Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 203 (2005) 731–743

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Communications overlapping in fast multipole particle dynamics methods

Jakub Kurzak [a], B. Montgomery Pettitt [a,b,*]

[a] *Department of Computer Science, University of Houston, Houston, TX 77204-5003, USA*
[b] *Department of Chemistry, University of Houston, 4800 Calhoun, Houston, TX 77204, USA*

Received 24 November 2003; received in revised form 6 July 2004; accepted 30 September 2004
Available online 11 November 2004

Abstract

In molecular dynamics the fast multipole method (FMM) is an attractive alternative to Ewald summation for calculating electrostatic interactions due to the operation counts. However when applied to small particle systems and taken to many processors it has a high demand for interprocessor communication. In a distributed memory environment this demand severely limits applicability of the FMM to systems with O(10 K atoms). We present an algorithm that allows for fine grained overlap of communication and computation, while not sacrificing synchronization and determinism in the equations of motion. The method avoids contention in the communication subsystem making it feasible to use the FMM for smaller systems on larger numbers of processors. Our algorithm also facilitates application of multiple time stepping techniques within the FMM. We present scaling at a reasonably high level of accuracy compared with optimized Ewald methods.

1. Introduction and motivation

The fast multipole method (FMM) was introduced by Greengard and Rokhlin [1] over a decade ago. Since then many performance improvements were incorporated into FMM. Diagonalization of the translation operators for harmonic functions replaced costly convolution of $O(p^4)$ complexity with an element-wise product of $O(p^2)$ complexity [2–5] (where p is the number of terms in multipole expansion). Careful error analysis [6] lead to the idea of spherically shaped interaction regions with super-blocks [7]. These techniques brought at least an order of magnitude improvement in the performance of FMM. At the same time,

* Corresponding author. Tel.: +1 7 13 743 3263; fax: +1 7 13 743 2709.
E-mail address: pettitt@uh.edu (B.M. Pettitt).

introduction of the technique of macroscopic expansion (ME) in a natural way extended applicability of FMM to systems with periodic boundary conditions [8]. The FMM became a clear alternative to Ewald summation based methods with a break-even point in serial mode of greater than 10 K atoms. Since its first appearance, parallelization of FMM has been an on-going effort [9–11]. Here we concentrate on a communications problem which can arise in any grid topology but is an accute problem for a switched network system.

2. Fast multiple method theory

Below, we provide a brief introduction to FMM techniques. The interested reader is refered to the literature for a more complete derivation and discussion [1–11].

2.1. Serial algorithm

The FMM utilizes hierarchical spatial decomposition of the simulation box into a number of levels organized in an octal tree. Each consecutive level refines its predecessor by subdividing each cell (parent) into eight smaller cells (children). The cells on the most refined (leaf) level house the particles subject to an electrostatic force calculation. The algorithm proceeds in the following steps (refer to Fig. 1 for graphical representation of steps 2–4):

(1) *Multipole expansion calculation* – each leaf cell calculates multipole expansion based on positions and charges of particles enclosed in that cell.
(2) *Upward pass* – each parent cell accumulates multipole expansions of its children by means of multipole to multipole (M2M) translation.
(3) *Well separated cell interactions* – cells on each level accumulate multipole expansions of well separated cells into their local expansions by means of multipole to local (M2L) translation.
(4) *Downward pass* – each parent cell distributes its local expansion to its children by means of local to local (L2L) translation.

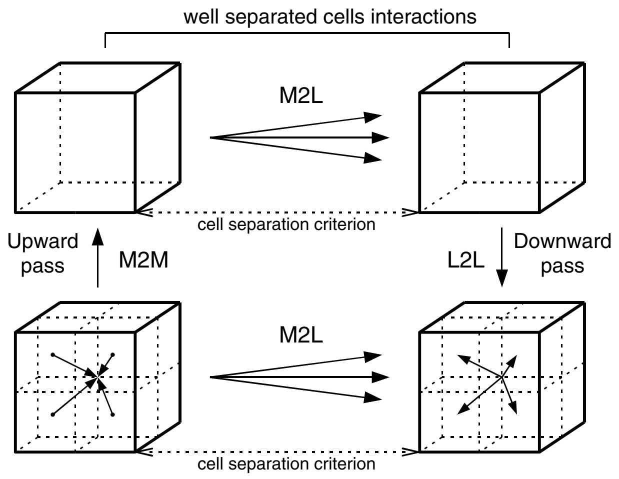
well separated cells interactions
M2L
cell separation criterion
Upward pass
M2M
L2L
Downward pass
M2L
cell separation criterion

Fig. 1. Multipole part of FMM with upward pass, well separated cells interactions and downward pass.

