IN9300071

BARC/1992/E/015

BARC/1992/E/015

CONCURRENT PARTICLE-IN-CELL PLASMA SIMULATION ON A MULTI-TRANSPUTER PARALLEL COMPUTER

by

A. N. Khare and A. Jethra
Electronics Division

and

Kartik Patel
Multidisciplinary Research Section

1992

BARC/1992/E/015

BARC/1992/E/015

GOVERNMENT OF INDIA
ATOMIC ENERGY COMMISSION

CONCURRENT PARTICLE-IN-CELL PLASMA SIMULATION ON A MULTI-TRANSPUTER PARALLEL COMPUTER

by

A.N. Khare, A. Jethra
Electronics Division

&

Kartik Patel
Multidisciplinary Research Section

BHABHA ATOMIC RESEARCH CENTRE
BOMBAY, INDIA

1992

BARC/1992/E/015

BIBLIOGRAPHIC DESCRIPTION SHEET FOR TECHNICAL REPORT

(as per IS : 9400 - 1980)

01	Security classification :	Unclassified
02	Distribution :	External
03	Report status :	New
04	Series :	BARC External
05	Report type :	Technical Report
06	Report No. :	BARC/1992/E/015
07	Part No. or Volume No. :	
08	Contract No. :	
10	Title and subtitle :	Concurrent particle-in-cell plasma simulation on a multi-transputer parallel computer
11	Collation :	36 p., 4 figs., 2 tabs.; appendices
13	Project No. :	
20	Personal author(s) :	1) A.N. Khare; A. Jethra 2) Kartik Patel
21	Affiliation of author(s) :	1) Electronics Division, Bhabha Atomic Research Centre, Bombay 2) Multidisciplinary Research Section, Bhabha Atomic Research Centre, Bombay
22	Corporate author(s) :	Bhabha Atomic Research Centre, Bombay - 400 085
23	Originating unit :	Multidisciplinary Research Section, BARC, Bombay
24	Sponsor(s) Name :	Department of Atomic Energy
	Type :	Government
30	Date of submission :	May 1992
31	Publication/Issue date :	June 1992

Contd... (ii)

40	Publisher/Distributor :	Head, Library and Information Division, Bhabha Atomic Research Centre, Bombay
42	Form of distribution :	Hard Copy
50	Language of text :	English
51	Language of summary :	English
52	No. of references :	11 refs.
53	Gives data on :	

60 Abstract : This raport describes the parallelization of a Particle-in-Cell (PIC) plasma simulation code on a multi-transputer parallel computer. The algorithm used in the parallelization of the PIC method is described. The decomposition schemes related to the distribution of the particles among the processors are discussed. The implementation of the algorithm on a transputer network connected as a torus is presented. The solutions of the problems related to global communication of data are presented in the form of a set of generalized communication functions. The performance of the program as a function of data size and the number of transputers show that the implementation is scalable and represents an effective way of achieving high performance at acceptable cost.

70 Keywords/Descriptors : ALGORITHMS; PARALLEL PROCESSING; PLASMA SIMULATION; DATA TRANSMISSION; COMPUTER NETWORKS; MESH GENERATION; DECOMPOSITION; PERFORMANCE; COMPUTERISED SIMULATION; P CODES; LAGRANGIAN FUNCTION; TWO-STREAM INSTABILITY

71 Class No. : INIS Subject Category : G5130

99 Supplementary elements :

1. INTRODUCTION

The widespread availability of concurrent processors promises to deliver computers with processing power greater than that of vector supercomputers but at much lower cost. However this power can be realised only if the programs implement parallel algorithms which fully exploit the concurrent execution capability of the machines. Efficient utilisation of concurrency requires re-grouping of the problem in order to isolate those parts which can be solved independently of other parts. This means that most algorithms must be modified or replaced.

Three-dimensional electromagnetic plasma simulation is one of those problems whose solution demands high performance machines. The importance of this field of computational physics has led to much effort on the development of plasma simulation codes [1] and their implementation on vector supercomputers like the Cray as well as on parallel machines like the Hypercube [2]. These programs consist of many thousands of lines of code and require large amounts of run-time memory. Such programs are presently being used to study, for example, the behaviour of plasmas occurring inside high-power microwave cavities, magnetic confinement fusion devices and plasmas occurring in the earth's magnetosphere and beyond [3] [4].

We report the implementation of a two-dimensional electrostatic plasma simulation code on a parallel computer built

around the Inmos T800 transputer. Most of the basic issues involved in the parallel implementation of plasma simulation are present in the two-dimensional plasma model. As explained in section 3, these problems involve data decomposition among the various processors and inter-processor communication. The algorithm used is scalable so that the same code can be executed on a machine with different number of transputers.

