Published in final edited form as:
*J Comput Chem*. 2011 November 15; 32(14): 3005–3013. doi:10.1002/jcc.21882.

# The Distributed Diagonal Force Decomposition Method for Parallelizing Molecular Dynamics Simulations

**Urban Boršnik**[1], **Benjamin T. Miller**[2], **Bernard R. Brooks**[2], and **Dušanka Janežič**[1,3,*]

[1]National Institute of Chemistry, Hajdrihova 19, SI-1000 Ljubljana, Slovenia

[2]National Heart, Lung, and Blood Institute, National Institutes of Health, 5635 Fishers Ln, Rockville, MD 20892-9314

[3]University of Primorska, Faculty for Mathematics, Natural Sciences, and Information Technology, Glagoljaška 8, SI-6000 Koper, Slovenia

## Abstract

Parallelization is an effective way to reduce the computational time needed for molecular dynamics simulations. We describe a new parallelization method, the distributed-diagonal force decomposition method, with which we extend and improve the existing force decomposition methods. Our new method requires less data communication during molecular dynamics simulations than replicated data and current force decomposition methods, increasing the parallel efficiency. It also dynamically load-balances the processors' computational load throughout the simulation. The method is readily implemented in existing molecular dynamics codes and it has been incorporated into the CHARMM program, allowing its immediate use in conjunction with the many molecular dynamics simulation techniques that are already present in the program. We also present the design of the Force Decomposition Machine, a cluster of personal computers and networks that is tailored to running molecular dynamics simulations using the distributed diagonal force decomposition method. The design is expandable and provides various degrees of fault resilience. This approach is easily adaptable to computers with Graphics Processing Units because it is independent of the processor type being used.

### Keywords

parallelization; parallel methods; force decomposition; parallel molecular dynamics simulations; dynamic load balancing

## 1 Introduction

Molecular dynamics (MD) simulations are an important tool in studying large macromolecular systems on an atomic level.[1,2] To allow the study of increasingly longer simulations of ever larger molecular systems, programs have been developed that use many processors in parallel to decrease the time that is required to perform an MD simulation.[3,4]

In parallel MD simulations, the calculation of the simulation is distributed among multiple processors of a computer. While the processors perform their calculations independently, they must exchange data at every simulation time step. Updated coordinates are needed to calculate forces or energies and updated forces are needed to update the new positions.[5] Since there are two data transfers per time step, the communication time should be

*Corresponding author dusa@cmm.ki.si.

minimized. Any time spent waiting for the updated data to be communicated before starting the next calculation leads to a decrease of the parallel efficiency $E_p$, which is defined as the ratio

$$E_p = \frac{T_s}{PT_p}, \tag{1}$$

where $T_s$ is the execution time of a program on a single processor and $T_p$ is the execution time of the program running on $P$ processors. The speedup $S$

$$S = \frac{T_s}{T_p} \tag{2}$$

expresses how much faster the program runs on $P$ processors than on one. Efficiency and speedup are related

$$E = \frac{S}{P}. \tag{3}$$

The ideal parallel programs has a speedup of $P$ on $P$ processors. In this case the maximum parallel efficiency is 1 (100%). In practice the overall efficiency tends to decrease when increasing the number of processors; however, in some applications a superlinear speedup has been observed with a greater number of processors due to the smaller data sizes present on individual processors.[6]

The most computationally demanding part of an MD simulation is the calculation of non-bonding forces. There are $O(N^2)$ non-bonding atomic pairs in a system of $N$ atoms. Excluding self-interactions results in at most $N(N-1)/2$ distinct atomic interactions. In many commonly used force fields,[7] the atoms of these atomic pairs mutually exert both van der Waals and electrostatic forces. Parallel methods therefore focus on effectively parallelizing the force computation of these non-bonded interactions so that the required communication time is minimized. Such methods are often used in conjunction with other techniques to decrease the computational complexity of force calculations from $O(N^2)$ to a lower order of $O(N \log N)$ or even $O(N)$[8,9] Often, MD simulations employ an energy cutoff distance beyond which the interaction of two atoms is defined to be zero[10] so that only a small fraction of the $N^2/2$ forces need to be calculated. However, when combined with parallelization, the use of a cutoff distance may give rise to processor load imbalance.[11] Since all processors must wait for the most loaded processor to finish its calculations before exchanging data, any load imbalance reduces the parallel efficiency. The possibilities for load balancing when using a cutoff distance is an important differentiating factor among the methods.[3,12]

Several parallel methods for MD simulations have been developed. They vary primarily in their data distribution among processors, which correspondingly dictates the data exchange patterns used for communication during an MD simulation. The simplest approach is the replicated data method,[13,14] in which every processor has a copy (replica) of the coordinates for all atoms. The energy and force calculations are divided among the processors and occur in parallel, but the results are continuously replicated among all of the processors. The replicated data method is the easiest to implement, but has the largest communication cost because all of the processors must exchange forces and coordinates for all atoms.

By contrast, the spatial decomposition method distributes the simulation space into segments and assigns a processor to every such segment.[11,15] The processor calculates all of the interactions among the atoms present in that segment. In addition, the interactions among atoms that are within the cutoff distance but are present in different segments must be computed. Thus the spatial decomposition method is best suited to simulations in which a short distance-based cutoff distance is applied since it limits the communication only to neighboring or nearby processors. Load balancing is non-trivial due to edge effects and inhomogeneities in the system density throughout the simulation space.[12,16]

The force decomposition method distributes the calculation of interactions by partitioning the set of all atoms into disjoint sets called blocks. Processors are assigned to calculate the interactions among the pairs of blocks.[17,18] A processor exchanges data only with a limited set of other processors assigned to the same two blocks, and the data is limited to only one block of atoms. The communication requirement of the force decomposition method scales with the inverse of the square root of the number of processors regardless of whether a cutoff distance is employed. For larger numbers of processors, the communication requirement is thus less than that of the replicated data model.

A new class of neutral-territory methods have recently been developed.[12,15,16,19–22] They combine the communication advantages of both force decomposition and spatial decomposition. Most retain a communication bound that scales with the square root of the number of processors, yet they also take advantage of a cutoff distance if it is used to reduce communication to neighboring cells.

Besides better algorithms and improved parallelization strategies, an often complementary approach to increasing parallel efficiency is to use hardware that is tailored to the algorithms. One such way is to decrease communication time by using faster communication hardware[23–26] or even specialized hardware for the problem domain, such as the MD-GRAPE chip and computer[27] or the Anton machine.[26] Increasing the coupling between the parallelization method and the underlying parallel computer is an effective way to increase parallel efficiency.[28] However, even clusters of PCs are flexible enough to allow adapting the parallel computer to the parallel method being utilized.[29–31] An example of this is the recent use of Graphics Processing Units (GPUs) to speed up MD.[32] However, taking full advantage of GPUs can require adapting the necessary algorithms to the hardware being utilized.[33] Therefore, adapting hardware and software to one another is becoming increasingly important to driving performance improvements for MD simulations.

