.25/hr for 300/110 bps.

9. PRIVATE PORT = Waived for first Prime Port connection and $250.00 per month for additional ports.

10. MINIMUM CHARGES = $.25 minimum run charge for each execution.

11. PLOTTER COST = $20.00 per hour plus supplies used.

12. BLOCK TIME AND OTHER SPECIAL RATES = The cost for the system to be up during non scheduled hours is a minimum of $100.00 an hour with a four hour minimum guarantee. The Block Time Rates are $750.00/hour for the first two hours and $450.00/hour thereafter, with a two hour minimum guarantee. Scheduling for services must be made in advance with the Facility Manager.

13. SPECIAL SERVICES = Other services furnished by UCS, such as consulting, keypunching, file restoration, manuals, postage, shipping charges, computer programs, tapes, binders, etc., will be billed to the consumer at the then current time and materials rate structure.

REFERENCES

1. ANL-AUA Proposal Development Team, 1977. National Resource for Computation in Chemistry: Perspectives on Minicomputers. Mimeographed position paper.

2. Chester, G., Gann, R., Galagher, R. and Grimson, A. 1978. Computer Simulations of the Melting and Freezing of Simple System Using an Array Processor. In Lykos, 1978: 111-124.

3. Conaway, J. H. 1979. The Economics of Structural Analysis on Superminis. Proceeding 7th ASCE Conference on Electronic Computation, 1979:374-385.

4. Faggin, F. 1978. How VLSI Impacts Computer Architecture. IEEE Spectrum, May, 1978:28-31.

5. Freuler, R. J. and Petrie, S.L. 1977. An Effective Mix of Minicomputer Power and Large Scale Computers for Complex Fluid Mechanical Calculations. In Lykos, 1977: 218-234.

6. Inskeep, W. 1978. The Array Processor: A Floating-Point Computer with High Throughout. American Laboratory, September, 1978:41-49.

7. Knottek, N., 1978. Mini and Microcomputer Survey. Datamation, August, 1978:113-130.

8. Kuck, D. J. 1972. Supercomputers for Ordinary Users. Proc. Fall Joint Computer Conference, 1972:213-220.

9. Lykos, P. 1978. Computer Modeling of Matter. American Chemical Society.

10. Lykos, P. 1977. Minicomputers and Large Scale Computations. American Chemical Society.

11. Mai Tan, J. and Kamel H. A. 1980. Performance of Minicomputers in Finite Element Analysis Pre and Post Processing, Paper presented at ASCE Spring Convention, Portland Oregon, April 1980.

12. Mueller, G. E. 1977. Whither the Large Computer? Mini-Micro Systems, January, 1977:67-68.

13. Norbeck, J. M. and Certain, P. R. 1977. Large Scale Computations on a Virtual Memory Minicomputer. In Lykos, 1977:191-199.

14. Nordby, G. L. 1978. Minicomputer Acquisition Policy. A Process and a Conclusion. EDUCOM Bulletin, Summer, 1978:2-8.

15. Pearson, P. K., Lucchese, R. R., Miller, W. H., and Schaefer, H. F. III. 1977. Theoretical Chemistry Via Minicomputer. In Lykos, 1977:171-190.

16. Robinson, A. L. 1979. Array Processors: Maxi Number Crunching for a Mini Price. Science, 203, 12:156-160.

17. Stevenson, D. 1978. Parallel Computers in the 1980's Proc. 17th Annual Technical Symposium, ACM, June, 1978.

18. Storaasli, O. O. 1977. On the Role of Minicomputers in Structural Design. Computers and Structures, 7:117-123.

19. Storaasli, O. O. and Foster, E. P. 1978. Cost-Effective Use of Minicomputers to Solve Structural Problems. 1978:164-169.

20. Swanson, J. A. 1979 Present Trends in Computerized Structural Analysis. Computers and Structures:10, 33-37.

21. Taylor, T. D., and Widhoff, G. F. 1979. A Trend Toward Inexpensive Computer Simulations. Astronautics and Aeronautics, April, 1979: 34-76.

22. Wagner, A. F., Day, P., Van Buskirk, R., and Wahl, A. C. 1977. Computation in Quantum Chemistry on a Multi-Experiment Control and Data - Acquisition on a Sigma 5 Minicomputer. In Lykos, 1977:200-217.

23. Winningstad, C. N. 1978. Scientific Computing on a Budget. Datamation, October, 1978159-173.

24. Wolin, Louis. 1976 Procedure Evaluates Computers for Scientific Applications. Computer Design, November 1976:93-100.

END