J Math Chem (2009) 45:372–385
DOI 10.1007/s10910-008-9412-5

ORIGINAL PAPER

Interconnection networks for parallel molecular dynamics simulation based on hamiltonian cubic symmetric topology

Klavdija Kutnar · Urban Borštnik · Dragan Marušič · Dušanka Janežič

Published online: 31 July 2008

Abstract A class of interconnection networks for efficient parallel MD simulations based on hamiltonian cubic symmetric graphs is presented. The cubic symmetric graphs have many desirable properties as interconnection networks since they have a low degree and are vertex- and edge-transitive. We present a method for scheduling collective communication routines that are used in parallel MD and are based on the property that the graphs in question have a Hamilton cycle, that is, a cycle going through all vertices of the graph. Analyzing these communication routines shows that hamiltonian cubic symmetric graphs of small diameter are good candidates for a topology that gives rise to an interconnection network with excellent properties, allowing faster communication and thus speeding up parallel MD simulation.

Keywords Cubic symmetric graphs · Interconnection networks · Communication scheduling

1 Introductory remarks

Many current parallel computers for parallel computation, including MD simulation, use a variety of interconnection network topologies. The prevailing interconnection for Beowulf-type personal computer clusters is a network switch [1], which is based on a full graph, but the number of processor pairs that can concurrently communicate

K. Kutnar · D. Marušič
University of Primorska, FAMNIT, Glagoljaška 8, 6000 Koper, Slovenia

U. Borštnik · D. Janežič (✉)
National Institute of Chemistry, Hajdrihova 19, 1000 Ljubljana, Slovenia
e-mail: dusa@cmm.ki.si

D. Marušič
IMFM, University of Ljubljana, Jadranska 19, 1000 Ljubljana, Slovenia

is limited. Designing computer interconnection networks based on graph theory gives more efficient networks [2,3]. Some such networks are 2-dimensional meshes and tori [4–6], hypercubes [7,8], and others [9,10]. We have therefore decided to determine a class of interconnection networks that is tailored for parallel MD simulation, and that is suitable for a wide range of the number of processors.

Parallelization is an effective technique for speeding up MD simulations [11–14]. In parallel MD, the calculations are parallelized among many processors so that every processor performs only a part of the calculations. Ideally, using P parallel processors would result in a theoretical P-times speedup: the same calculation would be P-times as fast as on a single processor; however, mostly due to the communication required among processors, the speedup is less than P and the parallel efficiency is therefore less than 100%. Reducing the communication time increases the efficiency of parallel computation [15].

The design of a computer interconnection network patterned on the communication requirements of parallel processors leads to a reduced communication time [7,13,16]. To model computer interconnection networks with graphs topologies, the following correspondences are used: (1) Graph vertices model computer processors; and (2) Graph edges model connections between individual processors. In the design of interconnection networks we want to have a large number of processors without requiring a large number of connections at a single processor or incurring long delays in communication from one processor to another [17]. A desirable extra property of interconnection networks is that they should appear identical from any processor. This means that the graphs which describe interconnection networks should be vertex-transitive. In particular, the topologies considered in this paper are the hamiltonian cubic symmetric graphs from the Foster census [18,19] (for the definition see Sect. 2). It turns out that interconnection networks with such topologies have many attractive properties, such as a high degree of regularity, symmetry, and efficient communication of derived networks.

Due to practical limitations of computer hardware, the number of connections at a single processor should be as small as possible. Interconnection networks that have three connections at each processor seem to be a good choice. We will give a schedule of a communication (e.g., broadcasting) between processors in the interconnection network with a hamiltonian cubic symmetric topology. This schedule for broadcasting is based on the property that the graphs in question have a Hamilton cycle, that is, a cycle going through all vertices of the graph. In particular, we will use the fact that every hamiltonian cubic symmetric graph can be presented with the so-called LCF code [20]. Using this schedule we will then show that hamiltonian cubic symmetric graphs of small diameter are good candidates for a topology that gives rise to an interconnection network with desirable properties.

In this paper we present a class of interconnection networks for parallel MD simulations based on cubic symmetric graphs possessing a Hamilton cycle. In Sect. 2 we introduce the cubic symmetric graphs on which the interconnection networks are based. In Sect. 3 we present the communication requirements for parallel molecular dynamics simulations and present an algorithm to schedule parallel MD communication transfers on the interconnection networks based on the cubic symmetric graphs. In Sect. 4 we present the modeled communication requirements of parallel MD simulations on the studied interconnection networks and compare them to standard interconnection networks.

2 Cubic symmetric graphs

In this section we introduce cubic symmetric graphs which will be used throughout this paper for interconnection networks for parallel molecular dynamics simulations. By a graph we mean an undirected graph without loops or multiple edges. All graphs are assumed to be connected. For the graph theoretic terminology not defined here we refer the reader to the literature [21].

For adjacent vertices u and v in X, we write $u \sim v$ and denote the corresponding edge by uv. A graph is said to be *cubic* if all of its vertices are of degree 3, that is, every vertex has precisely three neighbors in the graph. A simple cycle that traverses every vertex exactly once is called a *Hamilton cycle* (Hamilton circuit). A graph is said to be *hamiltonian* if it possesses a Hamilton cycle.