The current trend in parallel computing architecture is a two-tier parallelism of shared and distributed memory. Nodes have many processor cores that share node-local memory. Parallelism within a node can be expressed using either shared-memory or distributed-memory programming techniques. The nodes are then interconnected into a distributed-memory parallel computer.

In this article we introduce the distributed diagonal force decomposition (DDFD) method.[34] The method, which is an improvement over the conventional Force Decomposition method, is explained in section 2. Its implementation in the CHARMM program[35] is described, demonstrating its ease of integration into an existing MD code and the performance improvements it provides when used to parallelize the MD simulation techniques already present. In section 3 the force decomposition machine (FDM) is introduced and its expandability and fault tolerance described. It should be noted that the FDM does not depend on any particular hardware architecture, and individual nodes can be built with any type of processor, including GPUs. In section 4 the performance of the DDFD method is compared to the replicated data method on standard clusters.

## 2 The Force Decomposition Method

The DDFD method achieves high parallel efficiency by assigning individual parallel processors to calculate the forces among limited subsets of all atoms. The method is derived from the basic force decomposition method.[11,17,18]

The force decomposition method relies on the decomposition of the force matrix of an atomic system into disjoint subsets and assigning processors to calculate the forces in these subsets. A force matrix represents the interactions (forces) among atomic pairs, as shown in Figs. 1a and 1b. A pair of atoms $a_i$, $aj$ has an interaction force $\vec{f}_{ij}$ and an equal but opposite force $\vec{f}_{ji} = -\vec{f}_{ij}$. The total force $\vec{f}_i$ acting on an atom $a_i$ is the sum of the forces acting upon it due to interactions with all other atoms:

$$\vec{f}_i = \sum_{j \neq i} \vec{f}_{ij}, \tag{4}$$

which is illustrated in Figs. 1c and 1d, which also show the equal but opposite forces between atomic pairs.

The set of all $N$ atoms is divided into $B$ disjoint sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_B \in \mathcal{B}$ referred to as blocks. Assuming that the atoms are evenly distributed into blocks, each block contains approximately $N/B$ atoms. The $N \times N$ force matrix of all $N^2$ pairwise interactions is thus divided into a $B \times B$ grid of block products $\mathcal{B}_1 \times \mathcal{B}_1, \mathcal{B}_1 \times \mathcal{B}_2, \ldots, \mathcal{B}_B \times \mathcal{B}_B$. Each of these $B^2$ block products contains the interactions among the atoms that are members of the two blocks. A unique processor is assigned to each of the block products. It is responsible for calculating the interactions in its block product $\mathcal{B}_i \times \mathcal{B}_j$.

The processor needs only to have the atomic data for the atoms in the two blocks $\mathcal{B}_i$ and $\mathcal{B}_j$, i.e., for only $2N/B$ atoms. Processors need only a limited data set, which limits the communication requirements. All of the processors in any single row share the same row block and similarly all of the processors in any one column share the same column block. A processor therefore needs to communicate only with the other processors sharing its two blocks and the communication is limited only to the atomic data of the $N/B$ atoms in a block. Since $P = B^2$, the volume of data that must be transferred is limited to the order of $O\left(1/\sqrt{P}\right)$.

The force matrix is antisymmetrical, so half of it need not be calculated. Various methods for avoiding double calculation in the force decomposition method have been described.[6,17,36–38] The originally published method uses permutation of the atomic order in the force matrix to reduce communication and calculates half of the interactions in a checkerboard pattern of the force matrix[17,18] and the Under Triangle Force Block Decomposition[36,38] calculates only the interactions below and on the diagonal of the force matrix.

We have focused on the Under Triangle Force Block Decomposition[38] that was previously implemented in the CHARMM program.[36] Its force matrix is shown in Fig. 2a. The distribution of atoms into blocks is the same for both the rows and columns. The upper triangle is implicitly calculated from the lower triangle and processors are therefore not assigned to the block products in the upper triangle. Therefore every block product is now assigned a unique set of interactions; however, the block products are not of the same size: while the off-diagonal block products in the lower triangle all contain $(N/B)^2$ interactions,

the diagonal block products contain only $(N/B)^2/2$ interactions. Processors assigned to the latter products therefore have only half as many calculations to perform as the the processors assigned to the former block products, which leads to load imbalance. In the new Distributed-Diagonal Force Decomposition method, which is based on the Under Triangle Force Block Decomposition method, we do not assign processors to the diagonal block products, but instead distribute the force calculations in these block products to other processors.

## Distributed-Diagonal Force Decomposition Method

Examining the block products of the Under Triangle Force Block Decomposition in Fig. 2a reveals that the diagonal block products, $\mathcal{B}_i \times \mathcal{B}_i$, $1 \le i \le B$, differ from the off-diagonal ones in the lower triangle, $\mathcal{B}_i \times \mathcal{B}_j$, $1 \le j < i \le B$, in that the former block products contain only interactions among the atoms belonging to a single block while the off diagonal block products contain interactions among atoms belonging to two different blocks. Each of the $B$ blocks $\mathcal{B}_i$ has one 1 diagonal block product, $\mathcal{B}_i \times \mathcal{B}_i$, and $B-1$ full block products $\mathcal{B}_i \times \mathcal{B}_j$, $j \neq i$ with the other $B-1$ blocks. For example, Fig. 2a shows the 4 block products for block $\mathcal{B}_3$: the diagonal block product $\mathcal{B}_3 \times \mathcal{B}_3$ (number 6), and the off-diagonal block products $\mathcal{B}_3 \times \mathcal{B}_1$, $\mathcal{B}_3 \times \mathcal{B}_1$, $\mathcal{B}_3 \times \mathcal{B}_2$, and $\mathcal{B}_3 \times \mathcal{B}_4$ (numbers 4, 5, and 9). Since a processor assigned to any of these $B - 1$ full block products by definition contains the atomic data for all atoms in block $\mathcal{B}_i$, the calculation of interactions from $\mathcal{B}_i \times \mathcal{B}_i$ can be moved to any of the $B-1$ processors assigned to the $B - 1$ off-diagonal full block products. This assignment can be done statically at the beginning of simulation, or can be used to load balance the calculation with no communication costs during a simulation, as is explained in a later section. Each atom is therefore assigned to one processor that calculates its interactions with the other atoms in the block. The assigned processor is called the home processor and is used to parallelize $O(N)$ per-atom calculations, such as integration.

Fig. 2b shows how the interactions from block product $\mathcal{B}_3 \times \mathcal{B}_3$ (number 6) are distributed among the the other block products (numbers 4, 5, and 9). This distribution of the interactions from the $\mathcal{B}_i \times \mathcal{B}_i$ diagonal block product is performed for each of the $B$ blocks; its effect is illustrated in Fig. 2c. Finally, processors are assigned only to the block products that are fully in the lower triangle, as shown by assignment of block products to 6 processors in the final step, Fig. 2d. Every processor $p$ has the coordinates of the atoms in two different blocks ($\mathcal{B}_i$ and $\mathcal{B}_j$, $\mathcal{B}_i \times \mathcal{B}_j$) and calculates all of the forces for the interactions among the atoms of the two different blocks ($\mathcal{B}_i \times \mathcal{B}_j$), as well as a part of the interactions among the atoms from the first block (a subset of $\mathcal{B}_i \times \mathcal{B}_i$) and a part of the interactions among the atoms from the second block (a subset of $\mathcal{B}_j \times \mathcal{B}_j$).

