REPORT ON C-DOT PARALLEL PROCESSING SYSTEM

Shyam Khobragade

ABSTRACT

This report briefs an overview of CDoT Parallel Processing System (PPS) and software support, installation and parallelization of Software Development Environment (SDE) on CDoT PPS, system performance, its stability, limitations and merits.

---

1. System Overview

Parallel programming is a technique of breaking a large application into smaller ones and running it concurrently on multiple processors. The treatment of *Parallel Processing* in transputer systems is based on the idea of *communicating sequential processes*. In this model, a computing system is a *collection of concurrently active sequential processes* which can only communicate with each other over channels. Parallel programming is model dependent.

The architecture of CDoT PPS is based on so called SAMD model (Single Algorithm Multiple Data). It comes in between the SIMD and MIMD models and is expected to have advantages of both. A single user is allowed to run a single program on different data sets on all processing elements (PEs) at a particular instant of time.

A block diagram of the CDoT PPS is shown in **fig. 1**. The *architecture* consists of a Main Controller, several (upto 192) processing elements, an equal number of memory banks and a switching network interconnecting the two.

The *main controller* (MC) is the over-all master of the system. It assigns the work for the PEs and also takes care of distributing the data among them. It has the full knowledge of the flow of the program. It also controls the Multi dimensional access memory (MDAM). It has a parallel interface to the MDAM. The Main controller besides controlling the various actions in different parts of PPS, is a single processor computer by itself. It is built around Motorola 68030 microprocessor, runs UNIX version 5.3 and supports 1.5 Gbytes disk, 8 Mbytes memory and Ethernet interface. It also supports Mass storage controller (SCSI interface), a floppy drive and a I/O controller that takes care of the communication between the main controller CPU (MC) and the PEs.

The main controller complex also has a memory management unit (MC68051), a floating point unit (MC68882) and a DMA controller that can effect memory to memory data transfer.

The system has a *global memory* that is common to all the processors. This memory is *multi dimensional access memory (MDAM)*. This memory is connected to the processing elements (PEs) through an interconnecting network. The global memory is used only for storage of data. No computation is possible on this memory. The PEs can not directly address this memory. This global memory is organized as the number of banks equal to the number of PEs. Each bank has 256 Kbytes of memory. The MDA memory allows the PEs to access the memory through interconnection network (by software reconfiguration during run-time) as in different modes like either by row, column, diagonal or any other structure that might be required by the application. Each bank of memory has a *communication controller* that is used to transfer the data from the MDAM to the PEs. The interface between the MDAM and MC is parallel. Thus data can be transferred between MC and MDAM by a DMA operation. The

data rate between MC-MDAM is 8 Mbits/sec, and a cycle time is 450 nanoseconds.

The *inter-connection* is based on the principle of Time-Space-Time (TST) switch. This switch can be assumed to be a full fledged cross-bar switch 128 inputs and 128 outputs (in case of 64-PE PPS). The 128 inputs are the 64 PEs and the 64 MDA banks. Thus any PE can be connected to any PE or to any MDA bank. The switch settings are controlled by the main controller by *high-level system library routines* during run-time. The 16-PE machine uses the Time-Space (TS) switches, whereas the 64-PE system uses Time-Space-Time (TST) switches. By the time this report was written, only same-ID PE-MDAM link was supported by library calls. Data transfer rate is supposed to be 8 Mbits/sec.

Each *processing element (PE)* is self contained with its own floating point unit, memory for program and data and a high speed I/O link interface that is connected to the switch. Each PE is controlled by a processor that takes care of the communication over-heads. The PE is relieved of all other functions and is involved in the processing alone. The *PE control processor* (PEC) takes care of all the protocols with both towards the switch and towards the Main controller. Once this I/O is fixed to a particular standard, the PEs can be built out of any chips (like DSP chips or processor FPU combination or CPU with built in FPU).

Each PE uses the transputer as floating point engine which is expected to give computation rate of 2.5 MFLOPS with 64-bit floating point capability. It is provided with 4 Mbytes of dynamic memory and 8 Kbytes of Dual Port Memory.

The *communication controller* based on Motorola 68010 provides the basic control for the FPE in terms of handling the protocol while downloading the code/data from MC and in the inter-PE communication. So the PE hardware mainly has two communication links at 8 Mbits/sec.

As per the design philosophy, all PEs run the same program (algorithm) at a given instant of time. These programs operate on different data sets. After synchronization, the PEs may be loaded with a different program. PEs are controlled by MC by sending commands (messages) on the high speed broadcast link. Synchronization is achieved by means of hardware signals.

2. Software Support

The philosophy of this architecture is to *parallelize the data* and not the *code*.

To exploit the *parallelism* in the architecture, a user requires software support to manipulate the code and data across the subsystems. It could be done by providing extensions to the programming language or by library calls. The later approach has been adopted in CDoT PPS.

Two different types of libraries are supported - the system library and the application library. The system library is again divided into two parts : higher-level system routines (HSR) and lower-level system routines (LSR). An application programmer need not to know about LSR because that is used by the internal system and also in HSR. The system library is entirely implemented in C. Each routine expects its arguments to be passed by reference. If this convention is adhered to, then the MC program can be written in PASCAL or FORTRAN-77 or any other language whose implementation allows linking with external C routines.