Given a graph X we let $V(X)$, $E(X)$, $A(X)$ and Aut$X$ be the vertex set, edge set, arc set, and the automorphism group of X, respectively. For any vertices $v, w \in V(X)$, we let $d(v, w) = d(w, v)$ be the distance between v and w. The *diameter* of a graph X is defined by $diam(X) = \max\{d(v, w)|v, w \in V(X)\}$. The automorphism group Aut$X$ is said to be *vertex-transitive*, *edge-transitive*, and *arc-transitive* provided it acts transitively on the sets of vertices, edges, and arcs of X, respectively. A graph is said to be *vertex-transitive*, *edge-transitive*, and *arc-transitive* if its automorphism group is vertex-transitive, edge-transitive, and arc-transitive, respectively. Moreover, a graph is said to be *symmetric* if its automorphism group is vertex-transitive, edge-transitive and arc-transitive. The first result linking vertex and edge-transitivity to arc-transitivity is due to Tutte [22] who proved that a vertex-transitive and edge-transitive graph of odd degree is necessarily arc-transitive. Hence every cubic vertex-transitive and edge-transitive graph is also symmetric. Cubic symmetric graphs have spurred quite a bit of interest in the mathematical community resulting in extensive research using a variety of techniques from algebra, combinatorics, and topology [23–28].

In the computer interconnection networks, the diameter of a network based on a graph is the maximum internode distance; that is, it is the maximum number of links that must be traversed to send a message to any processor along a shortest path. The lower the diameter of a network the shorter the time needed to send a message from one processor to the processor farthest away from it. Therefore the topology of a network should be a graph with a small diameter. In view of the fact that the number of connections to a single processor should be as small as possible, we restrict our attention to computer interconnection networks based on cubic graphs. In particular, we investigate several cubic symmetric graphs from the Foster census [18,19] with a small diameter. Hereafter the notation FnA, FnB etc. will refer to the corresponding graphs in the Foster census of all cubic arc-transitive graphs [18,19] where the symbol FnA is sometimes conveniently shortened to Fn. In Fig. 1 the graphs F6, F8, F14, F16, and F20A are given. Figure 2 shows a computer interconnection network based on the complete graph $F4$.

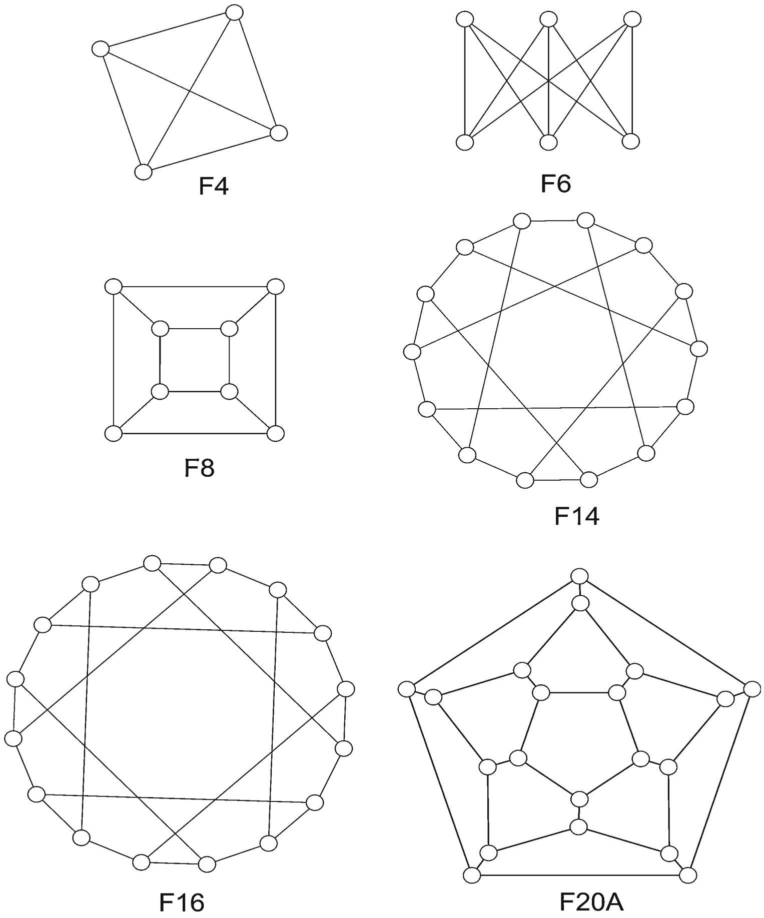
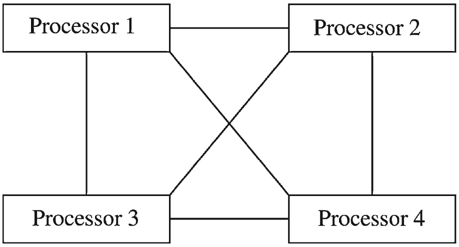


Fig. 1 The cubic symmetric graphs

Fig. 2 A computer interconnection network based on the F4 graph. The four processors correspond to the four vertices of the graph. The connections between the processors correspond to the edges between pairs of graph vertices



We will consider only the hamiltonian cubic symmetric graphs. (In fact, apart from the trivial example K_2, there are only four known connected vertex-transitive graphs that do not contain a Hamilton cycle.) All of the considered graphs can be represented with the LCF code (sometimes called the LCF notation after the authors' initials), which is a convenient notation devised for representing cubic hamiltonian