## Collective Operations

In standard MD, every time step requires two collective operations: summation and distribution of forces after the force calculation, as well as the broadcast of updated coordinates after the integration step.[5,14] In the DDFD method the two collective operations are implemented as $B$ separate operations on blocks of atoms. Since the blocks are disjoint, they can performed simultaneously. Because the operations are block-based, a processor communicates only with $2\times(B-2)$ processors sharing its two blocks.

During the calculation of the nonbonded energies and forces in standard MD, only data within a block must be exchanged. Communication is therefore limited to processors sharing the same block; however, for some parts of an existing program that is not fully parallelized, all data must be replicated among all processors. To achieve this, two classes of collective operations are defined:

**intra-block mode—**The standard DDFD operation in which data exchange is limited to single blocks. It replicates only the block data within that block or sums atomic data only within one block, respectively.

**inter-block mode—**The operation needed to supplement an intra-block collective operation to achieve an equivalent global, all-to-all collective operation among all processors. It provides full replication of all atomic data among all processors or the sum of atomic data from all processors. It is equivalent to the global collective operations seen in the replicated data method.

The inter-block operation uses the same communication pattern used for the intra-block operation but they differ in the data that is being communicated at each step. As a result, the subset of processors communicating with each other remains limited: only processors sharing a block communicate. The shared communication pattern between the two modes of operation is exploited by the topology of the force decomposition machine as described in section 3.

To fully replicate data, an inter-block operation is performed following the intra-block operation. The result of combining these two operations in sequence is the same as an equivalent all-to-all broadcast or summation among all processors in the replicated data parallelization approach. The benefit of defining the inter-block mode is that it only supplements the data transfer already achieved by an intra-block operation. Instead of invoking a global all-to-all collective operation involving all processors, the inter-block mode is used to achieve the equivalent result.

We have implemented the intra-block and inter-block collective sum and broadcast operations using the MPI point-to-point message transfer primitives. The collective operations of standard MPI message-passing libraries are blocking, which limits their ability to be used concurrently,[39] even though individual collective operations may be implemented efficiently for a given hardware platform. In order to keep our approach as general as possible, we have chosen to implement the collective operations by using non-blocking point-to-point message transfers among processors belonging to the same block.

## Performance Analysis

The DDFD method inherits the general performance characteristics of the force decomposition method. As such, its communication scales with $O\left(1/\sqrt{P}\right)$ on a parallel computer with $P$ processors. More specifically, given a parallel computer with $P$ processors (selected from the set of triangular numbers 1, 3, 6, 10, 15, …) and a molecular system with $N$ atoms, the atoms are evenly split into $B$ blocks of size $N/B$ atoms, such that

$$P=\frac{B(B-1)}{2}. \tag{5}$$

Each of the B blocks is assigned to a subset of B–1 processors and every processor belongs to exactly 2 blocks.

Data exchange occurs only within blocks during the intra-block collective operations that is used to sum forces and broadcast coordinates in every MD time step. For every one of the $B$ blocks, only its $B-1$ processors perform an intra-block collective operation (sum or broadcast) for only its subset of $N/B$ atoms. Therefore a processor communicates twice with $B-2$ other processors, exchanging a volume of $N/B$ data each time. Due to the independence of the blocks, the data exchanges can occur simultaneously. Including the assumption that a

processor can only participate in one collective operation at once, the communication time on any processor is therefore proportional to $2N/B = O(N/\sqrt{P})$.

Compared to other approaches, the DDFD method has lower communication requirements due to its higher ratio of blocks-per-processor ratio. For the same number of blocks $B$, the Under Triangle Force Block Decomposition method requires $P = B(B+1)/2$ processors[38] and the original force decomposition method requires $P = B^2$ processors,[18] while our approach needs only $B(B-1)/2$ processors. In the DDFD method only $B-1$ processors are involved in intra-block communication while in other methods $B$ processors are involved.

Assuming a full force matrix with no cutoff distance, then each processor calculates $c_d = (N/B)^2 + 2(N/B)^2/(2(B-1)) = (N/B)^2 + (N/B)^2/(B-1)$ interactions in the DDFD method. The first term is one full block product and the second term is the diagonal block product distributed among the $B-1$ processors in the block. Not distributing the diagonal means that $B-1$ blocks would have to be used for the same number of processors. Off-diagonal processors would then calculate $c_n = (N/(B-1))^2$ interactions while diagonal processors would calculate $(N/(B-1))^2/2$ interactions. There is a load imbalance between off-diagonal and diagonal processors. More importantly, compared to the DDFD method with the same number of processors, the computational load is greater by a factor of $c_n/c_d = B/(B-1)$ if a perfect distribution is assumed in both cases. The limit of the ratio is 1 with increasing processor counts but is non-negligible for practical processor counts. For example, even using 45 processors when $B = 10$ for the DDFD, not distributing the diagonal would lead to a 10% greater computational time.

## Load Balancing

The force calculation can be optimally balanced when forces are calculated for every interacting pair of atoms; however, when a distance-based cutoff is employed for force calculations, the force of any interaction between two atoms that are separated by a distance greater than the cutoff is defined to be zero.[5,10] The calculation of these forces is skipped. Since the number of non-zero interactions differs among the block products, the processors have different computational loads, which creates a load imbalance.

In the DDFD method, the act of distributing the diagonal moves interactions from diagonal block products $\mathcal{B}_i \times \mathcal{B}_i$ to full, non-diagonal, block products $\mathcal{B}_i \times \mathcal{B}_j, j \neq i$. Nominally, they are distributed equally among the $B-1$ processors belonging to block $\mathcal{B}_i$; however, the distribution need not be equal among the processors. By changing the placement of these interactions from the diagonal block products, it is possible to balance the processors' load. More interactions are moved to the block products having a smaller number of non-zero interactions, while block products with a higher number of non-zero interactions are avoided. By appropriately adjusting the diagonal redistribution, the load among the processors is evened out, thus negating the load imbalance. Load balancing in this way is possible only because there is a choice as to which processors the interactions from the diagonal block products can be distributed.

As the atoms move during an MD simulation, the distances between atomic pairs vary. When the atoms cross the cutoff distance boundary, the force matrix changes depending on whether the atoms moved apart or closer together. Due to these changes in the force matrix, a load imbalance may arise. The DDFD method allows dynamical load balancing to keep the computational load equal among processors.

In our implementation the load of a processor is modeled as the number of interactions calculated on a processor. All processors within a block count the load of all processors in the block. The placement of diagonal interaction calculations is then changed to balance the

load. Diagonal interaction calculations are moved from processors with greater loads to those with lesser loads. No communication is required in this implementation since all processors within a block perform the same operations. To change the processor that calculates an interaction is a matter of updating an array value that is replicated among all processors in a block.

