Optimization and Parallelization of the cosmological N-body Simulation on Many-core Processor

GuiYing Zhang[1], Hao Yang[1,2,*] and ZhiNi Li[1]

[1] School of Computer Science,
Chengdu University of Information Technology, Chengdu, China
[2] School of Information and Software Engineering,
University of Electronic Science and Technology of China, Chengdu, China
*Corresponding Author: Hao Yang. Email: vhaoyang@gmail.com

Abstract. N-body numerical simulation can solve the formation and evolution of large-scale structure in space, and then help to study the essence of the most essential problems, including the distribution of matter in the universe, the formation of galaxies, dark matter and dark energy, etc.[1] Thus, the transplantation and application of the N-body simulation software on the Sunway TaihuLight platform are of great significance. In order to enhance the entire performance of the N-body simulation software photoNs 2.0, a variety of methods were utilized to achieve it, such as using Athread thread library to exploit the potential capability of the many-core node of sw26010 system, tuning compile options, rewriting the transcendental function with lookup table (LUT) and leveraging the vectorization optimization instruction provided by Sunway TaihuLight. In addition, we extended the application to run with a mode of host and slave working simultaneously. A detailed performance evaluation was performed and the results showed that the proposed parallelization method achieve good accelerated effect of 13x when evaluating up to 512 MPI processes on Sunway TaihuLight, and correspondingly the calculation of P2P kernel of fast multipole method was tremendously improved by the strategies mentioned in this paper.

Keywords: Fast Multipole Method; Particle Mesh Method; N-body Simulations; Sunway TaihuLight; Many-core Computing; Heterogeneous Computing

1 INTRODUCTION

N-body problem is one of the basic problems of celestial mechanics and general mechanics, it determined the motion of N particles which interact with a force, such as gravitation or electrostatics [19]. Astrophysical N-body simulations played a very significant role in the study of nonlinear structure formation in the universe. Such simulations are usually called as cosmological N-body simulations, which simulate each single particle's move according to the interaction of all the other particles in the simulative system [2].

The most straightforward algorithm of N-body simulation is direct summation, which calculate the acceleration of a particle according to the forces given by the rest of the particles in the system. Assuming that a simulative system contains *N* particles, while *N* is large ($N > 10^6$), this method appears to bring unaffordable $O(N^2)$ computations [1]. It boils down to that faster, better algorithms with some approximation are needed to alleviate the computation. One of the most famous algorithms is the Fast Multipole Method (FMM), which is derived from tree code and widely used in cosmological N-body simulations [22]. The basic idea of this method is organizing all the particles in the simulative system into a hierarchical tree structure. Other algorithms like particle mesh (PM) method is also widely used, which aim at reduce memory usage and improve the computational efficiency [3]. On the one hand, benefiting from the fast Fourier transform (FFT) libraries, such as FFTW3, the speed is quite high, on the other hand, the load-balancing problem is well-addressed because the distribution of particle is rather homogeneous. Some famous N-body simulation software like GADGET, Barnes-Hut simulation, ExaFMM, Puma-EM and some analogues like them are developed base on the FMM or PM algorithm. The photoNs 2.0 combined those two methods, it is, the PM algorithm handle the long-range gravitation, while the FMM algorithm deal with the short-range gravitation.

During the past decade, with the rapid development of supercomputers, methodologies and techniques to exploit higher performance of numerical simulations on state-of-the-art supercomputers is extraordinary needed [4]. Some kernel like N-body simulation is of great significance to port onto those supercomputers. This is especially true for the Sunway TaihuLight supercomputer, which was released on June, 2016 as the first supercomputer with over 100 PFLOPS peak performance. It uses the many-core SW26010 processor with a relatively high aggregated performance and a relatively low memory bandwidth [5]. Taking account of the fact that photoNs is a compute-intensive program, there existing a great potential to porting the program on the Sunway TaihuLight and to optimize it.

This paper introduces the methods to port photoNs 2.0 onto Sunway TaihuLight ,and illustrate the prime algorithms and several techniques we used, including the load balance of computation task, rewrite of the transcendental function, the decoupling between communication and computation, the parallelization of PM algorithm and FMM algorithm, and the vectorization of transcendental function. Following those methods, the result of performance enhancement of the many-core acceleration is also provided.

