The performance prediction and improvement of SPH with the interaction-list-sharing method on PEZY-SCs

Natsuki Hosono[1,2][0000−0002−6638−7223] and Mikito Furuichi[1]

[1] Japan Agency for Marine-Earth Science and Technology, 3173-25, Showa-machi, Kanazawa-ku, Yokohama, Kanagawa, 236-0001, Japan
[2] RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan

Abstract. The demands for the optimization of particle-based methods with short-range interaction forces such as those in smoothed particle hydrodynamics (SPH) is increasing, especially for many-core architectures. However, because particle-based methods require large amount of memory access, it is challenging to obtain high efficiency for low-byte/FLOP many-core architectures. Hence, an efficient technique, the so-called "multiwalk" method, was developed in an N-body gravitational field. The key of the multiwalk method is in sharing of the interaction lists with multiple particles to offer an efficient use of the cache memory in the double-loops operation for calculating the interactions and reducing the main memory access. However, such performance improvement is not clear for the problems with short-range interaction forces such as those in SPH. In this paper, we proposed a theoretical performance model to examine the tradeoff relations between the memory and the cost of floating point operations to optimise the SPH code. We also validated the model with the wall-clock time spent on the PEZY-SCs (SC1 and SC2).

Keywords: MIMD processors · smoothed particle hydrodynamics

1 Introduction

Scientific computing is a common technique to solve complex problems which are hard to carry out by laboratory experiments. One important example is that of numerical hydrodynamics for solving fluid-motion problems. The smoothed particle hydrodynamics (SPH) [1,2] is one of the most widely accepted particle-based method which has advantages for problems with large deformations of fluid surface.

One of the disadvantages of particle-based methods compared to mesh-based methods is the calculation cost. When N particles are introduced into a system, the construction cost of interaction lists for all particles would be $\mathcal{O}(N^2)$ which prevents us from performing large-scale and/or high-resolution numerical simulations. A common technique to speed up calculations is to use a many-core device which is a peripheral unit to CPUs that has a number of microprocessors. The use of many-core devices for SPH simulations poses an interesting challenge in the high-performance computing [3,4]. However, it is not trivial to obtain high efficiency of computation for particle-based methods with recent many-core devices, because a native implementation shows intensive memory access cost compared to that of floating point operations and easily suffers from the memory-bandwidth problem especially in many-core devices with low-byte/FLOP.

One of the clever techniques to address the above-mentioned problem is the "interaction-list-sharing" method that was first developed for the N-body gravitational field problem [5,6]. Let us consider a "group" of particles (i-particles) located close to each other such that they have very similar interaction lists. To calculate the force acting on the i-particles, we need to "sweep" their interaction lists. The simplest implementation of this step is to use a double-loop: the loop sweeping particles in the interaction list inside the loop for the number of i-particles in the group. During this process, the size of the data transfers from the main memory can be reduced, because the data on the cache memory can be reused multiple times for each i-particle in the group.

Because SPH is a particle-based method, the interaction-list-sharing method can be applied. However, gravitational force is a long-range force; any pairs of two particles interact with each other even if they are infinitely distant. On the other hand, SPH involves a short-range force: a particle interacts only with its surrounding particles. Consider N_i particles that share one "shared" interaction list. When we increase N_i, the number of main memory accesses decrease and the size of the shared interaction list increases. However, when we set a large N_i two particles can be assigned to the same group despite them having largely different interaction lists. Such a situation results in an increase of unnecessary computational cost.

These tradeoff relations indicate that there should be a "desirable" value of N_i that minimises the wall-clock time spent on a many-core device which would depend on the choice of the SPH kernel and the computing device we use. Hence, in this paper, we built a model to predict the optimal value of N_i for the efficient computation on the many-core devices. Our performance model depends on the bandwidth and FLOP/s of the device, and arithmetic intensity of the SPH kernel. To validate the proposed model, we performed a benchmark test utilising PEZY Computing's PEZY-SC, which won the 1st place in GREEN500 of November 2018.

This paper is organized as follows. In Section 2, we briefly introduce the SPH method and proposed performance-prediction model. In Section 3, we outline the overall procedure of the interaction-list-sharing method for SPH. In Section 4, we show the validated result. In Section 5, we conclude this study.

