Parallel Computation for Electronics Design

ERNEST G. ZAIDMAN

Abstract—The use of parallel computational techniques for the design of vacuum electronic devices is presented with applications. The capabilities of parallel machines with an overview of the environment of the Connection Machine at NRL are discussed. An overview of particle-in-cell numerical techniques with application to both one-dimensional and three-dimensional codes is presented.

I. INTRODUCTION

Due to the increasing complexity of the design of vacuum electronics, the vacuum electronics branch at the Naval Research Laboratory has been developing an increased computational support base. The extension of this support to parallel architectures is in development on the NRL Connection Machine. A distinction is made between vector machines such as the Cray series and the many processor machines such as Goodyear’s Massively Parallel Processor and Thinking Machine’s Connection Machine series.

The use of parallel computational processing for the study of the motion of electrons began before the use of digital computers with the use of Hartree’s differential analyzer, an analog system. By dividing the workload among the processors, i.e., his students, the effective throughput was greatly increased. The use of computers to study the motion of electrons in vacuum electron devices dates back to work on magnetrons by Hartree and Buneman, and on the traveling-wave tube by Nordsieck in the 1940’s. These simulations were only able to include a small fraction of the system with greatly constrained physics but provided a significant improvement in our understanding of the nonlinear behavior in these devices. Today, available simulation tools are on the verge of becoming fully three dimensional with the complete set of Maxwell’s equations satisfied. The computational resources required by these codes include a significant amount of computer memory, speed, and cost. On a Cray class machine, with information organized in typically 64 bit words, computational speed measured in floating point instructions per second (flops), problems of interest with $10^5–10^7$ operations require megawords ($10^6$ words) and hundreds of millions of Flops (MFlops), which can cost hundreds to thousands of dollars to provide answers within an acceptable time. These constraints upon resources arise from both the computer architectures and available simulation codes that are constrained to work on these architectures. One possible approach to improving the simulation’s usefulness in design is the implementation of three-dimensional fully electromagnetic design codes on parallel architecture machines.

In the following, the advances of parallel computer architecture are summarized and compared to the advances in vector architectures in Section II. A discussion of particle-in-cell (PIC) simulation techniques with examples for a one-dimensional parallel electrostatic code is presented in Section III. Section IV presents possible future directions in the use of parallel computational techniques for vacuum electronics design.

II. COMPUTER ARCHITECTURE ADVANCES

As one tries to improve the performance of the computational hardware upon which simulations are to be done, several problems appear immediately in the traditional serial machine approach. The circuit switching speeds are limited by materials and fabrication techniques. The requisite compactness of design due to the finite speed of signal propagation gives heat dissipation difficulties. In practical devices, the rate at which heat can be removed places a maximum circuit density limit that, for a given material, is an important factor in determining the net computational speed available. These problems are largely avoided by using many less densely packed processors, substituting instead problems of definition of solution methodology, memory access conflicts, and interprocessor communications.

The vendors in the vector supercomputer arena include Fujitsu, Hitachi, NEC, ETA, CDC, and Cray. Consider the advancements in the Cray systems for example. Just considering the processor speeds, the improvement has been from $\approx 10^5$ flops in the Cray-1 to a projected speed of $\approx 10^{12}$ flops in the proposed Cray-4. The first Cray-1 [1] was delivered to Los Alamos Scientific Laboratory in early 1976. Its increase in speed over the then current computers was made possible to a large extent by the inclusion of vector pipelining, a technique to keep the CPU busy. To further increase performance in the Cray-2, the removal of heat in the more compact geometry was addressed by immersion of the boards into liquid refrigerant. The Cray-3 uses GaAs technology to improve switching speeds as does the proposed Cray-4 (with $\approx 10^{12}$ flops projected) which is to contain 64 processors. The vector supercomputers all use relatively few high-speed CPU’s, but the instructions to each CPU are pipelined so that par-
allel-type instructions may be completed rapidly with bus-based communications.

The opposite approach is to marshal the powers of many less powerful processors to attack the computations in parallel. A seminal paper for parallel architectures is *The SOLOMON Computer* by Slotnick et al. [2]. They proposed a design with 1024 processors under the control of a central control unit with a single instruction stream. The SOLOMON concept forms the basis of several parallel machines, including the ILLIAC IV [3], International Computers Ltd. Distributed Array Processor (ICL-DAP) [4], Goodyear Aerospace Corporation’s Massively Parallel Processor (MPP) [5], and Thinking Machines Corporation’s Connection Machine (CM) [6].