2 PLATFORM OVERVIEW AND PROBLEM DESCRIPTION

2.1 Platform Overview

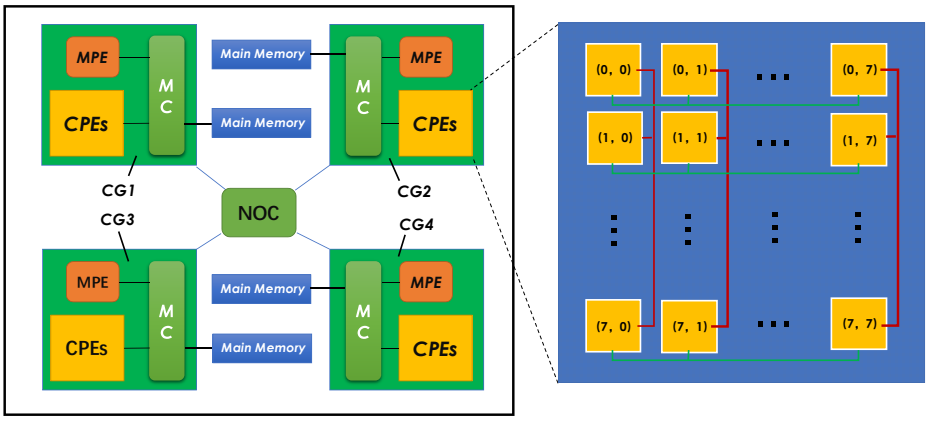


Fig. 1. The architecture of SW26010 processor

The Sunway TaihuLight is designed as a supercomputer with heterogeneous architecture base on SW26010 processor [6], which comprise of connected 40960 nodes, as show in Fig. 1, each node has one SW26010 processor. The processor consists of four "core groups" (CGs), one CG including a management processing element (MPE) and a computing processing element (CPE) cluster with 64 CPEs organized as an 8 by 8 mesh network. Both the MPE and CPEs are 64-bit RISC processors and have almost the same architecture [7]. Both MPE and CPEs have instruction caches. MPE has L1 and L2 data caches, while each CPE only has local data memory (LDM, 64 KB) and no cache memory [7]. Each CPE can still perform load/store operations to the main memory, and they can also issue DMA transfers between LDM and the main memory [7]. The need for explicit control of data movement between LDM and main memory makes the porting of the existing codes rather complicated. On the other hand, the possibility of explicit control makes performance tuning relatively straightforward [7].

Due to the fact that the operation system and user program both run in the MPE, we need use the OpenACC or Athread to utilize the CPEs, the former one is easily used by add compile directive in the code, while the latter one must be used elaboratively to design a new program or port an existed program onto the Sunway TaihuLight.

Both of MPE and CPEs are run with a clock speed of 1.45Ghz. Benefitting from the 256-bit wide vector registers provided by SW26010, the MPE and CPEs can perform four double-precise multiply-and-add operations simultaneously in one clock cycle. It means that the theoretical peak performance of one processor is 3.016 TFlops, thus, that of the entire machine with 40960 nodes is 123.5 PFlops. The bandwidth of 8GB DDR3 memory of each CG only reached 34GB/s, it is relatively lower than the

computation of processors. Thus, it is crucial to improve the ratio of computation to memory access [7].

2.2 Problem Description

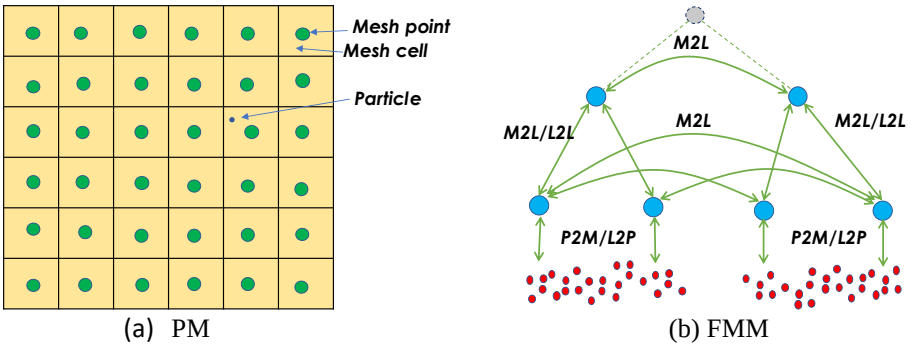


Fig. 2. PM method and FMM algorithm illustration. In PM method, the particles are portioned to each single box. And in FMM algorithm, particles are organized into the leaves of a tree.