The load balancing may be performed occasionally and not at every time step since the load imbalance does not change drastically within a few MD time steps. Since it inherently depends on the movement of atoms, we have coupled the load balancing to other non-bonding updates, the frequency of which is definable in the simulation input file.

### Non-Triangular Processor Counts

The DDFD method uses a number of processors from the set of triangular numbers 1, 3, 6, 10, 15, …, but computers often have a different number of processors. By varying the distribution of the diagonal block products, the DDFD method can be made to work with any number of processors, though with some loss of efficiency.

Instead of every processor belonging to two blocks, we allow a small number of processors to belong to just one block. For any number of available processors $P$, we find the largest $P'$ that is from the triangular set and is smaller or equal to $P$, i.e., for which $P \geq P' = B(B-1)/2$. The first $P'$ processors are assigned to the product of two blocks, according to the DDFD definition. The remaining $P-P'$ processors are only assigned to one block as shown in Fig. 3. These diagonal processors calculate intra-block interactions from their own blocks.

The load balancing techniques described in the preceeding section are used to delegate computations onto the $P-P'$ diagonal processors. Since the processor belong to only one block, they can only be assigned intra-block interactions. For simulations with a large cutoff in comparison to the simulation cell volume, the newly assigned diagonal processors have fewer calculations than the nondiagonal processors. This may lead to a load imbalance similar to the case when the diagonal is not distributed.

### Multi-Atom Interactions

The parallelization of the 2-D force matrix by the force decomposition method is based on, and optimized for, interactions between atomic pairs (pairwise interactions). A general MD program must also correctly handle other interaction types; for example the CHARMM force field also includes interactions among sets of three or more atoms[35] such as the bending potential of an angle formed by three atoms.

The dominance of the electrostatic and van der Waals pairwise interactions over the bonded interactions in the workload justifies the parallelization focus on atomic pairs. The relatively small number of other interaction types allows the data transfers for these interactions to be treated differently from the default data transfers.[26]

To correctly compute any type of interaction in the force field, the parallelization method must be able to map any interaction onto the force matrix in such a way that all of the atomic data needed to calculate the interaction is present together on a single processor. In the DDFD method, we select two atoms $a_i$, $a_j$ from the $n$, $n > 2$, atoms of a multi-atom interaction and select these to be the primary pair. The primary pair defines two base blocks $a_i \in \mathcal{B}_i$, $a_j \in \mathcal{B}_j$, which are not necessarily distinct. The block product $\mathcal{B}_i \times \mathcal{B}_j$ defines the processor $p$ responsible for the interaction calculation. The remaining $n-2$ atoms (orphan atoms) $a_k$, $a_k + 1$, $\ldots a_n$ are not necessarily members of either of the two base blocks $\mathcal{B}_i$ or

$\mathcal{B}_j$. Their coordinates must be transferred to processor $p$ and later the calculated forces must be summed with their corresponding home blocks.

To enable these supplementary data transfers, an orphan atom $a_k \in \mathcal{B}_k$ (and $\mathcal{B}_k$ is distinct from $\mathcal{B}_i$ and $\mathcal{B}_j$) is temporarily assigned to either of the two base blocks, e.g., $\mathcal{B}_i$. The processor $q$, assigned to block product $\mathcal{B}_i \times \mathcal{B}_k$, adds the atomic coordinates of atom $a_k$ to the set of coordinates of other atoms in block $\mathcal{B}_i$. They are then propagated along to processor $p$ during the regular intra-block data exchange for block $\mathcal{B}_i$. Processor $p$ can therefore calculate the interaction among the atoms $a_i$, $a_j$, $a_k$, … After the force calculation, the forces acting on atom $a_k$ are then summed back onto processor $q$ in the atom's home block in the reverse manner. Fig. 4 depicts the required data transfers for correctly treating orphan atoms.

The number of interactions that must be handled in this manner remains low and incurs only two additional transfers of $O(N/P)$ size in every MD time step. One is the transfer of the coordinates of the orphan atoms and occurs before the general coordinate distribution. The other is the transfer of calculated forces after the force calculation and before the general global force summation.

## 3 The Force Decomposition Machine

The force decomposition machine (FDM) is conceived as a network topology that would complement the distributed diagonal forces decomposition (DDFD) method. It uses the property of the DDFD method that all communication is local to a block. Processors communicate only with a limited subset of other processors sharing a block.

In general, for any switching interconnect technology, several switches with fewer ports are more affordable than a bigger switch with an equivalent number of ports, or the bandwidth available per unit price is greater. The network architecture of the FDM therefore uses a number of smaller switches to connect subsets of the processors. Any technology could be used, such as Ethernet or InfiniBand. One switch is used for each of the $B$ blocks and the $B-1$ processors of the block are connected to the switch. Every processor is connected to two switches. A total of $2P$ ports are necessary among all the switches. This is twice the $P$ ports that would be needed for directly connecting each processor to one bigger switch, but the advantage of the FDM is that it uses only $B$ smaller switches that each have at least $B - 1$ ports and the $B$ is on the order of the square root of $P$ since $P = B(B-1)/2$.

The topology of the FDM thus complements the data transfer characteristics of the DDFD method in which communication is block-based. All of the data transfers needed for an intra-block collective operation pass solely through the block's assigned switch. The topology of a 5-block, 10-processor FDM is illustrated in Fig. 5. This figure also highlights the switches and processors that take part in intra-block collective communication.

In this particular topology, there are only a specific number of processors that can be used, depending on the target number of atomic blocks. For $B$ blocks, the number of processors $P$ in the FDM is $P=\frac{B(B-1)}{2}$. The smallest practical number of processors is 3 (when 3 blocks are used). Other possible numbers of processors are 6, 10, 15, …. Particularly interesting are FDMs with 10 and 36 processors (5 and 9 blocks, respectively) because in these the number of processors in a block is a power of two. Having $2^n$ number of processors in a block simplifies the intra-block collective communication and allows for useful implementations using multi-CPU and multi-core computers.

The basic FDM uses only the *B* switches, each with *B*−1 connections; however, it is useful for a FDM to also use an additional larger common switch (or several bridged switches) to which all of the *P* processors are connected. The purpose of this switch is to provide default connectivity from external computers and to ease the administration of the computers. Its presence or performance does not affect the performance of MD simulations using the DDFD method.

### Cluster expandability and fault tolerance

The FDM uses many smaller switches instead of relying on a single larger multiport switch. This confers several unique advantages to the cluster, including expandability, a degree of fault-tolerance, and greater affordability.

