Performance of particle in cell methods on highly concurrent computational architectures

M.F.Adams¹ , S. Ethier² and N. Wichmann³

¹ Columbia University, APAM, 500 W. 120th St. Rm 200, MC 4701, New York, NY 10027, USA (ma2325@columbia.edu)

² Princeton Plasma Physics Laboratory

³ Cray Inc.

Abstract. Particle in cell (PIC) methods are effective in computing Vlasov-Poisson system of equations used in simulations of magnetic fusion plasmas. PIC methods use grid based computations, for solving Poisson's equation or more generally Maxwell's equations, as well as Monte-Carlo type methods to sample the Vlasov equation. The presence of two types of discretizations, deterministic field solves and Monte-Carlo methods for the Vlasov equation, pose challenges in understanding and optimizing performance on today large scale computers which require high levels of concurrency. These challenges arises from the need to optimize two very different types of processes and the interactions between them. Modern cache based high-end computers have very deep memory hierarchies and high degrees of concurrency which must be utilized effectively to achieve good performance. The effective use of these machines requires maximizing concurrency by eliminating serial or redundant work and minimizing global communication. A related issue is minimizing the memory traffic between levels of the memory hierarchy because performance is often limited by the bandwidths and latencies of the memory system. This paper discusses some of the performance issues, particularly in regard to parallelism, of PIC methods. The gyrokinetic toroidal code (GTC) is used for these studies and a new radial grid decomposition is presented and evaluated. Scaling of the code is demonstrated on ITER sized plasmas with up to 16K Cray XT3/4 cores.

1. Introduction

A plasma is a gas with a high enough temperature that electrons are striped from the atoms resulting in at least two species of particles (eg, electrons and deuterium ions). The Vlasov equation governs the behavior of each species in a plasma.

$$\frac{\partial f_\alpha}{\partial t} + \mathbf{v} \cdot \nabla f_\alpha + \frac{q_\alpha}{m_\alpha} (E + \mathbf{v} \times \mathbf{B}) \frac{\partial f_\alpha}{\partial \mathbf{v}} = \mathcal{C}_{\alpha,\beta}(f_\alpha), \quad (1)$$

where $f_\alpha \equiv f_\alpha(\mathbf{x}, \mathbf{v}, t)$ is the distribution function which represents the density of particles of species α found at any point in the domain of six-dimensional phase space $(\mathbf{x}, \mathbf{v})$; and $\mathcal{C}_{\alpha,\beta}(f)$ is the Fokker-Planck collision term between species α and β . The term $q(E + \mathbf{v} \times \mathbf{B})$ is the Lorentz force due to electric and magnetic fields. A collisionless plasma is described by this Vlasov equation, without the Fokker-Planck term, along with Maxwell's equations, or in our case simply Poisson's equation.

An effective method for analyzing the Vlasov-Poisson system of equations, used in simulating fully ionized plasmas, is the particle in cell (PIC) method. PIC methods evolve plasma dynamics

self-consistently by alternately *pushing* charged particles and *solving the fields* governed by Maxwell's equations. These methods enable the study of plasma microturbulence on global scales. This microturbulence (eg, scales of a few millimeters to centimeters or even larger) is important in understanding transport phenomenon in magnetically confined plasmas and is best understood with a model that includes the effect of gyro motion of charged particles in magnetic fields. The basic idea behind the gyrokinetic simulation method is to time-average rapid precessing motions, and only to push the guiding center motion for the particles [1]. The finite Larmor (gyro) radius effects also enter the system through the *gyrokinetic Poisson equation* [1, 2].

The gyrokinetic Poisson equation which with some simplification is of the form:

$$\frac{\tau}{\lambda_D^2} (\Phi - \tilde{\Phi}) = 4\pi e (\delta\bar{n}_i - \delta n_e) \quad (2)$$

where Φ is the electrostatic potential, $\delta\bar{n}_i$ and δn_e are the fluctuation part of the ion and the electron guiding center charge density, τ is the ratio of electron to ion temperature (assumed to be 1.0 herein), λ_D is the Debye length, and $\tilde{\Phi}$ is the second gyro-phase averaged potential [2]. With the adiabatic electron approximation ($\delta n_e = n_0 e \Phi / T_e$) and a small parameter expansion approximation a symmetric positive definite, diagonally dominate system of equations results and must be solved at each time step: $(1 - \rho_i^2 \nabla^2) \Phi = 4\pi e \delta\bar{n}_i$, where ρ_i is the ion gyroradius. A Padé approximation can also be used without significant change in the character of the equations resulting in the system: $(1 - 2\rho_s^2 \nabla^2) \Phi = 4\pi e (1 - \rho_i^2 \nabla^2) \delta\bar{n}_i$.

