Performance Study of the 3D Particle-in-Cell Code GTC on the CRAY X1 and Earth Simulator Computers

Stéphane Ethier, *Princeton Plasma Physics Laboratory, Princeton, NJ*

Abstract

The 3D particle-in-cell (PIC) code GTC, developed to study plasma turbulence in magnetic confinement fusion devices, has been ported to both the Earth Simulator and the CRAY X1 parallel vector computers. The gather and scatter operations, which are inherent to the PIC algorithm, turn out to be a challenge for both platforms. The most time consuming routines were streamed and vectorized, including the scatter operation, which required the use of the work-vector method. Impressive performance was achieved, although the memory footprint of the vectorized code is significantly larger than that of the super-scalar version. One of the tests ran on 64 processors on the Earth Simulator and the CRAY X1 is shown to be over 20% faster than a 1,024-processor run on the IBM SP Power3.

KEYWORDS: particle-in-cell, PIC, X1, Earth Simulator, gather-scatter, vectorization.

1 Introduction

The Gyrokinetic Toroidal Code (GTC) is a 3D particle-in-cell (PIC) code developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic confinement fusion [1, 2]. Turbulence is believed to be the main mechanism by which energy and particles are transported away from the hot plasma core in fusion experiments with magnetic toroidal devices. An in-depth understanding of this process is of utmost importance for the design of future experiments since their performance and operation costs are directly linked to energy losses.

The GTC simulations are computationally intensive, using up to 80,000 processor-hours for certain runs on the IBM SP Power3 at NERSC (80 hours on 1,024 processors). However, only about 10% of the peak performance of the Power3 processor is delivered to GTC during the run, and it gets worse with the newer Power4 processor where only 5% is delivered on average. This has been an increasingly frustrating trend among modern superscalar computers [10]. The new vector machines, such as the CRAY X1 and the Earth Simulator, have been designed to address this problem of ever decreasing ratio of sustained performance over peak performance. In this study, the GTC code was ported to both the Earth Simulator and the CRAY X1 in order to assess its performance and efficiency compared to current superscalar machines for which GTC was designed. The porting optimization details on the vector machines are described below, along with performance results. While the highest performances were achieved on the vector computers, GTC revealed to be quite a challenge even for them.

2 The GTC code

GTC is FORTRAN 95 code that solves the non-linear gyrokinetic equations [3, 4] in toroidal geometry for a system of charged particles in a self-consistent electrostatic field, and an externally imposed magnetic field. The gyrokinetic equations are essentially the Vlasov-Maxwell system of equations for which the fast, circular motion of the charged particles around the magnetic field lines has been integrated out. This gyrating motion is still taken into account but its effect is averaged over the circular path of the plasma particles. This allows for a longer time step since the fast motion around the field lines is not resolved. GTC makes use of a δf scheme [5] to reduce the statistical fluctuations associated with the initial sampling of the equilibrium distribution function. In this approach, only the perturbed part of the distribution function is evolved, while the initial equilibrium part is fixed.

In the version of the code presented here, the electrons are not treated separately but follow the ions adiabatically. By using the PIC method, the non-linear partial differential equation describing the motion of the particles in the system becomes a simple set of ordinary differential equations that can be easily solved in the Lagrangian coordinates [6, 7]. To further improve the efficiency and numerical accuracy, all calculations are done in the field-line following coordinates, so the mesh itself follows the magnetic field lines. The self-consistent electrostatic field driving this motion could be conceptually calculated directly from the distance between each pair of particles using N^2 calculations, but this method quickly becomes computationally prohibitive as the number of particles increases. The PIC approach drastically reduces the computational complexity to N by using a grid on which the particles deposit their charge to a limited number of neighboring points according to their range of influence. The electrostatic potential is then solved everywhere on the grid using the Poisson equation, forces are gathered back to each particle, and then used to move those particles along the characteristics (gather-push step). The charge deposition step is complicated by the fact that we are retaining the effect of the particles gyration around the magnetic field lines. In GTC, a particle actually represents a charged ring for which we follow the center of its circular motion. The radius of this motion varies with the velocity of the particle and the magnetic field strength. Charge deposition is done using the four-point average method [4]. We pick four points on the particle's circular orbit and distribute a fraction of its charge to the neighboring grid points (see figure). The Poisson equation is also solved using a four-point average method [8].