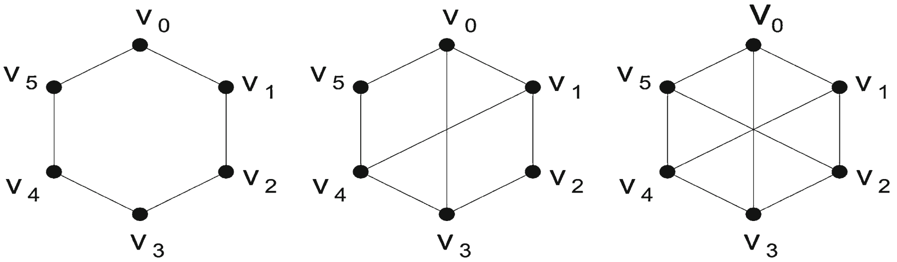


Fig. 3 The construction of the complete F6A graph from its LCF code, $[3, -3]^3$. We expand the code to $[3, -3, 3, -3, 3, -3]$ so that its length equals the graph order. We start with the Hamilton cycle, as shown in the first diagram. To each of the vertices we then add the number in the LCF code at the position of the vertex. In the middle diagram, we have connected vertex v_0 to its adjacent pair v_3 (because the sum of the number of the first vertex, 0, and the first number in the LCF code, 3, is equal to 3) and vertex v_1 to v_4 (because the sum of the number of the second vertex, 1, and the second number in the LCF code, −3, is equal to 4 modulo 6). The last diagram shows the completed graph with all of its edges

graphs [29,30]. The LCF code of a hamiltonian cubic graph relative to one of its Hamilton cycles $(v_0, v_1, \ldots, v_{n-1}, v_0)$ is a list LCF$[a_0, a_1, \ldots, a_{n-1}]$ of elements of $i \in Z_n \backslash \{0, 1, n-1\}$ such that v_i is adjacent to v_{i+ai} for every $i \in Z_n$ (Note that Z_n denotes the set of residues modulo n.) In addition, if there exists a proper divisor k of n such that $a_i = a_{i+rk}$ for all $i \in Z_n$ and $r \in \{1, 2, \ldots, n/k - 1\}$ then the notation is simplified to LCF$[a_0, a_1, \ldots, a_{n-1}]^{n/k}$.

Let us clarify the LCF code with an example. The F6A graph is specified with the LCF code LCF$[3, -3]^3$, which is expanded to $[3, -3, 3, -3, 3, -3]$ with six elements, equal to the graph order. Each of the six vertices, serially numbered v_0, v_1, v_2, v_3, v_4, and v_5, of the F6A graph is adjacent to the vertex given by the sum (modulo 6) of its vertex number and the LCF code at the corresponding position: e.g., the first vertex v_0 is adjacent to the vertex $v_{0+3} = v_3$, since the first number in the LCF code is 3. The second vertex v_1 is adjacent to vertex $v_{1-3} = v_4$, and so on. Or, the expanded code $[3, -3, 3, -3, 3, -3]$ is added element-wise to the sequential vector [0,1,2,3,4,5] to yield the sum [3,4,5,0,1,2]. The vertices in corresponding positions of the sequential vector and the sum are adjacent: 0 and 3, 1 and 4, 2 and 5. The generation of graph edges from the LCF code is illustrated in Fig. 3.

In Table 1 we give the LCF codes of cubic symmetric graphs that we have considered.

3 Communication in parallel molecular dynamics simulation

The calculation of MD simulation is inherently serial: the sequence of time steps must be calculated in consecutive order since the results of the current step are needed to calculate the proceeding time step [31]. In parallel MD, the calculations in each time step are parallelized. Every MD step consists of two parts: one is the calculation of forces and the other is the calculation of new atomic coordinates. Data from one part is needed for the next: the total forces acting on an atom must be known to calculate the new coordinates; then, the updated coordinates must be known to calculate the forces

Table 1 The LCF codes of cubic symmetric graphs that have been considered as interconnection networks