The PM algorithm and FMM algorithm constitute the main body of photoNs. Fig.2 portray the main idea of those two methods. In this section, we will introduce those two algorithms.

The main idea of PM algorithm is to create a grid on the box, each grid cell has a grid point, solve the potential at the grid point, and then calculate the potential gradient to get the force at each grid cell [8]. For far field, the principle steps of the mesh calculation are as follows:

1. Assign "mass" to the mesh ("particle mass" becomes "grid density"),
2. Solve the field potential equation (e.g. Poisson's) on the mesh,
3. Calculate the force field from the mesh-defined potential,
4. Interpolate the force on the grid to find forces on the particles.
5. Now like the PP: integrate the forces to get particle positions and velocities.
6. Update the time counter.

For step 2, the PM method solves the Poisson equation:

$$\nabla^2(\varphi) = \pi G\rho$$

where G is Newton's constant and ρ is the density (number of particles at the mesh points). the gravitational potential φ is trivial to solve by using the FFT technique [8,9]. The transform is computed by multiplication with a suitable Green's function for the Poisson equation which contains a spatially-varying part of the mass density field. Periodic boundary conditions are usually used to simulate an "infinite universe" [8,9,10,11,18,21].

The Fast Multipole Method (FMM) is a tree code method that uses two representations of the potential field, the far field (multipole) and local expansions [14,15,20].

This FMM calculate the potential filed extremely fast. From basic physics that the force is just the negative of the gradient of the potential, it indicates that the force is a vector while the potential φ (x, y, z) is a scalar. It's computationally easier to deal with the potential than that of the force [12,13,14,15,16].

In this method, the particles are organized into a k-dimensional (k-d) tree, whose leaves are package of particles while each one of nodes store two multipoles M and L. The leaf accepting the effect from other leaves is called target leaf, and the leaf applying impact on other leaves is refer to as source leaf. Via this structure, the calculation of forces among those particles are convert into the interaction of those multipoles. There are six operators in FMM: particle to multiple (P2M), multipole to multipole (M2M), multipole to local (M2L), local to local (L2L), local to particle (L2P) and particle to particle (P2P). M2L and P2P are hotspots of computation among these operators.

3 PARALLEL ALGORITHMS

In order to balance the performance and generality, the separation of concerns method was used to implement the following algorithms. Because the algorithms described in this section are not involving much architecture characteristic, it means that they can be apply to next generation and other similar platform. The algorithms including port the hotspot function to CPEs, calculate the number of tasks before perform p2p kernel to obtain load balance, and rewrite the exponential function to avoid global store (GST) and global load (GLD), Uncouple the communication and computation via a task queue, parallelize the PM and FMM algorithm by making MPE and CPEs work simultaneously, and vectorize the exponential function by means of the vector registers provide by SW26010.

3.1 Porting the P2P on CPEs

Sunway TaihuLight is typical accelerator-rich system, the overwhelming part of computing power of it is gained by the CPEs. Thus, CPEs' utilization is key point of attain good performance. Thus, it's essential to take advantage of the CPEs to handle the heavy computation task of p2p. The origin version of p2p adopt a straightforward pattern which firstly traverse the k-d tree, and then judge if two leaves are interact with each other, if so, finally apply p2p operation on them. This is fine for MPE, but taking account for CPEs, the situation might become much tougher, if CPEs want to take the same pattern of MPE, the traversal of k-d tree has to elaborately redesign to accommodate to the CPEs' characteristic, which means it will greatly increase the complexity of

programming. the P2P kernel is the computation between two leaves, the maximal particles in a leaf is 16, thus, another solution seems is assign the computation between two leaves to 64 CPEs. However, single P2P operation might not enough to distribute to 64 CPEs, meanwhile the overhead of thread scheduling is too expensive. Instead of the two former methods, as the pseudocode show in Algorithm.1, we employ a strategy, which firstly traverse the tree to collect the information required by all p2p computation, then assign the computational task to CPEs, finally complete these tasks and store the results. The data structure is show in Fig.3, the ileaf denote target leaf while the jleaf denote source leaf.