Smaller Ethernet networking switches usually have 8, 12, or 16 ports. The number of ports ultimately limits the number of processors that can be used with them in the FDM. For example, if 8-port switches are used, then at most 9 blocks can be used, which yields a cluster of 36 processors; 12-port switches limit the cluster size to 78 processors and 16-port switches limit it to 136 processors. If larger switches are used than required by the initial block sizes, then expanding the FDM cluster entails merely adding another switch, an appropriate number of computers, new connections connecting the new computers to the new switch, and one connection from each new computer to one existing switch. A 36-processor cluster that was built with 12-port switches is expanded to 45 processors by adding 9 processors and another switch. It can be expanded up to 78 processors in this manner. When the number of ports on a single switch limits expandability, a multilevel scheme may be used for each block.[30]

In addition to expansion, the FDM provides a measure of fault tolerance. The chances that any one processor fails increases with an increased level of parallelism. While currently a program using the DDFD method can not survive hardware failures during code execution, the ability to quickly recover the FDM from a failure without swapping any hardware is welcome. With advances in fault-tolerant software, such as checkpointing with fault-tolerant MPI or application-level support, such recovery while maintaining performance will prove essential.[40,41]

Additional standby processors must be available to enable failover operation. If a larger common networking switch is available, then the standby processors are also connected to the common networking switch along with all of the other FDM processors. When a processor fails, a standby processor is assigned to replace it. Then the processors that share the failed processor's two blocks update their networking routes to the failed processor's replacement via the external switch. More than one standby processor can be supported in this manner, but the communication contention in the common switch impacts the communication times and reduces parallel efficiency.

Another option to enable failover is to use an additional small standby switch. In this case a common switch is not needed. The standby processors (and only these) are all connected to the additional standby switch, which is in turn connected to every one of the per-block switches. If a processor fails, then a standby processor replaces it. It is reachable from all other processors, but all messages must now traverse through an additional switch, which imposes a slight latency penalty for communication. Additional standby processors are possible, but each additional failure leads to increasing contention on the standby switch.

### Implementation of the Force Decomposition Machine

For the pupose of testing the DDFD method, we have developed a force decomposition machine based on a cluster of personal computers (PCs).

The FDM is implemented as a 55-processor cluster based on 11 blocks. It is comprised of 55 3.06 GHz Intel Xeon CPUs, each in a separate computer, and connected with 1 Gb/s Ethernet technology. The 11 per-block switches are Linksys SRW2024 models and 55 Intel 82546GB dual-port PCI-X network cards are used to connect the computers to the switches. The operating system is the CentOS 4.3 Final Release, using the 2.6.9–42.0.2.ELsmp Linux kernel.

The FDM processor numbering order follows the scheme from Fig. 5 and the appropriate networking routes are defined in every computer to allow intra-block processor communication. Care must be taken to match the parallel library's processor ordering to the actual processor numbering order, otherwise the advantage of the FDM network topology is lost. For example, most MPI implementations have a way of specifying processor ordering when starting a parallel program. All of the computers are also connected to a multiport non-blocking gigabit Ethernet switch for administrative purposes.

## 4 Results and Discussion

The Distributed Diagonal Force Decomposition (DDFD) method as implemented in the CHARMM program[35] was compared to the performance of the replicated data (RD) method in the same program. The comparison of the DDFD and RD methods was performed on a standard PC cluster consisting of 64 nodes, each with dual Intel Xeon 3.06 GHz processors with 2 GB of RAM and connected via gigabit Ethernet. One processor per node was used for these benchmarks.

Two molecular systems were used to compare the different methods and clusters. The smaller system (system A) consists of a solvated carboxymyoglobin protein with 14,026 atoms total and the larger system (system B) consists of a solvated HIV-1 protease enzyme[42] with 54,212 atoms total. For both systems, 1000 steps of molecular dynamics simulation were run, preceeded by a standard CHARMM setup procedure for electrostatic and van der Waals computation and a short five step dynamics run. Timings given include the time taken by the setup procedure. Simulations of both systems used switching nonbonded interactions starting at 10Å with interaction and list cutoffs at 12Å and 14Å respectively.

MD simulations of the two systems A and B on the standard cluster were used to compare the DDFD and the RD methods. In this comparison of the methods, CHARMM version c33b2 and its internal CMPI hypercube-based communication[13] were used for the RD method simulations. For the DDFD simulations, the block-based DDFD communication was used in a custom version of CHARMM c32a2 and collective communication of non-atomic data was performed with the MPI implementation's collective communication routines. The MPICH 1.2.7[43] library was used in both codes. Fig. 6a shows the speed up and Fig. 6b shows the communication time of the simulations performed.

In both cases the new DDFD method, as implemented in CHARMM, required less communication time than the replicated data method. The communication time for the DDFD method increased much slower than for replicated data, and in the case of the larger system generally even decreased. As would be expected based on the lower communication time, the DDFD method yielded a higher speedup. This demonstrates the advantages of this method.

Since the DDFD method has lower communication times than the RD method, the DDFD is expected to have a higher parallel efficiency and speedup compared to the RD method. The data in Figure 6a clearly shows that for any number of processors the DDFD method's efficiency and speedup is higher. However, since communication is only part of the total simulation time, the differences in efficiency and speedup between DDFD and RD for the

whole simulation performance is not as drastic as for the communication time. For example, the DDFD communication times for system B are approximately 5–10% of the total running time for 6 or more processors. In addition, not all parts of the CHARMM program are fully parallel and there is no overlap between computation and communication. Although the efficiency of the DDFD method decreases with an increasing number of processors, the speedup continues to increase for all processor numbers tested, except for one drop when moving from 36 to 45 processors on the smaller system (A). A drop in the speedup marks the point where it is counterproductive to add more processors. For the target cluster sizes ranging up to 64 processors, the DDFD method provides a measurable speedup compared to the existing RD method.

## 5 Conclusions

Parallelization is an important tool in achieving faster molecular dynamics simulations. The newly described DDFD (distributed diagonal force decomposition) method is an efficient method that extends the standard force decomposition method. The DDFD method has a lower communication cost than the replicated data approach and allows dynamic load balancing throughout the MD simulation. As with all force decomposition methods, its best use is for simulations of atomic systems with inhomogeneous density or when a very large force cutoff distance, or none, is employed; very large atomic systems with short force evaluation cutoff distances might be better served using other parallelization methods.

The DDFD method can be easily implemented in existing codes for MD simulation and we have described its implementation in the CHARMM program. Thus, the existing MD methods and tools implemented in the CHARMM program are immediately usable in combination with the new DDFD method.

The force decomposition machine (FDM) is designed after the communication patterns of the DDFD method. The FDM uses standard, widely available components. It is easily expandable to a certain limit and offers several options to plan for quickly recovering from failed PCs in the cluster. The FDM is independent of processor architecture and can be constructed out of heterogenous nodes that include GPUs or other types of co-processors.

## Acknowledgments

The authors would like to acknowledge the financial support of the Slovenian Research Agency under grant No. P1-0002. This research was supported in part by the Intramural Research Program of the NIH, NHLBI. The authors would also like to thank Dr. Milan Hodošček for his valuable insights and Rishi P. Singh for assistance assembling the FDM.

## References

[1]. Allen, MP.; Tildesley, DJ. Computer Simulation of Liquids. Oxford University Press; New York: 1987.

[2]. Leach, AR. Molecular Modeling: Principles and Applications. Addison Wesley Longman Limited; Essex: 1996.