Graph	Vertices	LCF code
F004A	4	$\mathrm{LCF}[2, -2]^{2}$
F006A	6	$\mathrm{LCF}[3, -3]^{3}$
F008A	8	$\mathrm{LCF}[3, -3]^{4}$
F014A	14	$\mathrm{LCF}[5, -5]^{7}$
F016A	16	$\mathrm{LCF}[5, -5]^{8}$
F018A	18	$\mathrm{LCF}[5, 7, -7, -5, -7, 7]^{3}$
F020A	20	$\mathrm{LCF}[10, 7, 4, -4, -7, 10, -4, 7, -7, 4]^{2}$
F020B	20	$\mathrm{LCF}[-5, 9, 5, -9]^{5}$
F024A	24	$\mathrm{LCF}[5, -9, 7, -5, 9, -7]^{4}$
F026A	26	$\mathrm{LCF}[7, -7]^{13}$
F030A	30	$\mathrm{LCF}[-7, 9, 13, -13, -9, 7]^{5}$
F032A	32	$\mathrm{LCF}[-5, 13, -13, 5]^{8}$
F038A	38	$\mathrm{LCF}[15, -15]^{19}$
F040A	40	$\mathrm{LCF}[15, 9, -9, -15]^{10}$
F042A	42	$\mathrm{LCF}[9, -9]^{21}$
F048A	48	$\mathrm{LCF}[-7, 9, 19, 7, -9, -19]^{8}$
F050A	50	$\mathrm{LCF}[9, 11, -11, 11, -11, 11, -11, 11, -11, -9]^{5}$
F054A	54	$\mathrm{LCF}[-11, 11, 13, -13, -11, 11]^{9}$
F056A	56	$\mathrm{LCF}[11, 13, -13, -11]^{14}$
F060A	60	$\mathrm{LCF}[12, -17, -12, 25, 17, -26, -9, 9, -25, 26]^{6}$
F112C	112	$\mathrm{LCF}[11, -43, 43, -11]^{28}$
F126A	126	$\mathrm{LCF}[-47, 47, 49, -49, -47, 47]^{21}$
F152A	152	$\mathrm{LCF}[19, 21, -21, -19]^{38}$
F168A	168	$\mathrm{LCF}[5, -9, 55, 65, 9, -5, 5, -9, -65, -55, 9, -5]^{14}$
F208A	208	$\mathrm{LCF}[43, -59, 59, -43]^{52}$
F224A	224	$\mathrm{LCF}[-5, 45, 51, 5, -5, -51, -45, 5]^{28}$
F234A	234	$\mathrm{LCF}[41, 43, -43, 43, -43, -41]^{39}$
F248A	248	$\mathrm{LCF}[67, 69, -69, -67]^{62}$
F296A	296	$\mathrm{LCF}[27, 29, -29, -27]^{74}$
F304A	304	$\mathrm{LCF}[-21, 133, -133, 21]^{76}$
F312A	312	$\mathrm{LCF}[5, -9, 79, 89, 9, -5, 5, -9, -89, -79, 9, -5]^{26}$
F336C	336	$\mathrm{LCF}[-163, -9, 103, -55, 9, 163, -163, -9, 55, -103, 9, 163]^{28}$
F342A	342	$\mathrm{LCF}[-29, 29, 31, -31, -29, 29]^{57}$
F344A	344	$\mathrm{LCF}[131, 133, -133, -131]^{86}$
F350A	350	$\mathrm{LCF}[-29, 29, 31, -31, 31, -31, 31, -31, -29, 29]^{35}$
F378A	378	$\mathrm{LCF}[-173, 173, 175, -175, -173, 173]^{63}$
F392A	392	$\mathrm{LCF}[179, 181, -181, -179]^{98}$
F416A	416	$\mathrm{LCF}[-5, 173, 179, 5, -5, -179, -173, 5]^{52}$

The leading zeros after the initial F may be omitted: the notation $F006A$ and $F6A$ refer to the same graph

on atoms. In parallel MD simulation, in which the calculations—and the results—are distributed among processors, the appropriate calculated forces and coordinates must be transferred among the processors so that the following part of the MD calculation has the required data.

The calculation of the forces is the most computationally expensive part of an MD simulation [32]. As a specific case of the general N-body problem, it must account for the N^2 interactions among the N atoms of the molecular system [33].

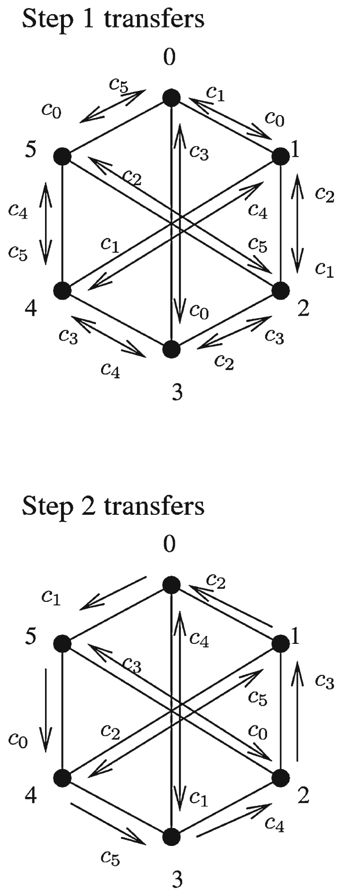
Several parallelization methods have been developed for calculating the force in parallel [32,34–36]. In all of the parallel methods for MD simulation, each processor calculates a part, N^2/P, of the interactions among atoms; collectively, the sum of these interaction calculations yields the total forces on the atoms, the same as would be calculated by a single processor. The calculation of new atomic coordinates is not as computationally demanding as calculating the forces, but can also be effectively parallelized: every processor is assigned a disjoint subset of N/P atoms and must update the coordinates of these atoms. Performing the sum of the forces after the force calculation, when the results are scattered among the processors, and the broadcast of the updated atomic coordinates after their calculation form the bulk of the data transfer in parallel MD.