2. THE PARTICLE-MESH REPRESENTATION

The collective behaviour of charged particles subject to their own as well as external electromagnetic fields is the domain of investigation in plasma physics. Plasmas exist in many parts of the universe, and plasma devices have many applications. Due to the non-linear nature of the basic equations governing the behaviour of many plasma systems, and the inherent complexity of the systems themselves, plasmas are now being increasingly investigated through computer simulation.

The model that is used to represent a plasma in a simulation depends on factors like charge density, the strength of the coulomb coupling between the charges, as well as the time and length scales of interest. Strongly coupled high density plasmas are better represented as conducting fluids, and they are investigated via numerical solutions of the coupled transport and field equations. Weakly coupled, low-density plasmas may be approximated as a collection of interacting particles. This

representation is well suited for the simulation of electrostatic as well as magnetohydrodynamic plasmas. We discuss the particle representation in more detail below.

In the particle representation, the plasma is approximated by substituting the large number of point size charges with a comparatively smaller number of finite size particles. Each particle carries more mass and hence a proportionally greater charge. The plasma is constructed out of these particles which move, subject to electromagnetic fields, within a simulation volume called the computational box. The computational box is discretised into a spatial mesh or grid, usually by sets of parallel lines. The intersection of these lines define the mesh points (sometimes called grid points) on which the electromagnetic field variables (charge density q, electric field $\vec{E}$ and magnetic field intensity $\vec{B}$) are defined. The lines themselves define the cells, through the boundaries of which the current density, $\vec{J}$ is computed. The electromagnetic fields are found by numerically solving the discretised representation of the Maxwell equations on the mesh, and particle positions and velocities are calculated at fixed intervals of time by using the Lorentz equation. The particles can thus move freely throughout the computational box, whereas the mesh points, once defined, remain fixed.

Each particle interacts with others only via the fields which are defined on the mesh points. Since there is no direct

interaction amongst the particles in this model, it is known as the Particle-Mesh representation, and the method of simulation is called the Particle-in-Cell (PIC) method.

We may describe the procedure formally as consisting of four sequential stages:

a) Charge assignment stage:

Depositing the charge and current densities on the mesh for the calculation of the new values of the field variables.

b) Field solution stage:

Calculating the field variables ($\vec{E}$ and $\vec{B}$) on the mesh points via the solution of the discretised version of the Maxwell equations

$$\nabla.\vec{E} = q/\epsilon_0$$

$$\nabla.\vec{B} = 0$$

$$\nabla x\vec{E} = \partial\vec{B}/\partial t$$

$$\nabla x\vec{B} = (1/c^2)(\partial\vec{E}/\partial t) + \mu_0\vec{J}$$

c) Velocity update stage:

Calculating the forces on the particles from the Lorentz equation

$$\vec{F}=q(\vec{E} + \vec{v}x\vec{B})$$

and solving the Newton's equation of motion to find their new velocities

$$d\vec{v}/dt = \vec{F}/m$$

d) Position update stage:

Using the new velocities to update the particle positions.

All these four stages are discussed in detail in a previous report [5].

The iterative execution of the above four stages results in a self consistent evolution of the plasma. For the case of an Electrostatic plasma the Maxwell equations reduce to the Poisson's equation.

3. PARALLELIZATION OF THE PIC METHOD

The most important feature in a parallel program is the distribution of the data among the available processors. This distribution is called *decomposition* and is governed by the parallel algorithm which is used. It determines how much work a particular processor does during the course of the simulation, and decomposition methods, which is tightly linked to *load balancing*, is an area where much research needs to be done.

The PIC representation of the plasma in any computer code is realized through two major data structures. These are the field and particle data arrays. In the case of the electrostatic plasma the field data consists of the voltage and charge density arrays, which are used to store the voltages and charge densities at each mesh point. The particle data resides in the position and velocity data arrays, which stores the particle positions and velocities. These are the largest arrays and take up the major part of available memory.

In a parallel program, however, other data arrays are also of vital importance. Since communication is a fundamental aspect of concurrent processing, a parallel program also maintains data arrays which are required for the interprocessor data communication. These structures contain field and particle information which may be required by neighbouring processors during the simulation. The exact nature of these structures depends upon the specific parallel scheme which is used in the implementation, and is discussed in more detail below.

The following points are of significance in the parallelization of the PIC method:

a) As explained earlier there is no direct interaction amongst the particles, hence once the fields at the mesh points are known, calculation on each particle in the system can be done independently and simultaneously.

b) During the evolution of the system, particles deposit their charges on the neighbouring mesh points. Fields on the mesh points produce forces only on neighbouring particles. That is, there is local data dependence among the particle and mesh data points.

c) Mesh points in the system are stationary and evenly distributed.

d) Particles are mobile; their distribution in the system is non-uniform and is constantly changing during the simulation.

