Particle-in-cell simulation of collisional plasmas using multi-processor architectures

Miles M. Turner and Huw Leggate

School of Physical Sciences and National Centre for Plasma Science and Technology, Dublin City University, Dublin 9, Republic of Ireland

Abstract

The low-temperature plasma physics community often employs particle-in-cell simulation in conjunction with Monte Carlo collisions, typically to model the interaction between the plasma and a background neutral gas, although more general collisional interactions can be handled as well. The use of this attractive procedure is limited by the computational cost. This paper describes various methods for accelerating particle-in-cell simulations using commodity computer hardware.

Introduction

Particle-in-cell simulation is widely used for modelling low-temperature plasma phenomena [1]. The attraction of the method is that it entails no assumptions about the form of particle distribution functions, or the nature of their interactions with the electromagnetic fields. This is useful, because plasma discharges often feature exotically non-Maxwellian distribution functions, and violently non-equilibrium interactions between the charged particles and the fields. The purpose of the present paper is to explore the use of modern desktop computers for particle-in-cell simulation. Such computers typically feature multiple processors, and multiple cores within processors, combined with a hierarchical memory structure. This usually means that a relatively capacious main memory is shared by all the processors, while each processor or core has one or more local memories, with less capacity but greater speed. Moreover, most modern processors have some capability for so-called single instruction multiple data (SIMD) operations, a kind of vectorisation. The approach discussed in this paper has three elements: (1) A reorganistion of the simulation data structure, with the twin aims of reducing memory bandwidth usage and facilitating vectorisation (2) introducing SIMD instructions to speed processing of particles and (3) using OpenMP constructions to enable multiple cores to be used. The combination of these measures can accelerate the simulation by as much as a factor of thirteen, and by a factor of ten over a wide range of conditions, on a common hardware configuration with two processors and eight cores.

Data structure and vectorization

The natural data structure for a particle-in-cell simulation is a set of arrays, each storing one of the particle coordinates, with no correlation between the logical position of a particle in the array and the physical location of the same particle in the simulated spatial region. The computational cycle of a particle-in-cell simulation consists of data transfers from particle positions to spatial mesh points and back again. If the particles are processed in sequential order, this leads to a random pattern of access to the mesh point data. Not only will this produce much movement of mesh data through the memory hierarchy, but vectorisation is impeded by simultaneous accesses to the same mesh location. These issues are addressed by introducing a data structure that associates particles with cells. We can then process all the particles belonging to a given cell in a group, and the mesh data can be stored in registers, economising on both memory bandwidth and pointer arithmetic when loading mesh data. This approach also facilitates vectorisation. Although there are other possibilities [2, 3], the data structure we have adopted is an unrolled linked list, a hybrid approach in which an array of data is stored at each node of a linked list. Of course, particles move between cells, and work is required to maintain the data structure. This is equivalent to sorting the particles, but this is a tolerable computational burden. Fig. 1 compares an orthodox particle-in-cell implementation with the present approach, for a simple case with one real space and one velocity space dimension. For a reasonably large number of particles per cell ($N_C \gtrsim 100$), the improvement achieved is about 20 % for the new data structure only, and about 50 % when vectorisation is also used. The orthodox implementation cannot be vectorised, for reasons already remarked. Maintaining the sorted data structure consumes 30-40 % of the computation time, with the implication that better results can be expected for more elaborate models, such as those involving three velocity components and magnetic fields.

Parallelisation

OpenMP [4] offers a lightweight framework for introducing parallelism into programs designed for shared memory architectures. The approach discussed above lends itself readily to parallisation using OpenMP, which can be combined with MPI to exploit systems that combine shared and distributed memory. Results obtained in this way are shown in Figs. 2 and 3. Fig. 3 shows that there is a regime where a superlinear speedup is obtained. This occurs because the available cache memory is increased by using more cores.

