

Space-Charge-Dominated Beam Dynamics Simulations Using the Massively Parallel Processors (MPPs) of the Cray T3D

Hongxiu Liu

Thomas Jefferson National Accelerator Facility
12000 Jefferson Avenue, Newport News, VA 23606, USA

Abstract. Computer simulations using the multi-particle code PARMELA with a three-dimensional point-by-point space charge algorithm have turned out to be very helpful in supporting injector commissioning and operations at Thomas Jefferson National Accelerator Facility (Jefferson Lab, formerly called CEBAF). However, this algorithm, which defines a typical N^2 problem in CPU time scaling, is very time-consuming when N , the number of macro-particles, is large. Therefore, it is attractive to use massively parallel processors (MPPs) to speed up the simulations. Motivated by this, we modified the space charge subroutine for using the MPPs of the Cray T3D. The techniques used to parallelize and optimize the code on the T3D are discussed in this paper. The performance of the code on the T3D is examined in comparison with a Parallel Vector Processing supercomputer of the Cray C90 and an HP 735/125 high-end workstation.

I. INTRODUCTION

Massively Parallel Processing (MPP) is of common interest for numerically intensive industrial and scientific calculations (1-3). It may provide a new approach to simulating three-dimensional space-charge-dominated beam dynamics with its larger amounts of CPU time and memory. Space charge simulation has turned out to be very helpful in supporting injector commissioning and operations at Thomas Jefferson National Accelerator Facility. It also is important for our future injector design and development.

Jefferson Lab's 4 GeV superconducting electron beam accelerator is providing new opportunities for pursuing new knowledge of internal structures of matter. It consists of a 45 MeV electron injector, two sections of north and south 400 MeV linear accelerators, and five recirculation passes. The injector design is unique in that it can deliver three CW high-intensity electron beams simultaneously to the main accelerator, which accelerates and finally separates the three beams for injection into three nuclear physics experiment halls. The quality of the beams is determined largely by the front end of the machine, i.e., the injector, where space charge is a dominating factor affecting beam generation, transport and bunching.

Space charge refers to the repulsive Coulomb interacting forces among the

charged particles that tend to blow up the beams. Its effects on the dynamics of a beam are so complicated, both in theoretical treatment and in experimental practice, that computer simulation often is the most effective shortcut in delivering the first and most accurate answers to the problems of concern. For example, the space charge effects on bunching of electrons in the Jefferson Lab's injector were clarified and corrected for a bunch length setting in use through computer simulation (4); beam transmission through this injector has been improved by a factor of two with the help of computer simulations.

The code we have been using for space-charge-dominated beam dynamics simulations is PARMELA (5) that has been maintained and modified by the author to meet various special needs for simulating electron injectors. Its space charge subroutine is based on a point-by-point method (6, 7), and has been used extensively in two contexts: designing a photoemission gun injector test stand (8-11) for high-power industrial free electron laser applications, and supporting the Jefferson Lab main accelerator's injector commissioning and operations (4). The algorithm was benchmarked (12) with the simulation results from the PIC code ISIS (13).

A longest space charge run we once had consumed a CPU time of 8 days on an HP 735/125 workstation with 8000 macro-particles that simulated beam transport and bunching through the entire FEL injector (11). Apparently, it is necessary to seek for a faster approach to running space charge jobs. Motivated by this, we have parallelized the code for using the MPPs of the Cray T3D. The techniques used to parallelize the code on the T3D are discussed in this paper. The performance of the code on the T3D is analyzed in detail.

II. PROCESSES IN A COMPLETE PARMELA RUN

A PARMELA input deck is composed of various elements as listed in Table 1. The elements chopper (#6), backb (#34), alpham (#36), poisson (#37), bfield (#38), ! (#39), nbend (#40), kicker (#41) and b_earth (#42) were added by the author for modeling chopper systems (14), backbombardment in microwave guns (15), α -magnets (16-18), finite-length solenoids, indexed-field bends, beam orbital corrections and the earth fields. The "!" element is used for inserting comments.