Algorithm 1 TRAVERSE P2P BETWEEN LEAVES

Input: A k-dimensional tree $\boldsymbol{T}$; a empty lists ($\boldsymbol{L}_1$,$\boldsymbol{L}_2$,$\boldsymbol{L}_3$,...,$\boldsymbol{L}_n$)
Output: Lists ($\boldsymbol{L}_1$,$\boldsymbol{L}_2$,$\boldsymbol{L}_3$,...,$\boldsymbol{L}_n$) of interaction leaves of each leaf

```
FOR MPE:
initialize: organize particles to k-dimensional tree T
for all ileaf in T do
    Initialize the L_i
    for all jleaf in T do
        if jleaf interact with ileaf then
            add jleaf to L_i
            i++
        end if
    end for
end for

FOR CPEs:
initialize: get corresponding data of target leaf from MPE by the id of CPEs
for ileaf from first_leaf to last_leaf do
    get the L_i according to ileaf
    for all jleaf in T do
        p2p(ileaf,jleaf)
    end for
end for
```

There is a phenomenon that first we port the p2p to CPEs, the performance was greatly decreased. By analysis the assembly code, we found too many GLD/GST instructions was generated, so the DMA technique was used to avoid the long latency of memory access between MPE and CPEs. Specifically, the particles' information store in memory on MPE is a kind of array of structure (AOS), instead access directly, we use the MPE to make the AOS to structure of array, in other word, putting particles' information, such as position, velocity, and so on, into several array store in MPE and then it can be accessed expediently by CPEs via DMA. The AOS to SOA operation, which traverse all the particles, is processed on MPE, so it cost some time, but for the sake of the DMA are much faster with continuously memory access, the entire performance is improved.

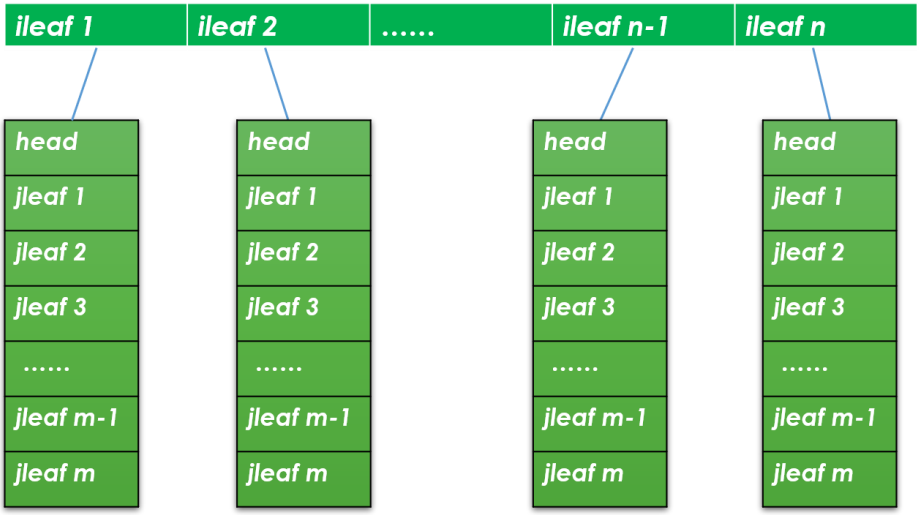


Fig. 3. The pattern of how the leaves in a k-dimensional tree perform the P2P operation, every target ileaf was interact with its source of jleafs

By using such a strategy, the computation was centralized, so the utilization efficiency of CPEs are greatly increased.

3.2 Load balance

Since the number of interactive leaves is not equal of each target leaf and the calculated amount is determined by both the source leaf and target leaf, so distributing the target leaves to 64 CPEs evenly can bring a load imbalance among those CPEs. Because of there is a barrier among all the them, the wall time of computation of P2P kernel on CPEs is determined by the CPE with longest time, thus, the uneven task distribution among CPEs becomes a big obstacle. We applied the following simple strategy to handle this problem.

1. calculate the average amount of target leaves a CPE should handle.
2. identify the lists of source leaves corresponding to the average target leaves
3. Assign these tasks to the CPEs

   By doing so, as show in Fig.4, the tasks were evenly distributed to CPEs.