Keeping the points mentioned above in mind, we can divide the decomposition into two parts: one, the primary decomposition, affecting the grouping of particles among the processors; and two, the secondary decomposition, affecting the grouping of the mesh points. The aim of decomposition is to distribute the data and hence the computational load among the processors in such a fashion as to obtain load balance and meet data dependency requirements.

3.1 THE PRIMARY DECOMPOSITION

The distribution of the particle data is a complicated operation. This is because, as noted above, the particles are not stationary, and their relationship with the mesh points is in the nature of many-to-one. That is, many particles which lie close together may interact with the same mesh point (we use the term interaction in this context to mean data dependence). There are two main types of primary decomposition, known as Lagrangian and Eulerian [6].

3.1.1 *The Lagrangian Decomposition Scheme*

The particles are distributed into groups with equal number of particles in each group. Typically the number of groups will be equal to the number of processors. These groups are initialised at the beginning of the simulation and assigned to different processors for the duration of the entire run.

If it so happens that two particles which lie in the same

cell are put into separate groups (which implies different processors) then the same mesh point will be required during computation on each particle. If the mesh points are also distributed among the processors in exclusive sets, the required mesh point will thus lie in at most one of the processors, hence making it necessary for the other to import its value. During the simulation, as the particle positions change, the mesh points required for the computation of the force and assignment of charge need to be communicated to each processor during each time step.

There is, however, no way of knowing *a priori* when a processor will require a particular mesh point for its calculations. Hence, to simplify matters, each processor keeps a copy of the entire mesh with its associated voltage and charge density data arrays in its local memory, which is updated via global communication at each time step.

Data management is thus easy, because each processor has a fixed number of particles and mesh points to manage. There are no local messages to generate except those related to global communication, and the load is always balanced during the simulation.

This method may be simpler to implement in a fully connected parallel computer, when each processor can talk to any other processor, but in MIMD systems such as those built around transputers, connectivity is limited only to nearest neighbours.

Because global communication in such a system requires that the interprocessor links be used as a shared resource by every processor, such communication becomes a bottleneck as the number of processors increase. Also since global communication overheads increase with the number of processors, this will give problems associated with scaling-up.

3.1.2 The *Eulerian Decomposition Scheme*

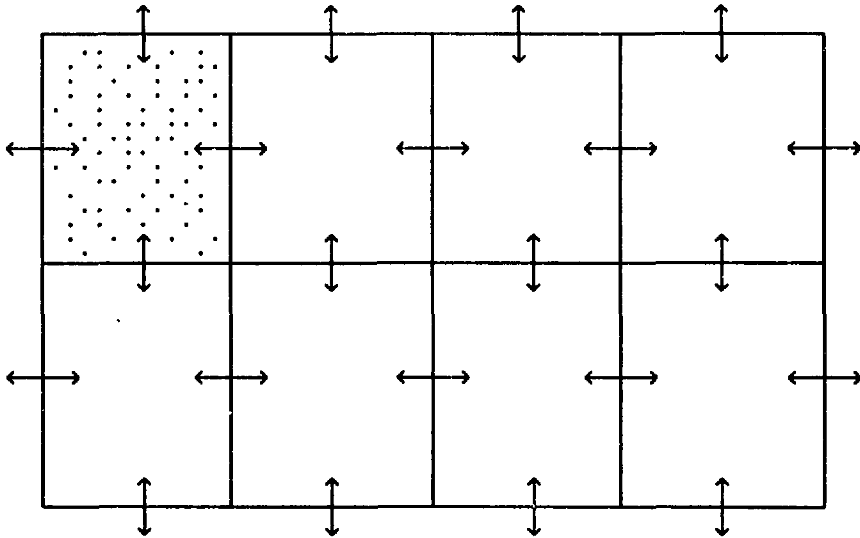
In this scheme the computational box is divided into non-overlapping subdomains. Typically the number of subdomains equals the number of processors, and each processor manages only those particles which lie within its subdomain. The particles whose coordinates fall within the same subdomain are thus grouped together. During the course of the simulation, particles which cross the boundaries of a subdomain are transferred to the group which lies within that subdomain, that is, handed over to the processor which is responsible for that subdomain. This requires local communication between processors in order to transfer the data related to the migrating particles. Each processor maintains those values of the mesh data arrays which lies within its subdomain, and which is updated at each time step. See Fig 1.

During the simulation different groups in general will not contain the same number of particles. The memory requirement for the particle arrays at each processor will thus dynamically change. A necessary consequence of the migration of particles is

Figure 1

The computational box decomposed into subdomains.

Arrows indicate particle migration across subdomain boundaries. Particles crossing the boundaries of the box reenter from the opposite end (periodic boundary conditions). Each subdomain is assigned to a different processor.



the maintenance of message buffers, creation and consumption of inter-processor messages and the manipulation of the dynamic data structure for the particles.

With the change in the number of particles in each group the

computational load on the processors will gradually become unbalanced. This drawback is removed in the Adaptive Eulerian decomposition scheme in which the subdomains are reconstituted after some time interval so as to equalize the number of particles in each group.