TABLE 1. Elements constituting a PARMELA input deck

(drift,1)	(solenoid, 2)	(quad, 3)	(bend, 4)	(buncher, 5)	(chopper, 6)
(cell, 7)	(tank, 8)	(trwave, 9)	(coil,10)	(run,11)	(input,12)
(output,13)	(title,14)	(scheff,15)	(zout,16)	(adjust,17)	(start,18)
(restart,19)	(continue,20)	(save, 21)	(end, 22)	(limit, 23)	(errors, 24)
(change, 25)	(rotate, 26)	(sbload, 27)	(cfield, 28)	(dpout, 29)	(cathode, 30)
(design, 31)	(pipe, 32)	(foclal, 33)	(backb, 34)	(wiggler, 35)	(alpham, 36)
(poisson,37)	(bfield, 38)	(!, 39)	(nbend, 40)	(kicker, 41)	(b_earth, 42)

The processes in a complete run are threaded in Fig. 1. After initialization, an electron beam is generated with N macro-particles. The six coordinates of the particles are stored in a two-dimensional array named `cord`. Then, the subroutine `pardyn` is called to initiate the processes of solving particle dynamics equations. Each process contains two major loops, one on time steps, and the other on particles. The space charge subroutine `scheff` is called once on all the particles, with the resultant momentum increments superimposed to those from external elements to advance each particle during the present time step. When all the processes come to the end of a beamline successfully, the program exits the subroutine `pardyn` and saves the coordinates of the particles before ending execution.

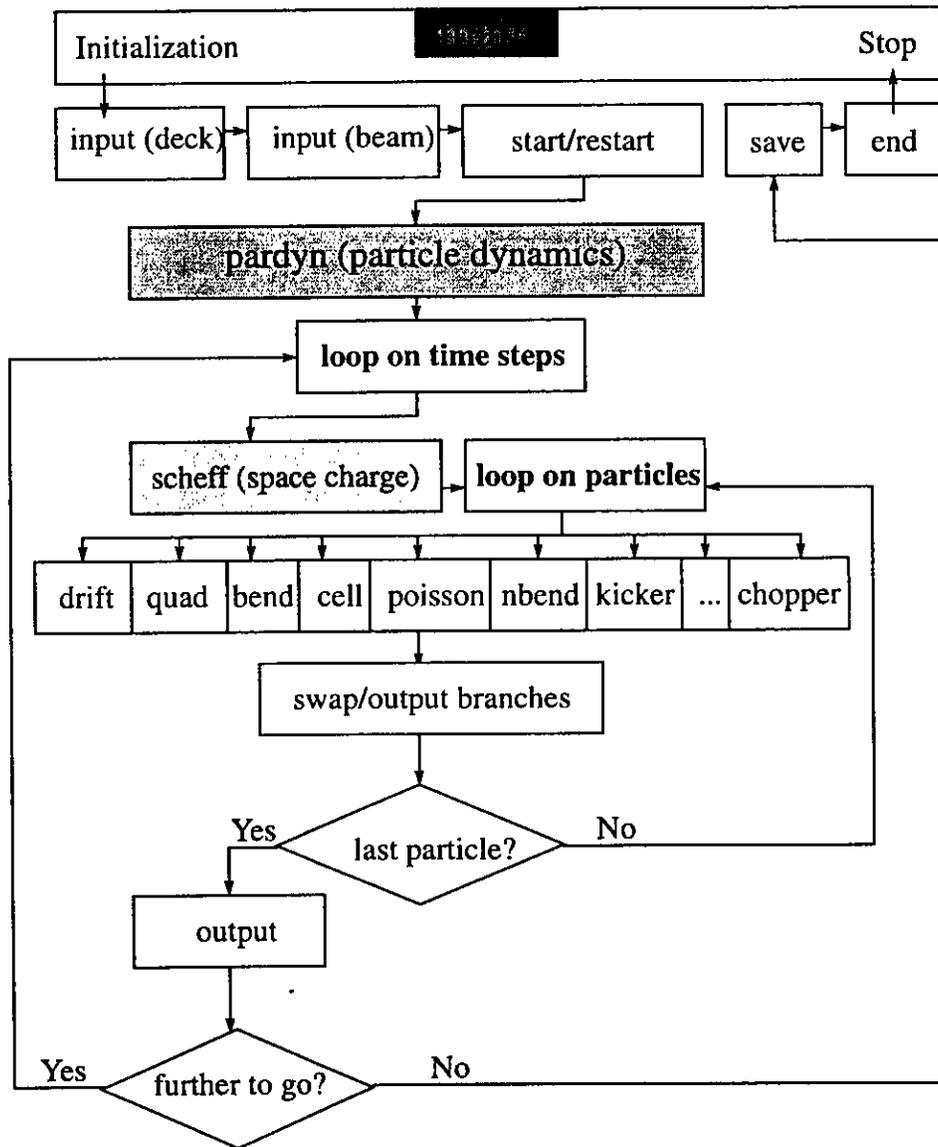


Figure 1. Processes in a complete PARMELA run.