The system library itself is designed and implemented using sequential code and in principle the operating system (O.S.) need not support multi-processes. The approach greatly simplifies the design and the implementation of the system library.

In keeping with the hardware architecture of the system, the user splits the program into two levels. The user sees the system as comprising of an MC, 16 or 64 PEs and a global MDA memory. He/she

compiles his/her source program and runs it on MC. This program reads, manipulates, writes data in the global store and schedules *calls* to procedures to be executed in the processing elements (PEs). All these operations are supported by system library routines and are made completely transparent to the user.

To facilitate execution of user programs in above manner, a multi-level software support is provided which can be categoried as follows :

- Support for compiling/debugging source program.
- Support for typical data manipulation and mathematical functions. This includes familiar Matrix operations like transpose, inverse, multiplication, row-col extraction, FFT computations and Maths operations. (This application library part is still under development).
- Support for defining global data items in MDAM and manipulating them.
- Support for using major system resources - the MDAM and PEs and controlling data transfer between them by setting up the switch.
- Support for controlling execution of programs in PEs.

All communications across the system are managed by messages and control signals. This forms the lowest level of data transfer commands and responses.

The software for diagnostics forms that part of system library which is not directly related to execution of programs. This comprises developing software for both on-line and off-line testing. On-line testing will be restricted to detection of faults and isolation of probable causes. Off-line testing will be (i) testing individual cards, or (ii) system-wide tests, testing different paths, memories and PEs.

The O.S. running on MC is UNIX. Some extra features are added to it to suit the PPS architecture. These are :

- Driver for MDAM which provides interface between MDAM and MC.
- Driver for ICN which provides interface between switch and MC.
- An interface to communicate to PEs on the common link.
- An interface to Ethernet - to be used as link with host.

Apart from these, UNIX provides drivers for console, graphics terminal and hard disk.

The O.S. on PEs is of special type and has limited functions. It is named as **PECOS** (PE Controller O. S.). It has to :

- control execution of code.
- interact with MC programs.
- send/receive data on one of its links.

It does not have to deal with devices, processes or perform complex I/O. Hence, it is designed like a monitor performing the above mentioned tasks. it forms the basis of the protocol of data transfer. it interprets MC requests and executes program loaded.

The PE-level program is not allowed to involve any I/O operations. This may be a FORTRAN-77 program or a C program, which is to be compiled using INMOS 3L-Fortran or 3L-C compiler, and then this executable code is again converted to PE-compatible code using a system command

ppcd_conv <filename>

The UNIX running on MC supports Fortran-77 and C compilers.

3. SDE on CDoT PPS

The Software Development Environment (SDE) is a *data reduction package* for Radio astronomy. It was designed by T.J.Cornwell and Pat Moore in 1986-87. System software was written by Moore and the application code, since then, has been written by Cornwell. However, it is a personal package of Cornwell et.al. and is not supported by NRAO.

It gives all the functionality of other Radio Astronomy related data reduction packages (like AIPS). Besides the standard algorithms that are used in Radio Astronomy data reduction, this is the only package which provides algorithm for *3D inversion* and reduction of the visibility data.

From the software design point of view, it is highly *modular*, with the different modules the interfacing is very well defined and easy to implement interface. The dependencies between the various modules are minimum. The system level code is in C, which is FORTRAN callable and is controlled by the application code which is in FORTRAN-77. This gives a high level dynamic data structure to the application programmer while he/she still writes code in FORTRAN-77. The complexity of code at the system level and the C-FORTRAN interface is not visible to the application programmer.

For *parallel processing*, one needs to parallelize only the compute intensive code which is always the *PIXEL* level code.

The design of SDE looks as a three stage software :

- the *task* stage
- the *image* stage
- the *pixel* stage

The **task** stage is basically the *user interface*. This is a rudimentary *parser*. The user is allowed to set the various *user selectable* parameters of the various applications, *save* or *load* these parameters to/from a file on the disk and finally run the application. A mechanism is also provided for getting *help* for the various applications. This stage is the one which builds up the system Database in memory, *interacts* with the user and *sets* the values of the various parameters of the task. The name of this Database is hard coded and is therefore visible to the entire application.

The **image** stage uses these parameters and does *consistency checks*. If everything is fine, it proceeds with the reading of the data files (if any) from the disk.

The **pixel** stage is the stage where the actual algorithm exists. The code at this stage sees only FORTRAN-77 data structures and does not have to worry about the source or the final destination, the format, etc. of the data. It is this stage which is most interesting from the *parallelization* point of view and the entire SDE design becomes so attractive because of this very clear decomposition. Also, at this stage, there is extensive support of various *pixel* operations, and adding more to it is straight forward.

