Computer Physics Communications 194 (2015) 18–32

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computational performance of a smoothed particle hydrodynamics simulation for shared-memory parallel computing

Daisuke Nishiura *, Mikito Furuichi, Hide Sakaguchi

Department of Mathematical Science and Advanced Technology, Japan Agency for Marine-Earth Science and Technology, Kanagawa 236-0001, Japan

ARTICLE INFO

Article history:
Received 14 November 2013
Received in revised form
11 March 2015
Accepted 9 April 2015
Available online 18 April 2015

Keywords:
SPH
Particle simulation
OpenMP
CUDA
MIC
GPU

ABSTRACT

The computational performance of a smoothed particle hydrodynamics (SPH) simulation is investigated for three types of current shared-memory parallel computer devices: many integrated core (MIC) processors, graphics processing units (GPUs), and multi-core CPUs. We are especially interested in efficient shared-memory allocation methods for each chipset, because the efficient data access patterns differ between compute unified device architecture (CUDA) programming for GPUs and OpenMP programming for MIC processors and multi-core CPUs. We first introduce several parallel implementation techniques for the SPH code, and then examine these on our target computer architectures to determine the most effective algorithms for each processor unit. In addition, we evaluate the effective computing performance and power efficiency of the SPH simulation on each architecture, as these are critical metrics for overall performance in a multi-device environment. In our benchmark test, the GPU is found to produce the best arithmetic performance as a standalone device unit, and gives the most efficient power consumption. The multi-core CPU obtains the most effective computing performance. The computational speed of the MIC processor on Xeon Phi approached that of two Xeon CPUs. This indicates that using MICs is an attractive choice for existing SPH codes on multi-core CPUs parallelized by OpenMP, as it gains computational acceleration without the need for significant changes to the source code.

1. Introduction

Large-scale parallel computing is important for numerically reproducing actual measurement results and dynamics of phenomena in various science and engineering areas, such as civil engineering [1], bioengineering [2], pharmaceuticals [3], and earth sciences [4,5]. The computational performance of parallelized software tools plays a critical role in such simulation studies, as these improve the computational accuracy relative to the simulation resolution within a limited computation time. Recent massively parallel computer systems based on shared- and distributed-memory architectures employ various types of arithmetic processors. Current processor designs are known to exhibit totally different computational performance depending on the numerical algorithms and implementation methods employed. Thus, it is important to investigate and compare different numerical algorithms and code tuning techniques for each type of processor.

Currently, parallel computing generally uses either a multi-core central processing unit (CPU), graphics processing unit (GPU), or many integrated core (MIC) processor. Multi-core CPUs have traditionally been used in high-performance computing, whereas GPUs were originally designed for computer graphics with many arithmetic cores [6]. Nowadays, the purpose of GPUs is more generalized, and they are used in many of the parallel computer systems on the TOP 500 list [7]. MIC is a new hardware accelerator used in processors such as Intel's Xeon Phi [8], which consists of up to 61 cores. The MIC architecture is used in the cluster systems of Tianhe-2 and Stampede, which were ranked first and sixth, respectively, in the 2013 TOP 500 list. These recent supercomputers employ distributed/shared hybrid memory systems designed for inter/inner-node hierarchically parallelized applications.

The common progress of current processor designs is the increase in the number of cores using vector operations such as single-instruction–multiple-data (SIMD). In such a situation, the shared-memory parallelization plays a basic but critical role in dealing with

* Corresponding author.
E-mail address: nishiura@jamstec.go.jp (D. Nishiura).

http://dx.doi.org/10.1016/j.cpc.2015.04.006

the increasing number of arithmetic cores in an efficient manner. However, parallel computing often cannot be performed efficiently on shared memory owing to memory-access conflicts, whereby parallel threads concurrently write to the same memory address. In addition, for multi-core processors, memory allocation must be carefully considered, because data locality significantly influences the memory access speed. For many-core processors such as GPUs, data alignment should also be appropriately implemented, as this affects the global memory access speed.

Numerical simulation methods used in science and engineering include the finite difference method (FDM), finite element method (FEM), finite volume method (FVM), boundary element method (BEM), and particle simulation method (PSM). Among these, PSM has a benefit of being mesh-free, allowing the computation of large-scale deformations and fractures of a continuum body without expensive remeshing tasks. As a PSM, smoothed particle hydrodynamics (SPH) is often used for a range of applications including wave breaking, tsunami simulations, etc. [9–12] because of its robustness in free-surface fluid dynamics. However, PSM programs must be implemented carefully to avoid write-access conflicts under shared-memory parallelization, especially when calculating a resultant force. In general, the inter-particle interaction force $\mathbf{F}_{ij}$ between particles i and j is calculated once per interacting pair, and, according to the action–reaction law, the calculated force is distributed between the two particles as $\mathbf{F}_{ij}$ and $\mathbf{F}_{ji}$ $(= -\mathbf{F}_{ij})$. Conflicts may arise on the shared memory when the interaction forces are distributed to each particle i and j in parallel, because different parallel threads may concurrently add the force to the same particle.