For local memory parallel machines with limited connectivity, such as transputer based computers, the Eulerian scheme is preferred because of its minimization of global communication, and its easy scalability.

3.2 THE SECONDARY DECOMPOSITION

Efficient solution of the field equations requires decomposition of the mesh points among the processors. The method chosen to solve the field equations depends on the particular set and the nature of the system in which they are being solved. For example, if only the solution of Poisson's equation is required in a rectangular computational box with periodic boundary conditions and without any internal electrodes, the Fourier Transform method [6] [3] is one of the fastest available, provided the mesh size is not very large. Parallel FFT's will thus provide the best means of calculating the fields.

On the other hand if the mesh size is large, then difference methods may be used, especially if there are internal electrodes whose arrangement has to be changed repeatedly.

Whatever be the method, the secondary decomposition will

divide the computational box into subdomains with equal number of mesh points in each region. These subdomains will be assigned to the processors which will then proceed with the solution of the field equations.

In the case of the 2-D Poisson's equation in a rectangular computational box which is being solved by using the SOR difference method, the box is divided into non-overlapping subdomains of equal area which contain equal number of mesh points. Each processor contains the data for the charge and mesh array for the region which is assigned to it. It is ensured that neighbouring subdomains are assigned to neighbouring processors, that is, the 2-D computational box is mapped onto a similar processor array. Each processor carries out the iteration over those points which lie within its region. At the end of the iteration the mesh values at the boundaries are exchanged via local communication and the iteration resumed. The progress is checked by the usual means and terminated when the solution is reached.

In the case when this method is used together with Eulerian decomposition for the particles, the subdomains for both the decompositions may be kept the same. This avoids global communication required in the Lagrangian or Adaptive Eulerian schemes.

4. IMPLEMENTATION

The PIC code for a two dimensional electrostatic plasma was implemented with the Eulerian decomposition scheme used for primary decomposition. The field equations were solved using the SOR difference method.

4.1 HARDWARE

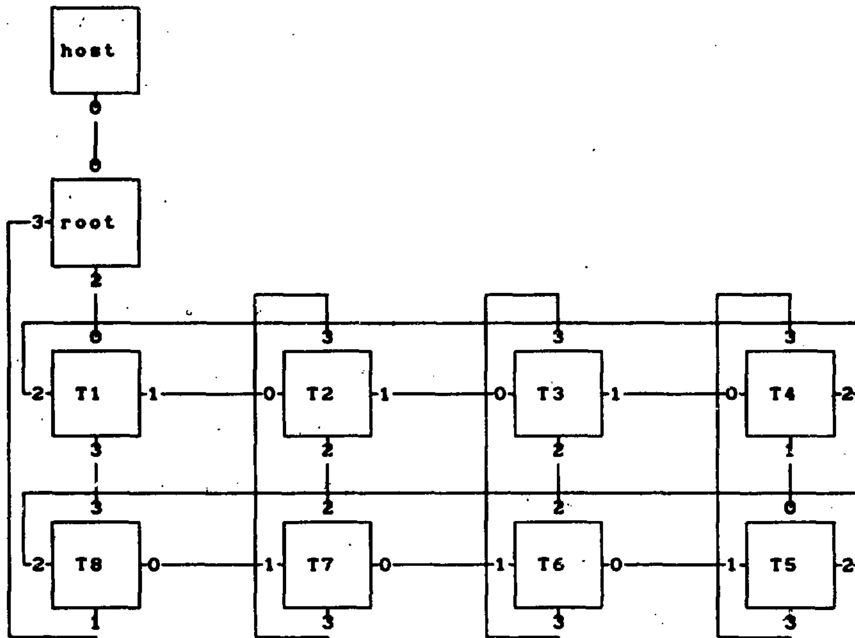
The development work was carried out on a system which is a network of 8 transputers (T800) with each node having 1M byte of memory (600 ns access time) [7]. Two links of each transputers are fixed - connected to form a ring of transputers. The remaining two links can be configured in desired manner by software instructions. In addition to this there is a root transputer which is also included in the ring.The remaining two links of root are used for interprocessor communication other than that required for solving the problem. One is used for interfacing with the Host and other to a Network Configuration Unit (NCU). Hence only two links from the root are available for the program to map the interprocessor communication.

The 8 transputers in the system are connected as an (m × n) torous (here 4 × 2) to implement the 2 dimensional simulation with periodic boundary conditions (see Fig. 2). The root is connected to the front end system where all I/O is managed.