The GTC code contains two levels of parallelism: a 1D coarse-grain domain decomposition using message passing interface (MPI) constructs, and a fine-grain loop- level parallelization controlled with shared memory OpenMP directives. While the MPI part scales linearly, the loop-level parallelism efficiency depends on the amount of work in the parallel loops. This efficiency can go up to 98% in the case of large problem sizes. In this work, only the MPI model was used. The limited stay at the Earth Simulator Center did not allow enough time to study the OpenMP model. Preliminary tests revealed a significant increase in memory usage when using the OpenMP. Also, the CRAY X1 already implements streaming at the MSP level, which competes directly with OpenMP loop-level parallelism. However, it is possible the use OpenMP to further split the work between all the MSPs or SSPs on each X1 node but this has not been explored yet.

The most computationally intensive parts of GTC are the charge deposition and gather- push steps. Both involve large loops over the number of particles, which can reach several million per domain partition. When running with MPI only on superscalar architecture, the percentage of time spent in these two sections is almost equally split and totals about 80%. The time spent in the charge deposition step is usually a percent or two higher than the gather-push step. The remaining 20% is split between solving the Poisson equation (7%), doing MPI communication, field evaluation, smoothing, and diagnostics. Communication time is usually between 3 to 9% depending on the type of interconnect and the speed and number of processors used for the run.

3 Porting GTC to the vector computers

3.1 The Earth Simulator and SX-6

The GTC vectorization work started on the single-node SX-6 at ARSC [9] as part of an LBNL project on comparing modern vector computers to current superscalar architecture [10]. Since both the SX-6 and the Earth Simulator (ES) share the same compiler and operating system, most of the porting work for the ES could be done on the single-node machine. The SX-6 has a good and mature FORTRAN compiler with a very useful loopmark listing that helps identify the vectorized and unvectorized parts of the code. The initial porting was fairly easy although some special compiler options were needed to make the code run (e.g. "-Wf'-pvctl vwork=stack"). Initial performance was very low since a very few loops were vectorized. By profiling the code with the "-ftrace" compiler option, it became clear that the charge deposition routine was again the most time consuming routine.

While being the basis for the success and power of the PIC method, the grid-based charge deposition operation is also the source of the PIC codes limited processor efficiency observed on all superscalar and vector computers. Randomly localized particles deposit their charge on the grid, causing a low cache reuse on cache-based architecture since two successive stores to the grid array are usually in different cache lines. This effect is even more damaging on vector systems since two or more particles can contribute to the change at the same grid point, creating a memory dependency that prevents vectorization. Fortunately, several methods have been developed to address this issue during the past two decades. Our approach uses the work-vector method [11], where a temporary copy of the grid array is given an extra dimension corresponding to the vector length.

Each vector operation, acting on a given data set in the register, writes to a different memory address, entirely avoiding memory dependencies. After the main loop, the results accumulated in the work-vector array are gathered to the final grid array. The only drawback of this method is the increase in memory footprint. The final version, including other temporary arrays created by the compiler, has a memory usage 2 to 8 times higher than the superscalar version of the code. However, full vectorization of the most important loops in this routine was achieved and performance increased dramatically. Other methods address this memory dependency problem by sorting the particles [12, 13], increasing the amount of computation instead of memory, and resulting in a longer time to solution compared to the work-vector method.

Other improvements to the charge deposition routine consisted of eliminating some memory bank conflicts caused by an access concentration to some particular addresses. This arises because the same elements of a few small 1D arrays are accessed repeatedly inside the same loop. This can be viewed as good cache optimization on a cache-based architecture, but has the opposite effect on the cache-less memory architecture of the SX- 6. A memory bank has a finite busy time and bank conflicts arise when a bank is accessed before the busy time from the last access is over. Doing this repeatedly leads to poor memory performance. Use of the ”duplicate” compiler directive alleviated this problem by instructing the compiler to create multiple copies of chosen 1D arrays across several memory banks. This way, the same array element can be read from different memory banks, effectively hiding the latency of the access time. This method significantly reduced the bank conflicts in the charge deposition routine and increased its performance by 37%. However, it further increased the memory footprint of the code.

Optimization changes to other subroutines were not nearly as extensive as what was done in the charge deposition routine. For example, adding a single compiler directive along with inverting the dimensions of a few arrays was sufficient to achieve 99.4% of vector operation ratio with a nearly perfect average vector length of 255.9 in the gather-push routine. During the limited time spent at the Earth Simulator Center, the last version of the code achieved an overall vector operation ratio of 98% with an average vector length of 241 on the most efficient tests.

