2010 Ninth International Conference on Grid and Cloud Computing

Parallel FMM algorithm based on space decomposition

Jinshi Zhu, Yongmei lei[1], Jianchen Shan
School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China
{jinshi, lei}@shu.edu.cn, shu.vagabond@gmail.com, [1]Corresponding author

***Abstract*—In this paper, a parallel computational model and algorithm based on space decomposition is constructed and implemented, which supports the dynamically resource allocation under cluster environment. The major aim is to explore the new space decomposition scheme that can solve computation intensive problem. The fast multipole method (FMM) is an algorithm for rapid evaluation of the potential and force fields in the system involving large numbers of particles. Based on the serial FMM algorithm, a parallel implementation entitled SDPFMM is presented in the paper. The proposed algorithm is characterized by scalability and flexibility. We carried out the experiment on the high-performance computer ZQ3000 with Intel Trace Analyzer and Collector integrated into SDPFMM, and analyzed the experimental data and MPI performance of SDPFMM. The results demonstrate that the proposed algorithm is satisfying in both efficiency and solution quality.**

***Keywords*—space decomposition, FMM, SDPFMM, MPI, ZQ3000, Intel Trace Analyser and Collector**

I. INTRODUCTION

The fast multipole method (FMM) [1] is invented in 1987 by Greengard and Rokhlin at Yale University, for the rapid evaluation of the gravitational or Coulombic interactions between all pairs of particles in the system with large numbers of particles. The FMM algorithm is widely used in various research fields such as astrophysics, molecular dynamics and fluid mechanics. The above problems are usually referred to as the N-body problem[7]. The most direct approach to solve the N-body problem is to directly calculate the interactions between particles, but the running time of this method grows in quadratic level, and therefore people put forward many fast algorithms for solving N-body problem. Most of these fast algorithms are represented by the Barnes-Hut algorithm [2], and the Fast Multipole Method.

The FMM algorithm is an effective method to solve the N-body problem, so it is very necessary to study its parallel scheme for the large-scale N-body problem. The FMM algorithm has potential in the aspect of parallelization, which can be directly parallel itself. And the potential of parallelization implicit in the FMM algorithm is enormous. This paper presents a parallel strategy based on space decomposition, which minimizes the communication data on the premise of system load balancing. The running time and communication time under this strategy in the experiment are recorded in order to verify the high efficiency of the strategy.

II. THE DESCRIPTION OF FAST MULTIPOLE METHOD

A. The principle of FMM algorithm

The N-body problem is a well-known scientific issue, which is described in mathematical language as follows:

$$f(y) = \sum_{i=1}^{N} W_i K(y, x_i),\ i = 1,2,\dots,N,$$

The time complexity to evaluate f in the N-body system is $O(N^2)$. In order to speed up the calculation of the N-body problem, several fast algorithms have been put forward. Among them the typical is the Barnes-Hut algorithm and FMM algorithm. The FMM algorithm is used to accelerate the computing speed of the N-body problem with $O(N)$ time complexity. The idea of the FMM algorithm is that the original space is divided into different levels of space, and then the computational domain is divided into near-domain and far-domain. Near-domain uses direct calculation, while the calculation of far computational domain introduces a single formula to be gradually solved by approximating the impact on space by a cluster of particles. This process is described mathematically as:

$$f(y) = \sum_{i=1}^{N_{near}} W_i K(y, x_i) + \sum_{i=1}^{N_{far}} W_i K(y, x_i),\ i = 1,2,\dots,N$$

When computing remote domain, two key expressions are introduced as follows: Multipole Expansion (ME), Local Expansion (LE). The nature of the two expressions is the Taylor series. During calculation, ME will be converted to LE, which is referred to as M2L. Additionally there are two conversions: M2M and L2L. M2M consists of two steps: Translation of a multipole expansion: it is a transfer of particle cluster center for obtaining a higher level of ME; Sum of MEs: sum of the MEs to obtain a higher level of ME. L2L is the translation of a local expansion: it is a transfer of particle cluster center for obtaining a lower level of ME, L2L and M2L is alternating.