3.1 Global broadcast

The communication pattern for the *global broadcast* operation arises from the need to have updated atomic coordinates on every processor. After the calculation of new atomic coordinates, the results are scattered among the processors: a processor has the updated coordinates only for the atoms for which it has calculated the coordinates, i.e., the coordinates of the atoms that are assigned to it. Every processor i must therefore broadcast c_i, the updated coordinates for its N/P atoms, to the other P-1 processors. A routine in the standard parallel library MPI that performs this all-to-all broadcast is named MPI_allgatherv [37] but for simplicity we shall refer to it as the *global broadcast* operation. An example of the broadcast operation is shown in Fig. 4. In the example from Fig. 4, before the operation every processor has the updated coordinates, c_i, for its subset of $N/6$ atoms. Upon completion of the operation, every processor must have the complete set of coordinates $\{c_0, c_1, c_2, c_3, c_4, c_5\}$ for all N atoms. In the first step, a processor exchanges its coordinates with all three neighbors; e.g., at the end of the first step processor 0 would have the data set of coordinates $\{c_0, c_1, c_3, c_5\}$. In the second step, a processor i receives from its neighbors $i + 1$ and $i + 3$ the coordinates from their neighbor $i + 1$ (i.e., c_{i+2} and c_{i+4}) and sends the appropriate coordinates to its two neighbors. (Processor numbers are taken as modulo 6.) For example, processor 0 receives coordinates c_2 from processor 1 and c_4 from processor 3 and sends c_1 to processor 5 and c_1 to processor 3. After this second step, the *global broadcast* operation is complete and all of the processors have the complete data set of coordinates for all N atoms.

3.2 Global sum

The communication pattern for the *global sum* operation arises from the need of a processor to have, after the force calculation, the total force acting on the N/P atoms that are assigned to it, so that it can then calculate the new coordinates for these atoms. After the parallel force calculation, every processor has as a result partial forces acting on all N atoms. Let f_n^i denote the partial force calculated by processor i acting on atoms assigned to processor n. After the force calculation, each processor i has a set of such partial forces $\{f_0^i, f_1^i, \ldots, f_P^i\}$. To be able to calculate new atomic coordinates,


Step 1 transfers

Step 2 transfers

Initial state

Processor	Data present
0	$\{c_0\}$
1	$\{c_1\}$
2	$\{c_2\}$
3	$\{c_3\}$
4	$\{c_4\}$
5	$\{c_5\}$

State between steps 1 and 2

Processor	Data present
0	$\{c_0, c_1, c_3, c_5\}$
1	$\{c_0, c_1, c_2, c_4\}$
2	$\{c_1, c_2, c_3, c_5\}$
3	$\{c_0, c_2, c_3, c_4\}$
4	$\{c_1, c_3, c_4, c_5\}$
5	$\{c_0, c_2, c_4, c_5\}$

Final state

Processor	Data present
0	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$
1	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$
2	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$
3	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$
4	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$
5	$\{c_0, c_1, c_2, c_3, c_4, c_5\}$

Fig. 4 The *global broadcast* operation on the F6A graph. The three tables on the right show the data that the six processors numbered 0–5 have before the start of the *global broadcast* operation, between the two steps, and after the completed operation. The diagrams on the left indicate the data transfers that occur at the two steps of the *global broadcast* operation, with arrows from the sender to the receiver and the transferred data written by the arrow head. Before the operation, every processor i, $0 \leq i \leq 5$ has the updated coordinates of the atoms that are assigned to it, c_i. After the operation, every processor has the updated coordinates of all of the N atoms, the complete set $\{c_0, c_1, c_2, c_3, c_4, c_5\}$

it must have the total force $\mathbf{F_i}$ acting on its assigned atoms, which is the sum of the partial forces $\mathbf{F_i} = \{f_i^0 + f_i^1 + \cdots + f_i^P\}$ acting on its assigned atoms; these partial forces are scattered among the other processors. The standard MPI library routine MPI_reduce_scatter performs the a sum-and-distribute operation that we call the *global sum* [16]. The *global sum* operation, which is the opposite from that of the *global broadcast* operation, is explained in Fig. 5. In the example with six processors, before the sum operation, a processor i has the partial forces $\{f_0^i, f_1^i, f_2^i, f_3^i, f_4^i, f_5^i\}$ for all N atoms. In the first step, a processor i sends its partial forces f_{i+2}^i to its neighbor $i + 1$ and f_{i+4}^i to its neighbor $i + 3$. Processor 0, e.g., would now have the partial sum $\{f_0^0, f_1^0 + f_1^3 + f_1^5, f_2^0, f_3^0, f_4^0, f_5^0\}$. In the next step, a processor i sends the partial sum f_{i+3}^i to its neighbor $i + 3$, f_{i-1}^i to $i - 1$, and the partial sum $f_{i+1}^i + f_{i+1}^{i-1} + f_{i+1}^{i+3}$ to $i + 1$. At the conclusion of this last step, every processor i has $\mathbf{F_i}$, the total sum of forces for all of the N/P atoms that are assigned to it. They can all now calculate the new atomic positions.

Both the *global broadcast* and *global sum* operations are implemented as a series of individual message exchanges between two processors, as illustrated in Figs. 4 and 5. These messages contain either the coordinate data c_i or the partial forces f_n^i. The processor-to-processor message exchanges can occur in parallel, since the connections between processors are physically separate connections. Also, full-duplex communication is assumed, so that two processors can concurrently send and receive. To achieve the lowest communication time, the order in which processors exchange which data must be tailored to the topology connecting the processors [38].

After introducing the *global broadcast* and *global sum* operations we can develop an algorithm, shown in Fig. 6, for scheduling individual data transfers among processors. For any computer interconnection network based on hamiltonian cubic symmetric graphs, the algorithm prepares a schedule that the processors follow to complete a *global broadcast* or *global sum* operation.

