Progress in Parallelizing XOOPIC

Peter J. Mardahl, John P. Verboncoeur
Cory Hall #173, EECS Department
University of California
Berkeley, CA 94720-1772, USA

Abstract

XOOPIC [1] (Object Oriented Particle in Cell code for X11-based Unix computers) is presently a serial 2d 3v particle-in-cell plasma simulation. This effort focuses on using parallel and distributed processing to optimize the simulation for large problems. The benefits include increased capacity for memory intensive problems, and improved performance for processor-intensive problems.

The MPI library enables the parallel version to be easily ported to massively parallel, SMP, and distributed computers. The philosophy employed here is to spatially decompose the system into computational regions separated by "virtual boundaries", objects which contain the local data and algorithms to perform the local field solve and particle communication between regions. This implementation reduces the impact of the parallel extension on the balance of the code.

Specific implementation details such as the hiding of communication latency behind local computation will also be discussed, as well as code features and capabilities.

1 GOALS FOR PARALLEL XOOPIC

XOOPIC has been successful as a single-processor code, and is able to simulate many interesting devices including relativistic klystron oscillators, electron guns, DC discharges with gas chemistry, plasma display panel cells, and highly relativistic beams in accelerators. However, particle-in-cell simulations are very computationally intensive, and on a single processor, some problems may take months to complete. The goals, therefore, for parallel XOOPIC are:

- Reduce run-times for large, complex simulations from weeks to days.
- Distribute memory demands across machines, allowing larger simulations than possible otherwise.
- Cross platform portability (networks of workstations, massively parallel machines, and SMP machines).
- Identical usage and feature set for parallel and non-parallel versions of XOOPIC, and largely shared source code.
- Complete source code availability to the general public.

2 PARALLELIZATION STRATEGY

The strategy for parallelization of XOOPIC is a coarse-grained spatial decomposition of the physical model into computational regions, as shown in Figure 1. Each computational region has its own mathematical mesh and particle arrays. A coarse-grained partitioning as shown has advantages over other partitioning strategies: in order to update the electromagnetic fields on the mesh points, it is necessary to know the fields on neighboring mesh points. If the neighboring mesh points are on other CPUs, communication is required between CPUs, and typically, communication is slower than computation on a parallel machine. In a block-cyclic decompositioning, for example, many more mesh points would have to have non-local data communicated in order to update fields, than would with a coarse-grained partitioning.

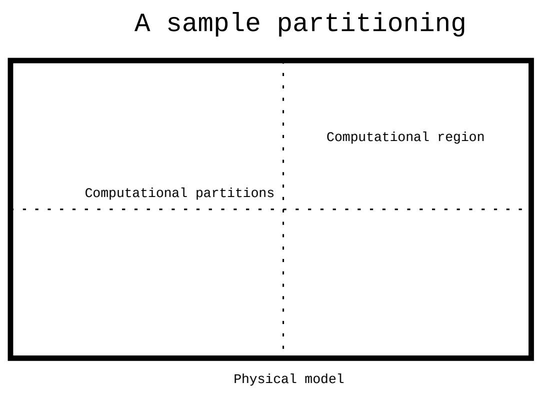

Figure 1: Example of partitioning a model into computational regions.

Another advantage of coarse-grained partitioning is re-use of code. Boundaries of the computational regions are treated as boundary conditions. Therefore, the special case of updating the fields on a computational boundary may be encapsulated in a new boundary condition called SpatialRegionBoundaries (SRBs), minimizing the number of modifications to the existing, already tested non-parallel version of XOOPIC.

Coarse-grained partitioning also allows ready identification of mesh points with only local dependencies: i.e., mesh points which require no data from remote CPUs to update. These "interior" mesh points may be updated while data from other CPUs is transmitted, allowing useful work to be done while messages are in transit. This is an important optimization to perform if parallel resources are to be used efficiently, and has been done in parallel XOOPIC, as shown in Figure 2.