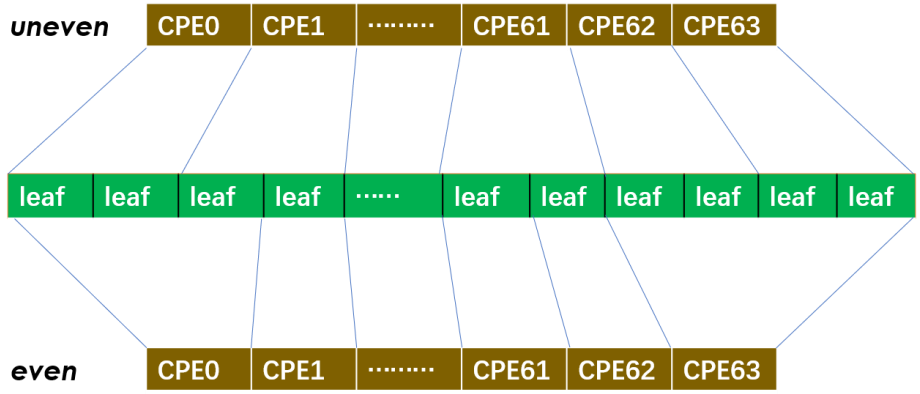


Fig. 4. Load balance of CPEs, the "leaf" is the source leaf, its interactive leaves (it's not show in the picture) is not equal. The "uneven" means the task distribution is not fair for 64 CPEs, the even means task distribution is approximately even for 64 CPEs.

Although the number of particles in a leaf is always not equal, but the difference is very small, so this algorithm works quiet well.

There also exists a method, which distribute the successive 64 leaves from first leaf to last leaf to 64 CPEs until all leaves get processed, it is feasible, but it consequently increases the GLD operations. The memory accesses of GLD operations are serialized when issued from multiple CPEs simultaneously, leading to the high latency (~5376 cycles if n = 64) and poor bandwidth [17].

3.3 Rewrite the exponential function

Lots of exponential function are used in the P2P operation, it cost most of the time. In the origin implement of exponential function on the Sunway TaihuLight, it takes the method of look-up table, which store results of the function in memory. Unfortunately, the local date memory (LDM) is quite small with the size of 64KB, so the table is store at the main memory. Every time the CPE call the exponential function will lead a GLD, as mentioned in [16], the latency is extremely high comparing with the direct memory access (DMA). So, we rewrite the function as follows:

$$e^x = e^{p*ln2} = 2^p * e^q$$

For the term 2^p, we can set p as exponential of a float point number, while the e^q can be approach by polynomial $P(x)$, the polynomial is calculate with LUT, which store in LDM. By eliminating the GLD, the performance is largely improved.

3.4 Porting the P2P among processors on CPEs

The inter-processors P2P operation is not the same as in-processor P2P operation illustrated above. Because the inter-processor P2P only happened when two leaves belong to different processor and close enough, one can infer that not all leaves engage in the p2p operation. So, it is not feasible to apportion the target leaves to CPEs, there is a need to change the strategy of task distribution, it is, as show in Algorithm 2, we collect the all leaf pairs that apply the P2P operation by the order they happened, and then distribute them to the CPEs. For one collection, the number of pairs might less than 64, in this case, we directly compute them on MPE. In algorithm 2, the TREE_{rs} is the trees receive from other processor, the LETs are abbreviation for Local Essential Tree, which store the information of particles that may involve in inter-processor p2p operation.

Algorithm 2 P2P_EX

Input: $TREE_{rs}$ which receive LETs , TASK[n] which store the leaves pairs
Output: None

```
1: initialize: Set i = 0
2: for all TREE_r in TREE_rs do
3:     for all ileaf in TREE_r do
4:         for all jleaf in TREE_r do
5:             if jleaf interact with ileaf then
6:                 add (ileaf,jleaf) to TASK[i]
7:                 i++
8:             end if
9:         end for
10:    end for
11:    if (i<64) then
12:        MPE handle task[ ]
13:    else
14:        assign TASK[ ] to CPEs
15:    end if
16: end for
```

Like task assignment, the DMA access also quite different with the in-processor p2p. The in-processor p2p operation can traverse all particles to put all the information that CPEs needed to several array, while the inter-processors p2p operation is make by small amount of the particles, it indicates the traverse operation might spend more time than that of in-processor p2p operation. We also notice that two leaves involved in inter-processors p2p operation is adjacent in logical index. And combining the fact that the particles' in information are continuously stored in memory, as a trade-off, we choose to leverage DMA channel to transfer the structures to CPEs for p2p operation, which transfer lots of useless data in these structures, but it is still much faster than the SOA to AOS operation on MPE.