Installation of SDE

It was a micro-depth interactive but nice experience with SDE while *installing* it on Main Controller (MC) of CDoT PPS. CDoT PPS is an UNIX machine. So any software which is portable on UNIX machine can be ported on this machine too. SDE has already been ported on many machines such as SUN, CRAY, MWAY, IBM, etc., so there should not be any problem to port it on PPS MC . But when installation of SDE was started, the basic and main problem was Fortran compiler (Greenhill's). Already SDE is ported on many different machines using different compilers. But with Greenhill's Fortran compiler, very strange errors were observed such as *Internal compiler error* and *Horrible IO error*, etc. which took about a week to sort out the problem. Then the need of modifications of some syntax in original programs was recognized.

Further, it was observed that the *main routines* in SDE software package were not working properly because of some Wrong Code Generation or some different logic of implementation of some of the Character variables. This problem was solved by using one SDE routine *STRLEN* in so many other places in many programs. This took another week. All these modifications are listed separately.

Next problem in SDE installation was that some system calls were not supported by Greenhill's compiler which were used in SDE. For this, new system calls namely : *getenvn, exit, fdaten, idaten* and *etimen* were supported.

So in next weeks time, all these problems were sorted out and SDE was prepared to run.

To test SDE, some sequential programs namely CLEAN, UVMAP, etc. were run successfully.

Parallelization of SDE programs

Then the concept of *parallelization* of SDE programs came up. CLEAN-HOGBOM-2D-REAL (pix2drcl) routine which is computational intensive was chosen for parallelization. This is a part of whole SDE-CLEAN program. Remaining part of SDE-CLEAN contains I/O, data & parameters type-checking, database-creation, history-cards, etc.

A parallel version of 'pix2drcl' routine coded by S.Bhatnagar for CDAC machine, was modified for CDoT PPS. This was then included in whole SDE program. This first version of program was tested on different number of Processing Elements (PEs). This time the speed-up ratio observed for 1-to-16 PEs was just about three times which is not satisfactory.

Next, the program was optimized without changing the basic algorithm of the program. Just overheads were removed. In this case, the speed-up ratio for 1-to-16 PEs was about 7-to-8 times faster which is not that bad. This testing was done on available data - Map size: 128 x 128 and Beam size : 128 x 128.

After a result-oriented discussion with M. Periasamy and Vivek Buzruk in CDoT, it was concluded that there is a need to optimize this program further to reduce the communication part over PEs-MDAM-MC and then MC-PEs paths for a small amount of data transfer in each iteration which slows down the speed-up ratio. Accordingly, the basic program was modified with some change in the algorithm. The concept of local iterations at PE-level was involved in this new version. With this modifications, the speed-up performance was further improved. But the next question was whether this modification in the algorithm be theoretically proved ?

All this CLEAN testing was performed on 128 x 128 Map and Beam sizes each. The program was tested on 16-PE machine for different number of PEs namely 1,4,8,9 and 16. The results are tabulated separately. Results were satisfactory apparently.

After the visit of TIFR delegation (C.R.Subramanya, Vasant Kulkarni and Ramesh Sinha) to CDoT, The size of the Map and Beam was expanded to 512 x 512 and it was decided not to change the basic algorithm at this time and just try with greater size of data.

With this too, the results on both machines (16-PE and 64-PE PPS) are satisfactory to some extent. The performance of both of the machines is given beneath this.

It was analyzed that with same amount of data, if the number of PEs are increased, then the speed-up ratio gets saturated.

The timing analysis is tabulated and speed-up performance for both the CDoT PPS are plotted separately.

The SDE-CLEAN timings on PPS-MC (MC68030), rohini (SUN-SPARC-1), ipcone (SUN-SPARC-IPC) and gmrt (SUN-4) are also compared.

How to enter a new Parameter in SDE Database

(i) Make entry in *.inf* file

(ii) Make entry in *.cur* file

(iii) Make following entries in *SDEMAIN* routine

For example,

(Let us enter the parameter **Nodes** in SDE-CLEAN program)

```
INTEGER NODES

CALL USRGETI ('Nodes', NODES, 1, NDUMMY)
CALL DATPUTI ('Components', 'NODES', NODES, 1)
```

(iv) Then make following entries in *img* level program

```
INTEGER NODES

IF (DATEXIST (STRM2(CLN, 'NODES'))) THEN
  CALL DATGETI (CLN, 'NODES', NODES, 1, NDUMMY)
ELSE
  NODES = 1
  CALL DATPUTI (CLN, 'NODES', NODES, 1)
END IF
```

(v) Finally pass this parameter to *PIXEL* level routine.

4. Performance Analysis

The *performance analysis* quoted here is related to both of the CDoT PPSs (the 16-PE and the 64-PE systems), on the basis of one SDE program (CLEAN-HOGBOM) run on those systems. Analysis comprises different *timing comparisons*, and *speed-up ratios* for *sequential* as well as *parallel* version of the program.

Now there are three kinds of *timings* noted here :

- *timex* (real, User, Sys)
- *pphstm* (elapsed time)
- *sdetime* (User, Sys)

The *unit* for all these parameters is *second.*

The *timex* (which is an *alias* to "/usr/5bin/time") is a UNIX command which returns the elapsed time during the command (real), the time spent in the system (User) and the time spent in the execution of the command (Sys). So the total execution time of the command is the sum of *User* and *Sys* times.