The gyrokinetic toroidal code (GTC) is an implementation of the gyrokinetic PIC method used for toroidal magnetic confined burning plasma devices [3]. GTC has been run effectively, with scaling up to 32K cores, on all of the recent high performance computational architectures [4]. The GTC programing model uses FORTRAN 90 and MPI, and uses PETSc for the potential solves in Poisson's equation, which is discretized with linear finite elements. Note, standard gyrokinetic ordering implies that Poisson's equation need only be solved on each *poloidal plane* (perpendicular to the toroidal direction in the torus), resulting in series of independent 2D grid linear solves. This paper presents recent performance results of GTC, with scaling to 16K Cray XT3/4 cores. This report proceeds by introducing the primary algorithmic elements of PIC codes in §2, followed by a discussion of the new radial decomposition of the mesh in §3 and finally numerical experiments on the Cray XT3/4 are presented in §4.

2. Gyrokinetic PIC algorithm

The basic PIC algorithm consists of depositing the charge from the particles onto the grid, computing and smoothing the potential, computing the electric field and pushing particles with the Lorentz force. Figure 1 shows a schematic of the PIC algorithm.

Performance is governed by three basic types of operations in this algorithm: 1) grids work (ie, Poisson solve), 2) particle processing (eg, position and velocity updates and 3) interpolation between the two (ie, charge deposition and field calculation in particle pushing). The dominant computational cost depends on the number of particles use; for the δf methods [5], used in GTC the grid work is a significant minority of the overall cost of the simulation, the cost being dominated by particle pushing. Earlier version of GTC parallelized the grid work with a shared memory model (Open MP) which is adequate if the grid is small enough and the size of the shared address space for each poloidal plane is large enough. As devices get larger, and finer grids are desired for some types of simulations, performance degrades with this model because more address spaces are required for each plane and the larger grid that needs to be stored puts more pressure on the cache and the entire memory system. This performance degradation is due to increased pressure on the cache in the charge deposition and redundant work required

```

( $\mathbf{x}, \mathbf{v}$ )  $\leftarrow$  Initialize           – Initialize (ion) particle position and velocity
time loop  $t = 1 : \delta t : T$ 
   $\delta \bar{n}_i \leftarrow \text{Charge}(\mathbf{x}, q)$  – Deposit ion charge ( $q$ ) on grid
   $\Phi \leftarrow (1 - \nabla_{\perp}^2)^{-1} \delta \bar{n}_i$  – Potential solve (small parameter expansion)
   $\Phi \leftarrow \text{Smooth}(\Phi)$          – Smooth potential
   $E \leftarrow \nabla \Phi$              – Compute electric field on grid
   $\mathbf{v} \leftarrow \mathbf{v} + \delta t \cdot qE/m$  – Update velocity (interpolate  $E$  from grid)
   $\mathbf{x} \leftarrow \mathbf{x} + \delta t \cdot \mathbf{v}$    – Update ion position
end loop

```

Figure 1. Schematic of the PIC algorithm

in the solve, field calculation and smoothing phases. The large fusion devices that now need to be modeled and the small amount of shared memory parallelism available on the newest large machines (ie, Cray XT4 and IBM Blue Gene have essentially no shared memory) requires a domain decomposition of the grid with MPI parallelism. This report discusses an MPI parallel decomposition method for the grids in GTC.

3. GTC mesh and decomposition

Figure 2 (left) shows a diagram of a typical magnetic fusion tokamak device. GTC simulates the core plasma; this is a toroidal domain with magnetic field lines with strong components in the toroidal direction but with some component in the poloidal plane resulting in a “twisting” magnetic field. GTC generates meshes for poloidal planes where Poisson’s equation is solved. Thus, charge deposition first interpolates a particles charge to the two planes on each side of the *toroidal domain* in which it is located. This charge on the poloidal plane is then interpolated onto the mesh points. Figure 2 (center) shows a sample set of poloidal mesh points on a poloidal plane. Figure 2 (right) shows a sample global grid with the field line following grid lines. These

Figure 2. Fusion tokamak design schematic (left), GTC discretization of toroidal domain (right)

planes form a natural 1D decomposition of the computational domain, which is the primary decomposition in GTC. The next section describes a new (radial) decomposition of the particles

and grid. This significantly complicates the parallel model in that the grid on the poloidal plane must be decomposed with explicit MPI parallelism.