To orchestrate the computations among the processing elements, a communications network is established. To be useful, this communications network must be relatively efficient and cost effective. The approach of connecting all of the processors together directly is seen by most as being prohibitively expensive. The ILLIAC IV computer was delivered to NASA Ames Research Center in 1972, but was not really operational until 1975 due to the fact that the state-of-the-art techniques incorporated into its manufacture were too ambitious at that time. The delivered machine had 64 processing elements with each processor connected to its nearest neighbors. Fig. 1 illustrates nearest neighbor connectivity with a two-dimensional grid. The International Computers Ltd. Distributed Array Processor (ICL-DAP) was introduced in the 1970’s with 4096 processors in a 64 by 64 array with nearest neighbor communication. The Goodyear Aerospace Corporation’s Massively Parallel Processor (MPP) with 16384 simple processors, each with 1 kbit of local memory, also uses nearest neighbor communication. A compromise between the direct connection of all processors and the nearest neighbor approach is to connect each processor to each of \(2^n\) others. This Boolean n-cube or hypercube connectivity is more widely used and is the communication network used in the Connection Machine (CM). The Boolean n-cube connectivity is perhaps best illustrated recursively with the connections per processor increased by one for every doubling of the total number of processors as seen in Fig. 2. The CM series has up to 65 536-bit sliced processors each with 64-kbit local memory with reconfigurable data routing among the hypercube connection paths as well as nearest neighbor. The bit sliced processor works with data 1 bit at a time rather than with words of, for example, 64 bits as in a Cray. The design complexity and cost of such a processing element can thus be relatively small, a necessary attribute for machines with so many of them. Well established techniques were used for the construction of the CM, avoiding the ILLIAC IV difficulties.

A comparison between a typical coarse division of computation, a two-processor Cray X-MP/24 and the most finely divided division in a CM-2 may be instructive. The Cray performance ranges from about 5 Mflops for an unvectorized code to around 480 Mflops for highly vectorized cases. The CM-2 peaks at 10 000 Mflops for an essentially fully parallel dot product. The word size on the X-MP is 64 bits while on the CM-2 a bit-sliced approach is used, but 64-bit floating point coprocessing chips can be used. The amount of memory for the X-MP/24 is 4 Mwords of 64 bit words and the CM-2 can have 64 Mwords of 64 bit words but the processors are really bit sliced. The cost of these machines is roughly $8.5 million for the X-MP/24 and roughly $1 million for the CM-2. The cost per instruction per second on the CM-2 approaches 0.1c compared to about 8.5c for the X-MP/24. At NRL’s CM-1 and X-MP/24 for FFT calculations, the crossover point in CPU performance comes at about 4000 processors operating simultaneously [7]. The ability to develop algorithmic techniques to distribute the computational load (load balancing) and access memory in an efficient fashion across processors determines the applicability of such architectures.

III. PARTICLE IN CELL SIMULATION TECHNIQUE

The use of the particle-in-cell (PIC) method [8] is particularly effective for vacuum electronic device simula-
tion because the fundamental Maxwell and Newton–Lorentz equations are used to calculate the particle-field interactions self-consistently. The principle bottleneck for sequential computer architectures (including vector architectures) is the massive number of particles required for realistic PIC simulations. The use of multiple CPU’s to perform particle calculations simultaneously could alleviate this difficulty. Buzbee [9] examined fusion directed particle codes and found a large degree of parallelism. The parallel architecture CHI system by Dawson et al. [10] showed that a significant performance improvement with reasonable cost could be achieved in practice with a parallel PIC model. More recently, in a hypercube geometry, Liewer et al. [11] implemented a one-dimensional electrostatic PIC code on the JPL Mark III Hypercube. The particle calculations were done in parallel, with an efficiency (speedup/number of processors) for the particle push of 84 percent, resulting in an overall code performance of 1/7 the speed of a Cray 2 for a model problem.

The principle algorithmic areas in a PIC simulation code include: 1) conversion of particle quantities such as charge and current into quantities defined on a computational grid; 2) solution of the field quantities at these grid positions; 3) interpolation of these fields back to the particle locations, so that 4) the particles may be advanced forward in time.

With the advent of particle ↔ grid simulation techniques, the number of operations required to solve the system at each step forward in time went from \( \approx N_{\text{particle}} \) to \( \approx N_{\text{grid}} \log N_{\text{grid}} \). For parallel architectures, when the communication pattern is limited as is the nearest neighbor scheme, extreme programming care is required as the interprocessor communications time can easily eliminate any gains due to a multiple processor distribution of the computation. Perhaps one could not use a grid at all and instead use discrete Fourier transform techniques with a resulting operation count \( N_{\text{particle}} N_k \) for \( N_k \) Fourier modes. Since \( N_{\text{particle}} \gg N_{\text{grid}} \) the gridless system requires more steps, but if each particle can be assigned to its own processor, the operation count goes to \( \approx N_k \). When connections are more readily established between CPU’s, the grid based approach becomes attractive again as the element with which the geometry of the device can be mapped onto the computer.