The *pphstm* is higher-level system routine (HSR) which returns the number of calls for which any particular HSR call was invoked. Also it returns the total number of clock-ticks and the total elapsed time

for each of HSR call and the accumulated time too.

The *sdetime* is a total execution time (which is a sum of *User* and *Sys* times). This time is returned by SDE calls.

It can be noted that the *times* returned by all the three types of calls are almost the same.

The *timex* and *pphstm* times for CLEAN-HOGBOM program on both of the CDoT PPSs are tabulated in **Table-1**. **Table-2** contains the listing of *sdetime* for both of the systems for the same program. For apparent analysis, we can look into **Table-3** which contains the speed-up performance of both systems. More clearly, this Table compares the *pphstm* and *sde-parallel-time* in terms of speed-up ratio for both the systems.

Speed-up ratio = T1 / Tnpe

where **T1** is the time when a single processing element (PE) was used and **Tnpe** is the time for N-number of PEs.

Speed-up Performance of the 16-PE PPS

The *parallel* version of SDE-CLEAN-Program was run on the 16-PE PPS using 1, 8 and 16 number of PEs separately. **Table-3** reveals that the speed-up ratio using 8 number of PEs was **6.96** (for *pphstm*) and **5.85** (for *sdetime*). For 16 number of PEs, it was **11.18** (for *pphstm*) and **8.60** (for *sdetime*). There is some difference in speed-up value for *pphstm* and *sdetime*. This is because these two different calls were invoked at two different places in the program.

Speed-up Performance of the 64-PE PPS

The same SDE-CLEAN (parallel version) program was run on this system also. In this case, different number of PEs were used at different times - 1, 4, 16, 32 and 51 PEs. The *speed-up* values are as follows :

Num. of PEs	pphstm-speed-up	sdetime-speed-up
01	1.00	1.00
04	3.20	2.70
16	8.99	7.28
32	11.19	8.56
51	11.69	9.60

Although, the same program was tested on both the systems, there is some difference in the speed-up values. After a lot of confusions, the reason was sorted out by CDoT team. The reason is twofold :

- The 64-PE PPS is still under treatment. In the hardware architecture of both the systems, T800 transputers are used at PE-level which run on 17.5 MHz. In the 16-PE PPS, all the 16 transputers run on 17.5 MHz, but in the 64-PE PPS, CDoT team had used the very first transputer (ID-1) which was running on 20 MHz and all other 63 transputers were running on 17.5 MHz. (Of course, this strange change was done during testing phase. But we came to know this after the results were taken and my time-limit was over and I was supposed to be back. So I couldn't take the fresh results.) Because of this difference in clock- frequencies, for a single PE (ID-1), the execution was faster, but using more number of PEs, the execution was comparatively slower.

Since the speed-up ratio is **1 : N** number of PEs, the overall value was less.

- Secondly, there is some system over-head in case of 64-PE PPS as compared to the 16-PE PPS.

The difference in *pphstm* and *sdetime* speed-up values is again because of the system calls invoked at two different places in the program.

Now from **Table-3, Graph-1 and Graph-2**, it is clear that keeping the same amount of data, if the number of processing elements (PEs) are increased, then the speed-up ratio goes down and tends to saturation.

Table-4 involves *pphstm* and *timex* times for a modified algorithm (parallel version) of SDE-CLEAN. This program involved the concept of *local iterations* which was then kept aside for further analysis. Using this modified algorithm also, we got **11-times** speed-up for 1-to-16 number of PEs.

Next, for comparison of *main processors' speed-ups*, the original SDE-CLEAN program was run on PPS-MC (MC68030), rohini (SUN-SPARC-1), ipcone (SUN-SPARC-IPC) and gmrt (SUN-4).

The *sdetime* (User and Sys) and *timex* (real, user, sys) values are tabulated in **Table-5**.

Now, the PPS-MC to different SUN-Systems speed-up values are as follows:

Sr.No.	System name	sdetime-speed-up
1.	PPS-MC	1.00
2.	rohini	3.98
3.	ipcone	4.96
4.	gmrt	6.95

5. Stability of the 16-PE CDoT PPS

After a cordial relation for about two months with the 16-PE CDoT PPS, it can be said that this is a stable system in its operation. Before coming to this conclusion, it was tested with several applications such as *SDE-CLEAN* (which is an Astronomical Image CLEANing program), *Weather forecasting program* (which is 10000 lines of PE-level code - highly computational), etc. were successfully run hundreds of times, in spite of some known failures which were then sorted out and abolished.

To test the *stability* as well as the *performance* of the system, some programs namely *1-D FFT, 2D-FFT, Integration, etc.* were run by Vasant Kulkarni and the results were satisfactory. Also some *Pulsar related programs* were tested by Yashwant Gupta.

When I started using the 16-PE PPS some two months back, it had some problems in MC-MDAM-PE path and MC-PE broadcast path as well. Out of these, some problems were hardware related and some software related.

At the initial stages, the software problems were with the high-level library calls namely *prdmat, pwrmat, ppdbdt, dispcd, and dispdt*. The diagnosis operation was intensified and the problems related to software as well as hardware were sorted out and abolished.