[3]. Phillips JC, Braun R, Wang W, Gumbart J, Tajkhorshid E, Villa E, Chipot C, Skeel RD, Kalé L, Schulten K. J. Comp. Chem. 2005; 26:1781. [PubMed: 16222654]

[4]. Shaw DE, Maragakis P, Lindorff-Larsen K, Piana S, Dror RO, Eastwood MP, Bank JA, Jumper JM, Salmon JK, Shan Y, Wriggers W. Science. 2010; 330:341. [PubMed: 20947758]

[5]. Frenkel, D.; Smit, B. Understanding Molecular Simulation: From Algorithms to Applications. Academic Press; San Diego: 2002.

[6]. Trobec R, Šterk M, Praprotnik M, Janežič D. Int. J. Quant. Chem. 2001; 84:23.

[7]. Cornell WD, Cieplak P, Bayly CI, Gould IR, Merz KM Jr. Ferguson DM, Spellmeyer DC, Fox T, Caldwell JW, Kollman PA. J. Am. Chem. Soc. 1995; 117:5179.

[8]. Barnes J, Hut P. Nature. 1986; 324:446.

[9]. Lambert CG, Darden TA, Board JA Jr. J. Chem. Phys. 1996; 126:274.

[10]. Loncharich R, Brooks B. Proteins: Struct. Funct. Genet. 1989; 6:32. [PubMed: 2608658]

[11]. Plimpton, S.; Hendrickson, B. Parallel Computing in Computational Chemistry. Mattson, TG., editor. American Chemical Society; 1995. p. 114-132.

[12]. Fitch, BG.; Rayshubskiy, A.; Eleftheriou, M.; Ward, TJC.; Giampapa, M.; Pitman, MC.; Germain, RS. SC 2006 Conference, Proceedings of the ACM/IEEE; 2006. p. 44

[13]. Brooks BR, Hodošček M. Chemical Design Auto. News. 1992; 7:16.

[14]. Borštnik U, Hodošček M, Janežič D. J. Chem. Inf. Comput. Sci. 2004; 44:359. [PubMed: 15032512]

[15]. Bowers K, Dror R, Shaw D. J. Phys. Conf. Ser. 2005; 16:300.

[16]. Bowers K, Dror R, Shaw D. J. Chem. Phys. 2006; 124:184109. [PubMed: 16709099]

[17]. Plimpton SJ. J. Chem. Phys. 1995; 117:1.

[18]. Plimpton SJ, Hendrickson BA. J. Comp. Chem. 1996; 17:326.

[19]. Snir, M. A note on N-body computation with cutoffs Technical report. IBM T. J. Watson Research Center; 2001.

[20]. Snir M. Theory Comput. Systems. 2004; 37:295.

[21]. Shaw DE. J. Comp. Chem. 2005; 26:1318–1328. [PubMed: 16013057]

[22]. Bowers K, Dror R, Shaw D. J. Chem. Phys. 2007; 221:303.

[23]. Sterling, T.; Becker, DJ.; Savarese, D. Proceedings, 24th International Conference on Parallel Processing; 1995. p. 11-14.

[24]. Boden NJ, Cohen D, Felderman RE, Kulawik AE, Seitz CL, Seizovic JN, Su W-K. IEEE Micro. 1995; 15:29.

[25]. Adiga NR, Blumrich MA, Chen D, Coteus P, Gara A, Giampapa ME, Heidelberger P, Singh S, Steinmacher-Burow BD, Takken T, Tsao M, Vranas P. IBM J. Res. Devel. 2005; 49:265.

[26]. Shaw DE, Deneroff MM, Dror RO, Kuskin JS, Larson RH, Salmon JK, Young C, Batson B, Bowers KJ, Chao JC, Eastwood MP, Gagliardo J, Grossman JP, Ho CR, Ierardi DJ, Kolossváry I, Klepeis JL, Layman T, McLeavey C, Moraes MA, Mueller R, Priest EC, Shan Y, Spengler J, Theobald M, Towles B, Wang SC. SIGARCH Comput. Archit. News. 2007; 35:1.

[27]. Narumi T, Susukita R, Ebisuzaki T, McNiven G, Elmegreen B. Mol. Sim. 1999; 21:401.

[28]. Grossman, JP.; Young, C.; Bank, JA.; Mackenzie, K.; Ierardi, DJ.; Salmon, JK.; Dror, RO.; Shaw, DE. Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/ Software codesign and system synthesis; ACM. 2008.

[29]. Reschke, C.; Sterling, T.; Ridge, D.; Savarese, D.; Becker, DJ.; Merkey, P. Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing; IEEE. 1996. p. 626-636.

[30]. Dietz, HG.; Mattox, TI. Extreme Linux track of the 4th Annual Linux Showcase. 2000.

[31]. Borštnik U, Janežič D. J. Chem. Inf. Model. 2005; 45:1600. [PubMed: 16309260]

[32]. Stone JE, Hardy DJ, Ufimtsev IS, Schulten K. J. Mol. Graph. 2010; 29:116.

[33]. Hardy DJ, Stone JE, Schulten K. Parallel Comput. 2009; 35:164. [PubMed: 20161132]

[34]. Borštnik, U. University of Ljubljana; 2007. Parallel Computer Simulations on Clusters of Personal Computers PhD thesis.

[35]. Brooks BR, Brooks CL, Mackerell AD, Nilsson L, Petrella R, Roux B, Won Y, Archontis G, Bartels C, Boresch S, Caflisch A, Caves L, Cui Q, Dinner A, Feig M, Fischer S, Gao J, Hodošček M, Im W, Kuczera K, Lazaridis T, Ma J, Ovchinnikov V, Paci E, Pastor RW, Post CB, Pu J, Schaefer M, Tidor B, Venable RM, Woodcock HL, Wu X, Yang W, York DM, Karplus M. J. Comp. Chem. 2009; 30:1545. [PubMed: 19444816]

[35]. Hwang Y, Das R, Saltz JH, Hodoscek M, Brooks BR. Computat. Sci. & Eng. 1995; 2:18.

[37]. Murty R, Okunbor D. Parallel Comput. 1999; 25:217.

[38]. Shu JW, Wang B, Chen M, Wang JZ, Zheng WM. Comput. Phys. Comm. 2003; 154:121.

[39]. Hoefler, T.; Lumsdaine, A.; Rehm, W. In proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07; IEEE Computer Society/ACM. 2007.

[40]. Bosilca, G.; Bouteiller, A.; Cappello, F.; Djilali, S.; Fedak, G.; Germain, C.; Herault, T.; Lemarinier, P.; Lodygensky, O.; Magniette, F., et al. Proceedings of the Supercomputing 2002 Conference; ACM/IEEE. 2002. p. 29-29.

[41]. Geist A, Gropp B, Kale L, Kramer B, Snir M. Int. J. High Perf. Comp. Appl. 2009; 23:374.

[42]. Eurenius KP, Chatfield DC, Brooks BR, Hodošček M. Int. J. Quant. Chem. 1996; 60:1189.

[43]. User's guide for mpich, a portable implementation of MPI. Gropp, WD.; Lusk, E. Mathematics and Computer Science Division. Argonne National Laboratory; 1996.