III. POINT-BY-POINT SPACE CHARGE ALGORITHM

In the point-by-point space charge algorithm that we have been using (6, 7), there are two major loops, the i -loop and the j -loop, to calculate the electric (\vec{E}) and magnetic (\vec{B}) fields produced by a moving charge Q

$$\vec{E}_{ij} = Q\gamma_j \vec{r}_{ij} / [r_{ij} \cdot \vec{r}_{ij} + (r_{ij} \cdot (\gamma\beta)_j)^2]^{3/2}, \quad \vec{B}_{ij} = \vec{\beta}_j \times \vec{E}_{ij}, \quad (1)$$

where j denotes the particle applying electromagnetic forces to the i th particle. The i -loop specifies the source particles, and the j -loop calculates and sums up the E and B fields from all other particles for the i th particle. A final loop applies the impulses obtained from the previous two loops to each particle. As is seen, the CPU time spent executing this algorithm is proportional to N^2 , where N is the number of macro-particles, which defines a typical N^2 problem in CPU time scaling. It becomes more and more time-consuming as N goes up rapidly.

IV. PARALLELIZATION ON THE T3D

The Cray T3D is an MPP supercomputer with 256 processing elements (PEs) (19–21). On the T3D, four MPP programming methods (22) are available: data sharing, work sharing, message passing, and explicit shared memory. In order to minimize the frequency of data exchange among the PEs, the arrays that contain the net momentum increments for each particle, `dbgx`, `dbgy` and `dbgz`, instead of `cord`, are chosen as the shared ones. The cyclic data distribution mechanism is used as follows

```
CDIR$ GEOMETRY G(:BLOCK(1))
      real dbgx(imaa), dbgy(imaa), dbgz(imaa)
CDIR$ SHARED (G) :: dbgx, dbgy, dbgz
```

where `imaa = 16384` specifies the maximum number of particles that can be loaded in a run, which must be a power of 2 on the T3D.

Work sharing is achieved by adding the following `DOSHARED` directive

```
CDIR$ DOSHARED (i) ON dbgx(i)
```

prior to the i -loop. This way, the j -loop is not parallelized since it is a nested loop inside the i -loop. The last loop applying the impulses to all the particles is split into two, and a `SHMEM` subroutine `shmem_put` (23) is used for the Master PE to collect all other particles from all other PEs. Finally, the master task copies the arrays `cord` and `gam` across to the other tasks or PEs.

The loop on particles in `pardyn` (see Fig. 1) has been parallelized as well. The major problem encountered was to deal with output and particle loss/swap. This

requires extra caution for correct communication among all the nodes; otherwise, particles would walk into wrong PEs, leading to wrong results. Output is controlled using the MPP intrinsic function MY_PEO which designates a master PE region. When a call to output is made, the Master PE collects all other particles it does not have in its own memory from all other PEs, and then conducts all statistic computations. LOCKs are used to count the particles to determine whether all the alive particles have passed the last exit and the program execution should stop.

V. PERFORMANCE

In this section, we examine the code performance on the T3D in comparison with an HP 735/125 workstation and the Cray C90. The major hardware/software configurations of each computing vehicle are listed in Table 2.

TABLE 2. Comparison of configurations on different machines

Configuration	HP 735/125	Cray C90	Cray T3D
CPU Type	PA-RISC 7150	Custom-made	DEC EV4 a (21064)
Number of CPU(s)	1	16	256
Precision for real (bits)	32	64	64
Memory	64 MB	268 Mw	256 x 8 Mw
Clock Period (MHz)	125	240	150
MFLOPS/CPU	201 (SPECfp92)	960 (peak)	150 (peak)
FP Computation	IEEE + sqrt	-	IEEE
Primary Cache	256KBI+KBD off-chip	Fast memory and vector registers with no cache (25)	8 KB on-chip
Secondary Cache	none		none