Now in view of *commercial system* is concerned, the software support is not adequate at this stage. The problems like *manual hardware reset*, only single user can compile the MC-level program at a time, all data transfer is through *one-dimensional* arrays, there is no direct way to see or diagnose the PE-level

program, etc still exist.

No doubt, the CDoT team is in progress to give enough software support to the system, may be in next version.

6. Limitations of CDoT PPS

(i) As per the working principle of the system, only single user can run a program on PPS at a time and other users can do editing or compilation or any other tasks which run only on MC and do not use MC-MDAM-PE or MC-PE paths or stages.

(ii) Since the PPS architecture is based on *Single Algorithm Multiple Data (SAMD)*, only one PE-program is allowed to run on all used PEs on different data sets.

(iii) No I/O operations are allowed in PE-program.

(iv) There is no run-time debugging facility provided for PE-level program.

(v) The PE-level program memory is limited to 4 Mbytes, out of which about 2.5 Mbytes of memory is available for data.

(vi) The MDA memory per bank is limited to 256 Kbytes.

(vii) The Main Controller (MC) has got memory of size 8 Mbytes.

So the application is supposed to fit within these limits.

(viii) Data types supported for data transfer between MC to PEs and vice versa through direct broadcast mode or through MDAM path are *real* and *integer*. So the user is supposed to use these two data types only.

(ix) Among all, the main limitation or disadvantage is that the user is supposed to know the basic architecture and the data flow through the system.

7. Merits of CDoT PPS

(i) There is no need to develop a parallelizing compiler.

(ii) User continues to use sequential programming languages like FORTRAN-77, C, etc.

(iii) Ease of programming.
Only few higher-level system routines (HSR) need to support parallelism.

(iv) Simple synchronization at the end of the PE-program execution.

8. Conclusions

It is clear that the 16-PE CDoT PPS is now stable. In spite of some limitations, the *parallel programming* is very easy on this system. This system has got *merits* too, which are very special. Now, the next time, the SDE package is easily portable on this machine. From the results, it is transparent that the PPS can give better performance. Further, there is a need to *parallelize* the PIXEL-level SDE-routines and *optimize* them for better speed-up. For *optimization*, a programmer should see that the *compute to communication ratio* is very high. That means the communication part must be reduced while re-writting the parallel version of the tasks. One *limitation* of the SDE design is that all the information is held in the memory through out the lifetime of the task.

There is a need to write a general-purpose and optimized routine for *data splitting*. Also a *run-time debugging tool* should be written for PE-level program.

As per the SDE design, a beginner programmer can directly start *parallelizing* the PIXEL level code without knowing much about the rest of the SDE. An application programmer is expected to know about the basic architecture, data flow through the system and the higher-level system routines (HSR) to get the *parallelism* on CDoT PPS.