The input to the algorithm is a the LCF code of the graph of the computer network. The output of the algorithm is the broadcast schedule for the graph. It is a sequence of messages exchanges to be performed, including the step of the operation, the source and destination processors, and the data to be transferred. The data transfers for the *global broadcast* and *global sum* operations shown in examples on Figs. 4 and 5 were generated by this algorithm. The same, yet reversed, schedule is used for the sum operation. The steps are taken in descending order and the sending and receiving processors are reversed, while the data to be transferred refers to the partial forces instead of the coordinates. The processor sums the receiving forces to the partial sums it already has.

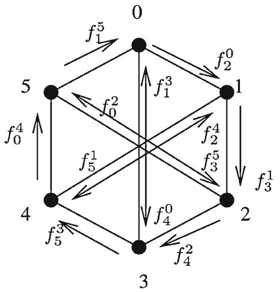
4 Analysis of communication requirements

We have used the algorithm from Fig. 6 to calculate the communication requirements of the *global broadcast* and *global sum* operations. The modeled time is governed by two variables: the *latency*, which is the delay between the time when the sending processor begins to send a message and the time when the receiving processor begins to

Initial state

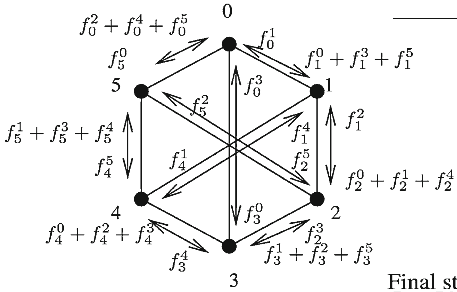
Processor	Data present
0	$\{f_0^0, f_1^0, f_2^0, f_3^0, f_4^0, f_5^0\}$
1	$\{f_0^1, f_1^1, f_2^1, f_3^1, f_4^1, f_5^1\}$
2	$\{f_0^2, f_1^2, f_2^2, f_3^2, f_4^2, f_5^2\}$
3	$\{f_0^3, f_1^3, f_2^3, f_3^3, f_4^3, f_5^3\}$
4	$\{f_0^4, f_1^4, f_2^4, f_3^4, f_4^4, f_5^4\}$
5	$\{f_0^5, f_1^5, f_2^5, f_3^5, f_4^5, f_5^5\}$

Step 1 transfers



State between steps 1 and 2

Processor	Data present
0	$\{f_0^0, f_1^0 + f_1^3 + f_1^5, f_2^0, f_3^0, f_4^0, f_5^0\}$
1	$\{f_0^1, f_1^1, f_2^0 + f_2^1 + f_2^4, f_3^1, f_4^1, f_5^1\}$
2	$\{f_0^2, f_1^2, f_2^2, f_3^1 + f_3^2 + f_3^5, f_4^2, f_5^2\}$
3	$\{f_0^3, f_1^3, f_2^3, f_3^3, f_4^0 + f_4^2 + f_4^3, f_5^3\}$
4	$\{f_0^4, f_1^4, f_2^4, f_3^4, f_4^4, f_5^1 + f_5^3 + f_5^4\}$
5	$\{f_0^2 + f_0^4 + f_0^5, f_1^5, f_2^5, f_3^5, f_4^5, f_5^5\}$



Final state

Processor	Data present
0	$\{\mathbf{F_0}, f_1^0 + f_1^3 + f_1^5, f_2^0, f_3^0, f_4^0, f_5^0\}$
1	$\{f_0^1, \mathbf{F_1}, f_2^0 + f_2^1 + f_2^4, f_3^1, f_4^1, f_5^1\}$
2	$\{f_0^2, f_1^2, \mathbf{F_2}, f_3^1 + f_3^2 + f_3^5, f_4^2, f_5^2\}$
3	$\{f_0^3, f_1^3, f_2^3, \mathbf{F_3}, f_4^0 + f_4^2 + f_4^3, f_5^3\}$
4	$\{f_0^4, f_1^4, f_2^4, f_3^4, \mathbf{F_4}, f_5^1 + f_5^3 + f_5^4\}$
5	$\{f_0^2 + f_0^4 + f_0^5, f_1^5, f_2^5, f_3^5, f_4^5, \mathbf{F_5}\}$

Fig. 5 The *global sum* operation on the $F6A$ graph. The three tables on the right show the data that the six processors numbered 0–5 have before the *global sum* operation, between the two steps, and after the completed operation. The diagrams on the left indicate the data transfers that occur at the two steps of the *global sum* operation, with arrows from the sender to the receiver and the transferred data written by the arrow head. Before the operation, every processor i, $0 \leq i \leq 5$ has partial forces for all N atoms, $\{f_0^i, f_1^i, f_2^i, f_3^i, f_4^i, f_5^i\}$. After the operation, every processor has the sum of these partial forces, $\mathbf{F_i} = \{f_i^0 + f_i^1 + f_i^2 + f_i^3 + f_i^4 + f_i^5\}$ for the N/P atoms assigned to it, along with residual partial forces for atoms that are not assigned to it; these residual partial forces are unneeded and eventually discarded