3.5 Uncouple the communication and computation

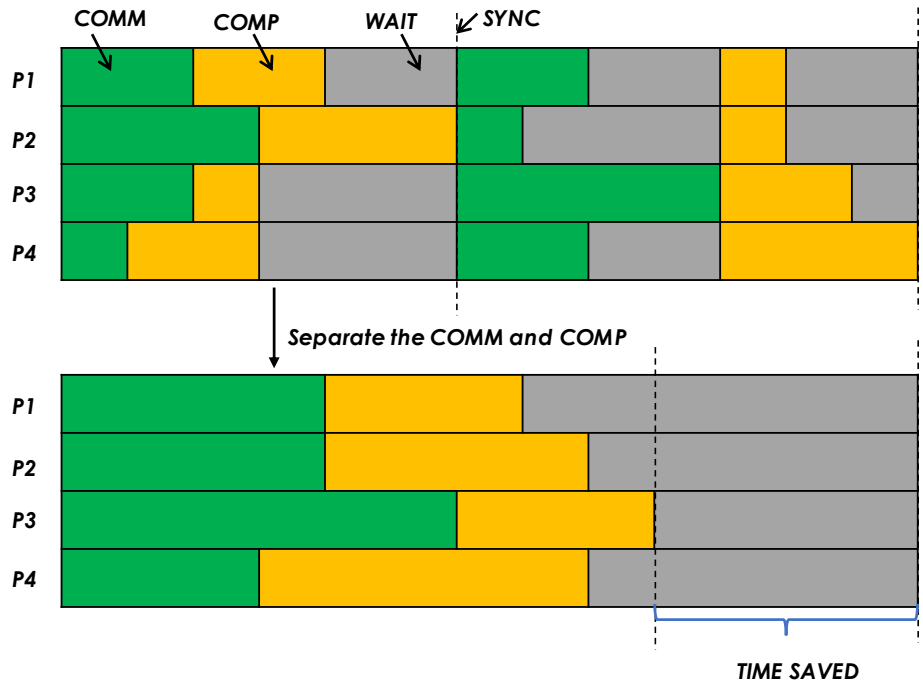


Fig. 5. The figure illustrates the behavior of send and receive LETs and inter-processor P2P kernel. P1, P2, P3, P4 represent four processors, and the "COMM" means communication, "COMP" means computation base on the data received LETs, "WAIT" means wait for synchronization, "TIME SAVED" means the time lessened after separate the communication and computation.

In the initial implement of inter-processor P2P operation, in order to apply P2P operation on the leaf belongs to other processor, the MPE must sent and receive a local essential tree (LET) to other processors, as illustrate in the upper half part of Fig. 5, the communication and computation are alternately processed, and the data dependency exists between communication and computation, the computation must resumed after receive the LETs and the communication among MPEs have a synchronization, therefore, even P1, P3 or P4 have already finish their computational job, it must wait P1 end its computation. The circumstance gets more terrible when MPI processors increased, as a result, the performance was greatly declined by waiting each other to end computation. Rather than communication and computation processing alternately, we split it individually. Specifically, all MPE firstly receive the LETs from other processors, meanwhile, use a queue data structure to store the information of computation task generate by the LETs, then after the communications are finished, the computation task in the task queue can be handled.

3.6 Parallelize the PM and FMM algorithm

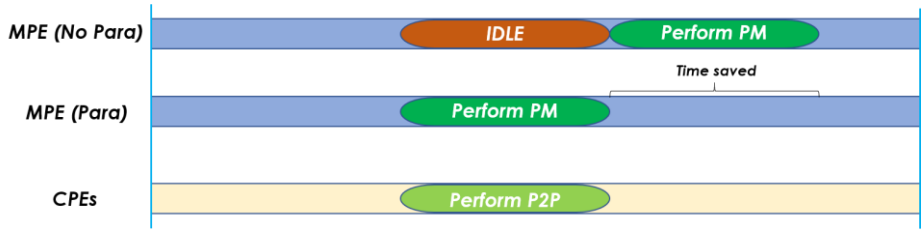


Fig. 6. MPE CPEs Parallel

For the Sunway TaihuLight, the common schema of the MPE and CPEs is that MPE assign work to CPEs and wait for the CPEs finish their work. During the period of waiting the MPE wait for the CPEs finish, the MPE is do nothing but blocked in joining the thread. Given that there is no data dependence between PM and FMM algorithm, we utilize the MPE to handle PM algorithm rather than idle waiting for CPEs get the FMM done. As show in Fig.6., the PM algorithm takes relatively little time compared with FMM, thus, the time of PM is completely hidden.