We choose an input, shown in Fig. 2, that does a real job of simulating beam quality degradation due to space charge effects. This input is set up to reflect all the aspects that parallelization must deal with. The emittance values of 7.380, 30.081, 47.710, and 62.840 in the x -direction are used to check that the same results are obtained at the same exits despite of any changes and/or modifications to the code.

```

TITLE - mpp test (cap96_1.in)-
RUN /IRUN=1 /IP=1 /FREQ=1497. MHZ /Z0=-3.5 CM /W0= 0.511 MEV /LTYPE=1
OUTPUT 6
DRIFT /L=2.1 /APER=2.54 /IOUT=1
DRIFT /L=6.7 /APER=2.54 /IOUT=1
DRIFT /L=6.6 /APER=2.54 /IOUT=1
DRIFT /L=8.9 /APER=2.54 /IOUT=1
ZOUT
INPUT 16 /NP=999 10.0 0.0 0 10.0 0.0 0 31.8 0.0 0 0 0 0 0 1.0
SCHEFF /BEAMI=-1025.E8 /RMESH=2.5 /ZMESH=8.0 /NR=10 /NZ=300 0. 0 1.5 0 /POINT=3. /SS=1.0 0
START /WTO=0. /DWT=7.2 /NSTEPS=999999 /NSC=1 /NOUT=40
END

```

Figure 2. An input deck with 10^3 particles for testing the performance of the code.

The quantitative approach to measuring the performance of a computer differs from workstation to supercomputer. Workstations are benchmarked using "SPECfp" with which the MFLOPS number is a geometric mean from a number of application runs, while supercomputers are rated using "peak performance," which is sort of "the speed of light" that the manufacturers guarantee that nobody can exceed. In this paper, we simply use the following formula to calculate the MOPS contained in the input shown in Fig. 2

$$MOPS = 70 \times M \times N^2, \quad (2)$$

where 70 is the total number of operations per step per particle contained in the inner loop, M is the total time steps executed from start to finish for the job, and N is the number of macro-particles. With the input shown in Fig. 2, $M = 90$, $N = 1000$, so it needs 6300 MOPS total. This number will be used as the same measure in the following context to calculate MFLOPS numbers for all the platforms.

The baseline performance of the code on a single node is shown in Table 3. For each platform, different compile flags have been tested to make sure that the FORTRAN compiling optimizer built with each computer has been used to the largest possible extent. It is seen that an optimization functionality may make a huge difference in speeding up the CPU time that this input may take. On the C90, the inner loop is vectorized and the outer loop is autotasked (parallelized), making it ideal for running this algorithm. The wall-clock time for this input on the C90 is about 7 seconds only with a usage of ~ 50% of 16 CPUs.

TABLE 3. Baseline performance on a single node

	$(\Delta t)_{cpu}$ (s)	MFLOPS (% of peak)	Compile Flags
H P	412	15 (7.6 of SPECfp)	f77 -R8 +e +E1
	134	47 (23 of SPECfp)	f77 -V -R8 +OP -WP, -o=4 +e + E1
C 9 0	52	121 (12)	cf77 -Zp -Wd"-du -l tmp.l" -c
	61	103 (10)	cf77 -Wd"-du -l tmp.l" -c
	148	43 (4.3)	cf77 -O vector0 -c
	561	11 (1.1)	cf77 -O scalar0 -O vector0 -c
T 3 D	417	15 (10)	cf77 -Wl"-lfi" -c
	389	16 (11)	cf77 -Wf"-o unroll" -Wl"-lfi" -c
	388	16 (11)	cf77 -Wf"-o aggress" -Wl"-lfi" -c
	392	16 (11)	cf77 -Wf"-o unroll -o aggress" -Wl"-lfi" -c
	390	16 (11)	cf77 -Wf"-o unroll -o noieeeedivide" -Wl"-lfi" -c
	389	16 (11)	cf77 -Wf"-o unroll " -Wl"-lfi -D rdahead=on" -c

The difference in CPU time between default optimization and maximized optimization is 1.2 on the C90, 1.1 on the T3D, and 3.1 on the HP. Aligning cache boundaries was tried on the T3D, but seemed not very helpful, possibly because the number of variables involved in a single loop is much larger than the cache size, and cached data cannot be reused effectively. The single node performance is 12% of peak on the C90, and 11% of peak on the T3D. Using the single node CPU time spent on each platform, we see that a single C90 CPU is 2.6 times faster than the HP, the HP is 2.9 times faster than a single T3D CPU, and a single C90 CPU is 7.5 times faster than a single T3D CPU, which is close to the peak performance ratio (6.7) of the C90 to the T3D.