B. The executing step of FMM algorithm

The FMM algorithm is essentially a calculation of the Taylor expansion of different stages to finally complete the evaluation of interactions. The FMM algorithm uses a hierarchical spatial decomposition method to break down the original space box into a number of levels. The key task

978-0-7695-4313-0/10 $26.00 © 2010 IEEE
DOI 10.1109/GCC.2010.43

flow of the FMM algorithm is divided by functional decomposition as follows: ① Initialization; ② P2M (or: ME); ③ Translation of a multipole expansion, Sum of MEs; (called M2M) ④ M2L ; ⑤ L2L; ⑥ Computation of Local expansion (referred to as LEC); ⑦ direct calculation (referred to as PP); ⑧ Sum (sum of near-domain and far-domain force). The above tasks are carried out in sequential order. They are data-related.

The complete process of the FMM algorithm is shown in Figure 1. The FMM algorithm can be divided into four phases: initialization, upward, downward, calculation. The labels in Figure 1 indicates: ① tree construction ② initialization of tree ③ solve neighboring nodes and interaction list ④ P2M ⑤ M2M ⑥ M2L ⑦ L2L ⑧ L2P ⑨ PP ⑩ sum. In the Initialization phase, FMM is mainly to locate the input particles, initialize various attributes of particles and solve neighbor nodes and interaction list of grid units of different space levels. Initialization phase includes ①②③ shown in Figure 1. ①②③ is the initialization steps of the algorithm. ②③ can work in parallel, with no data dependence between the two. ④⑤⑥⑦⑧ are key steps of the FMM algorithm. The six steps are data-related, and the order is determined, so they can only be executed in the sequential order. ⑨ step is independent and can be executed in parallel with from ④ to ⑧.

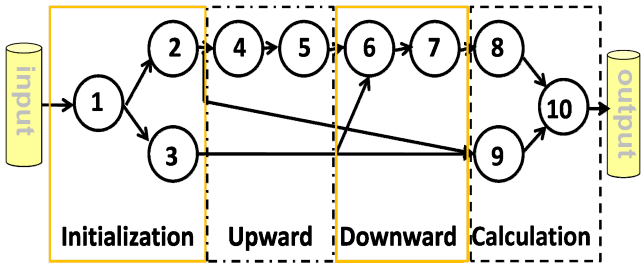

Figure 1. Process of the FMM algorithm

III. FMM ALGORITHM BASED ON SPACE DECOMPOSITION

A. *Location of particles and Morton encoding*

In this article, only a two-dimensional situation is discussed. SDPFMM uses the linear quadtree to construct the tree. In fact, the decomposition of the geometric space of FMM algorithm and the relation between parent-child nodes in the quadtree are the same in nature. Unlike the conventional quadtree, the linear quadtree maps a two-dimensional space to a one-dimensional space to overcome the disadvantage that the conventional quadtree needs more storage space. In addition, in the linear quadtree, the parent nodes can be generated from child's encoding to avoid using pointers to link them. To sum up, the basic idea of the encoding of linear quadtree is: No record of intermediate nodes and no use of pointers; only the leaf node is recorded, and the location of leaf nodes is denoted with address code. The input data of FMM algorithm is the particles, including various properties of the particles, one of which is the coordinates of the particles. To use a linear representation, a location code is needed to identify the quartrees. The location code contains information about the position and level of the quadtree. Such location code is known as Morton encoding. The following describes how to locate the particles into a grid, and then to locate them using Morton-ordered one-dimensional array.

Figure 2 shows one number for level=3. The sequence in Figure 2 is filled in Z pattern. It's a space filling curve. Figure 3 shows the grid coordinates for level=3. First, the algorithm finds the corresponding grid P belongs to according to the coordinate of P (a, b). The coordinates of the particles and that of the corresponding grid have the relation: P $(a, b) \in B(\lfloor a \rfloor, \lfloor b \rfloor)$.

