An Optimal Hypercube Direct N-body Solver on the Connection Machine

Jean-Philippe Brunet
Jill P. Mesirov
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142
617-876-1111

Alan Edelman*
CERFACS
42 Avenue Gustave Coriolis
31057 Toulouse CEDEX
France
(33) 61.07.96.96

Abstract

The direct method for solving N-body problems maps perfectly onto hypercube architectures. Unlike other hypercube implementations, we have implemented a direct N-body solver on the Connection Machine CM-2 which makes optimum use of the full bandwidth of the hypercube. When $N \gg P$, where P is the number of floating-point processors, the communication time of the algorithm is negligible, and the execution time is that of the arithmetic time giving a P-fold speed-up for real problems. To obtain this performance, we use "rotated and translated Gray codes" which result in time-wise edge disjoint Hamiltonian paths on the hypercube. We further propose that this communication pattern has unexplored potential for other types of algorithms. Timings are presented for a collection of interacting point vortices in two dimensions. The computation of the velocities of 14,000 vortices in 32-bit precision takes 2 seconds on a 16K CM-2.

1 Introduction

The N-body algorithm is a critical kernel in a wide variety of application areas including astronomy, molecular biology, and fluid dynamics. The direct method (as opposed to local correction [1], hierarchical [2, 3], or multipole [8] methods) runs in $O(N^2)$ serial time for N bodies since all pairwise interactions are computed; the force on each body, v_i, is given by

$$\text{Force}(v_i) = \sum_j F_{v_j}(v_i),$$

where $F_{v_j}(v_i)$ is the force exerted on v_i by v_j.

In this paper, we show how the direct method is ideally suited for the hypercube architecture. For non-direct methods on the hypercube, see [5, 7, 13]. Our algorithm divides the N bodies evenly over the $P = 2^d$ processors of a d-dimensional hypercube. For simplicity, assume the number of bodies in each node, N/P, is an integer multiple of d. To compute the N^2 interactions, the data for each body must be transmitted to all of the other processors. If the processors have enough memory, then this transmission can be performed initially, and afterwards, all the arithmetic can be performed locally. This type of communication in a hypercube has been called **all to all broadcasting** [9] and **multinode broadcast** [6].

We do not follow this strategy which wastes memory; in particular, we can interleave pieces of the computation with the communications and have no need to store the data for all the bodies at each node. Thus, we can use the idea of "rotated and translated Gray codes" described in [9]. Although this method has "limited potential for pipelining", we show that it is just right for the N-body problem, and possibly many other as yet unexplored applications. [1].

The idea of rotated and translated Gray codes is to produce $d2^d$ time-wise edge disjoint Hamiltonian paths through the hypercube, d of them starting at each node. A **Hamiltonian path** in a graph visits all the nodes in the graph only once. **Time-wise edge disjoint** means that, although the paths themselves share edges of the hypercube, no two paths traverse the same edge on the same communication step. Consequently, there is no contention for communication channels. The data is circulated on these paths, but not retained, avoiding the memory waste. Furthermore, from a programming point of view, the "rotated and translated Gray codes" are very simple since one can take advantage of the symmetry of the hypercube. The combination of the simplicity and the optimal use of communication bandwidth makes this communication pattern one of the fastest which the Connection Machine®[2] CM-2 [3] can perform.

In Section 2, we will describe the Connection Machine architecture, and the slicewise model of which we make use. Section 3 contains an explanation of how the Hamiltonian paths are generated as well as the data motion of the algorithm. More detailed implementation issues are discussed in Section 4, and Section 5 presents the timings for a sample application.

*This work was mostly performed while the author was at Thinking Machines Corp.

[1] See [10] for one example

[2] Connection Machine is a registered trademark of Thinking Machines Corporation

[3] CM-2 is a trademark of Thinking Machines Corporation

CH2916-5/90/0000/0748/$01.00 © IEEE

2 The Connection Machine Architecture

The CM-2 is composed of a microsequencer and a maximum of 64K single-bit processing elements. The processors run in SIMD mode with the instruction stream broadcast by the sequencer. The sequencer is controlled by an external front-end machine, usually a SUN®[4], SYMBOLICS®[5] Lisp Machine, or VAX®[6].