3.7 Vectorize the exponential function

The exponential function is still a hotspot after rewrite it, thus, we take full advantage of the vector instruction provided by SW26010 to vectorize the exponential function. But using the vectorized exponential function directly is inefficient, because of the number of particles in a leaf is less than 16, it's means that many scalar computations will be performed. Thus, we devise a collect-compute-replace strategy as show in Algorithm 3 to efficiently exploit the vectorization component of SW26010.

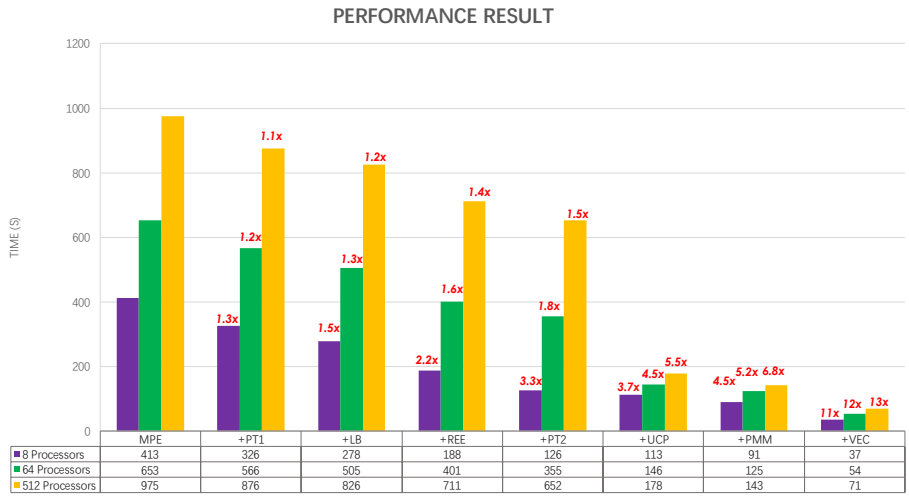
Algorithm 3 VECTORILIZE P2P

Input: ileaf, jleaf
Output: None

```
1: initialize: Set i = 0
2: for all particle1 in ileaf do
3:     for all particle2 in jleaf do
4:         collect x which x perform exp(x) to data[ ]
5:     end for
6: end for
7: result[ ]=vec_exp(data[ ])
8: for all particle1 in ileaf do
9:     for all particle2 in jleaf do
10:        put result[ ]
11:    end for
12: end for
```

4 VALIDATION AND PERFORMANCE ANALYSIS

To evaluate the effects of the many-core acceleration, we use 4,16,128 SW26010 processors to measure the performance improvement of seven optimization versions. The number of particles of one MPI processor is 32768, thus, the total number of 8, 64, 512 MPI processors is 262144, 2097152, 16777216 respectively. The results are presented below.



	MPE	+PT1	+LB	+REE	+PT2	+UCP	+PMM	+VEC
8 Processors	413	326	278	188	126	113	91	37
64 Processors	653	566	505	401	355	146	125	54
512 Processors	975	876	826	711	652	178	143	71

Fig. 7. Performance result. MPE is the baseline, the PT1 means the porting of P2P, the LB means load imbalance, REE rewrite exponential function, PT2 means porting of remote P2P, UCP means uncouple the communication and computation, PMM means parallelize the FMM and PM. VEC means vectorize the exponential function.

By analyzing Fig. 7, a basic CPEs parallelization of P2P kernel only leads to the speedups of 1.26x, 1.2x, 1.1x of 8, 64, 512 processors respectively, after solving the load imbalance among CPEs, speedups of 1.3x, 1.2x, 1.1x of 8,64,512 processors are achieved separately. By using the CPE version of exponential function which we re-written, the speedups ascend to 2.19x, 1.2x, 1.1x of 8, 64, 512 processors separately. A further 3.3x, 1.2x, 1.1x of 8, 64, 512 processors improvement of performance is achieved respectively results from the parallelization of remote P2P kernel on CPEs. For the method of uncoupling the communication and computation, the speedups of 8, 64, 512 processors are 3.7x, 4.5x, 6.8x respectively, which are not approximate to that of the pervious method mentioned, this is because the further quantity of MPI processors will leads more communication waiting, by utilizing uncoupling of communication and computation, the efficiency of was greatly enhanced. In addition, by taking full advantage of the MPE, we arrange the PM task to it when CPEs handle the FMM task, it brings speedups of 4.5x, 1.2x, 1.1x of 8, 64, 512 processors separately. Finally, an

impressive speedup of 11.1x,1.2x,1.1x of 8,64,512 processors are sustained respectively by means of the 256-bit wide vector registers of SW26010.