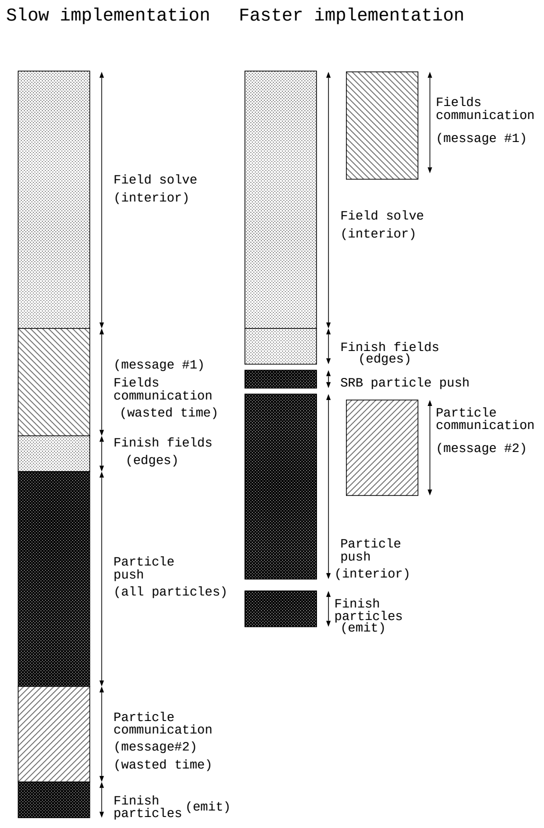

Figure 2: Hiding communication time behind local computation.

Particles also require field data in order to update their positions. Communicating field data every time a particle needed it would be a fatal mistake if performance were a consideration. However, particles only need field data from adjacent mesh points, so if particles are assigned to the same CPU on which the fields it needs reside, communication is not required to update their positions. The exception to this is when particles cross from one region to another: in this case, the particle data is communicated to the destination CPU. In XOOPIC, particles are distributed this way, so that the fields are nearly always local, but it is not possible to easily identify which particles will cross an SRB. This is unfortunate, because it makes another optimization difficult: if crossing particles could be identified early enough, they could be moved first, communicated to their destination, and while the data was in transit, particles needing only local fields could be updated (see *Faster implementation*, Fig. 2).

XOOPIC at present uses the *Faster implementation* for the fields, and the *Slower implementation* for the particles.

Coarse-grained partitioning strategies lend themselves to the use of parallel libraries such as MPI [2] (Message Passing Interface) or PVM (Parallel Virtual Machine). The MPI library in particular is widely portable and gives good performance, and is the library used for parallel XOOPIC. Other tools for parallel programming exist, such as split-C and High Performance Fortran, but these require special compilers which are often not freely available on platforms of interest. Use of the split-C and HPF compilers would also require extensive modifications to the XOOPIC code.

3 SPATIAL REGION BOUNDARIES

As remarked above, much of the work in parallelizing XOOPIC is encapsulated in a special boundary condition: the Spatial Region Boundary (SRB). Figure 3 shows the fields on the mesh near and on an SRB. SRBs are created in linked pairs, with each virtual boundary (a boundary defining a computational region, not a physical boundary) requiring two SRBs, one per computational region. Each SRB is responsible for sending and receiving the necessary fields and passing and receiving particles which cross them.

In order to compute the field components correctly on the SRB (the fields in the dotted box), the field components external to the region must be communicated to it (the fields lE1, lE2, lE3, lB1, lB3, and the currents, J1 and J3, which are in the same locations as E1 and E3 respectively).

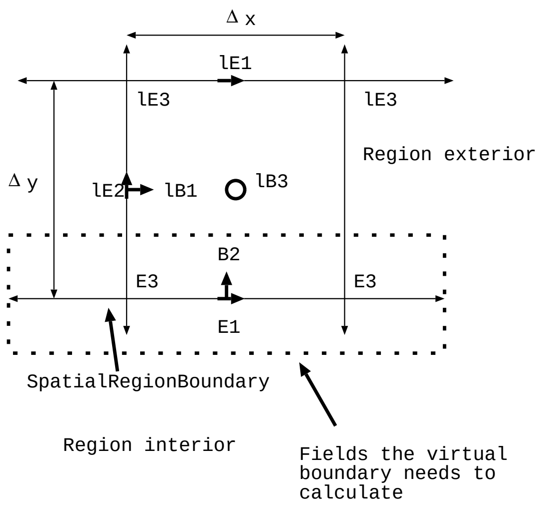

Figure 3: Fields on and near a horizontal Spatial Region Boundary.