receive it; and the connection *bandwidth*, which is the speed at which data is transferred between the sender and receiver. The time to transfer a single message is therefore the $latency + message\,size \times bandwidth$. For the *global broadcast* or *global sum* operations, the total time is $steps \times latency + bandwidth \times data\ volume$, where the

```
Require: $LCFGraphCode$, the $LCF$ graph code
Ensure: $schedule$, the completed schedule
  $vertex \leftarrow$ createVertexFromLCCF($LCFGraphCode$)
  $has \leftarrow \emptyset$
  $hasUpdate \leftarrow \emptyset$
  for $p := 0$ to $processors - 1$ do
    $has(p, p) \leftarrow$ True
  end for
  $allHaveAll \leftarrow$ False
  $step \leftarrow 1$
  while $\neg allHaveAll$ do
    for $me := 0$ to $processors - 1$ do
      for $data := 0$ to $processors - 1$ do
        if $\neg has(me, data)$ then
          for all $hop \in \{$spoke, less, more$\}$ do
            $you \leftarrow$ vertex$(me, hop)$
            if $\neg hasUpdate(me, data)$ then
              $hasUpdate(me, data) \leftarrow$ True
              $schedule \leftarrow schedule \cup \{(step, you, data, me)\}$
            end if
          end for
        end if
      end for
    end for
    if $hasUpdate \neq \emptyset$ then
      $has \leftarrow has \cup hasUpdate$
      $hasUpdate \leftarrow \emptyset$
    else
      $allHaveAll \leftarrow$ True
    end if
  end while
```

Fig. 6 An algorithm for creating a broadcast schedule among vertices (processors). The input is the graph's LCF code and the output is a broadcast schedule that specifies which data a processors sends and receives at each time step

steps is the number of communication time steps needed for the operation and *data volume* is the sum over all of the steps of the maximal message size exchanged by any two processors.

The modeled times according to the algorithm are gathered in Table 2. For each graph, identified by the Graph Name, we list the number of processors—corresponding to the graph order—in the interconnection network, the number of communication time steps, which is equal to the graph diameter, and the data volume.

The communication time steps does not increase monotonically with an increasing graph order, nor does the data volume. Therefore there are a number of networks that, while connecting a larger number of processors, have a relatively low time required for communication. Among the larger networks, the ones based on graphs F304A and F336C have such low requirements compared to other similarly-sized networks.

The F060A graph, which is comparable in size to a 6-dimensional hypercube of order 64, has fewer communication time steps, 5 compared to the hypercube's 6 [16]. Also, the data transfer time is 25 compared to the hypercube's 63.

Table 2 Communication requirements for a number of interconnection networks

Graph name	Number of processors	Communication steps	Data volume
F004A	4	1	1
F006A	6	2	3
F008A	8	3	4
F014A	14	3	7
F016A	16	4	8
F018A	18	4	11
F020A	20	5	8
F020B	20	4	13
F024A	24	5	17
F026A	26	5	13
F030A	30	4	15
F032A	32	5	16
F038A	38	5	19
F040A	40	6	20
F042A	42	6	21
F048A	48	6	34
F050A	50	7	25
F054A	54	6	27
F056A	56	7	28
F060A	60	5	25
F112C	112	7	56
F126A	126	10	63
F152A	152	11	76
F168A	168	12	84
F208A	208	9	104
F224A	224	13	112
F234A	234	14	117
F248A	248	15	124
F296A	296	15	148
F304A	304	11	152
F312A	312	16	156
F336C	336	12	168
F342A	342	16	171
F344A	344	17	172
F350A	350	17	175
F378A	378	18	189
F392A	392	17	196
F416A	416	19	208

For each network, the graph name, the number of communication time steps, and data volume is given

5 Conclusions

We have presented a class of computer interconnection networks based on hamiltonian cubic symmetric graphs. The cubic symmetric graphs have many desirable properties for use as interconnection networks since they have a low degree and vertex- and edge-transitivity. We have developed an algorithm for scheduling the data transfers for the *global broadcast* and *global sum* operations, which are needed for parallel MD simulation, on networks derived from any of these graphs. We have shown that the computer interconnection networks based on cubic symmetric graphs are very scalable, enabling MD simulations to be run on parallel computers with a large number of processors.

Acknowledgements The authors acknowledge the financial support from the state budget by the Slovenian Research Agency under Grants Nos. P1-0285 (K.K. and D.M.) and P1-0002 (U.B. and D.J.).