5 CONCLUSION REMARKS

In this paper, we have expounded the details for efficiently accelerating the processes stem from the cosmological N-body simulations on the gradually mature Sunway TaihuLight platform. By conducting the optimization from details of code to the algorithm, we gain a good performance consequently. In the future, we intend to go deeper of the N-body simulation and we hope these algorithms we proposed in this paper can be generalized to more areas.

6 ACKNOWLEDGMENTS

This source code comes from the photoNs2 developed by National Astronomical Observatories of China (NAOC). we also got a lot of help of the National supercomputer center in Wuxi and the scientific researcher Wang Qiao of NAOC. This study was supported by the Scientific Research Foundation (KYTZ201718) of CUIT.

References

1. Springel V , Yoshida N , White S D M . GADGET: A Code for Collisionless and Gasdynamical Cosmological Simulations[J]. New Astronomy, 2000, 6(2):79-117.
2. Ishiyama T, Nitadori K, Makino J. 4.45 Pflops Astrophysical N-body simulation on K computer——The gravitational trillion-body problem. In the SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012. New York ACM, 2012.
3. Yu H R , Pen U L , Wang X . CUBE: An Information-optimized Parallel Cosmological N -body Algorithm[J]. The Astrophysical Journal Supplement Series, 2017, 237(2).
4. K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen,J. Wawrzynek, D. Wessel, and K. Yelick, "A View of the Parallel Computing Landscape," Commun. ACM, vol. 52, pp. 56–67, Oct. 2009.
5. H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang,W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang,W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "TheSunway TaihuLight supercomputer: System and applications,"Sci. China Inf. Sci., pp. 1–16, Jun. 2016.
6. Ao Y , Yang C , Wang X , et al. 26 PFLOPS Stencil Computations for Atmospheric Modeling on Sunway TaihuLight[C]// IEEE International Parallel & Distributed Processing Symposium (IPDPS'17). IEEE, 2017.
7. Iwasawa M , Wang L , Nitadori K , et al. Global Simulation of Planetary Rings on Sunway TaihuLight[J]. 2018.
8. Edmund Bertschinger and James M. Gelb, "Cosmological N-Body Simulations," Computers in Physics, Mar/Apr 1991, pp 164-179.

9. R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles, Institute of Physics Publishing, 1988.
10. A.L. Melott, Comment on "Nonlinear Gravitational Clustering in Cosmology", Physical Review Letters, 56, (1986), 1992.
11. P.J.E. Peebles, A.L. Melott, M.R. Holmes, and L.R. Jiang, "A Model for the Formation of the Local Group," Ap J, 345, (1989) 108.
12. William D. Elliott and John A. Board, Jr., "Fast Fourier Transform Accelerated Fast Multipole Algorithm," SIAM Journal on Scientific Computing, March 1996, Volume 17, Number 2.
13. Leslie Greengard, "The Numerical Solution of the N-Body Problem, " Computers in Physics, Mar/Apr 1990, pp 142-152.
14. This algorithm was first published in "A Fast Algorithm for Particle Simulations", L. Greengard and V. Rokhlin, J. Comp. Phys., v 73, 1987.
15. Greengard's 1987 Yale dissertation "The Rapid Evaluation of Potential Fields in Particle Systems" won an ACM Distinguished Dissertation Award.
16. S. L. W. McMillan and S. J. Aarseth (1993). "An O(NlogN) Integration Scheme for Collisional Stellar Systems", Astrophys. J., Vol. 414, p. 200-212.
17. Xu Z , Lin J , Matsuoka S . Benchmarking SW26010 Many-Core Processor[C]// 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2017.
18. Klypin A , Holtzman J . Particle-Mesh code for cosmological simulations[J]. Physics, 1997, 145(1):1-13.
19. Moscardini L , Dolag K . Cosmology with Numerical Simulations[J]. 2011.
20. Singh J P , Holt C , Hennessy J L , et al. A parallel adaptive fast multipole method[C]// Supercomputing '93. Proceedings. IEEE, 1993.
21. Hellwing W . A short introduction to numerical methods used in cosmological N-body simulations[C]// Introduction to Cosmology. Introduction to Cosmology, 2015.
22. Ambrosiano J . The fast multipole method for gridless particle simulation[J]. Computer Physics Communications, 1988, 48(1):117-125.