(5) *Local expansion evaluation* – each cell calculates for each particle the contribution to force and potential from the local expansion.
(6) *Direct particle to particle calculation* – each cell calculates for each particle the contribution to force and potential from particles in nearby cells.

Steps 1–5 are referred to as the multipole calculations; step 6 is referred to as the direct calculations. Step 3, well separated cell interactions, is the most costly operation in the multipole part. Since the M2L translation is a convolution, one of the methods to improve its performance is to replace it with an element-wise product in Fourier space (Fig. 2). We utilized the method proposed by Elliott and Board [3]. In the serial algorithm step 3 is preceded by fast Fourier transform (FFT) of the multipole expansions and followed by inverse FFT of the accumulated local expansions. This technique results in a performance improvement of the M2L translation of about four times. Although a newer technique exists [4,5], it involves considerably more cumbersome mathematical apparatus.

We also use the speedup mechanism of spherically shaped interaction regions with super-blocks (also called parental conversion) and flexible separation criterion [7]. In this solution the interaction region of a cell has the corners cut off. Also a number of interactions with cells on the same level are replaced with interactions with their parents. For typical separation criterion of 2 cells this technique decreases the size of the multipole interaction region from 875 to 315 cells and the size of the direct interaction region from 125 to 93 cells, with little impact on the numerical accuracy.

To implement periodic boundary conditions we utilized the ME [8]. In non-periodic FMM the tree is build from the root down by subdividing the domain. Analogously in ME another tree is build from the root up by combining periodic images of cells into higher levels of aggregate cells. ME converges to a properly implemented Ewald summation (with dipole correction).

2.2. Parallel algorithm

For parallel execution, cells on all levels are spanned with a recursive Hilbert space-filling curve [12] and approximately even pieces of the curve are assigned to each processor. The cells assigned to a processor in this manner are referred to as host cells for that processor. These cells interact with other cells within the multipole interaction region and direct interaction region, which are assigned to other processors and are referred to as neighbor cells (Fig. 3). Neighbor cells have to be fetched from their host processors prior to computation involving these cells. Although much more sophisticated domain partitioning schemes exist [13,14], Hilbert space-filling curves suffices for all practical purposes.

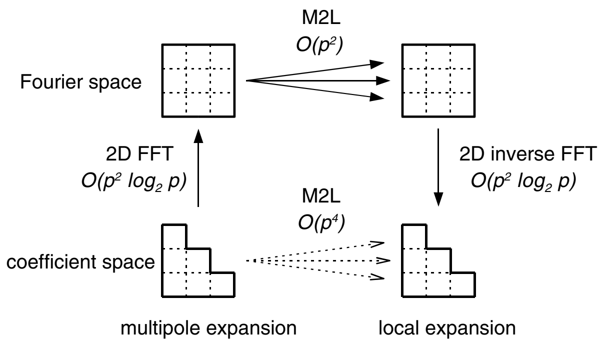

Fig. 2. FFT acceleration of FMM with convolution in coefficient space replaced with element-wise product in Fourier space.

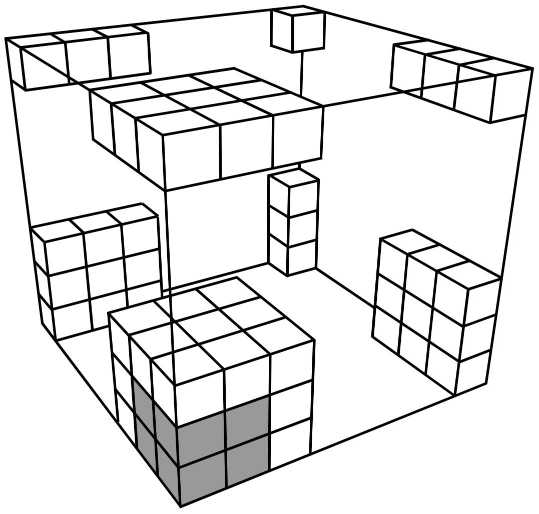

Fig. 3. Domain partitioning in parallel FMM with 512 leaf cells mapped to 64 processors. Shaded cells are cells assigned to processor 0 (host cells of processor 0). Not shaded cells are direct neighbors of processor 0 residing on other processors. For clarity of the picture minimum possible separation of 1 cell was chosen. We may notice the wrap-around effect of periodic boundary conditions.