42	43	46	47	58	59	62	63
40	41	44	45	56	57	60	61
34	35	38	39	50	51	54	55
32	33	36	37	48	49	52	53
10	11	14	15	26	27	30	31
8	9	12	13	24	25	28	29
2	3	6	7	18	19	22	23
0	1	4	5	16	17	20	21

Figure 2. Z-SFC for level=3

(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)

Figure 3. Grid coordinates for level=3

After locating every particle to its owned grid (referred to as leaf nodes in the quadtree), you can find the corresponding number in Figure 2 according to the coordinates of leaf nodes (hereinafter referred to as Morton code). In the above example, the Morton code of P (2.6,5,8) is 25. Computer programs can use some data structure to hold the correspondence between the leaf node coordinates and the Morton codes. For example, in C++ language, it can use map type or structure, and so on. But after study Morton codes have the following characteristics (figure 4). First of all, the (2,5) is turned into a binary form (0010,0101), and then the binary bits are interleaved: from left to right the

abscissa is on the odd position, the ordinate is on the even position. After interleaving the form of binary bits is changed to 00011001, which is precisely the decimal number 25 corresponding to Figure 2. Finally, The Morton encoding process is finished by appending binary code of corresponding level 011 in the above encoding to get the final encoding 00011001011.

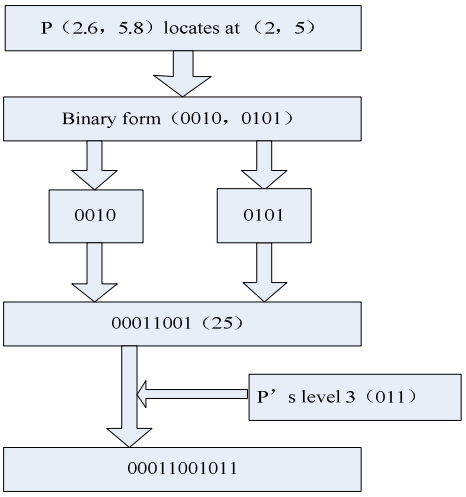

Figure 4. Morton encoding

After the leaf nodes are sorted in Morton encoding, the algorithm can generate a global array index (the linear quadtree) according to Morton ordering; each array element is an index and the index points to the corresponding spatial cell. The global array index is the foundation of the construction of the LET[4] described in the following.

B. The constrction of LET

In the SDPFMM, the core part is the construction of the locally essential tree (LET). LET is the sets of nodes including leaf nodes scheduled to some processor, their ancestor nodes and their interaction list nodes. The local tree (LT) only includes leaf nodes scheduled to some processor, their ancestor nodes. LET is the extension of LT. In Figure 5, the space is divided into four regions 1,2,3,4, representing the computing area initially allocated by each processor. These areas are LT. The cells marked with * are the sets of interaction list of area 2. We use LET (i) representing LET executed by the i processor. So a conclusion can be drawn that LET（2）=LT（2）+{cells marked with *}. This paper has mentioned the global array index in Morton ordering. Before the construction of LET, the algorithm will create the global array index of leaf nodes and their ancestor nodes in Morton ordering. The global array index of parent nodes and ancestor nodes are dynamically generated from the global array index of leaf nodes. When constructing LET, the algorithm will specify the k-th level for parallel computing, then divide the k-th level global array. The division can be equal length or unequal length. The equal length division is the simplest method, but it will cause the load imbalance. For the unequal length division, the algorithm will estimate in advance the total number of the interaction list of local arrays of each level, then find a way of division the task of each area is more or less be in balance. The figure 5 shows that for the LET in area 2, the * cells will send the data from 1,3,4 to 2 and 1,3,4 will receive corresponding data from other areas. But how do areas 1,3,4 determine which cells should be sent to area 2 ? Take example for figure 5. The cells be sent to area 2 are {cells | cell$\in$(LET(2) $\cap\ \Omega_1$)$\cup$(LET(2) $\cap\ \Omega_3$) $\cup$ (LET(2) $\cap\ \Omega_4$)}.

The above description shows that LT is the set of local array corresponding to each level in fact while LET is the set of local array and its interaction list. The conversion of LT into LET requires communication. The communications include the communication between the LETs, between LET and root Tree, and the communication between the neighbors which happens in the interaction between near-domain cells.