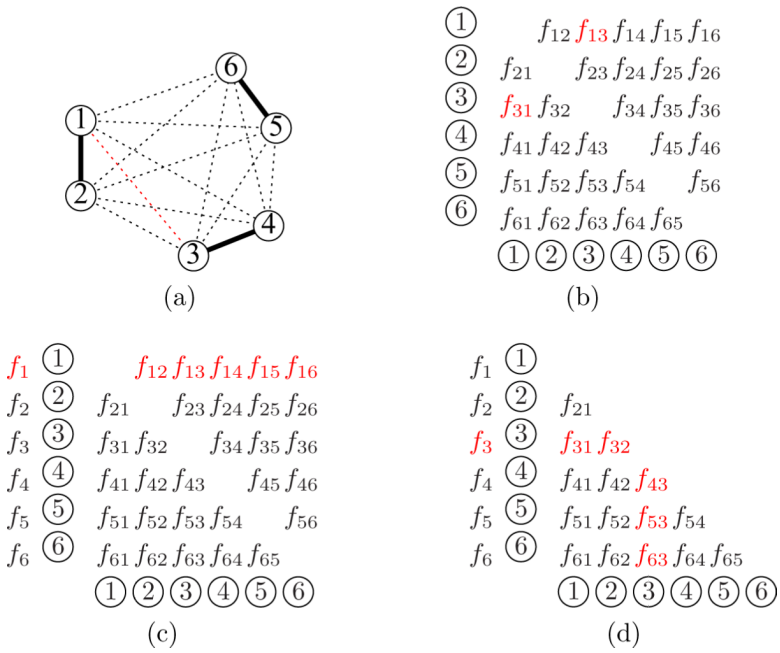


**Figure 1.**

Origin of the force matrix for an example system of 3 molecules with a total of 6 atoms. (a) The bonding interactions are shown with thick, solid lines and the non-bonding interactions with thin, dashed lines; the non-bonding interaction between atoms 1 and 3 is differentiated in red. (b) The force matrix corresponding to all of these interactions. The row and column labels correspond to atoms and every matrix element corresponds to an interaction between the atoms; for example, the interactions between atoms 1 and 3 are shown in red, $f_{13}$ being the force exerted by atom 3 on atom 1, while $f_{31}$ is the force atom 1 exerts on atom 3. (c) The total force acting on an atom is the sum of all partial forces due to individual interactions. The total force $f_1$ acting on the first atom is the sum of forces $f_{12}+ f_{13}+\cdots+ f_{16}$, shown in red. (d) Because every interaction has two equal but opposite forces, only one of them needs to be calculated. Force $f_3$ can be calculated as $f_{31}+ f_{32}- f_{43}- f_{53}- f_{63}$.

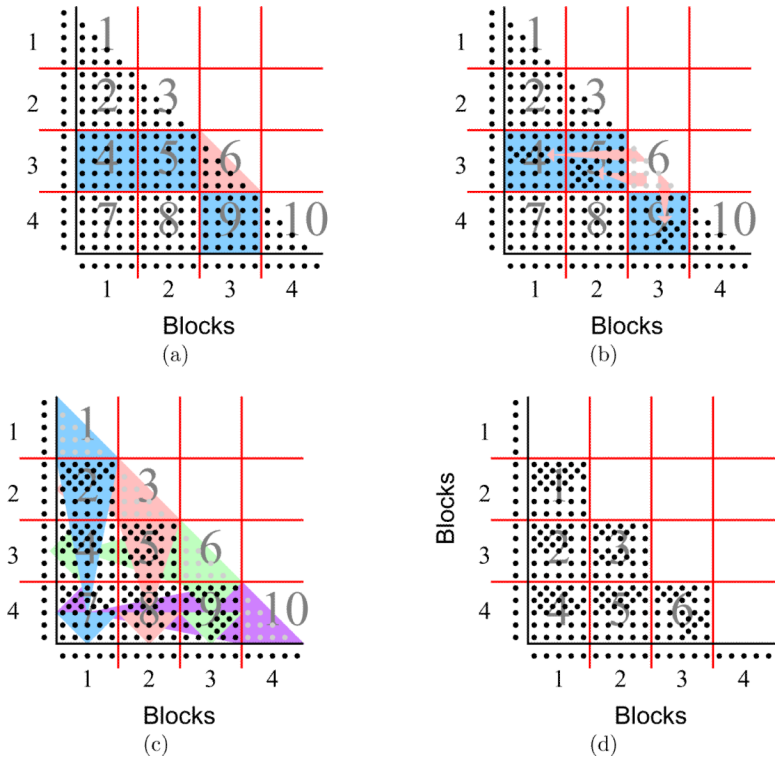


**Figure 2.**
Distribution of the diagonal in the Distributed-Diagonal Force Decomposition Method. Processor 6 has been assigned to calculate the intra-block interactions among the atoms of block 3 (a). However, any of the other processors 4, 5, 9 that share the same block 3, has all of the necessary data to calculate any of these intra-block interactions. The calculations of these interactions are therefore assigned to the three processors 4, 5, 9 (b), and processor 6 becomes superfluous. A similar distribution of interactions from the intra-block diagonal block products is performed (c), leaving the assignment of interactions to processors as shown in (d), where processors are assigned only to off-diagonal block products.

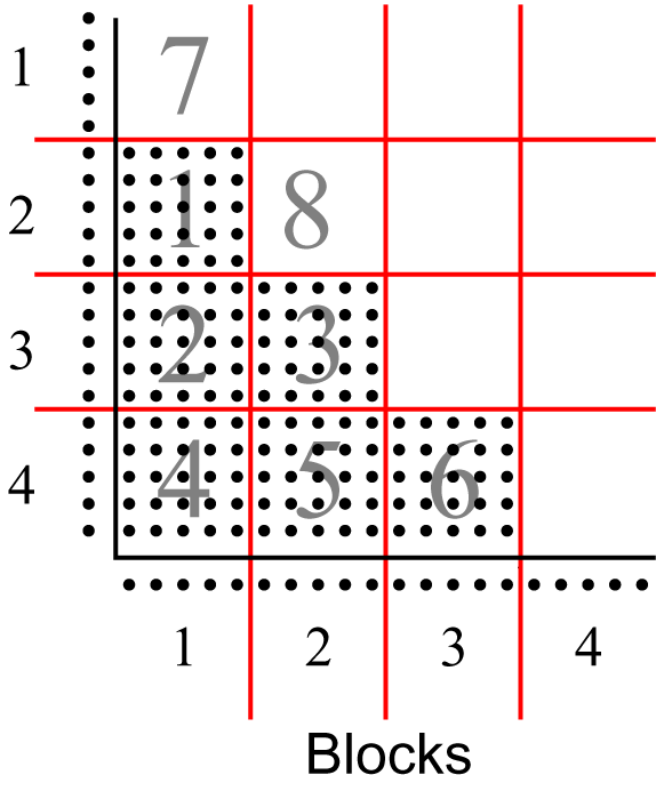


**Figure 3.**
Support for arbitrary processor counts. Extra processors that can not be assigned to two blocks, such as processors 7 and 8, are assigned to just one block. The load balancing algorithm assigns computations from the diagonal block product onto the extra processors.