One designated processor is responsible for ME calculations. For constant volume systems the cost of ME is just a few extra M2L translations which is negligible since every processor performs hundreds of them anyway. For constant pressure systems (non-constant volume) the problem is isomorphic to constant volume dynamics in the virtual variable space.

3. Problem description

Although FMM is often called an inherently parallel algorithm, its potential parallelism is limited by its considerable communication requirements. When applied to small systems (below 30 K atoms) the optimal hierarchical spatial decomposition of FMM is shallow (4 levels) dividing the simulation box into a relatively small number of cells (512 leaf level cells). With aggressive parallelization the number of cells per processor (host cells) goes down to just a few. At the same time the number of cells involved in any communication operation (neighbor cells) stays on the order of a few hundred (Tables 1 and 2).

Table 1
Numbers of host multipole cells and communicated multipole cells per processor for system with periodic boundary conditions, 4 levels of spatial decomposition and 585 total number of cells

Number of processors	Number of host multipole cells[a]	Number of multipole data communications[a]
1	585	0
2	292	293
4	146	439
8	73	512
16	36	549
32	18	542
64	9	494

[a] Approximate numbers. The actual numbers will slightly differ from processor to processor.

Table 2
Numbers of host multipole cells and communicated multipole cells per processor for system with periodic boundary conditions, 4 levels of spatial decomposition and 585 total number of cells

Number of processors	Number of host direct cells[a]	Number of direct data communications[b]
1	512	0
2	256	256
4	128	384
8	64	416
16	32	320
32	16	240
64	8	176

[a] Exact numbers.
[b] Approximate numbers. The actual numbers will slightly differ from processor to processor.

Typically the multipole and the direct part of FMM have the form presented in Fig. 4. In the processing phase the outermost loop iterates through the host cells and the innermost loop iterates through the neighbors of a particular host cell. With this approach most of communication has to take place before the first cell can be processed. The reason for this is that interaction regions for host cells overlap significantly. In fact with just a few cells per processor, the interaction region for a single cell covers approximately half of the whole interaction region for all host cells.

Naively, with this approach all the communication is initiated at the same time exposing the program to contention in the communication subsystem in its two major forms of network or switch contention and end-point contention. The first major type of contention is experienced on distributed systems built of SMP nodes connected with a switched network. Usually it is a high performance network with low latency and high bandwidth. However due to switching technology the effective bandwidth per processor decreases with the number of processors which is antiscaling. The second major type of contention is the end-point contention which occurs when several processors attempt to communicate with one other processor at the same time.

We see with the approach displayed in Fig. 4 there may not be enough computation to be placed between communication initiation and communication completion resulting in idle or wait time. If a multiple time stepping technique for integration of equations of motion is used the only computations that can be scheduled there are interactions between host cells which constitute only a small fraction of all interactions and therefore CPU time.

```
Initiate communication
...
Do some computation
...
Wait until communication completes

FOR each host cell I
  FOR each cell J within the interaction zone of I (host and neighbor)
    Accumulate in cell I the contribution from cell J
  END FOR
END FOR
```

Fig. 4. Pseudocode for the contention-prone communication scheme.

The diagonalization of translation operators, although very beneficial for performance of the sequential algorithm, introduces new challenges to parallelization of FMM. Data representation in the transformed space takes up more memory and creates bigger messages than in the coefficient space representation. In addition, the element-wise product is a very inefficient operation in the sense that a single data element is used just once. Overall the ratio of communication to computation increases drastically.

Another problem that we have to deal with is the communication library overhead.

From a simple point of view it is convenient to communicate each data set (multipole expansion, direct particle positions) in a separate message. Such approach also allows one to avoid buffering. However, by communicating a large number of short messages we are exposed to the overhead of the communication library calls and latency. This way we may also contribute to network contention by creating a lot of handshaking messages.

At this point it is crucial that the communication subsystem is utilized efficiently and contention is avoided. Communication systems are an architecture dependence that we will not explore fully in this work. Here we base this work on an all-to-all connectivity abstraction provided by the MPI communication model. The underlying architecture common to many available platforms is symmetric multiprocessors (SMPs) connected with a switched network of fat tree topology.

4. Method

4.1. Inverse processing

Typically in the FMM, step 3, well separated cell interactions, and step 6, direct particle to particle interactions, are organized according to Fig. 4. The outer loop iterates through host cells and the inner loop iterates through all neighbors of a given host cell. We can make a simple observation that these loops can be interchanged according to Fig. 5. The outer loop iterates through all neighbors of a given processor and the inner loop iterates through all host cells a given neighbor interacts with. Fig. 6 demonstrates the idea of this inversion.