3.2 The CRAY X1

As a start, the Earth Simulator version of GTC was put on the CRAY-X1 and compiled without changes. The compilation in MSP mode was straightforward and the default options did not make the code crash as in the SX-6 case. The loopmark listing generated by the compiler was very informative. The use of the work-vector method in the charge deposition routine was just as necessary on the X1 as on the ES to achieve full vectorization of the scatter loop (charge deposition). In MSP mode, this loop also streams nicely, although a few directives had to be inserted to allow it. The dimensions of the work-vector array for the scatter loop are the same on both machines despite the fact that the vector length on the X1 is 64 compared to 256 for the ES. The scatter loop is being split in 4 streams and each individual stream is being vectorized, so we need 4x64 copies of the grid array to avoid all possible memory dependencies in the loop.

After discovering that the FORTRAN intrinsic function ”modulo” was preventing the vectorization of an important loop in the gather-push routine, it was replaced by an equivalent but vectorizable statement using ”mod”. By adding a compiler directive to prevent the vectorization of a short inner loop, the gather-push routine achieved the highest performance of all the subroutines in the code, which was also the case on the Earth Simulator. However, the most time consuming routine on the X1 became the ”shift” subroutine, which was unexpected since this routine had a relatively low percentage (11%) of run time on the ES for the same test case. It was now using more than 54% of the time on the X1. This subroutine verifies the coordinates of newly moved particles to determine whether they have crossed a subdomain boundary and therefore require migration to another processor. The main loop over the particles in this routine contained nested ”if” statements that prevented vectorization of this loop on both the ES and the X1. However, the non-vectorized shift routine accounted for significantly more overhead on the X1 than on the ES. Even though both architectures have the same relative scalar to vector peak performance ratio (1/8), scalar loops can incur an even larger penalty on the X1. In the case of a scalar loop that cannot be streamed, only one of the four SSP scalar processors within an MSP does useful work, resulting in an effective scalar to vector ratio of (1/32). By converting the nested ”if” statements into two successive condition blocks easily recognizable by the compiler, the main loop now streamed and vectorized, decreasing the percentage of time spent in the routine from 54% to only 4%. This change has not been implemented on the Earth Simulator but it certainly would have a positive effect as well, although not as dramatic as for the X1.

At this point, the best test case on the X1 has a 99.7% vector operation ratio with an average vector length (AVL) of 62.4 (perfect AVL is 64).

4 Performance results

Table 1 shows the performance numbers of 2 GTC test cases ran on 32 and 64 processors. The first case uses 10 particles per grid cell, which is the normal value for production runs, and the second case uses 100 particles per grid cell, a much higher resolution that improves the overall statistics of the simulation but can be prohibitive on super-scalar architectures. The grid dimensions are the same for all cases. The same tests were ran on 4 different platforms: the IBM SP Power3 at NERSC, which is the platform currently used for the GTC production runs, the SGI Altix at ORNL, the Earth Simulator computer in Yokohama, Japan, and the CRAY-X1 at ORNL. The performance numbers shown in table 1 are based on the Earth Simulator results. Since it is difficult to get consistent values for the number of floating point operation per second between vector and scalar machines, times to solution are used instead. The time to solution, or wall clock time, is a better indicator of the overall performance of the computer since it includes all the overheads due to communication latency, maximum bandwidth, system work, etc. However, we express the wall clock time ratios in terms of a valid baseline flop count since it adds another level of performance information.

As the table show, GTC runs 4 to 11% faster on the CRAY-X1 than on the Earth Simulator, although there is a case for which the Earth Simulator has a 4% lead. As noted above, the version of the code that ran on the Earth Simulator did not include the vectorized ”shift” subroutine, which gives an advantage to the X1. The Mflops numbers given by the performance measuring tool on the X1 are actually higher than those shown in the table. The reason for this is that most tools measure flops per second in terms of cpu time instead of wall clock time. It is known that the MPI implementation on the X1 has a larger overhead than the one on the ES, and this accounts, in large part, for the lower efficiency. GTC would benefit from using lower overhead communication software such as Co-array FORTRAN, which is known to have a much lower latency on the X1. In terms of cpu time, the best efficiency achieved by GTC on the Earth Simulator was 1344 Mflops/sec per processor, or 17% of theoretical peak performance. On the X1, this number was 1871 Mflops/sec per processor, or 15% of peak if we take 12800 Mflops/sec as the maximum performance of an MSP. However, GTC runs in single precision (4-byte floats) and the theoretical peak for an MSP running a single precision code is twice that of a double precision code, although this cannot be achieved since the memory bandwidth is not sufficient. The gather/scatter operations in GTC make it even harder for the compiler to take advantage of this extra capability.