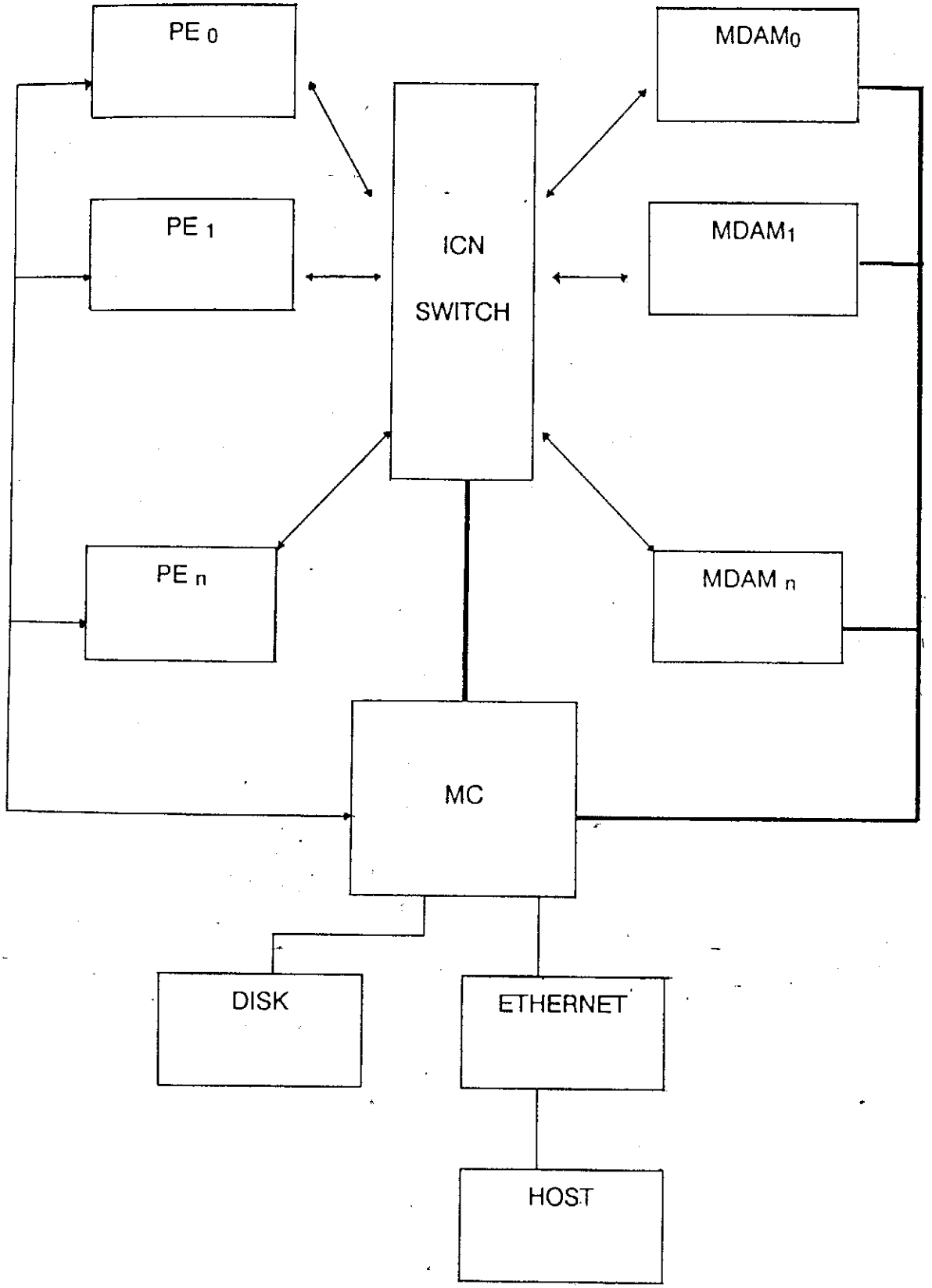

PE 0
MDAM0
PE 1
ICN
SWITCH
MDAM1
PE n
MDAM n
MC
DISK
ETHERNET
HOST

Fig- 1

CLEAN-HOGBOM-TIMINGS (pphstm-timex-times)

PPHSTM-TIMEX-Times for CLEAN-HOGBOM-Program					
Sr.No.	MC-or-NPE	PPHSTM Time (Seconds)		Timex (Real, User, Sys)	
		16-PE PPS	64-PE PPS	16-PE PPS	64-PE PPS
1.	MC	...	...	46:06.10 32:59.53 14.23	...
2.	1 PE	3721.5667	3197.5667	1:06:41.48 4:30.45 59:41.18	57:54.65 4:31.95 52:29.83
3.	4 PEs	...	998.000	...	21:24.611 4:33.85 16:14.21
4.	8 PEs	534.0666	...	13:29.55 4:34.41 8:45.30	...
5.	16 PEs	332.5833	355.6833	10:13.65 4:39.28 5:20.80	10:43.85 4:39.63 5:38.78
6.	32 PEs	...	285.8167	...	10:13.30 4:48.10 4:23.85
7.	51 PEs	...	274.050	...	9:41.85 4:59.63 4:23.03

Table-1 : *PPHSTM-TIMEX-timings for CLEAN-HOGBOM-Program.*

Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000

Note : *All results match Residual and Flux values.*

CLEAN-HOGBOM-TIMINGS (SDE-times)

(sdetime = Finished CLEAN time - Got images time)

{ Total time = User time + Sys time }

SDE-Times for CLEAN-HOGBOM-Program			
Sr.No.	MC-or-NPE	SDE-Time (secs)	
		The 16-PE PPS	The 64-PE PPS
1.	MC	1822.0633	...
2.	1 PE	3680.1234	3249.7833
3.	4 PEs	...	1207.1666
4.	8 PEs	628.55	...
5.	16 PEs	428.3669	446.3433
6.	32 PEs	...	379.4333
7.	51 PEs	...	338.6669

Table-2 : *SDE-timings for CLEAN-HOGBOM-Program.*

Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000

Note : *All results match Residual and Flux values.*

SPEED-UP PERFORMANCE OF C-DOT PPS

{ SDE-CLEAN-HOGBOM-Program }

(Speed-up ratio = T1 / Tnpe)

SPEED-UP PERFORMANCE OF C-DOT PPS					
Sr.No.	No.of PEs	PPHSTM Time		SDE-Parallel-Times	
		16-PE PPS	64-PE PPS	16-PE PPS	64-PE PPS
1.	1 PE	1.00	1.00	1.001	1.00
2.	4 PEs	...	3.20	...	2.70
3.	8 PEs	6.96	...	5.85	...
4.	16 PEs	11.18	8.99	8.60	7.28
5.	32 PEs	...	11.19	...	8.56
6.	51 PEs	...	11.69	...	9.60

Table-3 : *Speed-up performance of CDoT PPS*

T1 --> *Time for 1-PE*
Tnpe --> *Time for N-PEs*

Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000

Note : *All results match Residual and Flux values.*

CLEAN-HOGBOM-TIMINGS (pphstm-timex-times)

{ *Modified Parallel CLEAN Program involving Local Iterations* }

[**Results on 16-PE CDoT PPS**]