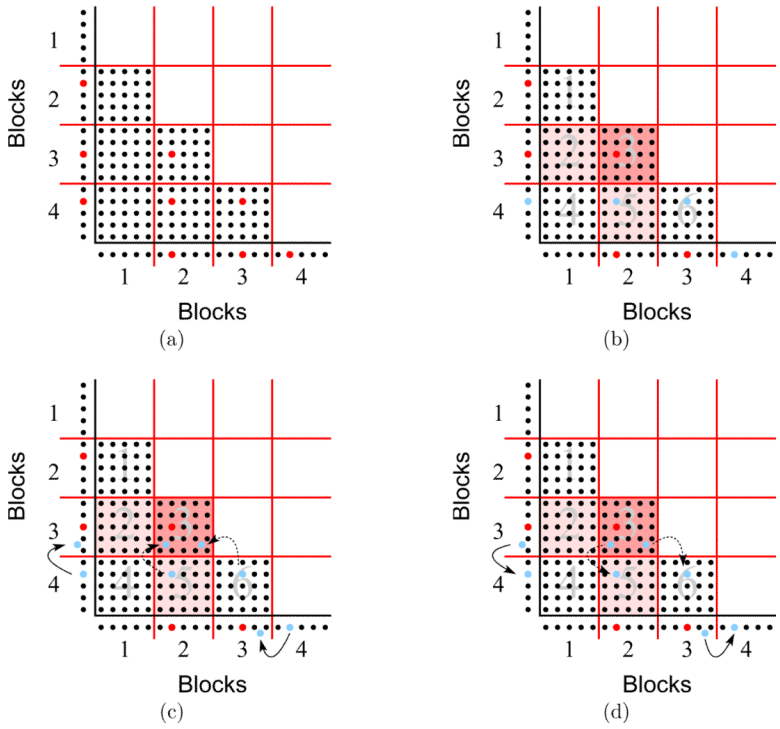


**Figure 4.**

The copying of data for multi-atom interactions. An interaction among three atoms in three distinct blocks, 2, 3, and 4, is depicted in (a). There is no block that holds all three atoms. Processor 3, assigned to the block product between blocks 2 and 3 is chosen to calculate the interaction. The two atoms in blocks 2 and 3 are colored red (as is their interaction), while the orphan atom in block 4 is colored blue (as is its interaction with the first two atoms) (b). Processor 3 has the coordinates for the two atoms from blocks 2 and 3, but it also needs the coordinates of the orphan atom from block 4. A processor (here processor 5) that is both in the orphan atom's block (block 4) and in either block 2 or 3 (here block 2) is selected. It sends the coordinates to processor 3 (c), in effect merging this interaction calculation onto processor 3. After the force calculation, the partial force acting on the orphan atom is copied back to processor 5 (d) so that the force contribution from the interaction on the orphan atom is appropriately added to the total force acting upon it.

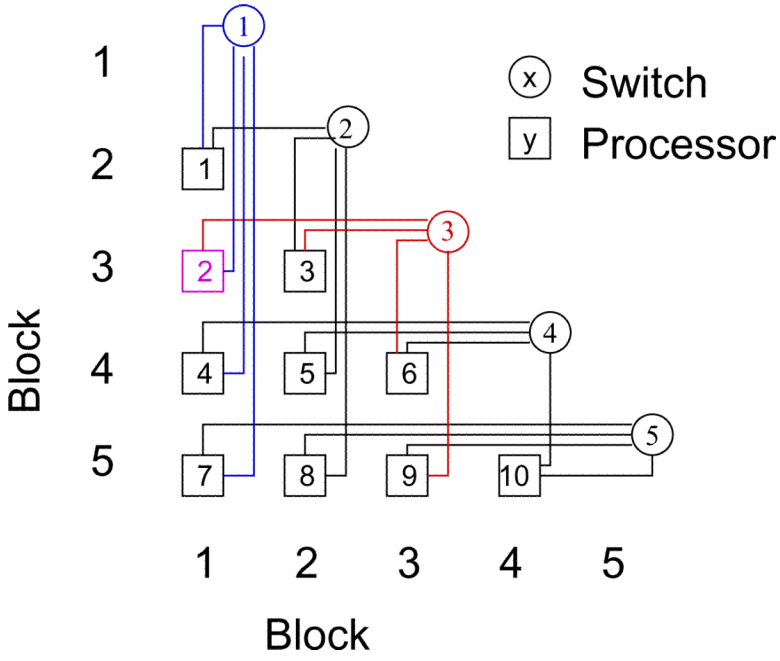


**Figure 5.**

A 5-block FDM. It has 10 processors (represented with squares) connected to 5 switches (represented with circles). Every one of the 5 blocks is assigned one of the 5 switches, to which 4 processors are connected. For example, processors 1, 2, 4, and 7, which belong to the first block, are connected to switch 1 (colored blue). Every processor is connected to two switches; e.g., processor 2, which calculates interactions among the atoms in blocks 1 and 3, is connected to the two appropriate switches, switches 1 and 3. A processor communicates only with the other processors in its two blocks: processor 2 (colored purple), for example, communicates only with the 6 other processors sharing its two blocks. It communicates with processors 1, 4, and 7, sharing block 1, using switch 1 (blue colors). It communicates with processors 3, 6, and 9, sharing block 3, using switch 3 (red colors).

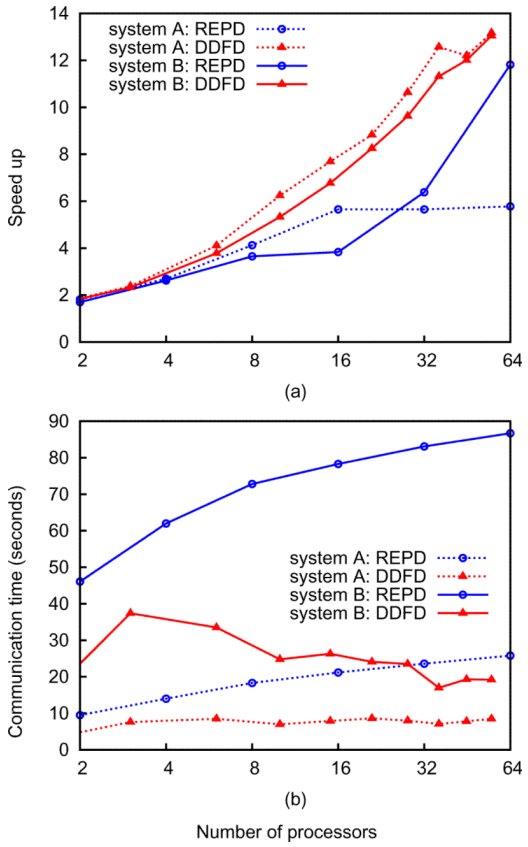


**Figure 6.**

Speedups (a) and communication times (b) for the standard cluster using CHARMM's replicated data method (REPD, blue lines with circles) and the DDFD method (DDFD, red lines with triangles) for systems A (dashed lines) and B (solid lines). The speedup for *N* processors was determined by dividing the wall time for one processor by the wall time for *N* processors.