3.1. Radial grid decomposition

The optimal grid decomposition for GTC is not obvious in that several efficiencies are impacted by the decomposition in different ways. Additionally, the optimal method may be more complex to implement than is necessary for acceptable performance in the range of device sizes (up to 10K radial grid cells) and the computers of interest in the next several years (say 640K cores and 128 poloidal planes, resulting in 4K processes per plane). The first thing to note is that the particles move primarily along the magnetic field lines and thus do not move much in the radial direction. Thus, a radial partitioning will result in minimal communication of particles within the poloidal plane. A fully 2D grid decomposition has the advantage that the same decomposition can be used for the Poisson solver – currently a separate unstructured 2D decomposition is used. Also a structured 2D decomposition is not as efficient for the solver as an unstructured one when large amounts of parallelism are required. Given these trade-offs and the relative simplicity in implementation we have opted for a structure radial grid decomposition (1D in the plane) for the grid/particle computations and a 2D unstructured decomposition for most of the grid computations (ie, the Poisson solver).

Our approach is to assume that the density of particles is a known function of radius (eg, a constant) and to compute a non-overlapping decomposition of the domain (ie, a geometric partitioning) that balances the particles exactly (given the assumed distribution). Figure 3 shows a small GTC grid of one plane (left) and a schematic of a geometric partitioning with four radial domains (processors or cores). Note, the GTC computational domain has an inner hole of radius “a0” and an outer radius “a1” in Figure 3. The local domain, that needs to be

Figure 3. Sample GTC mesh (left), schematic if non-overlapping geometric radial partitioning (right)

stored on each processor, must be extended to accommodate the charge deposition. The particle position stored in the gyrokinetic method is the guiding center of the particle – the gyrokinetic formulation models the gyro motion as a charged ring around particles guiding center. This charged ring is discretized with a few points (eg, four) on the ring; the charge at these points is deposited on the grid with linear or bilinear interpolation. A small but trivial optimization

is to extend each radial domain to line up with the radial grid points of the mesh before the grid extension for the gyro radius is computed. This enlarges the size of the domain that a particle can occupy on each processor. This (overlapped) radial decomposition defines the *valid* region in which a particle's guiding center must reside for a processor to be able to, in general, deposit the particle charge. That is, a particles guiding center must be in this region for it to be processed locally and must be sent to another processor if it strays outside of this region. Figure 4 (left) sketches the extension of the local domain to this extended valid region, that lines up with the grid (the grid is not shown). This valid region is then extended to accommodate the largest expected gyroradius of a particle for the charge deposition. This defines a partition of the grid that is stored on each processor. Figure 4 (right) shows a schematic diagram of this final domain extension as well as diagram of the four point gyro averaging technique used for the charge deposition.

Figure 4. Extension of geometric partition to grid aligned overlapped partitioning for processor P (left), further extension of processor grid to support charge deposition (right)

4. Numerical results

Data was collected on the 11,508 socket XT3/XT4 system at Oak Ridge National Laboratories (ORNL) called Jaguar. Each socket is a dual core AMD Opteron running at 2.6 Ghz. The main performance different between the XT3 and XT4 half is the XT4 has approximately 50% better main memory bandwidth. We collected data by running a benchmark problem from 64-4096 processor targeting either the XT3 or the XT4 exclusively, and from 64 to 16K processors allowing the scheduler to choose which processor type to use for any given PE. Finally, we ran the old version of GTC – without the radial domain decomposition of the mesh – up to 4K XT3 processors, which is the highest it can go because of memory constraints given the size of the reactors being simulated.

This benchmark problem uses 1mm grid spacing and the size of the device is scaled with the number of processors; the largest problem is an ITER sized plasma (ie, 2.2 meter minor radius). 20 particles per grid point were used and about 60,000 points were placed on each core. Table

1 shows the parameters for the scaling study.

Case	1	2	3	4	5
num. poloidal planes	64	64	64	64	64
num. θ (max)	625	1,250	2,500	5,000	10,000
num. radial points	100	200	400	800	1,600
major radius (m)	0.388	0.775	1.55	3.1	6.2
num. PEs	64	256	1,024	4,096	16,392

Table 1. Weak scaling benchmark GTC parameters

The minor radius is 0.32 times the major radius, the number of radial grid points and grid points in the θ direction are successively double. The plasma temperature is a constant at 2500 for all cases.

4.1. Weak scaling to 16K processor

Figure 5 shows the results of the weak scaling study running on 64 to 16K processors. The first

Figure 5. Weak Scaling to 16K processors

observation is that the new version of GTC is performing much better than the older version as the processor count, and thus the device size grows. This is primarily the result of the new decomposition scheme's and solver's ability to distribute the work associated with large grids. Scaling of the older version of GTC stopped at 4096 PEs because it could not run, due to

memory constraints, and ITER size device using 16K processors. Scaling to 4096 processors is good, but not as good as hoped for. Time approximately doubles while ideally it would stay flat. There does not appear to be much difference between the XT3 and the XT4, an early indication that the majority of the code is not sensitive to memory bandwidth. Moving to 16K PEs time almost doubles again. While a tremendous amount of science can be accomplished at this number of PEs, the performance degrades significantly.