It is important that instead of fetching all neighbors at the same time, before steps 3 and 6, they can be fetched one by one throughout steps 3 and 6 with communication taking place in the background of proceeding computation. This is analogous to a prefetch operation. By communicating a single multipole or direct neighbor at a time, we spread the communication throughout the whole computation phase without loss of synchronization and therefore determinism in the equations of motion.

4.2. Processing scheduling

In order not to loose determinism of computation each processor pulls the neighbors from other processors in an apriori order. It is easy however to create the situation where all processors fetch neighbors data from the same host processor at the same time. In such case the host processor becomes the communication bottleneck (Fig. 7(a)).

There is no need however to come up with complicated scheme. A simple solution is to employ circular neighbor processing (i.e. modulo number of processors) where processor n fetches and processes neighbors data in turn from processors $n + i$ mod no_processors ($i = 0, \ldots,$ no_processors $- 1$) (Fig. 7(b)).

As one quality measure of this computation scheduling scheme we consider the percentage of time when more than one processor processes the same neighbor cell. With our circular assignment this time stays on the order of 10% of the total computation time and no adverse effects of the ''conflicts'' on the communication are noticed.

```
FOR each host cell J
  FOR each host cell I that has cell J within its interaction zone
    Accumulate in cell I the contribution from cell J
    Call the communication subroutine*
  END FOR
END FOR

FOR each neighbor cell K
  FOR each host cell I that has cell K within its interaction zone
    Accumulate in cell I the contribution from cell K
    Call the communication subroutine*
  END FOR
END FOR
```

**The communication subroutine encompasses the task of initializing and completing communication events in such rate that the communication subsystem is not overloaded and the data is delivered on time for the computing part to process.*

Fig. 5. Simplified pseudocode for the contention-free communication scheme which distributes the communication throughout the computation.

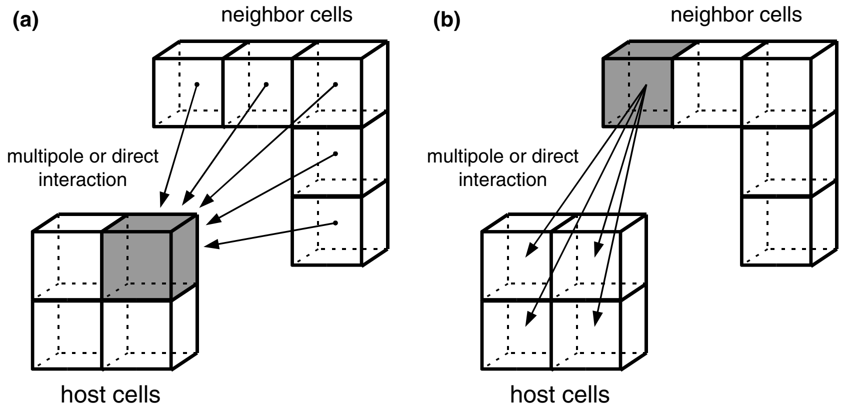
(a)
neighbor cells
multipole or direct interaction
host cells
(b)
neighbor cells
multipole or direct interaction
host cells

Fig. 6. (a) Classic FMM – each host cell in turn processes all its neighbors. (b) Inverse processing – each neighbor in turn passes its contribution to all host cells.

This scheme can be trivially implemented by sorting the neighbor cells according to their owner number and shifting the list. The advantage of this simple scheme is that all neighbors from a single processor are processed in a continuous manner and the resulting messages can easily be aggregated into bigger ones, or even a single big message. The message size can then be a target of optimization to take advantage of bandwidth and switch contention issues given the amount of time for the overlapping computation.

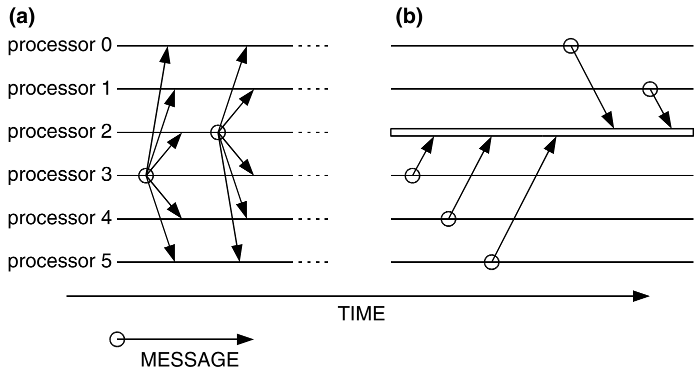

Fig. 7. (a) Worst case scheduling – all processors pull neighbors data from other processors in the same order. (b) Circular scheduling – processor n pulls neighbors data from processor $n + 1$, $n + 2, \ldots, N, 0, 1, \ldots, n-1$, where N is the total number of processors.