References

1. T. Sterling, D.J. Becker, D. Savarese, Beowulf: a parallel workstation for scientific computation. In *Proceedings of the 15th International Conference on Parallel Processing (ICPP)*, vol. 1 (1995), pp. 11–14
2. J.-C. Bermond, C. Delorme, J.-J. Quisquater, Strategies for interconnection networks: some methods from graph theory. J. Parallel Distrib. Comput. **3**, 433–449 (1985)
3. S.M. Figueira, V.J. Reddi, Topology-based hypercube structures for global communication in heterogeneous networks. In *Proceedings of the 11th International Euro-Par Conference*, August 30–September 2. Lecture Notes in Computer Science, vol. 3648 (Springer, Berlin, 2005)
4. R. Trobec, Two-dimensional regular d-meshes. Parallel Comput. **26**, 1945–1953 (2000)
5. S. Horiguchi, T. Ooki, Hierarchical 3D-Torus interconnection network. In *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'2000)* (IEEE CS Press, Richardson, TX, 2000)
6. B. Parhami, D.M. Kwai, Comparing four classes of torus-based parallel architectures: network parameters and communication performance. Math. Comput. Model. **40**, 701–720 (2004)
7. M. Hodoščcek, U. Borštnik, D. Janežič, CROW for large scale macromolecular simulations. Cell. Mol. Biol. Lett. **7**, 118–119 (2002)
8. V. Kutalek, V. Dvorak, On complexity of collective communications on a fat cube topology. J. Univ. Comput. Sci. **11**, 944–961 (2005)
9. F. Comellas, M. Mitjana, J.G. Peters, Broadcasting in small-world communication networks. Proc. Inform. **13**, 73–85 (2002)
10. T. Dobravec, B. Robič, J. Žerovnik, Permutation routing in double-loop networks: design and empirical evaluation. J. Syst. Arch. **48**, 387–402 (2003)
11. D.W. Heermann, A.N. Burkitt, *Parallel Algorithms in Computational Science* (Springer-Verlag, Berlin, 1991)
12. J.C. Phillips, G. Zheng, S. Kumar, L.V. Kalé, NAMD: biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*, Baltimore, MD (2002)
13. R. Trobec, M. Šterk, M. Praprotnik, D. Janežič, Parallel programming library for molecular dynamics simulations. Int. J. Quant. Chem. **95**, 530–536 (2004)
14. U. Borštnik, M. Hodoščcek, D. Janežič, Fast parallel molecular simulations. Croat. Chem. Acta **78**, 211–216 (2005)
15. H. Lu, S. Dwarkadas, A.L. Cox, W. Zwaenepoel, Message passing versus distributed shared memory on networks of workstations. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, CA, USA (1995), p. 37
16. U. Borštnik, M. Hodoščcek, D. Janežič, Improving the performance of molecular dynamics simulations on parallel clusters. J. Chem. Inf. Comput. Sci. **44**, 359–364 (2004)
17. B. Robič, B. Vilfan, Improved schemes for mapping arbitrary algorithms onto processor meshes. Parallel Comput. **62**, 308–318 (1995)
18. I.Z. Bouwer, W.W. Chernoff, B. Monson, Z. Star, *The Foster Census* (Winnipeg, 1988)
19. M.D.E. Conder, P. Dobcsanyi, Trivalent symmetric graphs on up to 768 vertices. J. Combin. Math. Combin. Comput. **40**, 41–63 (2002)
20. R. Frucht, A canonical representation of trivalent hamiltonian graphs. J. Graph Theory **1**, 45–60 (1977)
21. N. Biggs, *Algebraic Graph Theory* (Cambridge University Press, London, 1974)
22. W.T. Tutte, *Connectivity in Graphs* (University of Toronto Press, Toronto, 1966)
23. Y.Q. Feng, J.H. Kwak, Cubic symmetric graphs of order twice an odd prime-power. J. Aust. Math. Soc. **81**, 153–164 (2006)
24. Y.Q. Feng, J.H. Kwak, Classifying cubic symmetric graphs of order $8p$ or $8p2$. Eur. J. Combin. **26**, 1033–1052 (2005)
25. D. Marušič, T. Pisanski, Symmetries of hexagonal molecular graphs on the torus. Croat. Chem. Acta **73**, 969–981 (2000)
26. K. Kutnar, A. Malnič, D. Marušič, Chirality of toroidal molecular graphs. J. Chem. Inf. Model. **45**, 1527–1535 (2005)

27. A. Malnič, R. Nedela, M. Škoviera, Graph automorphisms by voltage assignments. Eur. J. Combin. **21**, 927–947 (2000)
28. A. Malnič, Action graphs and coverings. Discrete Math. **244**, 299–322 (2002)
29. J. Lederberg, DENDRAL-64: a system for computer construction, enumeration and notation of organic molecules as tree structures and cyclic graphs. Part II. Topology of cyclic graphs. In *Interim Report to the National Aeronautics and Space Administration*. Grant NsG (1965)
30. H.S.M. Coxeter, R. Frucht, D.L. Powers, *Zero-Symmetric Graphs: Trivalent Graphical Regular Representations of Groups* (Academic Press, New York, 1981)
31. A.R. Leach, *Molecular Modeling: Principles and Applications* (Addison Wesley Longman Limited, Essex, 1996)
32. S.J. Plimpton, Fast parallel algorithms for short-range molecular dynamics. J. Chem. Phys. **117**, 1–19 (1995)
33. M. Snir, A note on n-body computations with cutoffs. Theor. Comput. Syst. **37**, 295–318 (2004)
34. S.J. Plimpton, B.A. Hendrickson, A new parallel method for molecular-dynamics simulation of macromolecular systems. J. Comput. Chem. **17**, 326–337 (1996)
35. B.R. Brooks, M. Hodošček, Parallelization of CHARMm for MIMD machines. Chem. Des. Autom. News **7**, 16–22 (1992)
36. R. Murty, D. Okunbor, Efficient parallel algorithms for molecular dynamics simulations. Parallel Comput. **25**, 217–230 (1999)
37. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference* (The Massachusetts Institute of Technology Press, Cambridge, 1996)
38. T. Dobravec, J. Žerovnik, B. Robič, An optimal message routing algorithm for circulant networks. J. Syst. Arch. **52**, 298–306 (2006)