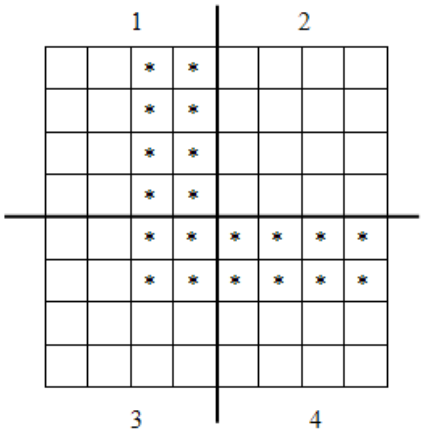

Figure 5. Set of interaction list of area 2

The following is the description of LET. Before the algorithm is described, several variables are introduced.

L_p: leaf nodes assigned for processor p;

$A(L_p)$: the ancestors of L_p;

$\Omega_{p'}$: the region controlled by processor p';

$I(\beta)$: the interaction list of β.

Algorithm description for LET construction:

Input: the set of particles: x

Output: LET on each processor p

1. $L_p = \text{PointsToQuadtree(x)}$
2. $B_p = L_p \cup A(L_p)$
3. $O_{pp'} := \{\beta \in B_p : I(\beta) \cap \Omega_{p'}\}$
4. $\forall p' : p' \neq p$
   p' Send $O_{pp'}$ to processor p
   p Receive $O_{pp'}$ from processor p'
   Put $O_{pp'}$ into B_p
5. Return B_p, the construction of LET is finished.

C. The description of Parallel FMM algorithm based on space decomposition

According to the foregoing description, one can know how to locate the particles, Morton encoding and the construction of LET. The following will completely describe the parallel FMM algorithm based on space decomposition,

which is the theoretical reference of SDPFMM programming realization. The SDPFMM algorithm steps are described as follows:

Step 1: Specify the initial parameters: d for the maximum depth of space division level, starting from the k-th level to carry out parallel computing;

Step 2: Locate the particles to corresponding grid according to the coordinates of the particles, then sort the leaf nodes in Morton encoding to generate the global array, and then dynamically generate Morton encoding of parent nodes and ancestor nodes from Morton encoding of leaf nodes; construct a Morton-order sorted array in each space level;

Step 3: According to the parameter k, divide the arrays of the k-th level to some local arrays which are sub-task array scheduled to each processors. The task corresponding to the arrays above k-th level is done by one appointed processor.

Step 4: Construct LET on each processor of which the height is $d - k$;

Step 5: Each processor computes MEs, M2Ms in upward order starting at the leaves, exchanges data between LETs, and sends communication data to the root tree;

Step 6: Calculate the M2Ms in root tree, and distribute communication data to each LET. Each processor calculates M2Ls, L2Ls and LEs in downward order, and exchanges data between LETs. Finally evaluate local expansion at each particle, namely solve the far-domain force of each particle.

Step 7: Each processor directly computes near-domain interaction forces, then sums near-domain and far-domain interactions to output the result.

IV. THE IMPLEMENTATION OF SDPFMM AND EXPERIMENTAL RESULTS

A. The description of the implementation of SDPFMM

The implementation of the SDPFMM algorithm uses MPI. As the SDPFMM algorithm uses the idea of space area division, each area is the parallel processing unit. In describing the SDPFMM algorithms, the calculation unit indexed by the local arrays sorted by Morton encode is essentially each region of decomposed space. As depicted in Figure 6, the global array corresponds to undivided area of space, then the global array is divided into (which can be equal-length or unequal-length) several local arrays corresponding to the divided area. In SDPFMM algorithm, there is communication among the areas when constructing LET, After the construction of LET, the memory in each processor acquires the copy of the memory in other processors.