For performing floating-point computations, better performance is usually obtained by looking at the machine in what is referred to as the **slicewise** model. That is, we consider processing nodes on the machine to be the ensemble of a floating-point unit and the memory of 32 associated physical processors of the CM-2. In the usual **fieldwise** model, the storage of a 32-bit word would be allocated in 32 sequential bits of a physical processor's memory. In the slicewise model, a word is stored in a 32-bit slice across the memories of the 32 processors in the node, i.e., 1-bit per processor. From this viewpoint, a 64K processor CM-2 becomes 2048 floating-point nodes connected as an 11-dimensional hypercube with two communication channels between connected nodes instead of one.

It is with this model of the CM-2 that we implemented our fast direct N-body solver. On a CM-2 with 2^{d+5} physical processors, there are $P = 2^d$ slicewise floating-point nodes. We divide the data for the N bodies evenly among the nodes; each node will be responsible for accumulating the forces for N/P bodies. We are thus left with the problem of how to optimally transmit the data for the bodies at each node to all of the other nodes. In the next two sections, we will describe how that is done.

3 Rotated and Translated Gray Codes

A d-dimensional hypercube is a graph with 2^d nodes labeled by the d-bit binary representation of the integers 0 to $2^d - 1$. Each bit in the d-bit representation is associated with a different dimension of the cube. There is an edge between two nodes, i and j, in the hypercube if and only if their binary representations differ in only one bit. We can think of the edges as traversing different dimensions of the hypercube.

A **Gray code** is a circuit of all binary d-tuples such that only one coordinate position changes at each step. Thus, a Gray code represents a Hamiltonian path on the hypercube. There are many ways to construct Gray codes, but the most famous is the binary reflected Gray code [11]. To simplify the terminology, we will refer to this code as **the** Gray code. We define the **transition sequence** [11] corresponding to a Gray code as the list of positions which change at each step of the code or the list of dimensions traversed on the hypercube at each step of the circuit. In Figure 1, we show the binary reflected Gray code for $d = 3$, the corresponding transition sequence, and the associated Hamiltonian path on the cube.

Given the Gray code beginning at 0, we can "translate" it to node i by taking the *exclusive or* of i with the d-tuples in the original code. The transition sequence of the translated Gray code is identical to the original. Figure 2 shows the translated code starting at node 5 = 101. Furthermore, given the Gray code beginning at 0, we can "rotate" it by performing circular shifts of the bits in the original code. In

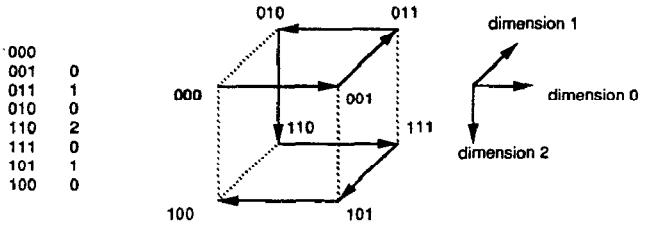

Figure 1: The binary reflected Gray code, the corresponding transition sequence, and the associated Hamiltonian path originating at node 0 on a three dimensional cube.

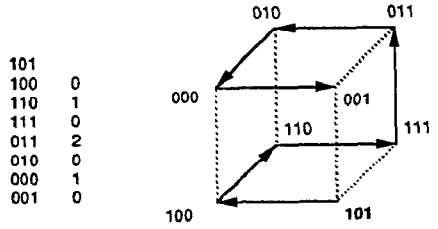

Figure 2: The transition sequence of Figure 1 is used to generate a translated code to node 101 = 5 and the associated Hamiltonian path.

the tables below, we list the three rotated Gray codes and the corresponding transition sequences when $d = 3$. Notice that in general, the transition sequence of the rotated code i is obtained by adding i modulo d to the transition sequence of the binary reflected Gray code.

Rotated Gray Codes								
Code 0	000	001	011	010	110	111	101	100
Code 1	000	010	110	100	101	111	011	001
Code 2	000	100	101	001	011	111	110	010

Transition Sequences							
Code 0	0	1	0	2	0	1	0
Code 1	1	2	1	0	1	2	1
Code 2	2	0	2	1	2	0	2

By taking all of the 2^d possible translations of the d rotated Gray codes beginning at 0, we can generate a set, H, of $d2^d$ Hamiltonian paths on the hypercube. This set H is

