Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Physics Communications 164 (2004) 456–458

Computer Physics Communications

www.elsevier.com/locate/cpc

Porting the 3D gyrokinetic particle-in-cell code GTC to the NEC SX-6 vector architecture: perspectives and challenges

S. Ethier [a,*], Z. Lin [b]

[a] *Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA*
[b] *Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA*

Available online 27 October 2004

Abstract

Several years of optimization on the cache-based super-scalar architecture has made it more difficult to port the current version of the 3D particle-in-cell code GTC to the NEC SX-6 vector architecture. This paper explains the initial work that has been done to port this code to the SX-6 computer and to optimize the most time consuming parts. After a few modifications, single-processor results show a performance increase of 5.2 compared to the IBM SP Power3 processor, and 2.7 compared to the Power4.

PACS: 02.70.Ns; 07.05.Tp

Keywords: Vector processor; Particle-in-cell; Code optimization

1. Introduction

The impressive performance achieved in 2002 by the Japanese Earth Simulator computer (26.58 Tflops, 64.9% of peak) [1] has revived the interest in vector processors. Having been the flagship of high performance computing for more than two decades, vector computers were gradually replaced, at least in the US, by much cheaper multi-processor super-scalar machines, such as the IBM SP and the SGI Origin series. Although the theoretical peak performance of the super-scalar processors rivals their vector counterparts, most codes cannot take advantage of this and can run only at a few percent of peak performance ($< 10\%$). When properly vectorized, the same codes can, however, reach over 30 or even 40% of peak performance on a vector processor. Not all codes can achieve such performance. The purpose of this study is to evaluate the work/reward ratio involved in vectorizing our particle-in-cell code on the latest parallel vector machines, such as the NEC SX-6, which is the building block of the very large Earth Simulator system in Japan [2]. This evaluation was carried out on a single node (8 CPUs) SX-6 located at the Arctic Region Supercomputing Center (ARSC) in Alaska (the SX-6 is distributed in the United States by CRAY Inc. under a special agreement with NEC). Early performance results are compared to the same tests performed on the IBM SP Power3 and Power4 machines, on which our particle-in-cell code, GTC, is normally run.

[*] Corresponding author.
E-mail address: ethier@pppl.gov (S. Ethier).

0010-4655/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2004.06.060

2. The Gyrokinetic Toroidal Code

The Gyrokinetic Toroidal Code (GTC) [3] was developed to study the dominant mechanism for energy transport in fusion devices, namely, plasma microturbulence. Being highly nonlinear, plasma turbulence is well described by particle codes for which all nonlinearities are naturally included. GTC solves the gyroaveraged Vlasov–Poisson system of equations (gyrokinetic equations [4]) using the particle-in-cell (PIC) approach. This method makes use of particles to sample the distribution function of the plasma system under study. The particles interact with each other only through a self-consistent field described on a grid such that no binary forces need to be calculated. This saves a great deal of computation, since it scales as N instead of N^2, where N is the number of particles. Also, the equations of motion to be solved for the particles are simple ordinary differential equations and are easily solved using a second order Runge–Kutta algorithm. The main tasks of the PIC method at each time step are as follows: The charge of each particle is distributed among its nearest grid points according to the current position of that particle; this is called the scatter operation. The Poisson equation is then solved on the grid to obtain the electrostatic potential at each point. The force acting on each particle is then calculated from the potential at the nearest grid points; this is the “gather” operation. Next, the particles are “moved” by using the equations of motion. These steps are repeated until the end of the simulation.

GTC has been highly optimized for cache-based super-scalar machines such as the IBM SP. The data structure and loop ordering have been arranged for maximum cache reuse, which is the most important method of achieving higher performance on this type of processor. In GTC, the main bottleneck is the charge deposition, or scatter operation, mentioned above, and this is also true for most particle codes. The classic scatter algorithm consists of a loop over the particles, finding the nearest grid points surrounding each particle position. A fraction of the particle’s charge is assigned to the grid points proportionally to their distance from the particle’s position. The charge fractions are accumulated in a grid array. The scatter algorithm in GTC is more complex since one is dealing with fast gyrating particles for which motion is described by charged rings being tracked by their guiding center [5]. This results in a larger number of operations, since several points are picked on the rings and each of them has its own neighboring grid points.

3. Porting to the SX-6

Initial porting of the GTC code to the SX-6 was straightforward. Without any modifications to the code, the initial single processor test run was only 19% faster than on the Power3 processor, which has a peak of 1.5 Gflops compared to the 8 Gflops SX-6 processor. The compiler on the SX-6 includes very useful analysis tools, allowing a quick identification of the code’s bottlenecks. Also, a source listing clearly indicates the loops that have not been vectorized and the reasons why. With these tools in hand, the scatter operation in the charge depositing routine was quickly identified as the most time consuming part of the code, and was not vectorized because of memory dependencies.