PPHSTM-TIMEX-Times for Modified CLEAN-HOGBOM-Program				
Sr.No.	MC-or-NPE	PPHSTM Time (Seconds)	Timex (Real, User, Sys)	Residual Value (Itr=141;Loc.Itr=50)
1.	MC	...	12:18.86 7:16.76 4.15	0.004
2.	1 PE	1045.32	17:53.10 27.50 16:53.30	0.004
3.	4 PEs	338.83	6:10.45 27.88 5:37.20	-0.0117
4.	8 PEs	174.92	3:23.23 28.31 2:53.88	0.0112
5.	9 PEs	156.67	3:05.13 28.03 2:34.23	-0.0122
6.	16 PEs	94.75	2:02.88 28.68 1:33.75	0.0093

Table-4 : *Results on 16-PE PPS for Modified CLEAN Program.*

Map size : 128 x 128
Beam size : 128 x 128
Bpatch : 64
BLC : 1, 1
TRC : 512, 512
ITER : 10000 (Total = Niter x Liter)

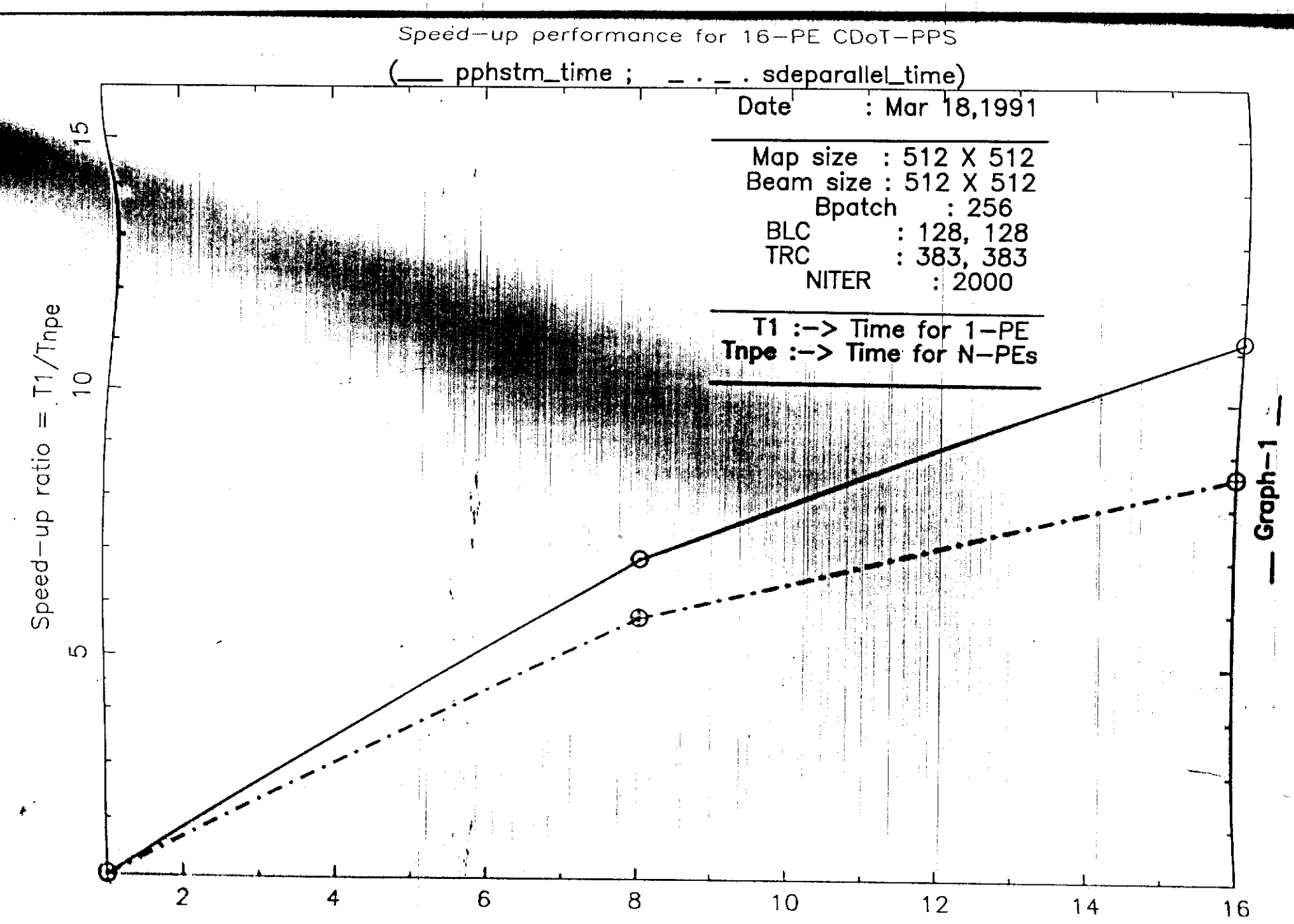
Note : *Residual values are also given in the Table.*