Both the ES and the X1 are from 6 to 11 times faster than the IBM Power 3 when running the same GTC tests. The 100-particles per cell case was also run on the Power3 in hybrid mode MPI/OpenMP with a total of 1,024 processors. This is how GTC is run for large production simulations. Even there, the 64-processor ES and X1 runs were still about 20% faster for the chosen test, showing the impressive performance of the vector processors. The Power 3 is a rather old processor by now so it is probably more appropriate to compare with the more recent Itanium 2 processor, which is coupled to a very fast and low latency NUMA-Link interconnect in the SGI Altix. Even there, GTC runs between 3.2 and 4.5 times faster on the ES and the X1 compared to the Altix,

As it stands at this point, GTC runs with the highest efficiency on the Earth Simulator computer although the CRAY-X1 has the shortest time to solution for most of the tests. In spite of that, the large amount of extra memory used by the vectorized version of the code on both these computers limits the problem size that can be run, especially on the Earth Simulator since it has only 16GB of memory per node (8 cpus). One solution would be to split the problem size even more by adding another dimension of domain decomposition to the code, but this is a non-trivial task that requires major code modifications. Table 1 shows that by going from 32 to 64 processors when keeping the problem size fixed, the performance on the Earth Simulator decreases by 7% in one case and 13% in the other. These numbers are even larger for the X1. The lower performance is mainly due to smaller loop sizes rather than poor MPI scaling. The super-scalar machines are not nearly as sensitive to this problem size splitting since they show a fairly constant performance in all the tests. Moreover, the hybrid GTC model is not yet functional on the ES and the X1 computers

Table 1: GTC per processor performance for 32 and 64-processor runs at 2 resolutions: 10 and 100 particles per cell. The Mflops/sec/cpu for the Earth Simulator serve as the reference for all other platforms. The numbers are actually indicative of the ratios of wall clock times, which include all overheads due to communication, system times,etc.

Part	#procs	Power3	Altix	Earth Simulator		CRAY X1	
cell		MF/s/cpu	MF/s/cpu	MF/s/cpu	AVL	MF/s/cpu	AVL
10	32	135	290	961	209.87	1000	58.63
10	64	132	257	835	184.18	803	56.46
100	32	135	333	1344	240.73	1496	62.38
100	64	133	308	1245	228.48	1359	61.72

but works fine on the IBM Power 3.

5 Conclusion

The 3D particle-in-cell code GTC was ported to both the Earth Simulator and the CRAY-X1. Several modifications to the code were required in order to achieve a high percentage of vectorization (>98%). One critical change was to use of the work-vector method to eliminate memory dependencies in the scatter loop that forms the core of the charge deposition step in PIC codes. Once these modifications were in place, impressive GTC performance was attained on both the Earth Simulator and the CRAY-X1.

Acknowledgements

The author would like to thank Dr. Leonid Oliker of LBNL for the collaboration that lead to this work. The author would also like to thank the staff of the Earth Simulator Center, especially Dr. T. Sato, S. Kitawaki and Y. Tsuda, for their assistance during his visit; D. Parks and J. Snyder of NEC America for their help in porting applications to the ES. Also, many thanks to Dr. J. Schwarzmeier and N. Wichmann of CRAY for their help in porting GTC to the CRAY X1. This work was supported by US DOE Contract no. DE-AC020-76-CH03073 and in part by the DOE SciDAC Plasma Microturbulence Project.

References

[1] Z. Lin *et al.*, *Science* **281**, 1835 (1998).

[2] Z. Lin, S. Ethier, T. S. Hamh, and W. M. Tang, *Phys. Rev. Lett.***88**, 195004 (2002).

[3] W.W. Lee, *Phys. Fluids* **26**, 556 (1983).

[4] W.W. Lee, *J. Comp. Phys* **72**, 243 (1987).

[5] S.E. Parker and W. W. Lee, *Phys. Fluids B* **5**, 77 (1993).

[6] R.W. Hockney and J.W. Eastwood, *Computer Simulation Using Particles*, McGraw Hill (1981).

[7] C.K. Birdsall and A.B. Langdon, *Plasma Physics via Computer Simulation*, IOP Publishing (1991).

[8] Z.Lin and W.W. Lee, *Phys.Rev. E* **52**, 5646–5652 (1995).

[9] Arctic Region Supercomputing Center, http://www.arsc.edu

[10] L. Oliker et al., in Proceedings SC'03 (2003)

[11] A. Nishigushi, S. Orii, and T. Yabe, *J. Comp. Phys.* **61**, 519 (1985).

[12] E.J. Horowitz, *J. Comp. Phys.* **68**, 56-65 (1987).

[13] D.V. Anderson and D.E. Shumaker, *Comp. Phys. Comm.* **87**, 16-34 (1995).