In the traditional serial and vector machines, the codes are often dominated by the particle calculations. When the number of processors becomes on the order of the number of particles, this is completely reversed. The bottleneck becomes the translation between the particle and grid quantities. Thus, areas 1 and 3 above, which depend heavily upon the details of memory access, define the new bottleneck. In area 2, the solution for the field quantities on the grid may be implemented with each grid node processed by a different CPU using a local formulation of the relevant equations. Area 4 may also be treated as a algorithmically local task. Some first steps toward implementing a vacuum electronic device design code were accomplished with the construction of a simple one-dimensional electrostatic code that addresses these four areas.

The one-dimensional electrostatic code developed for the Connection Machine environment was written in a variant of Common Lisp denoted "*Lisp" with a Symbolics 3600 series Lisp machine front end. The user interface resides on this front end and the serial computations are done there. (The Connection Machine was originally envisioned as a computer for parallel artificial intelligence applications, so Lisp was the first high-level language for it.) Some of the most powerful constructs on the CM are involved in the distribution of information from one parallel variable into another. A parallel variable such as the set of \( x \) positions for all particles is distributed among the processors with each particle’s attribute \((x \text{ position})\) at one processor (multiplied by the number of virtual processor layers if virtual processors are used). We look at the four areas starting with the particle-to-grid gathering. For each processor element that has a particle associated with it, that processor is turned on for the operation of selecting that particle’s charge and position. The particle’s position is the parameter that determines which processor receives the charge of that particle into its charge-accumulation grid variable. Although the target code was to use linear interpolation for each dimension, the feasibility of the overall approach was first looked at using a zeroth-order scheme. For the zeroth order nearest grid point interpolation (NGP) scheme on a unit-spaced grid, this all occurs in one line of *Lisp code

\[
(*\text{pset}:\text{add}(\text{!} \text{charge}) \text{charge-density}(\text{truncate}!! \text{x-particle})).
\]

An explanation of the various elements in this statement is as follows. The "*pset" is a *Lisp function that takes one parallel variable (pvar) and writes it into another. The "::add" qualifier to the "*pset" indicates that in writing into the destination pvar the values written are to be added to the values already in place. The value pvar "(!! charge)" is actually composed of the parallelizing operator "!!" operating on the normal variable charge. This forms a pvar with the value "charge" at each processor.

The uniform syntactic form between data and program in Lisp allows the substitution of this program, i.e., distribute charge into a pvar, for data, i.e., a value pvar. The destination pvar, "charge-density," is written into based upon the address pvar "(truncate!! x-particle)." The value pvar is written into the destination pvar of the processes addressed by the address pvar. Each processor can contain the elements of many pvars, so that processors can contain both particle elements and grid elements, one or more particles and/or grid nodes at each processor.

The process of scattering the field quantities back to the particles, area 3, is similar to the gathering step, area 1. The corresponding *Lisp code is
(*)set clx - particle
(preff!!(-!!
(preff!! electrostatic-potential(1 + !!((self-address!!))))
(preff!! electrostatic-potential(1 - !!((self-address!!)))
(truncate!! x - particle))).

This statement is only slightly more complicated than that for area 1. The ‘‘*set’’ is a *Lisp function that takes a
pvar and transfers its contents to another pvar in a purely
local sense, i.e., the element of the source pvar in pro-
cessor n is transferred into the element of the destination
pvar also in processor n. The ‘‘clx-particle’’ is the desti-
nation pvar. The source pvar is what remains in the above
expression, and it is here that the communications be-
tween processors is used. A finite-difference approxima-
tion to the electrostatic field from the electrostatic poten-
tial is used with the particular CPU address locations
referenced via ‘‘pref!!’’ located at right and left of the
self-address of the processor associated with the particle
in NGP fashion via the ‘‘(truncate!! x-particle)’’ con-
struct.