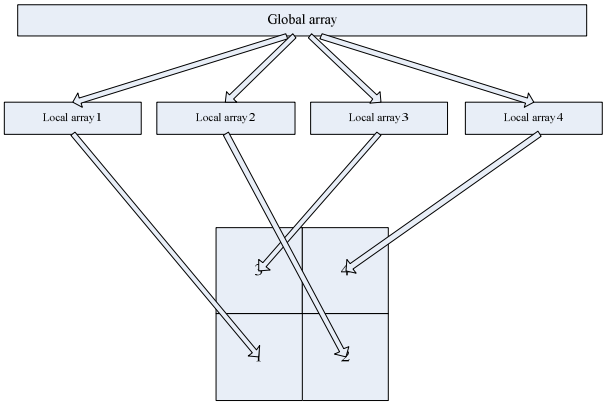

Figure 6. The relation between global array and local arrays

B. Experimental data and performance analysis

Runtime environment of our experiments is the high-performance computer ZQ3000 cluster of Shanghai University, which consists of 192 nodes, and each node has two 3.06GHZ Intel Xeon CPUs with a 2GB memory. The experimental procedures are implemented by C and C++ with MPI parallel library and PETSC library[6]. In addition, a graph partitioning tool such as Parmetis is introduced to create partitions. With the purpose of better understanding MPI application behavior, quickly finding bottlenecks and achieving high performance for parallel cluster applications, a powerful tool named Intel Trace Analyze and Collector is integrated into the SDPFMM program.

TABLE 1 EXPERIMENTAL DATA OF SERIAL FMM

Particles	Level	Time(s)	Minimum. time(s)
62500	4	113.61	40.64
	5	40.64	
	6	47.84	
140625	5	188.24	84.51
	6	84.51	
	7	174.94	
250000	5	492.76	166.07
	6	166.07	
	7	194.07	

In table 1, the number of particles selected is 62500, 140625, and 250000. Table 1 shows that the executing time for the same number of particles differs with each other as the value of level changes and there exists an optimal value making the time least. In order to facilitate comparison, the distribution of particles in the following experiment is well-proportioned. For each number, when running the FMM program we select the different levels of space

decomposition to record the running time of the different numbers of particles. Before carrying out the SDPFMM experiment, it is necessary to run serial FMM, which is instructive to SDPFMM. On the one hand the SDPFMM parameter selection will be as far as possible consistent with the serial program, making the parallel program run under the optimal parameters. On the other hand it is used to assess the performance of parallel programs.

TABLE 2 EXPERIMENTAL DATA FOR 2,4,6 PROCESSORS;LEVEL=6; K=3(64 SUB-TASKS)

Particles	Nodes	avg. none-com. time(s)	avg. com. time(s)	avg. time(s)	Speedup
62500	2	24.73	0.38	25.11	1.00
	4	9.02	1.47	10.49	2.39
	6	6.35	3.79	10.14	2.48
140625	2	44.51	0.66	45.17	1.00
	4	16.28	2.44	18.72	2.41
	6	18.78	4.23	23.01	1.96
250000	2	86.38	0.79	87.17	1.00
	4	23.45	2.49	25.94	3.36
	6	59.03	8.79	67.82	1.29

TABLE 3 EXPERIMENTAL DATA FOR 2,4,6 PROCESSORS; LEVEL=6;K=4(256 SUB-TASKS)

Particles	Nodes	avg. none-com. time(s)	avg. com. time(s)	avg. time(s)	Speedup
62500	2	25.21	1.22	26.43	1.00
	4	8.99	2.80	11.79	2.24
	6	15.05	5.90	20.95	1.26
140625	2	45.8	1.66	47.46	1.00
	4	16.44	4.25	20.69	2.29
	6	27.41	9.53	36.94	1.28
250000	2	89.63	2.14	91.77	1.00
	4	23.66	3.82	27.48	3.34
	6	54.35	9.85	64.20	1.43