2 Methods

In the standard SPH method, the equation of motion is formulated as follows

$$\frac{\Delta \boldsymbol{v}_i}{\Delta t} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla_i W(|\boldsymbol{r}_j - \boldsymbol{r}_i|; h_{ij}), \tag{1}$$

where $\boldsymbol{v}$, t, m, p, ρ, $\boldsymbol{r}$ and h are the velocity, time, the mass, the pressure, the density, the position, and the smoothing length that determines the size of an SPH particle, respectively. The subscript i indicates the label for each particle. h_{ij} is the arithmetic mean of h between particle i and j. Hereafter, we only focus on the performance of Eq. (1) because its computational cost is dominant in the general SPH solver.

The function W is the so-called “kernel function”. Herein, we used the Wendland C^2 kernel:

$$W(r, h) = \frac{21}{2\pi} \left(1 - \frac{r}{Hh}\right)_+^4 \left(1 + 4\frac{r}{Hh}\right), \tag{2}$$

where $(\cdot)_+ := \max(\cdot, 0)$. Note that H is a parameter that determines the cutoff size of the kernel function.

Hereafter, a particle that receives moments from its surrounding particles is referred to as i-particle, whereas the particles that give moments to an i-particle are referred to as j-particle. To calculate Eq. (1), an i-particle must have its p_i, ρ_i, h_i and $\boldsymbol{r}_i$, whereas a j-particle must have its m_j, p_j, ρ_j, h_j and $\boldsymbol{r}_j$. Assuming double precision, the sizes of an i-particle (S_i) and a j-particle (S_j) were specified as $8 \times 6 = 48$ and $8 \times 7 = 56$ bytes, respectively. The number of arithmetic operations for one interaction (N_{arith}) is 70, assuming that division and square root are 8 times more expensive than the basic arithmetic operations.

For building the interaction lists, we employed a framework named `Framework for Developing Particle Simulator` (`FDPS` [7]). `FDPS` takes full responsibility for the construction of interaction lists in parallel. The parts that the users must take care of are the kernel code that works on a many-core device and the communication between the host and the device. `FDPS` uses the tree method to construct interaction lists, which allows us to reduce the computational cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_8 N)$ [5,8,9]. After the constructions of interaction lists, `FDPS` provides the array of i-particles and their interaction lists to the kernel code written by the user. In the interaction-list-sharing method, `FDPS` makes an aggregation of groups of i-particles and their interaction lists for sending to a single many-core device. Then, the many-core device calculates interactions for particles in several groups simultaneously.

To predict performance improvement using the interaction-list-sharing method, we propose the following theoretical cost model. Consider the double-loop operations to update i-particles with j-particles. A particle in a group of N_i particles shares potentially interacting N_j particles with other particles in the same group. The number N_j depends on N_i and the kernel function’s cutoff radius H. The cost for the loop operations (C) contains the memory cost to load N_j j-particles

(C_{load}), to store N_i i-particles (C_{store}), the arithmetic cost for $N_i \times N_j$ pairs of interactions (C_{arith}), and other costs (C_{others}) (e.g., the kernel launching time). Those are summarised as follows:

$$C = C_{\text{load}} + C_{\text{store}} + C_{\text{arith}} + C_{\text{others}}, \tag{3}$$

$$C_{\text{load}} = \frac{N_j S_j}{B}, \quad C_{\text{store}} = \frac{N_i S_i}{B}, \quad C_{\text{arith}} = \frac{N_i N_j N_{\text{arith}}}{F}, \tag{4}$$

where B and F are the memory-bandwidth and the FLOP/s of a device, respectively. Hence, the cost to update a single particle deviated from the result with $N_j = 0$ can be written as

$$\frac{C}{N_i} - \frac{S_i}{B} = \frac{1}{B}\left(\frac{N_j}{N_i} S_j + \frac{B}{F} N_j N_{\text{arith}}\right). \tag{5}$$

Here we assumed that C_{others} is negligible and can be excluded. This cost model shows that the increase of N_i results in the decrease of the memory access $\propto N_j/N_i$ and the increase of the arithmetic costs ($\propto N_j$) to update a single particle. Note that N_j is calculated by counting the number of j-particles inside the spheres centred at each N_i particle position with a radius Hh. Eq. (5) indicates that for a given particle-based method, the dependence of the cost to update one particle can be characterized by the inverse of FLOP/byte (B/F). The size of the parameter H is generally chosen between 2.1 and 3.1.

To validate our performance predictions, we tested two PEZY-SC devices, viz., PEZY-SC1 and PEZY-SC2 which are MIMD-type architectures. The bandwidth and FLOP/s for SC1 are $B = 150$ GB/s and $F = 1.50$ TFlops, whereas for SC2 they are $B = 102$ GB/s and $F = 4.09$ TFlops, respectively. Note that PEZY-SC1 does not have floating operation units for division in double precision. Thus, for the PEZY-SC1, the approximate technique are used for square root operations [10] with double precision. For division, we first calculate the division with single precision and then apply Newton-Rhapson method to convert the result to double precision.