4.3. Message aggregation

Communicating small messages can be very inefficient. If we send each data set in a separate message we may suffer considerable overhead. We can aggregate data sets (multipole expansions, direct particle data) into bigger messages. This strategy works up to a point.

With increasing number of processors used, eventually we will be limited by the decreasing amount of information exchanged between each pair of processors. On the other hand, with a small number of processors maximum possible aggregation will create huge messages and expose us to the same contention phenomena that we are trying to avoid. As opposed to aggregation we will have to chop the data into smaller portions.

The right message size is a matter of architecture and will depend on the computer system and message passing library implementation. Our experience shows that on most large scale parallel systems the message size in the range between 20 and 50 kB gives good results. This corresponds to 8–24 multipole expansions in a single message. This can be retuned for a given platform.

4.4. Side effect – memory savings

As discussed in more detail in the following section for the multipole part, we have decided to communicate in coefficient space and perform FFTs on the receiving processor. In such a case there is a positive side effect of this communication scheme. We notice that once a multipole expansion in Fourier space is passed to all the host cells, we can reuse the memory it occupies. Because we process multipole expansions of the neighbor cells sequentially, we only need one multipole matrix in Fourier space for all the neighbors. Fig. 8 compares the amount of memory required for multipole expansions by a single processor with and without using this technique.

Potentially the same could apply to multipole matrices of the neighbors in coefficient space. However it is convenient to keep the coefficient space matrices of the neighbors due to message aggregation and the fact that more than one communication may be in progress at a time. One may still wish to consider using a-few-multipole-expansions-wide buffer instead of allocating space for the entire multipole neighborhood. This way we could potentially limit memory allocated for multipole expansions to the host cells.

These memory savings do not have as big impact as might be expected because of other memory requirements of FMM. Nevertheless superlinear speedups were observed on some machines with somewhat limited memory systems.

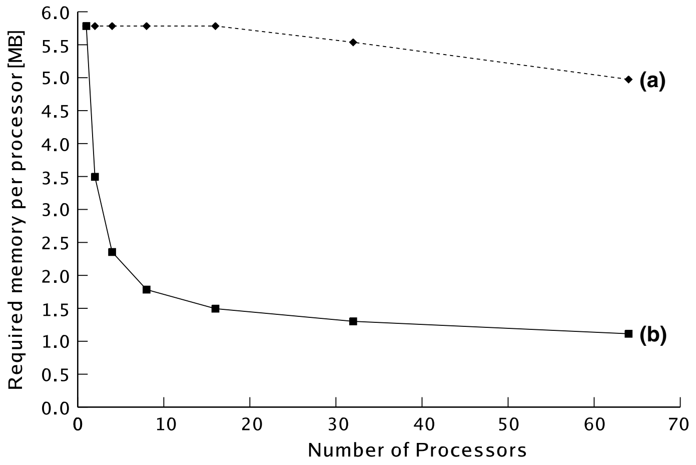

Fig. 8. Memory requirements per processor for storage of multipole expansions with: (a) memory allocated for multipole expansions in coefficient space and Fourier space for host cells and neighbor cells, (b) memory allocated for multipole expansions in coefficient space and Fourier space for host cells and for multipole expansion in coefficient space only for the neighbor cells.

5. Performance results and discussion

We present results which reflects our experiences with a system of around 13 K atoms 4 levels of spatial decomposition, expansion size of 16 and separation criterion of 2 cells. With the technique of spherical interaction regions and super-blocks this results in a multipole neighborhood size of 315 cells and direct neighborhood size of 93 cells. It is an example of a small system of practical interest to us that significantly benefits from FMM. Also a system of that size allows us to do aggressive parallelization of FMM and investigate its scalability bottlenecks without using many hundreds of processors.

All results presented here were collected on the PCS Compaq Alphaserver ES45 system with SMP nodes of 4 Alpha EV6.8CB processors and a Quadrics network connecting the nodes. The results show performance for a system of 13 K atoms, 4 levels of spatial decomposition, expansion size of 16 and separation criterion of 2 cells.