To address these issues, a number of parallel algorithms for shared memory have been developed [13–19]. One of the simplest methods is to use atomic instructions, which atomically access memory locations to parallelize the reduction operation by adding compiler directives to the program code. In general, however, such instructions are computationally expensive because of memory barriers. The use of private memory space on each thread is proposed for reducing the cost for such memory barriers [19], although this technique is useful only for a small number of threads. Another approach is to calculate the interaction twice [13–15], such that $\mathbf{F}_{ij}$ and $\mathbf{F}_{ji}$ are calculated separately in order to integrate the forces on particles i and j in parallel without using the action–reaction law. Although this method can avoid write-access conflicts, it requires double the arithmetic cost to evaluate the reaction force $\mathbf{F}_{ji}$. We have proposed parallel algorithms to avoid this problem [20]. Our algorithms use the action–reaction law to evaluate $\mathbf{F}_{ji}$ from $\mathbf{F}_{ij}$, and parallelize the interaction summation with a reference table to avoid memory access conflicts. Our method was found to show high computational performance on GPUs, but it is not clear whether our algorithms are also efficient on other current processors.

To suggest which processor and implementation is suitable for PSM, we investigate the parallel performance of an SPH program on various many- and multi-core platforms. First, we introduce several parallelization strategies for three major modules of SPH, namely neighbor particle pair list creation, interaction calculation, and updating particle information. The computation time of these modules is then examined to determine the best algorithm for each processor type: CPUs with/without SIMD instructions, MIC, and GPUs. Finally, we discuss the effective performance and power efficiency for the SPH simulation in high-performance computing.

2. Computational procedures for parallelized SPH simulations

2.1. Formulation of SPH simulation

SPH is a mesh-free simulation method that discretizes the field with explicitly tracked reference particles [21]. Each particle has a kernel function characterized by a spatial distance, called the "smoothing length". In this research, Wendland's function in three-dimensional space is used as the kernel function W_{ij} with smoothing length h:

$$W_{ij} = \frac{21}{16\pi h^3}\left(1 - \frac{|\mathbf{r}_{ij}|}{2h}\right)^4 \left(\frac{2|\mathbf{r}_{ij}|}{h} + 1\right) \quad 0 \le |\mathbf{r}_{ij}| \le 2h, \tag{1}$$

$$\nabla_i W_{ij} = \frac{105}{16\pi h^5}\left(\frac{|\mathbf{r}_{ij}|}{2h} - 1\right)^3 \mathbf{r}_{ij}, \tag{2}$$

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ is the relative position between particle i and particle j, with $\mathbf{r}_k$ denoting the position of particle k, and $\nabla_i W_{ij}$ is the gradient of the kernel function.

By discretizing the Navier–Stokes equations of fluid with the kernel function, the momentum equation and the continuity equation are given by

$$\frac{d\mathbf{v}_i}{dt} = -\sum_j m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right)\nabla_i W_{ij} + \sum_j m_j \left(\frac{\xi}{\rho_i \rho_j}\frac{4\mu_i\mu_j}{(\mu_i + \mu_j)}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{\mathbf{r}_{ij}^2 + \eta^2}\right)\nabla_i W_{ij} \tag{3}$$

and

$$\frac{d\rho_i}{dt} = \sum_j m_j \mathbf{v}_{ij} \nabla_i W_{ij} \tag{4}$$

respectively, where $\mathbf{v}_{ij}$ $(= \mathbf{v}_i - \mathbf{v}_j)$ is the relative velocity between particle i and particle j, and $\mathbf{v}_k$, P_k, ρ_k, μ_k, and m_k are the velocity, pressure, density, viscosity, and mass of the kth particle, respectively. η is a small parameter used for smoothing out the singularity at $\mathbf{r}_{ij} = 0$, which is set to $0.01h$. ξ is determined to be 4.96333 as per a calibration against the known exact transient solution of the Couette flow [22]. The local pressure in the first term on the right-hand side of Eq. (3) is given by the following equation of state, which is based on Tait's equation [23]:

$$P_i = \frac{c_0^2 \rho_0}{\gamma}\left[\left(\frac{\rho_i}{\rho_0}\right)^\gamma - 1\right] \tag{5}$$