4.2. Component Times

To get a better idea of where the time was being spent, we graphed the time spent in each of the major components at a function of the number of PEs. Figure 6 shows the *average* processor time spent in each component. Not surprisingly, the pusher (PUSHI) and charge routines dominate the time. What we do see is that the push routine is reporting large load imbalance.

Figure 6. Component Times

4.3. Load Imbalance

All of the communication for the particle pushing is done in SHIFTI, the routine that moves particles in the direction of the torus, or SHIFTI, the routine that moves particles radially. These shift routines contain soft synchronization points that could be collecting load imbalance from previous phases. To test whether the cause was the communication or load imbalance, we inserted a barrier just inside the shifter and then timed this barrier and the two shift components. The results can be seen in Figure 7.

The two shift routines remain quite flat across all PE counts. On the other hand, the barrier time is climbing dramatically as PE count increases. This indicates that the load imbalance is

Figure 7. Breakdown of time in shift

coming from the pusher. Figure 8 shows the time for each radial domain for eight planes of a test problems with 256 radial domains per poloidal plane. This data shows that indeed there is quite a bit of load imbalance in the pusher and that is very consistent across all planes. The cause of this is not clear in that the load balance (ie, number of particles and flops per plane in the pusher) is very good. To investigate this issue we look at more detailed performance data.

4.4. Instrument *PUSHI*

With scaling limited by a load balance problem in *PUSHI*, we collect information about that routine from hardware performance counters. To do that, we collected Opteron counter data using the Cray Pat performance tool. The first step was to simply plot the time spent in *PUSHI* as a function of PE number. Note, a smaller test problem, than that used in Figure 8, is used here and so the imbalance is smaller. One can see in Figure 9 (left) that there is about a 15-20 % difference in time depending on the PE number. Examination of the FLOP count and cache hit rate data showed that there was not sufficient variability in computational or memory work load to explain the variations in individual processor times. The number of translational lookaside buffer (TLB) misses, however, vary significantly. The TLB is used to map (translate) virtual

Figure 8. Push time vs. radial domain

Figure 9. Breakdown of time in shift (left), TLB misses on different PEs (right)

address to physical memory addresses and is similar to the memory cache in some respects (eg, there are many levels of TLBs and if an address is not found than an expensive “page fault” occurs).

Figure 9 (right) shows almost a factor of four difference in TLB misses depending on PE number, a variation that could potentially explain the variation in time. The question was if there was a correlation between TLB misses and PUSHI time. Figure 10 is a plot of PUSHI

Figure 10. PUSHI Time and a function of TLB misses

time and a function of TLB misses for every PE. One can see a very strong correlation. Any PE that took less than 100 million TLB misses ran in less than 290 seconds, and any PE that took more than 300 M TLB misses to more than 312 seconds. With this latest data, much of the focus of performance studies is trying to explain both the frequency of TLB misses and the distribution of TLB misses as a function of PEs. We do not currently have a good explanation for this data.

5. Conclusion

We have shown that by fully parallelizing the GTC code, via a radial mesh domain decomposition, we are able to significantly increase the performance of the code on large devices such as ITER. We have conducted device scaling studies, going up to high resolution ITER sized plasmas on 16K processors. We have identified some sources of parallel inefficiency that occurs at this scale, namely TLB miss induced load imbalance in the particle pushing phase and degradation in efficiency at 16K processors, which we are currently investigating.

6. Acknowledgments

This work is supported by the DOE SciDAC center for gyrokinetic particle simulations (GPSC) and in part by the DOE contract number DE-AC02-76CH03073.

References

- [1] W. W. LEE, *Gyrokinetic approach in particle simulation*, Phys. Fluids, (1983), pp. 556–562.
- [2] W. W. LEE, *Gyrokinetic particle simulation-model*, J. Comput. Phys., 72 (1987), pp. 243–269.
- [3] Z. LIN, T.S. HAHM, W. W. LEE, W.M.TANG AND R.B. WHITE, *Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations*, Science, 281 (1998), pp. 1835–1837.
- [4] L. OLIKER, A. CANNING, J. CARTER, C. IANCU, M. LIJEWSKI, S. KAMIL, J. SHALF, H. SHAN, E. STROHMAIER, S. ETHIER AND T. GOODALE, *Scientific Application Performance on Candidate Petascale Platforms*, In Proceedings IPDPS 2007, Long Beach, CA (2007) (Best Paper Award)
- [5] S.E. PARKER AND W.W. LEE, *A fully nonlinear characteristic method for gyrokinetic simulation*, Phys. Fluids B (1993), pp. 77–86.