The solution of the field at the grid positions, area 2,
may be accomplished any of several ways. These may be
broken down into the broad classes of local and nonlocal
algorithms. In the nonlocal class, a fast Fourier transform
technique may be used as an example. Here, Poisson’s
equation $\frac{\partial^2 \phi}{\partial x^2} = -\rho$ is transformed into the wave-
number domain for solution $\phi(k) = k^{-2}\rho(k)$, and then
an inverse transform on $\phi$ yields the desired result. Since
the relative speed of operations within a single processing
element to the speed when processor-to-processor com-
munications must be used is on the order of 1 $\mu$s to 1 ms,
which do not rely heavily on global communica-
tion may be preferred. This local class of algorithms
must still account for the boundary conditions of the de-
vice simulation, however. For local sampling, $k^2$ is
replaced by $k^2 \text{dif}(k\Delta x/2)$, which actually reduces to the
use of the three-point finite differencing of $\nabla^2$. For the
one-dimensional electrostatic code a relatively simple di-
rect tri-diagonal solver for periodic boundaries was im-
plemented for the solution of Poisson’s equation. If one
lets the potential at processor 0 be 0 (*Lisp starts counting
at 0) and uses charge neutrality to obtain a potential at
processor 1, the serial approach takes steps through each
of the remaining processors in turn to solve $\phi_1 = \rho_{1-1} +
2\phi_{1-1} - \phi_{2-2}$. The principle bottleneck in serial and vec-
tor machines, area 4, the particle advancement step, is the
most efficient on the parallel processor machine since all
of the required quantities are local to each processor and
no communication is required.

One of the initial test cases is the electrostatic two-
stream instability. The simulation length was 128 $\lambda_d$ with
2000 particles, $\omega_p = 1$, with streaming velocities
$\pm 0.2$. Timing information was not taken for these runs
as the implementation was not optimized. The wall clock
time was almost completely determined by the output of
graphics on the Symbolics front end. The simulation pro-
ceeded as anticipated with Fig. 3 showing the phase-space
as anticipated for the instability.

IV. DISCUSSION AND FUTURE DIRECTIONS

Experience with the design and development of the one-
dimensional electrostatic code was so encouraging with
respect to the ease of establishing flexible communication
patterns that the construction of a three-dimensional code
is in progress. The overall structure of this code is similar
to that described above. The major differences arise from
the use of three-dimensional generalized geometries. To
accommodate the wide variety of geometries that differen-
tiate among the devices that might be considered, a metric
is defined locally at each grid node allowing the use of con-
formal mapping of the grid to the device. A local Cartesian
space is used for particle pushing to avoid some numerical
difficulties. The solution of the field quantities uses local
convolution operators to solve the equations. The boundary
conditions for the fields are then used as a global condition
that is propagated through the system in an iterative man-
ner. The efficiency with which the boundary information
is propagated through the computa-
tion and the locality of these conditions is expected
to significantly influence the potential usefulness of the
parallel code implementations. The difficulties of load
balancing and memory management have been signifi-
cantly less important than previously expected. The as-
ignment of particles uniformly among processors distrib-
utes the loading, while the combination of nearest
neighbor and hypercube communication routing in a large
bandwidth path minimizes the penalty. To improve the
numerical noise and accuracy, digital filter techniques will
be utilized, with grid discreteness and aliasing to be min-
imized with the same approach as the convolution tech-
niques already in place.

From the user’s perspective, the code should provide
ease in describing the device to be simulated with a
clear exposition of the device characteristics. The imple-
mentation of a variety of diagnostic and device specific
algorithms needs to be done for the code described herein.
Perhaps the most difficult part of the code development
is the design of the interface to input the system that is to
be simulated and also in the design of the diagnostics that
inform the user of the progress and results of the simul-
ation. The design and use of an interactive graphical and
textual interface is expected to be necessary to achieve the
flexibility required. The use of standardized files from
computer-aided design packages is expected to help ease
the input burden and the use of a fast frame buffer inter-
faced to video output devices is expected to allow the dis-
play of a significant fraction of the information that can
be generated in the three-dimensional fully electromag-
netic simulation environment.

It now appears that the use of parallel computers can
make a contribution to the computational needs of elec-
ronic device design. In the short term, the implementation of the transverse field solver will complete the core of the physics component in the test bed code. The inclusion of spectral diagnostics and additional field diagnostics will then allow a more complete evaluation of the suitability of these parallel techniques.

ACKNOWLEDGMENT
The author acknowledges helpful discussions with Dr. R. H. Jackson.

REFERENCES

Ernest G. Zaidman was born in Tampa, FL, on November 15, 1954. He received the B.S. and M.A. degrees in physics from the University of South Florida in 1975 and 1979, respectively, and the Ph.D. degree in physics from the University of Texas at Austin in 1986.

He has been employed as a Research Scientist in the development of mapping and data interpolation software at Zycor, Inc. from 1980 to 1982. He is currently at the Naval Research Laboratory. His current field of interest is the development and application of computer simulation codes for the study of microwave devices.