Fig. 9 shows execution timeline for step 3, well separated cells interactions, for the above mentioned system on 8 processors. Fig. 10 presents performance results for the multipole part. We can see rapid performance degradation of the original algorithm (Fig. 4) with growing number of processors. Up to 16 processors our algorithm (Fig. 5) more efficiently hides the communication. However starting at 32 and more strongly at 64 processors the communication is of such volume that it cannot be handled anyway. To decrease the volume of communication we have decided to communicate multipole expansions in coefficient space instead of Fourier space and perform FFTs on the receiving processor. As Fig. 10 shows this way we are able to achieve much better scalability and the cost of extra FFTs is acceptable. Fig. 11 shows performance improvements from using the memory-saving technique and from message aggregation. Their effect is not big, but not insignificant either. Fig. 12 shows results for the direct part.

6. Implementation remarks

Building the inverted interaction list is not completely trivial. Because of periodic boundary conditions and depending on the cell separation criterion, the host cell may interact with the original neighbor cell as well as with a number of its periodic images. For the typical separation of 2 cells and the number of levels

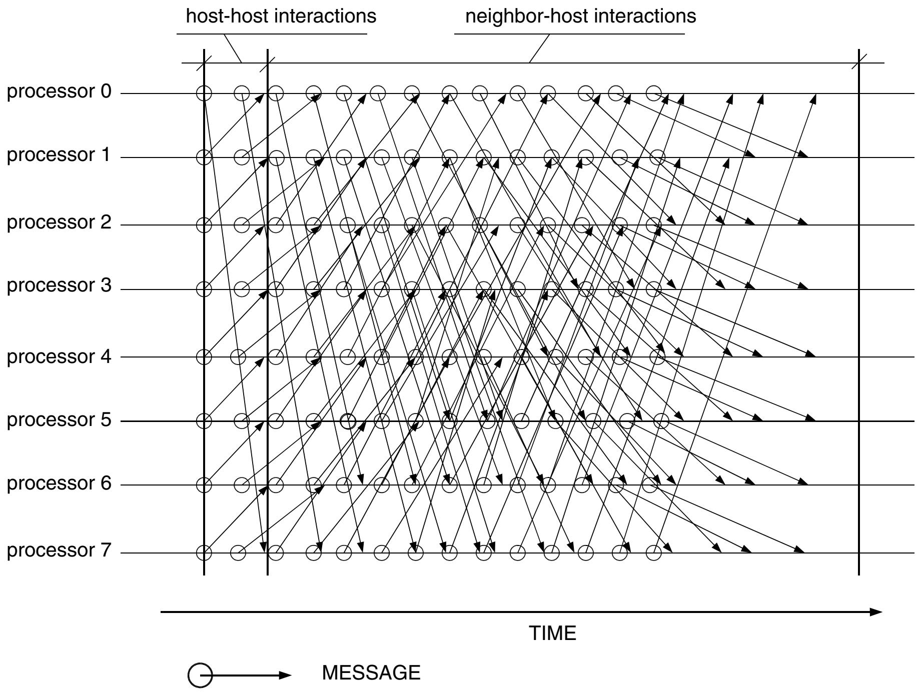

Fig. 9. Execution timeline of the multipole part of FMM with contention-free communication scheme (4 levels of spatial decomposition; message aggregation with message size of approximately 80 kB).

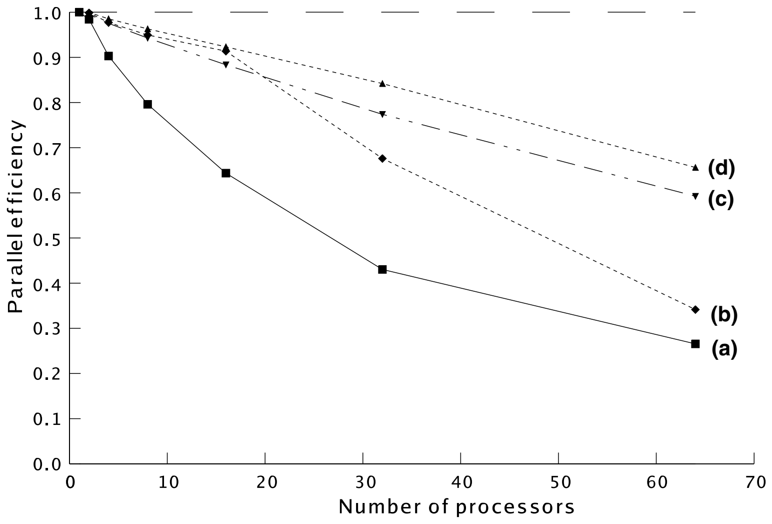

Fig. 10. Parallel efficiency relative to a single processor for the multipole part: (a) contention-prone scheme, Fourier space; (b) contention-free scheme, Fourier space; (c) contention-free scheme, coefficient space; (d) contention-free scheme, coefficient space, with the overhead of the redundant FFTs subtracted.