[4]SUN is a trademark of Sun Microsystems, Inc.
[5]SYMBOLICS is a trademark of Symbolics, Inc.
[6]VAX is a trademark of Digital Equipment Corporation

more elegantly described as the paths generated by applying a group of $d2^d$ symmetries (rotations and translations) of the hypercube to the binary reflected Gray code[7] starting from node 0. These symmetries are the subgroup of all the hypercube symmetries that rotate the dimension numbers modulo d. It is straightforward to verify that two distinct symmetries from this subgroup transform a directed edge into two distinct directed edges. It follows that the collection of paths H have the nice property of being time-wise edge disjoint: i.e., if $j \in 1, 2, \ldots, 2^d - 1$, then there is no overlap among the jth edges of the paths in H. Because of this nice property, if we have d packets of information in each node of the hypercube, we can circulate the data, one packet to a Hamiltonian path, in $2^d - 1$ communication steps. Furthermore, we can perform the arithmetic on the data in between each communication step eliminating the need for data storage.

4 Implementation on the CM-2

If we have d words at every node, and if we wish to circulate the data using the Hamiltonian paths described in the previous section, one might expect that we would need either to attach some labeling information to be transmitted with the data or else, one might imagine some kind of look up table that specifies over which dimension data needs to be sent. Neither of these kinds of operations is without cost. In our implementation, as we shall now describe, we avoid both of these types of methods. In fact, the information in the transition sequence for the binary reflected Gray code is all that is needed.

Using the slicewise model, the CM-2 architecture allows for the loading of as many as $2d$ words to be sent over the d dimensions simultaneously, two words in each dimension. Typically, one loads one set of d words for the d dimensions followed by another set of d words. For simplicity, we will concentrate on one set with the understanding that the other set follows exactly the same pattern. Another important feature of the CM-2 architecture is that it is particularly efficient to load consecutive words, or more generally, words with a constant stride, to be sent over consecutive dimensions.

To illustrate how the CM-2 works, let us first imagine that we only have one word in each processor and each of these words will follow the Hamiltonian path specified by the transition sequence of the Gray code: $g = \{0, 1, 0, 2, 0, 1, \ldots\}$. Further imagine we have only one memory location (location 0) in each node where the word is stored. Then the algorithm is simply:

[7] The use of the binary reflected Gray code is merely for convenience; any other Hamiltonian path will suffice.

For $k = 1, 2, \ldots, 2^d - 1$
  In each node, the data at location 0 is
  1. Loaded to be sent over dimension g_k
  2. Transmitted
  3. Stored in location 0

Now, if we have d words in each node at memory locations 0 through $d - 1$, we can quite readily make sure that the word at location j in each node follows the jth transition sequence. The method is to store this word at location j as it passes through each node like a traveler who prefers the same hotel in each city. Then, at step k, we send the data in each node at location j over dimension $g_k + j \pmod d$. To take advantage of the Connection Machine's ability to load consecutive words, we in fact use the algorithm in the box below.

For $k = 1, 2, \ldots, 2^d - 1$
  In each node
  1. Data at locations 0 through $d - g_k - 1$ is loaded to be sent over dimensions g_k through $d - 1$ respectively.
  2. Data at location $d - g_k$ through $d - 1$ is loaded to be sent over dimensions 0 through $g_k - 1$ respectively.
  3. Data is transmitted.
  4. Data received over dimensions g_k through $d - 1$ is stored at locations 0 through $d - g_k - 1$ respectively.
  5. Data received over dimensions 0 through $g_k - 1$ is stored at locations $d - g_k$ through $d - 1$ respectively.

Notice that steps 2 and 5 are vacuous if $g_k = 0$, which is half of the time. The actual code to perform this operation is not much longer than the boxed pseudo-code.

In the real algorithm, we assume there is a multiple of $2d$ bodies in each node, and each body may contain more than one word of information. In the example described in the next section, there are two words of coordinate information and also a coefficient, giving three words for each body. Since we can transmit $2d$ words simultaneously, we loop over all the data until it is all transmitted. After the data is transmitted, we perform a computation such as the one described in the next section, and then we repeat for $k = 1, 2, \ldots, 2^d - 1$.

5 Analysis and timings

5.1 A sample application

As an application example, we computed the velocities of a collection of interacting point vortices in two dimensions. This N-body problem is at the heart of the vortex method to solve the Navier-Stokes equation for incompressible viscous

flows [12]. A Lagrangian perspective of the vorticity-stream function formulation of the Navier-Stokes equation leads to the introduction of discrete vortex elements which interact according to a Biot-Savart type force law. To avoid potential numerical instability caused by very close encounters of particles, finite size vortex elements should be used. However, in these timing runs, we have considered the interaction of point vortices.