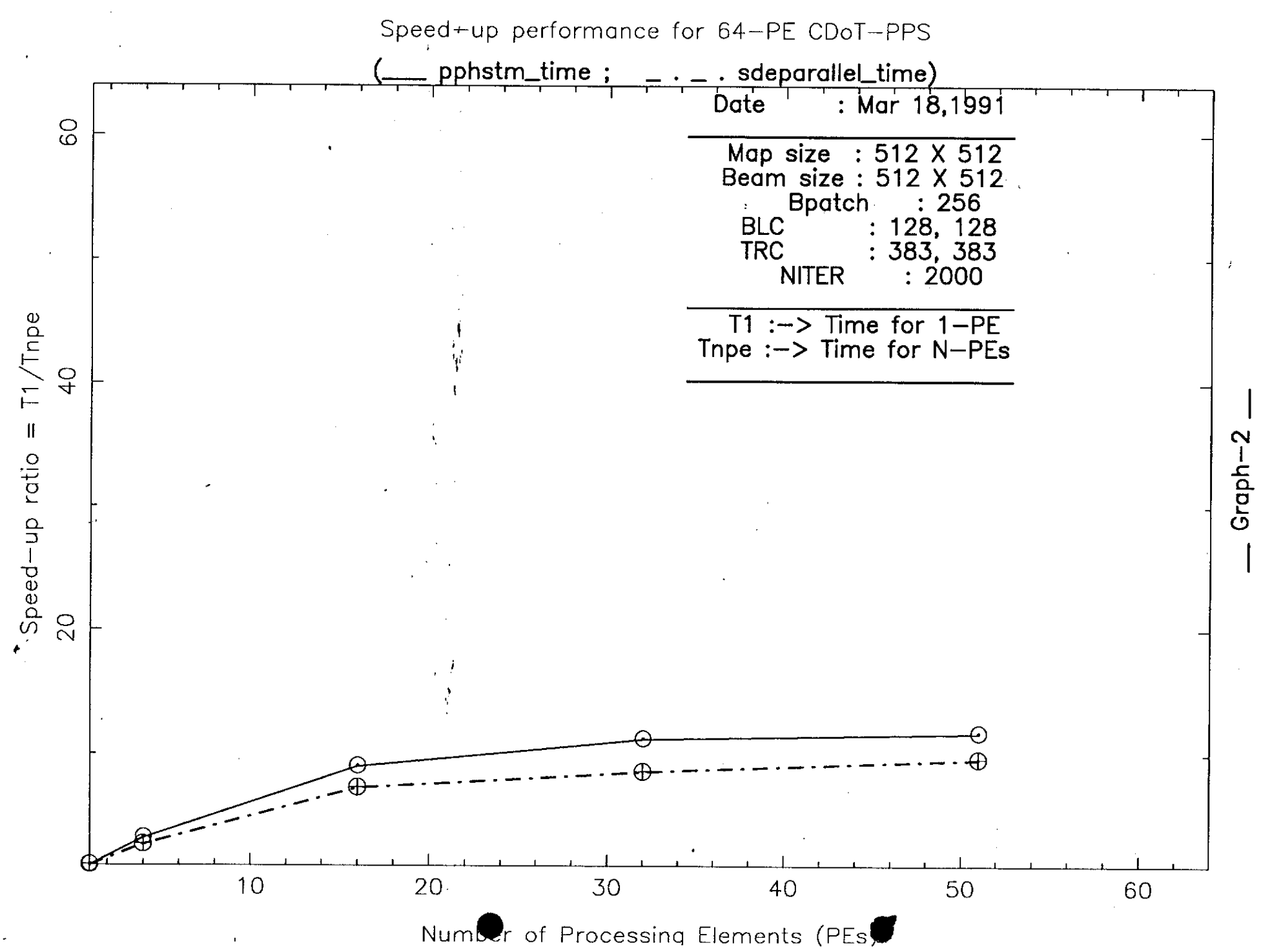
TIMINGS ON MAIN PROCESSORS			
Sr.No.	System Name : Processor Type	sde-times (user, sys)	Timex (real, user, sys)
1.	CDoT PPS : MC68030	1979.50, 14.18	46:06.10, 32:59.53, 14.23
2.	rohini : SUN-SPARC-1	420.20, 80.27	8:25.9, 7:00.4, 1:20.3
3.	ipcone : SUN-SPARC-IPC	334.58, 67.04	7:04.2, 5:34.5, 1:07.1
4.	gmrt : SUN-4/[illegible]	254.36, 32.47	5:08.8, 4:14.3, 32.4

Table-5 : *SDE-CLEAN Timings on different systems*

Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000

Note : *All results match Residual and Flux values.*


Speed-up performance for 16-PE CDoT-PPS
(___ pphstm_time ; _ . _ . sdeparallel_time)
Date : Mar 18,1991
Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000
T1 :-> Time for 1-PE
Tnpe :-> Time for N-PEs
Speed-up ratio = T1/Tnpe
5
10
15
2
4
6
8
10
12
14
16
— Graph-1 —


Speed+up performance for 64-PE CDoT-PPS
(___ pphstm_time ; _ . _ . sdeparallel_time)
Date : Mar 18,1991
Map size : 512 X 512
Beam size : 512 X 512
Bpatch : 256
BLC : 128, 128
TRC : 383, 383
NITER : 2000
T1 :-> Time for 1-PE
Tnpe :-> Time for N-PEs
Speed-up ratio = T1/Tnpe
Number of Processing Elements (PEs)
10
20
30
40
50
60
— Graph-2 —