The optimized performance, shown in Table 4, refers to the performance when the code is manually optimized to take full advantage of the architectural features of each machine. Based on the observations that pipelining and cache reuse are effective only when as few branches as possible are contained in a loop (2), the code has been rewritten on the C90 with the original two loops split into three parts: one is for calculating shielding factors for each particle in advance, one is to calculate interacting forces without image charge, and the third one includes image charge calculations. Due to limitations on the single node memory size, the shielding factors are not pre-calculated on the HP and T3D. After the algorithm has been recoded, we found that the code is 1.6 times faster than it was on the HP, 3 times faster on the C90, but only 1.2 times faster on the T3D.

TABLE 4. Optimized performance on a single node

	$(\Delta t)_{\text{cpu}}$ (s)	MFLOPS (% of peak)	Compile Flags
H P	345	18 (9 of SPECfp)	f77 -R8 +e +E1
	85	74 (37 of SPECfp)	f77 -V -R8 +OP -WP, -o=4 +e + E1
C 9 0	22	286 (29)	cf77 -Zp -Wd"-du -l tmp.l" -c
	23	274 (27)	cf77 -Wd"-du -l tmp.l" -c
	164	38 (3.8)	cf77 -O vector0 -c
	589	11 (1.1)	cf77 -O scalar0 -O vector0 -c
T 3 D	319	20 (13)	cf77 -Wf"-lfi" -c
	318	20 (13)	cf77 -Wf"-o unroll" -Wf"-lfi" -c
	342	18 (12)	cf77 -Wf"-o aggress" -Wf"-lfi" -c
	326	19 (13)	cf77 -Wf"-o unroll -o aggress" -Wf"-lfi" -c
	321	20 (13)	cf77 -Wf"-o unroll -o noieeeedivide" -Wf"-lfi" -c
	318	20 (13)	cf77 -Wf"-o unroll " -Wf"-lfi -D rdahead=on" -c

It is noticed that functional calculations of \sqrt{x} are significantly slower on the T3D than on the C90 and on the HP. On the T3D/T3E the standard "libm" "sqrt0" is

implemented in Alpha assembly language, and the routine has about two dozen floating point adds/multiplies; the latest `sqrt()` on the T3D takes about 129 clock periods, and on the T3E about 67 clock periods (24). The C90 vector `sqrt` function requires 21 floating point operations, 8 multiplies, 10 adds, and 3 reciprocal approximations plus some overhead to set up the vectors; for vectorized code with arrays on the order of 1 million elements, `sqrt` achieves around 700 MFLOPS, roughly 1 `sqrt` result per each 7.2 clock ticks (25). It is noticed that the DEC's Alpha architecture CPUs have no `sqrt` FP computation, unlike MIPS, PA-RISC, PowerPC and SPARC (26).

Table 5 shows the partition of CPU time among different parts of the program executed with one PE and 32 PEs respectively, based on apprentice analysis on the T3D (27). The measured elapsed time was 748 seconds with 1 PE and 35.5 seconds with 32 PEs in contrast with 318 seconds with 1 PE and 20 seconds with 32 PEs when compiled with no apprentice. Therefore, the program was slowed down by a factor of 1.78 with 1 PE, and a factor of 2.35 with 32 PEs by apprentice. The MFLOPS numbers are 8.7 for program and 14.7 for `scheff2` with 1 PE, and 137 for program and 384 for `scheff2` with 32 PEs, according to apprentice. However, these MFLOPS numbers from apprentice are less accurate, since `sqrt` has not been instrumented in counting MOPS, but its time is counted in calculating MFLOPS.