3 Results

In this section, we show the results of the validation tests. We put 10^6 SPH particles in cubic lattice with periodic boundary condition so that all particles have the same number of neighbour particles. All calculations are performed with double precision.

Figure 1 shows the predicted wall-clock time by our model and its dependence on N_i. We showed two cases; one is the high $B/F = 0.5$ case (K computer) and another is the low $B/F = 0.025$ case (PEZY-SC2). With the cases of PEZY-SC2 with $H = 2.1$ and 3.1, the curves show a gradual decrease of the wall-clock time with the N_i increases until $N_i \simeq 32$ and then, the wall-clock time increase for $N_i > 64$. This result suggests that the interaction-list-sharing method with $N_i^{\text{opt}} \simeq 32$ is a reasonable choice, where N_i^{opt} is the optimal choice for N_i.

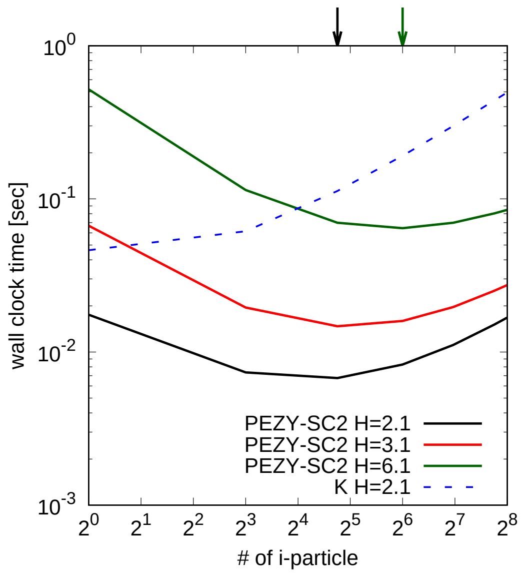

Fig. 1. Theoretical wall-clock time for PEZY-SC2 and K computer. The horizontal axis indicates N_i. The black arrow at the top of the figure indicates N_i to minimise predicted wall-clock time for the cases with $H = 2.1$ and 3.1. The green arrow also indicates optimal N_i, but for $H = 6.1$. The solid black, red, and green curves represent the predicted wall-clock time for the PEZY-SC2 for the cases with $H = 2.1, 3.1$, and 6.1, respectively. The dashed blue curve represents the predicted result with K computer and $H = 2.1$.

However, in the case of $H = 6.1$, the optimal choice becomes $N_i^{\mathrm{opt}} \simeq 64$. N_i^{opt} is found to increase with the kernel radius H (see two arrows in Fig. 1). The region $N_i < N_i^{\mathrm{opt}}$ indicates that the efficiency of the device is limited by the memory-bandwidth, whereas in the region $N_i^{\mathrm{opt}} < N_i$, the efficiency is limited by floating-point operations. Hence, N_i^{opt} increases as H increases.

Figure 1 also shows that the shape of the performance curve depends on a computer architecture characterized by B/F. For example, the blue dashed line shows the result with a high $B/F = 0.5$ architecture that predicts the performance in a K computer. The implementation of the interaction-list-sharing method does not result in a performance improvement because the predicted wall-clock time monotonically increases against N_i. In the case of $B/F = 0.5$, the wall-clock time is always limited by floating-point operations which means that the interaction-list-sharing technique is not efficient. Hence, N_i^{opt} would decrease as the B/F of a device increases.

The comparison between the predicted wall-clock time and the measured value are shown below. The results of the benchmark tests for PEZY-SC2 are shown in Fig. 2a. Our cost model characterized by B/F successfully captures