Figure 2

Layout of the transputer network as a (4 * 2) torous

(processor configuration)



T1,...,T8 : Network Transputers

root : Root transputer

host : The host computer (PC-AT)

0,...,3 : Transputer communication links

——— : Hardwired connections

4.2 SOFTWARE

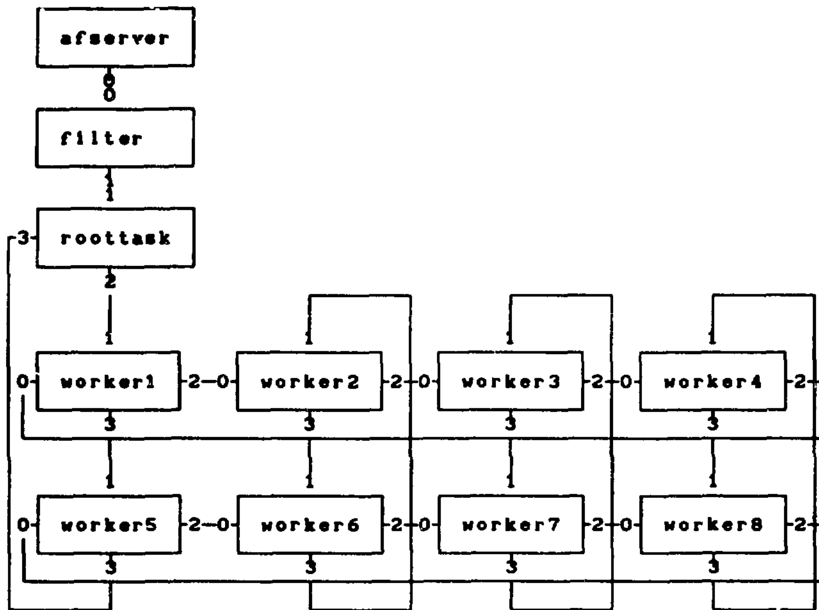
The programming was done in Parallel C and the 3L compiler was used [9]. The root transputer supports I/O performed on front end host and a separate program (roottask.c, see Appendix A) is written for that. All other transputers in the system are programmed with a generalised worker program (worktask.c, see Appendix B), in which location independent behaviour of all the workers is coded. The location dependent behaviour of each worker is programmed separately in four units (typeX.c, where X = 1, 2, 3, or 4. See Appendix C). Each of these units contains the same number of identically named functions, but which behave differently because they are location dependent. Depending upon the location of the worker in the overall network, the appropriate unit is linked with the worker task to achieve overall synchronous local and global communication.

The program which is to be executed on the network of transputers is generated by running a configurer which is an essential part of the 3L Parallel C compiler. The configurer takes as input a user-defined configuration file (see Appendix D). This file is essentially a formal description of the transputer network along with other details specifying the placement of the programs within it. The process-to-processor mapping is done automatically by the utility. It generates a single binary file which is loaded into the network by a server program (also a part of the compiler)

Figure 3

The transputer network as an array of processes

(process configuration)



afserver, filter	: processes running in the host
roottask	: root transputer process
workerX, X=1,...,8	: worker tasks running in the network transputers
0,...,3	: logical communication ports for interprocess communication

which runs on the host and which links the transputer network with the host and, ultimately, the user.

Figure 3 shows the process configuration mapped onto the transputer array and the location of the root and the worker programs

The simulation program takes run time parameters such as the number of particles to be simulated, the dimensions of computational box, constants such as the magnetic field, plasma frequency etc. from an input file and generates files for kinetic, electrostatic and total energy and the potentials on the mesh points. The diagnostics and timing analysis data is also generated during course of execution.

5. RESULTS

The scalability of the algorithm on T800 transputer array is demonstrated by the results obtained.

The tables below show the speed-up and efficiency of the parallel program as a function of number of processors and the data size (related to the number of plasma particles used for simulation). The observations were taken by running the program with the plasma configured to simulate the Two-Stream Instability.

The efficiency is given by

$$\eta = S_n/n$$

where S_n is the speed-up obtained for n transputers, and n is the total number of transputers in the network. S_n is derived from

$$S_n = T_1/T_n$$

where T_n is the execution time of the parallel program on n transputers and T_1 the time required to run the sequential program. I/O times have been excluded from the measurement of execution time.

During the initial phase the particles are uniformly distributed so that each processor handles the same number, that is, the load is perfectly balanced. Table 1 shows representative values of speed-up and efficiency for this initial phase of the simulation.

Table 1

Relative	4 nodes		6nodes		8 nodes	
Data Size	SpeedUp	η	SpeedUp	η	SpeedUp	η
1	2.96	0.75	4.07	0.68	5.58	0.7
3.8	3.10	0.77	4.66	0.78	6.18	0.77
8.5	3.15	0.77	4.6	0.77	6.3	0.79
14.8	3.16	0.79	4.61	0.77	6.31	0.79
20	3.16	0.79	4.6	0.77	6.3	0.79

A data size of 1 corresponds to 5048 particle

A data size of 20 corresponds to 1,07,648 particles

It is clear from the near constant value of the efficiency that the performance does not deteriorate as the number of transputers increase. The algorithm implemented in the program is scalable.