In two dimensions, the (u,v) velocity components of a point vortex at position (x,y) are given by the following formulae:

$$u(i) = \frac{1}{2\pi} \sum_{j=1}^{N} \frac{c(j)\Delta y(i,j)}{\Delta r(i,j)^2}, \quad i = 1,...,N$$

$$v(i) = \frac{-1}{2\pi} \sum_{j=1}^{N} \frac{c(j)\Delta x(i,j)}{\Delta r(i,j)^2}, \quad i = 1,...,N$$

$$\begin{aligned} \text{with} \qquad \Delta x(i,j) &= x(j) - x(i) \\ \Delta y(i,j) &= y(j) - y(i) \\ \Delta r(i,j)^2 &= \Delta x(i,j)^2 + \Delta y(i,j)^2 \\ \text{and} \quad c(j) &= \text{ circulation of point vortex } j. \end{aligned}$$

5.2 Analysis and Timings

The complexity analysis of our algorithm is straightforward. The computation time, T_{arith}, scales like $N^2/2^d$, where N is the total number of bodies and d is the dimension of the hypercube. The communication time, T_{comm}, scales like $\lceil \frac{N}{2^d 2d} \rceil (2^d - 1)$. Here $\lceil x \rceil$ denotes the ceiling of x, i.e., the smallest integer greater than or equal to x. Since there are $N/2^d$ bodies per node and $2d$ wires leaving each node, the first factor in T_{comm} represents the number of communication operations needed to empty a node of data. The second factor just represents the length of the path through the hypercube the data must traverse.

Given a fixed value of N such that $\frac{N}{2^d} \gg 2d$, T_{comm} is, at first, a linearly decreasing function of d. As the number of dimensions, d, increases, the number of required communication operations decreases to one, and the second factor in the expression of T_{comm} will dominate. Because at least one such communication operation must be performed, even if there are less than $2d$ data items at a node, we see that as $d \to \infty$,

$$T_{comm}(d) \simeq 2^d.$$

The function T_{comm}, therefore, exhibits a minimum which occurs when the bandwidth, $2d$, is roughly equal to the number of bodies per node. As the number of bodies per node decreases, one is making less and less effective use of the communications bandwidth of the hypercube, and T_{comm} becomes a more significant portion of the total execution time. In fact, when $\frac{N}{2^d} \leq 1$, i.e., the number of bodies is much smaller than the number of processors, another approach, which replicates data to increase processor utilization, becomes more cost effective [4].

We performed several calculations as described in the previous section for a varying number of particles on various machine sizes. Timing results are shown in Figure 3. Note that the runs were made on CM-2 configurations with 2^{d+5} processors for $d = 4,5,7,8,9$, which were the only ones available to us. Typically, 14,000 vortices interact in 2 seconds on a 512 floating-point node machine (16K processors). Counting the divide operation as one operation, the execution rate is about 5.2 Gflops on a fully configured CM-2 (2048 nodes and 64K processors). [8]

Keeping the number of bodies constant, the time to compute the velocity components appears inversely proportional to the number of floating-point nodes of the hypercube. This

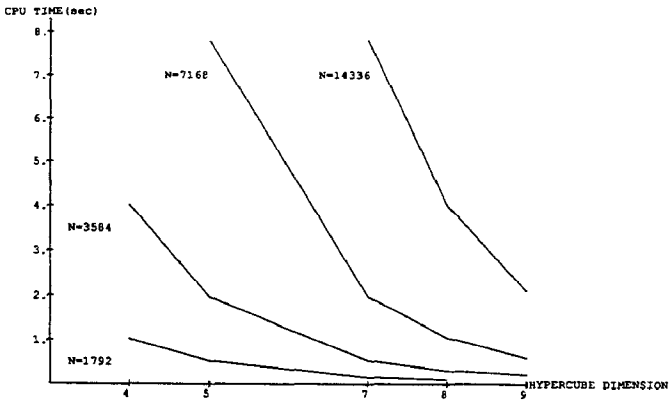

Figure 3: Total execution time for the direct interaction of N point vortices in two dimensions, with N = 1792, 3584, 7168, 14336. The hypercube dimensions 4, 5, 7, 8, 9 correspond to 512, 1024, 4096, 8192 and 16284 Connection Machine processors, respectively.

is the kind of behavior we expect given the analysis above. We get a more accurate picture of what is going on from Figure 4 which plots the actual speedup of the implementation against the optimal speedup of 2^d (the number of processors). We calculated the actual speedup by taking the ratio of T_{arith} times the number of processors and the total execution time of the calculation. Note that again the curves behave as predicted by the complexity analysis. For a fixed hypercube dimension, as the number of bodies increases the speedup approaches optimality. In particular, over the range of test cases we ran, T_{comm} varies from a few percent to thirty percent of the total execution time as the number of bodies approaches the number of processors.