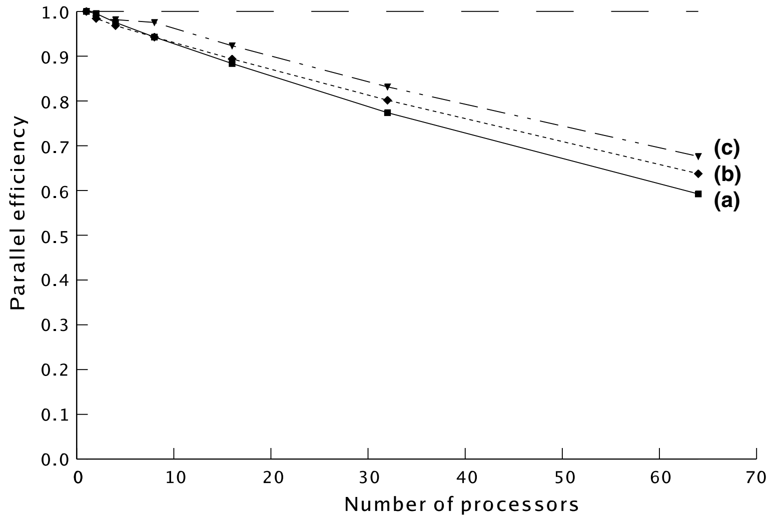

Fig. 11. Parallel efficiency relative to a single processor for the contention-free scheme for the multipole part with communication in coefficient space: (a) no memory saving, no aggregation; (b) memory saving, no aggregation; (c) memory saving, aggregation with maximum message size of 8 multipole expansions, what equals exactly 17 kB.

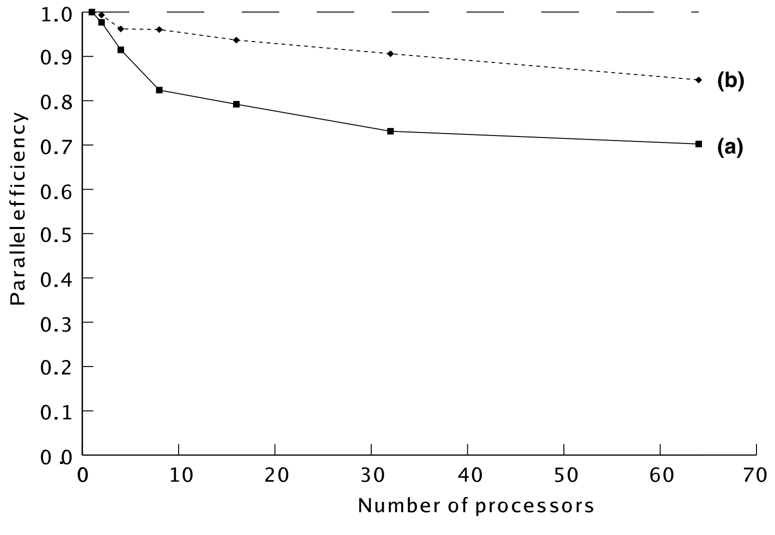

Fig. 12. Parallel efficiency relative to a single processor for the direct part: (a) contention-prone scheme; (b) contention-free scheme (no message aggregation).

greater than 3 this phenomenon will not occur for direct interactions. Also with cell separation of 2 and number of levels greater than 4 it will not occur for leaf-level multipole interactions. However it will always occur on higher levels of multipole interactions.

Using aggregation the sender will send the same cell data to many other processors, in fact, to each one in different configurations with other cells. It is impossible to send an aggregate message directly from a continuous memory block. However sending aggregate messages can be conveniently coded using the MPI mechanism of general data types.

Nevertheless, on the receiving processor the cells are always received in the same order, so we can allocate memory in such a way that the whole aggregated message will be received "in place" (which is the case in our implementation). The problem is in general irregular. We must take full advantage of all regularities. To this end it is essential that the communication scheme is coded using non-blocking persistent communication calls.

7. Conclusion

We have presented a simple idea that allows one to use the FMM more efficiently for smaller systems on more processors and also promises much better speedups for larger systems. Thanks to a few simple observations communication may proceed almost seamlessly in parallel with computation. More time is available for communication and contention is effectively prevented. The multipole and the direct part of FMM can enclose their own communication needs and be conveniently separated for multiple time stepping. The scheme has proved to be very robust on a number of different architectures. We will consider other architectures and problem scaling in future work.