The challenge with the scatter operation in all PIC codes, and on all types of processor, is the continuous non-sequential writes to memory. The particle array is being accessed sequentially with a fixed stride, but each of its elements, representing the position of a particle, corresponds to a random location in the simulation volume. This signifies that the grid array accumulating the charges gets written to in a random fashion, resulting in a poor cache reuse on a cache-based super-scalar processor. This problem is well known [6], and it gets amplified on a vector processor, since many particles will end up depositing some charge on the same grid points, thus giving rise to a classic memory conflict that prevents vectorization. Fundamentally, the requirement for vectorization is that all the individual operations making up a single vector operation must be independent from each other. The required number of independent individual operations is equal to the number of elements in the vector registers, or “vector length”. The simplest method to avoid the intrinsic memory conflicts of the scatter operation, and achieve vectorization, is to have each element in the vector register write to its own temporary copy of the charge accumulating array. One needs as many copies as the number of elements in the vector register, which is 256 for the SX-6. When the loop is completed, the information in the 256 temporary arrays must be merged into the real charge array. The increase in memory generated by this method is of at least VLEN*N_G, where VLEN is the processor's vector length and N_G is the number of grid points in the domain. This can be a very large number considering that GTC uses tens of million of grid points for an average simulation. However, the increased speed gained by the vectorization of the scatter operation compensates for the use of the extra memory. This algorithm was included in the GTC code, leading to the vectorization of the two most compute-intensive loops. Further modifications, mainly adding compiler directives, helped achieve the vectorization of the second most compute intensive routine in the code.

4. Results and discussion

Table 1 summarizes the results of the single processor test that compares the overall performance on the Power3, Power4, and SX-6 processors. One notes from the results that the Power3 processor gives the highest efficiency at 12% of the maximum theoretical speed. The fact that the Power4 runs GTC only at half the efficiency of the Power3 agrees with the extensive benchmarks done by the CSM group at the Oak Ridge National Laboratory [7], and which showed that the memory bandwidth of the Power4 was not sufficient to keep up with its high-clocked CPU. With the code modifications made so far, the SX-6 runs the test case at 715.7 Mflops with 96.7% of vector operation ratio and an average vector length of 180.3 (the ideal being 256). Although this is only 9% of the maximum 8 Gflops that the vector processor can deliver, the SX-6 already runs 5.2 times faster than the Power3 processor and 2.7 times faster then the Power4.

More can be done to further improve the efficiency of the PIC method on the vector processor. The modifications that have been performed on GTC, so far, are just the beginning. A deeper analysis of the charge accumulating loop shows that even though the vector operation ratio is 99.89% with a perfect vector length of 256, the loop runs only at 7% of peak speed. However, the analysis also shows that a large number of scalar operations are performed in that same loop, most likely arising from the indirect array indexing characteristic of the random writes done in this algorithm. By simplifying the array indexing and sorting the particles according to their position, one might expect to achieve a much better performance on the SX-6. The goal is to reduce, as much as possible, the number of scalar operations, while maximizing the vector operations.

Thus far, the work/reward ratio of porting the GTC code to the SX-6 computer is good. We showed that a few small changes plus a more involved but straightforward method involving temporary arrays already increases the performance by a factor of 5. However, more code modifications need to be made in order to achieve the high efficiency obtained by other codes on the SX-6 computer. One also needs to study the parallel scaling of the code, which is a very important aspect since GTC usually runs on up to 1024 processors.

Table 1
Single processor performance of GTC test run on the IBM SP Power3 and Power4, and on the NEC SX-6 vector processor

Processor	Max speed (Gflops)	GTC test (Mflops)	Efficiency (real/max)	Relative speed (user time)
Power3	1.5	173.6	12%	1
Power4	5.2	304.5	6%	1.9
SX-6	8.0	715.7	9%	5.2

Acknowledgements

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] Shingu, et al., A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth simulator, in: SC'02 Conference, Baltimore, MD, Nov. 2002; http://sc-2002.org/paperpdfs/pap.pap331.pdf.
[2] http://www.es.jamstec.go.jp/esc/eng/.
[3] Z. Lin, S. Ethier, T.S. Hahm, W.M. Tang, Phys. Rev. Lett. 88 (2002) 195004.
[4] W.W. Lee, Phys. Fluids 26 (1983) 556.
[5] W.W. Lee, J. Comp. Phys. 72 (1987) 243.
[6] V.K. Decyk, S.R. Karmesin, A. de Boer, P.C. Liewer, Optimization of particle-in-cell codes on reduced instruction set computer processors, Comput. Phys. 3 (1996) 290.
[7] http://www.csm.ornl.gov/evaluation/CHEETAH/index.html.