TABLE 5. Partition of CPU time among different parts of the program on the T3D

Parts	Δt in seconds	
	NPES = 1	NPES = 32
program/parmela/pardyn	368/0.04/3.21	755/26.9/19.5
scheff2/_sqrt/drift	218/145/0.668	268/145/0.677
shmem_broadcast/_barrier	0.008/0.003	294/79.7
shmem_put/set_lock/clear_lock	0.200/0/0006/0.0006	1.14/1.82/0.06

It is seen from Table 5 that `sqrt` is responsible for 39% of the CPU time in the case of one CPU run. According to Amdahl's law, the program is slowed down by a factor 1.5 with the assumption that $24/70 = 34\%$ of MOPS attributable to `sqrt` and that it is $(24 \times 129 \times 240) / (21 \times 7.2 \times 150) = 33$ times slower than on the C90. In addition, overhead resulting from communications among multiple PEs has increased significantly from the case of 1 PE to the case of 32 PEs, as indicated by large amounts of CPU time with 32 PEs from `shmem_broadcast` and `_barrier` shown in the table.

The performance of the code vs. the number of nodes on the T3D is shown in Table 6, with space charge on and off. It is seen that the performance is linearly scalable for time-consuming space charge runs. The T3D starts to outperform the C90 at 32 processors. In the near future, optimization and use of the code for our space-charge-dominated beam dynamics simulations will be switched to the NERSC's Cray T3E which is about six times faster than the T3D.

TABLE 6. Performance vs. number of nodes on the T3D

Number of nodes	Space charge on		Space charge off	
	$(\Delta t)_{\text{cpu}}$ (s)	MFLOPS	$(\Delta t)_{\text{cpu}}$ (s) ($N_P = 1000$)	$(\Delta t)_{\text{cpu}}$ (s) ($N_P = 16384$)
1	318	20	3.95	63.3
2	162	39	3.36	38.0
4	84	75	3.30	21.2
8	46	137	3.78	13.2
16	28	225	4.56	9.85
32	20	315	5.50	9.07

ACKNOWLEDGMENTS

I thank C. Sinclair and J. Bisognano for their support and S. Corneliussen for editing the manuscript. The access to the T3D and C90 was provided by NERSC/LLNL. This work was supported by DOE contract no. DE-AC05-84ER40150.

REFERENCES

1. A. Koniges and K. Lind, *Comput. Phys.*, **9**, 399 (1995).
2. V. Decyk, S. Karmesin, A. Boer and P. Liewer, *Comput. Phys.*, **10**, 290 (1995).
3. R. Ryne and S. Habib, *LHC95*, 15–21 October 1995, CERN, Geneva.
4. H. Liu and J. Bisognano, to be published.
5. Originally from K. Crandall and L. Young at Los Alamos.
6. K. McDonald, *IEEE Trans. Electron Devices* **ED-35**, 2052 (1988).
7. H. Liu, *Computational Accelerator Phys.*, AIP Conf. Proc. No. 297, 508(1994).
8. H. Liu et al., *Nucl. Instr. Meth. A* **358**, 475 (1995).
9. H. Liu et al., *Proc. of 1995 Particle Accelerator Conf.*, **2**, 942 (1995).
10. H. Liu, *Microbunches Workshop*, AIP Conf. Proc. No. 367, 56 (1995).
11. *CEBAF FEL Conceptual Design Report*, Chapter 4, 1995, unpublished.
12. H. Liu, *CEBAF Tech. Notes*, TN# 94-040, 1994.
13. M. Jones and B. Carlsten, *Proc. 1987 IEEE Particle Accelerator Conf.*, p. 1319.
14. H. Liu, *Computational Accelerator Phys.*, AIP Conf. Proc. No. 297, 385 (1994).
15. H. Liu, *Nucl. Instr. Meth. A* **302**, 535 (1991).
16. H. Enge, *Rev. Sci. Instr.* **34**, 385 (1963).
17. H. Liu, *Nucl. Instr. Meth. A* **294**, 365 (1990).
18. H. Liu, *J. of Electronics* **13**, 293 (1991).
19. *An introduction to the T3D at LLNL*, NERSC document, July 1995.
20. *Cray T3D system architecture overview*, Revision 1.B, March, 1993, CRI.
21. *Cray T3D software 1.0 release overview*, CRI manual HR-04033 Ro-5215 1.0, 1993, CRI.
22. *Cray MPP FORTRAN reference manual*, SR-2504 6.2.2, 1995, CRI.
23. *SHMEM Technical Note for FORTRAN*, Revision 2.3, October 25, 1994, CRI.
24. T. Welcome, private communication.
25. M. Stewart, private communication.
26. D. Bhandarkar, *Alpha Implementations and Architecture*, Digital Press, 1996.
27. *Introducing the MPP Apprentice tool*, IN-2511 1.2, 1994, CRI.