Conclusions

The methods employed here combine to produce a speedup that is in excess of thirteen at best, and above ten over a useful range of problem sizes. More modern hardware, with better

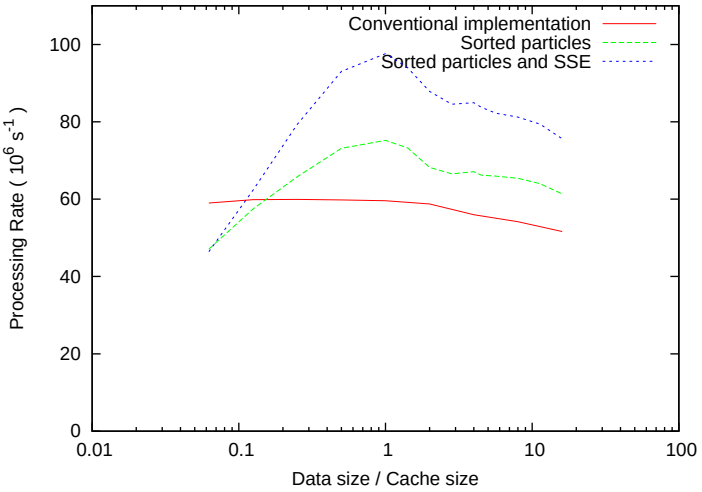

Figure 1: An orthodox particle-in-cell implementation with particle data stored in unsorted arrays compared with the present approach both with and without hardware vectorisation using SSE[5].

main memory bandwidth, would probably extend this range. Greater speedups are also likely to be obtained in conjunction with computationally more complex algorithms, because the book keeping will be a smaller proportion of the computational burden.

Acknowledgment

This work was supported by Science Foundation Ireland under grant 07/IN.1/I907 and by Association EURATOM-DCU.

References

[1] C. K. Birdsall. Particle-in-cell charged-particle simulations, plus Monte Carlo collisions with neutral atoms, PIC-MCC. *IEEE Trans. Plasma Sci.*, 19(2):65–85, April 1991.

[2] K. J. Bowers. Accelerating a particle-in-cell simulation using a hybrid counting sort. *J. Comp. Phys.*, 173(2):393–411, November 2001.

[3] D. Tskhakaya and R. Schneider. Optimisation of PIC codes by improved memory management. *J. Comp. Phys.*, 225(1):829–839, July 2007.

[4] The OpenMP API specification for parallel programming, 2008. URL `http://www.openmp.org`.

[5] Intel 64 and IA-32 architectures opimization reference manual, November 2009.

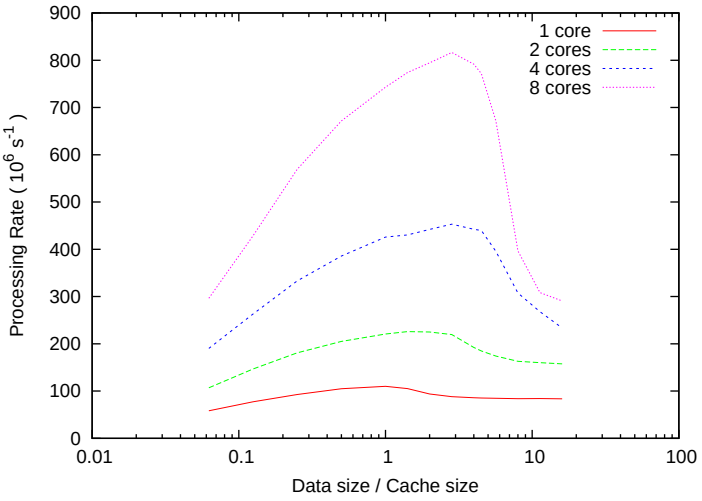

Figure 2: The present approach with parallelism using OpenMP, showing performance on a varying number of processor cores.

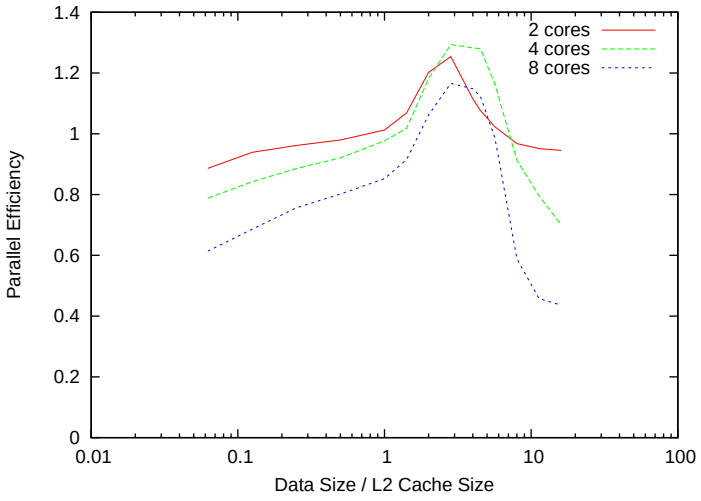

Figure 3: The same data as in Fig. 2, represented as parallel efficiency with respect to execution on a single core.