[8] On the 32-bit floating point unit of the CM-2, a divide operation requires the explicit implementation of two Newton-Raphson iterations of the initial value which is provided by an internal look-up table. This amounts to a total of six atomic operations which we count as one.

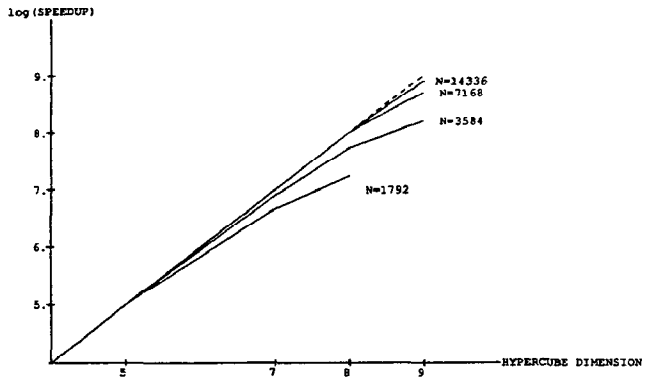

Figure 4: Speedup on the CM-2 for the direct interaction of N point vortices in two dimensions, with $N = 1792, 3584, 7168, 14336$. The speedup is defined as the ratio of the execution time using one floating point processor and the execution time using 2^d processors to solve the same problem; d is the dimension of the hypercube and a logarithmic scale is used. The dashed curve is the optimal speedup, i.e., the number of processors 2^d.

6 Summary

We have designed and implemented a hypercube algorithm for direct N-body solvers on the Connection Machine CM-2. The algorithm is optimal in the sense that, as long as there is sufficient data, it uses the full communication bandwidth of a hypercube of any dimension. When the number of bodies per node is large enough, the communication time for the implementation is negligible, i.e., less than 2%. In particular, this means that we obtain close to optimal speedup in this regime. This level of performance has enabled us to take advantage of the implementation for a relevant physical application [12].

References

[1] Anderson, C.R. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *SIAM J. Numer. Anal.* **22**, (1986), 413–440.

[2] Appel, A. An efficient program for many-body simulation. *SIAM J. Sci. Stat. Comput.* **6**, (1985), 85–103.

[3] Barnes, J. and Hut, P. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature* **324**, (1986), 446–449.

[4] Brunet, J-Ph., Edelman, A., and Mesirov, J.P. Two Hypercube Algorithms for Direct N-body Solvers, in preparation.

[5] Barnes, J. and Hillis, W.D. Programming a highly parallel computer. *Nature* **326**, (1987), 27–30.

[6] Bertsekas, D.P., Ozveren, C., Stamoulis, G.D., Tseng, P., and Tsitsiklis, J.N. Optimal communication algorithms for hypercubes. To appear.

[7] Fox, G.C., Hipes, P., and Salmon, J. Practical parallel supercomputing: examples from chemistry and physics. *Proceedings Supercomputing '89.* ACM Press, Reno, NV, 1989, pp. 58–70.

[8] Greengard, L. and Rokhlin, V. A fast algorithm for particle simulation. *J. Comp. Phys.* **73**, (1987), 325–348.

[9] Johnsson, S.L. and Ho, C.T. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers* **38**, (1989), 1249–1268.

[10] Johnsson, S.L. and Ho, C.T. Multiplication of arbitrarily shaped matrices on boolean cubes using the full communications bandwidth. Yale University Tech. Rep. YALEU/DCS/TR-721, Yale University, Dept. of Computer Science, New Haven, 1989.

[11] Reingold, E.M., Nievergelt, J., and Deo, N. *Combinatorial Algorithms*, Prentice-Hall, Englewoods Cliffs, NJ, 1977.

[12] Sethian, J.A., Brunet, J-Ph., Greenberg, A., and Mesirov, J.P. Two-dimensional, viscous, incompressible flow in arbitrary geometries on a massively parallel processor. To appear.

[13] Zhao, F. and Johnsson, S.L. The parallel multipole method on the Connection Machine. Thinking Machines Corporation Tech. Rep. CS89-6, Thinking Machines Corp., Cambridge, MA, 1989.