The field solve on the boundary is actually a three-step process: lE1, lE2, and lE3 from the region exterior are used to update lB1 and lB3, which are stored in the SRB. E1 and E3 on the SRB are then computed using the updated ghost values of lB1 and lB3. B2 on the SRB is updated using the updated E3s on the SRB by the interior field solve. It is only necessary for the SRB to set E3 properly. These are the computations performed by the SRB shown in Figure 3:

$$lB1^{t+\frac{1}{2}} = lB1 - \frac{\Delta t}{2\Delta y}(lE3 - E3) \tag{1}$$

$$lB3^{t+\frac{1}{2}} = lB3 - \frac{\Delta t}{2}\left(\frac{E1 - lE1}{\Delta y} + \frac{lE2_{j+1} - lE2}{\Delta x}\right) \tag{2}$$

$$E1^{t+1} = E1 + \frac{\Delta t}{\epsilon}\left(\frac{lB3 - B3_{k-1}}{\mu \Delta y} - J1\right) \quad (3)$$

$$\begin{aligned} E3^{t+1} &= E3 + \frac{\Delta t}{\epsilon}\left(\frac{B1_{k-1} - lB1}{\mu \Delta y}\right. \\ &\quad \left. + \frac{B2 - B2_{j-1}}{\mu \Delta x} - J3\right) \end{aligned} \quad (4)$$

where $J1$ and $J3$ are current densities from particle motions, Δt is the simulation time step, and the subscripts $j-1$, $j+1$,$k-1$ are mesh point indices. The update of B is split into two phases, a half-update of B, an update of E, and another half-update of B: this scheme achieves 2nd order accuracy for the field solve, and makes E^t and B^t available for the particle update. This is the reason for the use of $\Delta t/2$ in the update of B. Note that the both of the paired SRBs are redundantly calculating the fields on the SRB itself, to avoid the necessity of having to communicate the result.

XOOPIC tracks the trajectories of particles, and when particles cross boundaries, they are removed from the simulation and given to the boundary for disposition. Conducting boundaries simply remove the particles, and dielectric boundaries may collect the charge of the particles on them. SRBs, when given particles, transfer them to the paired SRB on the CPU which is handling the adjacent computational region, and completes the particle update, so that the particle motion continues unperturbed through the SRB.

Figure 4 shows a flow diagram for the XOOPIC code which details when the SRBs send and receive messages. In step 1, XOOPIC updates all the particle positions. Particles which are crossing virtual boundaries are identified by the fact that they intersect an SRB. At the end of this stage, when all the particles have been moved, the SRB sends crossing particles to its counterpart SRB (msg #2). When this message arrives (the dashed arrow), the SRB places any particles sent to it into the simulation (step 2.) Also in step 2, boundaries which emit particles (such as secondaries, a thermionic cathode, or a beam of particles) place their particles in the simulation.

When step 2 is completed, the field solve may begin. The current density, J, is required for the field solve, and cannot be computed until all particle positions are updated: it is when the particle positions are updated that the current due to particle motion is calculated. The first step of the field solve (step 3) is for the SRBs to communicate fields to their counterparts. A message is sent (msg #1) which contains the necessary field components. Computation is immediately begun on updating fields interior to the computational region, which proceeds while msg #1 is in transit, since those fields do not depend on external data. If computation reaches the “Boundary” stage before msg #1 has arrived, execution will halt until it arrives. Whether time is wasted or not depends on the comparison of the time it takes to update all interior points (T) and the time it takes to transmit

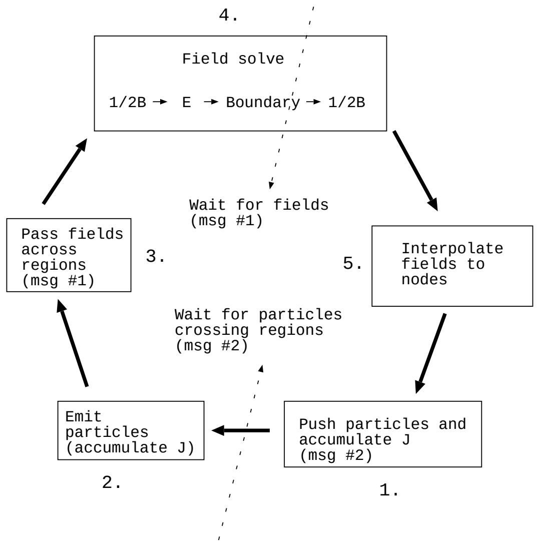
4.
Field solve
1/2B → E → Boundary → 1/2B
Wait for fields (msg #1)
Pass fields across regions (msg #1)
3.
5.
Interpolate fields to nodes
Wait for particles crossing regions (msg #2)
Emit particles (accumulate J)
Push particles and accumulate J (msg #2)
2.
1.

Figure 4: Flow diagram for parallel XOOPIC.

and receive the message (t). If $t > T$, time is wasted. T and t are both highly dependent on the specific CPU and parallel architecture, and on the ratio of boundary points to interior points. In general, performance is best when there are many more interior points than boundary points.