The new communication scheme allows us to have just a few cells per processor which creates a new load balancing challenge. The total number of cells that participate in the multipole part is a sum of cells on all levels, but only the leaf cells participate in the direct part. If all calculations associated with one cell are assigned to a single processor we cannot balance the multipole calculations and the direct calculations at the same time and we are going to suffer high load imbalance even for uniform particle distributions. We also notice that the computational load per cell in the multipole part is constant, when the load in the direct part is strictly dependent on the particle distribution. In future we plan to load balance the multipole part and the direct part separately which also addresses the problem of load balancing when multiple time stepping is used.

We find that the performance of the FFT affects not only our serial performance, but also directly affects scalability and parallel efficiency of our algorithm due to the fact that we transmit data in coefficient space and perform redundant FFTs on the receiving side. There are efficient algorithms for performing non-power of two FFTs in the range of our interest (8–32) and more efficient algorithms to perform 2 dimensional FFT than to perform 1 dimensional FFT by rows and ten by columns [15,16]. Also we can take advantage of the fact that the multipole expansion has the form a triangular matrix. Finally we can try to utilize schemes that attempt to dynamically adapt to particular architecture [17]. There are adaptable library technologies which offer some of these features but not all. The FMM routine will usually be used as a library invoked from MD packages. If such code is data parallel or uses different spatial domain decomposition than the FMM presented here, the design of an efficient interface may become non-trivial.

Acknowledgments

The authors thank Dr. Dragan Mirkovic for stimulating conversations. The authors acknowledge the NIH, Robert A. Welch Foundation and the Texas Institute for Intelligent Bio-Nano Materials and Structures for Aerospace Vehicles, funded by NASA Cooperative Agreement No. NCC- 1-02038. Calculations were made possible through generous computing time allocations of Pittsburgh Supercomputing Center and San Diego Supercomputing Center.

References

[1] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325–348.
[2] L. Greengard, V. Rokhlin, On the efficient implementation of the fast multipole algorithm, Technical Report RR-602, Department of Computer Science, Yale University, 1988 (unpublished).

[3] W.D. Elliott, J.A. Board Jr., Fast Fourier transform accelerated fast multipole algorithm, SIAM J. Sci. Comput. 17 (2) (1996) 398–415.
[4] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numer. (1997) 229–269.
[5] T. Hrycak, V. Rokhlin, An improved fast multipole algorithm for potential fields, SIAM J. Sci. Comput. 19 (6) (1998) 1804–1826.
[6] W.D. Elliott, J.A. Board Jr., Revisiting the fast multipole algorithm error bounds, Technical Report TR94-008, Department of Electrical Engineering, Duke University, 1994 (unpublished).
[7] W.D. Elliott, Multipole algorithms for molecular dynamics simulation on high performance computers, Ph.D. Thesis, Department of Electrical Engineering, Duke University, 1995.
[8] C.G. Lambert, T.A. Darden, J.A. Board Jr., A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles, J. Comput. Phys. 126 (1996) 274–285.
[9] L. Greengard, W.D. Gropp, A parallel version of the fast multipole method, Comput. Math. Appl. 20 (7) (1990) 63–71.
[10] J.F. Leathrum Jr., J.A. Board Jr., The parallel fast multipole algorithm in three dimensions, Technical Report TR92-001, Department of Electrical Engineering, Duke University, 1992 (unpublished).
[11] W.T. Rankin, Efficient parallel implementations of multipole based N-body algorithms, Ph.D. Thesis, Department of Electrical Engineering, Duke University, 1999.
[12] H. Sagan, Space-Filling Curves, Springer, New York, 1995.
[13] S. Teng, Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation, SIAM J. Sci. Comput. 19 (2) (1998) 635–656.
[14] Y.C. Hu, S.L. Johnsson, S. Teng, A data parallel implementation of the geometric partitioning algorithm, in: Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, 1997.
[15] R. Tolimieri, M. An, C. Lu, Mathematics of Multidimensional Fourier Transform Algorithms, Springer, New York, 1997.
[16] R. Tolimieri, M. An, C. Lu, Algorithms for Discrete Fourier Transform and Convolution, Springer, New York, 1989.
[17] D. Mirkovic, R. Mahasoom, S.L. Johnsson, An adaptive software library for fast Fourier transform, in: Proceedings of the 14th ACM International Conference on Supercomputing, 2000, pp. 215–224.