Table 2 shows the effects of load imbalance on efficiency. The Two-Stream Instability is simulated for a total of 250 time steps. During this period the plasma instability manifests itself and then dissipates. In this phenomenon nearly all the particles crowd into one half of the computational box for a short time. This means that in this interval half the number of transputers are doing all the particle related calculations while the remaining half are almost idle.

Table 2

Relative Data Size	4 nodes Efficiency, η	8 nodes Efficiency, η
1	0.52	0.37
3.8	0.68	0.59
8.5	0.74	0.68
14.8	0.76	0.72
20	0.77	0.74

A data size of 1 corresponds to 5048 particle

A data size of 20 corresponds to 1,07,648 particles

The efficiency varies from about 0.3 to 0.7. This is due to overheads in the parallelisation of the method, mainly associated with interprocessor communication. The interprocessor communication overheads are fixed when the number of transputers remain constant. Thus efficiency thus improves as the number of particles increase. It approaches the same value as for the ideal case (in Table 1) when the data size is sufficiently large. For a given data size the efficiency goes down as the number of transputers is increased. This is because increasing the number of transputers increases the interprocessor and global communication overheads. This behaviour is typical in MIMD systems and is also seen in other application areas such as image processing. Such systems are most efficiently utilized for large problems.

In passing, one may note that the processor load is a problem dependent feature. If one were to simulate an expanding plasma cloud the load would be more balanced throughout the calculation than for this particular problem.

6. CONCLUSIONS

It is clear that for large, computationally intensive problems, only a scalable parallel computer with a scalable algorithm to solve the problem will be able to deliver results at acceptable cost.

The results show that the algorithm is perfectly scalable

but its performance is affected by load imbalance. The load imbalance in a parallelised PIC code, though inevitable, is strongly problem dependent. It is a more severe problem in distributed memory parallel systems having limited connectivity as compared to shared memory systems. In contrast, scalability is limited in shared memory systems [8]. The load imbalance, however, can be minimised by applying adaptive methods at the expense of increase in program complexity [2] [11].

REFERENCES

1. "Methods in computational physics ", Volume 9 and Volume 16, B.Alder, S.Fernbach, N.Rotenberg, series editors, Academic Press, New York, 1970
2. "A general concurrent algorithm for plasma Particle-in-cell simulation codes ", P.C.Liewer, V.K.Decyk, Journal of Computational Physics, 85 ,302 (1989)
3. "Computer simulation using particles", R.W.Hockney, J.W.Eastwood, McGraw-Hill, New York, 1981
4. "Plasma physics via computer simulation", C.K.Birdsall, B.K.Langdon, McGraw Hill, New York, 1985.
5. "Particle simulation of a two dimensional electrostatic plasma", K.Patel, BARC Report-1489 (1989)
6. "Mapping Schemes of the particle-in-cell method implemented on the PAX computer", T.Hoshino, R.Hiromoto, S.Sekiguchi, S.Majima, Parallel Computing, 9, 53 (1989-90)
7. "Reconfigurable parallel processing system", A.N.Khare, et.al., Proceeding of workshop on parallel processing, BARC (1990)
8. "Supercomputers and their use", C.Lazou, 180-187, Clarendon Press, Oxford, 1986.
9. "Parallel C Users Guide", 3L Ltd.,1988.
10. "Modeling the performance of hypercube: A case study using the particle-in-cell aplication", M.Lubeck, V.Faber, Parallel Computing, 9, 37 (1989-90)

11. "Characterising the parallel performance of a large-scale, particle-in-cell plasma simulation code", Concurrency: Practice and experience, 2(4), 257-288, December 1990

Appendix A: The Roottask program

The functions of the program running on the Root transputer are twofold. Firstly it coordinates data flow to and from the network transputers . Secondly it acts as a connecting link in the torous in order to support interprocessor communication during particle exchange operations. Since this version of the program did not incorporate a parallel Poisson solver, it receives charge density values from the workers, solves Poisson's equation and sends the potential values back to the workers. The total time required for the solution of Poisson's equation is a small fraction of the time required to advance the particles, so this is an acceptable compromise. This function will be removed from the Root to the workers in later versions of the program.

It reads plasma related data supplied from the input data file and communicates them to the workers. It receives energy data from the workers and stores it to disk for later processing.

The root is fitted into the torous topology by breaking one link of the torous and connecting these two broken links to two links of the root (see Fig.2). Thus a logical Torous is formed

while maintaining the link with the host.

As a link in the torous it is used as a relay station to convey particle data from one network transputer to the other in order to fulfil the periodic boundary conditions necessary for the simulation.

In addition to this the program monitors the links for run-time error messages from workers at critical stages.