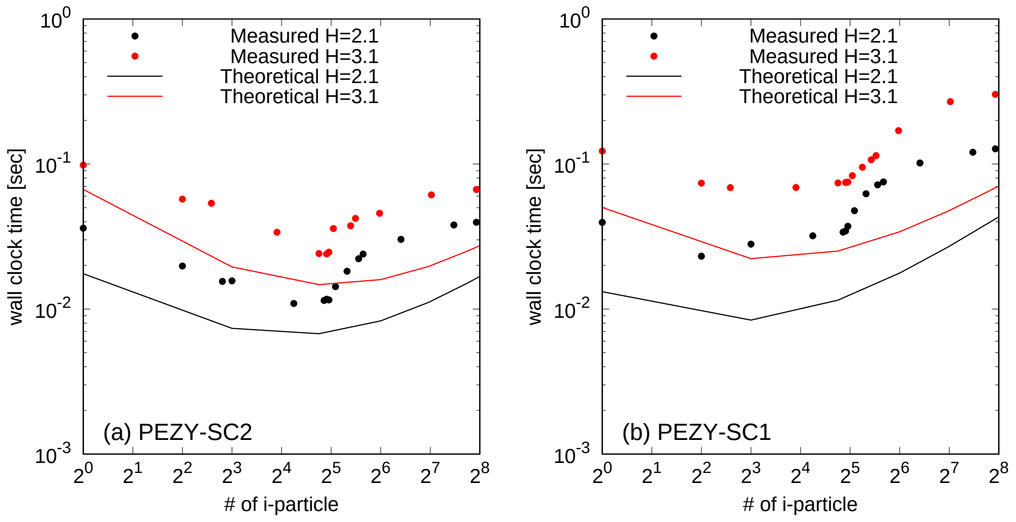

Fig. 2. The theoretical and measured wall-clock time for (a)PEZY-SC2 and (b)PEZY-SC1. The horizontal axis indicates the size of a group N_i. The solid black and red lines indicate the predicted wall-clock times for $H = 2.1$ and 3.1, respectively. The black and red circles indicate the measured wall-clock times for $H = 2.1$ and 3.1, respectively.

the trends of the predicted performance curve and predicts the improvement by using the interaction-list-sharing method. For example, the numerical experiment with $H = 2.1$ shows the 75% cost reduction when using $N_i = 32$ compared to $N_i = 1$, whereas our model predicts the 61% cost reduction with $N_i = 32$ compared to $N_i = 1$. Conversely, the observed cost in the numerical experiment is approximately 2 times larger than the theoretical cost due to approx. 50% efficiency of computation against theoretical peak performances.

Figure 2b shows a result of performance analysis with PEZY-SC1. The shape of the prediction curve is characterized by B/F; thus, our model predicts a smaller N_i^{opt} than that of PEZY-SC2. The measured wall-clock time shows that N_i^{opt} for PEZY-SC1 is 4, which is consistent with our prediction.

4 Conclusion

This paper examined the impact of the interaction-list-sharing method for SPH simulation. To predict the performance improvement quantitatively, a theoretical cost model was proposed. Through a series of benchmark tests, the proposed model was validated as a scaled performance model successfully suggesting the optimal interaction-list-sharing sizes for problems with a different interaction range H in various many-core architectures, including the MIMD-type architecture. While it appears not to be useful for high B/F machines, the interaction-list-sharing method can be a powerful optimization technique for accelerating particle simulations with short-range interactions especially for low B/F many-core architectures that are widely used in the modern devices.

In this paper, we focus on the optimization of the SPH method; however, the proposed approach to evaluate the performance improvement can be applied to other short-range methods. If we have a typical byte size of a particle, the number of arithmetic operations and the number of j-particles, then the optimal choice of N_i and the computational performance for a given many-core device can be predicted. Hence, we conclude that our prediction model can have a broad utility.

Acknowledgment

We used the super computer Shoubu system B at RIKEN. This work was supported by Post-K Issue 3 “System for Integrated Simulation of Earthquake and Tsunami Hazard and Disaster”.

References

1. R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977.
2. L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astrophysical Journal*, 82:1013–1024, December 1977.
3. D. Nishiura and H. Sakaguchi. Parallel-vector algorithms for particle simulations on shared-memory multiprocessors. *Journal of Computational Physics*, 230(5):1923 – 1938, 2011.
4. D. Nishiura, M. Furuichi, and H. Sakaguchi. Computational performance of a smoothed particle hydrodynamics simulation for shared-memory parallel computing. *Computer Physics Communications*, 194:18 – 32, 2015.
5. J. E. Barnes. A modified tree code: Don’t laugh; It runs. *Journal of Computational Physics*, 87:161–170, March 1990.
6. T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, Nov 2009.
7. M. Iwasawa, A. Tanikawa, N. Hosono, K. Nitadori, T. Muranushi, and J. Makino. Fdps: A novel framework for developing high-performance particle simulation codes for distributed-memory systems. In *Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, WOLFHPC ’15, pages 1:1–1:10, New York, NY, USA, 2015. ACM.
8. J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, December 1986.
9. L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *The Astrophysical Journal Supplement*, 70:419–446, June 1989.
10. D. Kushner. The wizardry of id [video games]. *IEEE Spectrum*, 39(8):42–47, Aug 2002.