4 PRESENT STATUS OF PARALLEL XOOPIC

Parallel XOOPIC is presently available to the general public at http://ptsg.eecs.berkeley.edu/xoopic/xoopic.html. The parallel version has these additional features over the non-parallel version:

- Partitioning in either x or y direction. Partitioning in both simultaneously is implemented but not tested.
- Parallel electromagnetic field solve.
- Parallel particle push.
- Particle passing across virtual boundaries.
- Automatic partitioning of a given model.
- Diagnostics by computational region.

Parallel XOOPIC has been tested on a well-load-balanced case on several SMP machines, and on a single-processor DEC Alpha. Near-linear speedup or even super-linear speed up has been observed (Figure 5). In particular, 2 CPUs on a 4-CPU Pentium Pro machine performed more than twice as well as 1 CPU. This may be due to the larger amount of cache available. The two 8-processor runs exhibit the difference between the GNU g++ compiler and the proprietary Sun compiler on an Ultra Enterprise 5000. 940,000 particles were used in all tests, so the 8-CPU tests had roughly 120,000 particles per CPU.

The parallel version (as of XOOPIC 2.51) has these shortcomings:

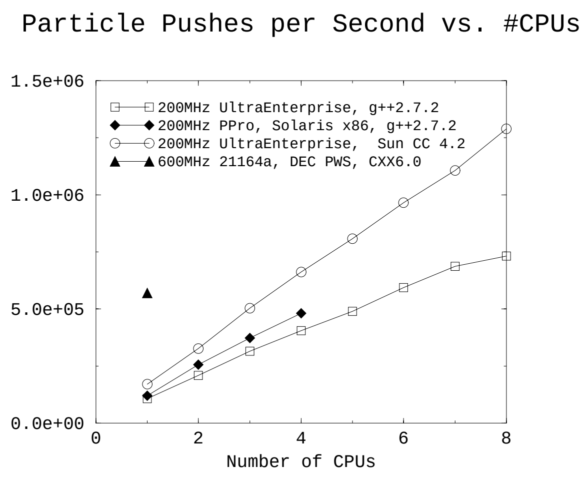

Figure 5: Parallel XOOPIC displays near-linear scaling with number of CPUs.

- No electrostatic models work.
- Dielectric triangle objects are not split correctly across regions.
- Beam emitter boundaries cannot be split by virtual boundaries.
- Particles see nearest-grid-point B3 near SRBs rather than linearly weighted B3.

5 FUTURE WORK

Efforts to extend parallel XOOPIC are ongoing. Areas being worked on include:

- Fixing identified bugs.
- Checkpointing of simulations to allow recovery of data should XOOPIC be interrupted.
- Improved diagnostics.
- A parallel Poisson solve, so that electrostatic models may be treated.
- Dynamic load balancing, for best use of parallel resources.
- Elimination of the need to wait for the particle update to complete before identifying which particles must be communicated.
- Extension of XOOPIC to 3-dimensional models.

Checkpointing of simulations is an important capability. Parallel machines have a statistically increased probability of hardware failures, simply because there are more components which may fail. Having the simulation preserve its state to disk may allow recovery of most computation: only the computation done since the last checkpoint would be lost.

Implementation of the parallel Poisson solve is under way, using the the PETSc library [3] with BlockSolve95 [4]. This will allow parallel electrostatic models as well as electromagnetic models. Completion of the parallel Poisson solve is anticipated by early 1999.

Elimination of the need for the particle update to complete and dynamic load balancing are performance optimizations which will extend the usefulness of parallel XOOPIC. Presently, scalability to many CPUs is limited: all computation must wait for the particles to be transferred between SRB pairs, and no dynamic load balancing is done. Efficient use of parallel resources thus requires some planning on the part of the user at present: the type of auto partitioning used can ensure good load balancing, and for some models, reasonable load balancing may not be possible. Dynamic load balancing will improve matters by distributing workload at runtime.

6 ACKNOWLEDGMENTS

This work has been funded by AFOSR F49620-94-1-0387, and AFOSR/MURI F49620-95-1-0253. We are also grateful to Jim Havranek (formerly at AFRL) and Gerry Sasser (AFRL) for their useful comments an suggestions.

7 REFERENCES

[1] J.P. Verboncoeur, A.B. Langdon, and N.T. Gladd, “An object-oriented electromagnetic PIC code.” Computer Physics Communications 87 (1995) 199-211.

[2] “The Message Passing Interface (MPI) Standard”, http://www.mcs.anl.gov/mpi/

[3] “The PETSc Library”, http://acts.nersc.gov/petsc/main.html

[4] “BlockSolve95”, http://www.mcs.anl.gov/mpi/