Appendix B: The worktask program

The workers are programmed in two parts. One is a position independent part and the other is a position dependent part in the form of a *communication type*, which is described in Appendix C.

The workers hold both the particle and mesh data strutures as described below.

Since the computational box is represented as a two-dimensional array of mesh points, it has to be decomposed into suitable blocks and assigned to the workers as their domains. Particles which lie within a particular domain will be the responsibilty of the processor in charge of that domain.

The mesh points per worker are decided from the run time parameters taken from the input data file. These values are received from the root at the beginning of the program. The worker initialises three arrays for keeping the mesh related data. Two of

these store the potentials and charge densities at the mesh points. The third array is required to store intermediate results.

In the simulation there are two streams of particles and the particles of each stream are handled seperately in two data strutures. Associated with each particle four parameters are stored, namely, the two position coordinates and the two velocity components. The number of particles per node during the simulation is a variable quantity, hence these structures are varied dynamically at each time step. After initialising the particles the worker enters the main program loop.

In the main loop the worker program computes the charge density on the mesh points which lie within it. These values are sent to the Root, which returns the potential values after solving Poisson's equation. This is used to compute the force acting on each particle, which in turn is used to compute the acelleration of the particle and its subsequent new position. The worker checks the particle position against its knowledge of the domain boundaries. If the particle has crossed a domain boundary it puts the particle parameters in the appropriate transfer array. There are four such arrays, one for each boundary. After all the particles have been scanned the tranfer arrays are sent off to the neighbouring transputers. Here the Root transputer acts as a relay for two transputers in the torous. The incoming transfer arrays from the neighbours are also accepted. The particle arrays are

reallocated and the incoming particles placed in them. In this process the arrays may shrink or grow, depending on whether more particles have left or entered. Empty spaces in the particle array are removed. At this stage if a memory allocation error occurs, the Root is informed and the program terminated.

The loop begins again by the recalculation of the charge density on the mesh points.

In this loop certain other diagnostic data are also calculated and sent to the Root for storage to disk and later processing. These are energy and velocity data which are indicators of the progress of the simulation, and of course the reason for running the program in the first place.

Appendix C: The Global Communication functions

Position dependence in the workers occurs as a result of the communication with the root which is neccessary for the extraction of data. Workers which are at the extreme end (as seen from the viewpoint of the Root) need to receive and send data along one link, whereas the inner workers need to send and accept their own, as well as act as a relay to pass on data which are generated by and meant for the other workers.

The typeX.c units, as mentioned earlier, provides the coding of position dependent logic. The node can be catagorised into one

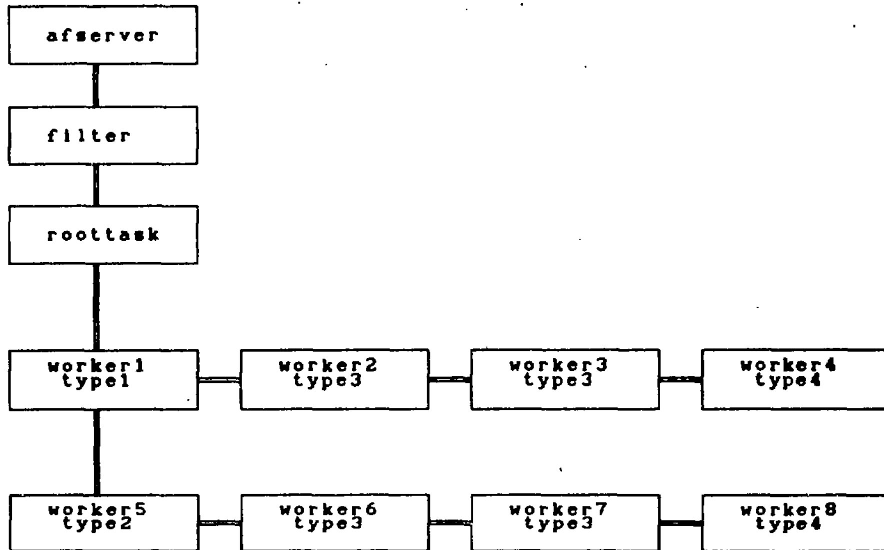
of the four types depending upon its position with respect to root.

A dataline is superimposed on the network in vertical horizontal distribution way (Fig. 4). The roottask-worktask communication data flows through this dataline. The same dataline is used for any global communication between the worker nodes. The node can be in a T-joint position (type1.c), L-joint position (type2.c), pipe position (type3.c) or terminator position (type4.c). Depending upon the node type, the appropriate unit is linked to the worker. Figure 3 shows the 4x2 torous network along with the dataline and the nodes assigned with their types.

Similarly nodes of any torous can linked in order to achieve global communication. This solves the position dependency of the global communication code in a very general way and the same units can be reused for a network of any size.