In the table 2 and 3, the non-communication time is the sum of running time of each processor's non-MPI code while the communication time is the sum of the running time of each processor's MPI communication function. The analysis of the above two tables show that the average none communication time of SDPFMM reduces as the number of the processors increases on condition that the parameter level and k are fixed, but are not directly proportional. The time trendline would be a degressive wavy line if we draw a fitting curve when having more processors run the parallel program. When the number of particles is different, the number of CPUs is same, and the parameter level and k are fixed, the average communication time of SDPFMM increases as the number of the processors increases. On the condition of the same number of particles with parameter k fixed, the average running time of SDPFMM is least when the number of CPUs is 4. This is because it takes less time for SDPFMM to deal with the construction of linear quadtree. The parmetis library generates different partitionings according to the number of processors. For 2,4 and 6 CPUs, 4 CPUs is optimal for parmetis, so the speedup is maximal when the number of CPUs is 4. As the number of processors increases, four CPUs may be not the optimal choice. Additionally, with the size of the N-body problem increased, more processing nodes are needed, and the different program parameters have great influence on the results, so the optimal number of nodes is associated with the size of the problem and the parameter selection of the program. But it is not the focus of this paper. In general, the SDPFMM performance is satisfying.

V. CONCLUSIONS

The fast multipole method is a fast algorithm to solve the N-body problem. In order to make FMM programs run in high-performance parallel computers, many scholars put forward some parallel methods of FMM and implement them with a variety of parallel programming methods. In this paper, we put forward a parallel FMM algorithm based on space decomposition, describe the idea and implementation method of SDPFMM, and carry out some experiments on ZQ3000 cluster. By analyzing the experimental data, we come to a conclusion that SDPFMM algorithm has good performance, and its design is scientific and rational. The future work will focus on introducing pipelining to parallelizing communication and computation.

VI. ACKNOWLEDGEMENT

This work is supported by National High-tech R&D Program of China under Grant 2009AA012201, Major Technology R&D program of Shanghai under Grant

08dz501600, and Shanghai Academic Leading Discipline Project J50103.

REFERENCES

[1] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. Journal of computational physics 73,325-348(1987)

[2] J. Barnes and P. Hut. A Hierarchical O(N logN) Force-Calculation Algorithm. Nature 324, 446–449 (1986).

[3] Felipe A. Cruz and L. A. Barba. Characterization of the errors of the Fast Multipole Method approximation in particle simulations.

[4] Ilya Lashuk, Aparna Chandramowlishwarn and Harper Larper Langston. A massively parallel adaptive fast-multipole method on heterogeneous architectures. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. Nov.2009

[5] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. http://math.nyu.edu/faculty/greengar/

[6] S. BALAY, K. BUSCHELMAN, W. D. GROPP, D. KAUSHIK, M. KNEPLEY, L. C. MCINNES, B. F. SMITH, AND H. ZHANG, PETSc home page, 2001. http://www.mcs.anl.gov/petsc

[7] Pangfeng Liu. Experiences with Parallel N-Body Simulation. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 11, NO. 12, DECEMBER 2000

[8] B. K. Alpert and V. Rokhlin. A Fast Algorithm for the Evaluation of Legendre Expansions. SIAM J. Sci. Stat. Comput. 12, 158−179 (1991).

[9] Jakub Kurzaka and Montgomery Pettitta. Massively parallel implementation of a fast multipole method for distributed memory machines. J. Parallel Distrib. Comput. 65 (2005) 870–881.

[10] Xiaobai Sun and Nikos P.Pitsianis. A Matrix Version of the Fast Multipole Method. SIAM Review Vol.43 No.2.pp.289-300

[11] Jakub Kurzak and B.Montgomery Pettitt. Communication overlapping in fast multipole pariicle dynamics methods. Journal of Computational Physics 203(2005) 731-743

[12] C. R. Anderson. An Implementation of the Fast Multipole Method Without Multipoles. SIAM J. Sci. Stat. Comput. 13, 923–947 (1992).

[13] Andrew W. Appel. An effcient program for many-body simulation. SIAM Journal on Scientific and Statistical Computing, 6(1):85-103,1985.

[14] C. R. Anderson. An Implementation of the Fast Multipole Method Without Multipoles. SIAM J. Sci. Stat. Comput. 13, 923–947 (1992).

[15] R. K. Beatson and W. A. Light (1996). Fast evaluation of Radial Basis Functions: Methods for 2–dimensional Polyharmonic Splines. Toappear in IMA J. Numer. Anal.