Figure 4

The Global communication dataline



——— : Path used for global communication

workerX : The location independent worker task

typeX : The location dependent communication task

(where X=1,2,3,4)

Appendix D: Configuration of the program on given network

In Appendix A the code for Root transputer was discussed. In Appendix B the code for workers was described which runs on each network transputer. In Appendix C the location dependent code was briefly described. These components are combined by means of a configuration file for generating a complete program which can run on any given mxn torous network.

A configuration file is created in which the processes and their interconnection pattern is formally stated. The transputers in the system along with their connectivity are also defined. The configuration utility of the Parallel C compliler is applied on this configuration file and the executable program to run on the complete network is generated. The configuration file for 4x2 torous is given here. In order to run the program on a different network, say 4x4, all one needs to do is to create a similar file for that network and use the same utility to create the required executable program.

Listing of Configuration file

```
!give names to the processors in the network
Processor Host
Processor Root
Processor T1
Processor T2
Processor T3
Processor T4
Processor T5
Processor T6
Processor T7
Processor T8
```

```
!define hardwire connections between processor links
Wire ? Host[0] Root[0]
Wire ? Root[2] T1[0]
Wire ? Root[3] T8[1]
Wire ? T1[1] T2[0]
Wire ? T1[2] T4[2]
Wire ? T1[3] T8[3]
Wire ? T2[1] T3[0]
Wire ? T2[2] T7[2]
Wire ? T2[3] T7[3]
Wire ? T3[1] T4[0]
Wire ? T3[2] T6[2]
Wire ? T3[3] T6[3]
Wire ? T4[1] T5[0]
Wire ? T4[3] T5[3]
Wire ? T5[1] T6[0]
Wire ? T5[2] T8[2]
Wire ? T6[1] T7[0]
Wire ? T7[1] T8[0]

!define tasks and the number of associated in and out channels
Task Afserver Ins=1 Outs=1
Task Filter Ins=2 Outs=2 Data=10K
!roottask
Task roottask Ins=4 Outs=4 Data=?
!worker tasks
Task work1_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work2_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work3_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work4_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work5_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work6_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work7_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code
Task work8_8 Ins=4 Outs=4 Stack=2K Heap=? Opt=Stack Opt=Code

!define logical connections between tasks
Connect ? Afserver[0] Filter[0]
Connect ? Filter[0] Afserver[0]

Connect ? Filter[1] roottask[1]
Connect ? roottask[1] Filter[1]

Connect ? roottask[2] work1_8[1]
Connect ? work1_8[1] roottask[2]

Connect ? roottask[3] work5_8[3]
Connect ? work5_8[3] roottask[3]
```

```
connect ? work1_8[0] work4_8[2]
connect ? work4_8[2] work1_8[0]

connect ? work1_8[2] work2_8[0]
connect ? work2_8[0] work1_8[2]

connect ? work1_8[3] work5_8[1]
connect ? work5_8[1] work1_8[3]

connect ? work2_8[2] work3_8[0]
connect ? work3_8[0] work2_8[2]

connect ? work2_8[3] work6_8[1]
connect ? work6_8[1] work2_8[3]

connect ? work2_8[1] work6_8[3]
connect ? work6_8[3] work2_8[1]

connect ? work3_8[1] work7_8[3]
connect ? work7_8[3] work3_8[1]

connect ? work3_8[2] work4_8[0]
connect ? work4_8[0] work3_8[2]

connect ? work3_8[3] work7_8[1]
connect ? work7_8[1] work3_8[3]

connect ? work4_8[1] work8_8[3]
connect ? work8_8[3] work4_8[1]

connect ? work4_8[3] work8_8[1]
connect ? work8_8[1] work4_8[3]

connect ? work5_8[0] work8_8[2]
connect ? work8_8[2] work5_8[0]

connect ? work5_8[2] work6_8[0]
connect ? work6_8[0] work5_8[2]

connect ? work6_8[2] work7_8[0]
connect ? work7_8[0] work6_8[2]

connect ? work7_8[2] work8_8[0]
connect ? work8_8[0] work7_8[2]

!define location of tasks in the transputer network
Place Afserver Host
Place Filter Root
```

```
Place roottask Root

Place work1_8 T1
Place work2_8 T2
Place work3_8 T3
Place work4_8 T4
Place work5_8 T8
Place work6_8 T7
Place work7_8 T6
Place work8_8 T5
```

Acknowledgements

We gratefully acknowledge the keen encouragement and advice given by Shri B.R.Bairi, Head, ELectronics Division and Dr. P.R.K.Rao, Head, MDRS.

We are also grateful to Shri M.D.Ghodgaonkar, Electronics Division without whose active help and cooperation this work would not have been possible.

We are also grateful to Dr. R.Chidambaram, Director, BARC, at whose suggestion this work was initiated.

Published by : M. R. Balakrishnan Head, Library & Information Services Division
Bhabha Atomic Research Centre Bombay 400 085