Optimal Evaluation of Array Expressions on Massively Parallel Machines

SIDDHARTHA CHATTERJEE
Research Institute for Advanced Computer Science
JOHN R. GILBERT
Xerox Palo Alto Research Center
ROBERT SCHREIBER
Research Institute for Advanced Computer Science
and
SHANG-HUA TENG
Xerox Palo Alto Research Center

We investigate the problem of evaluating Fortran 90-style array expressions on massively parallel distributed-memory machines. On such a machine, an elementwise operation can be performed in constant time for arrays whose corresponding elements are in the same processor. If the arrays are not aligned in this manner, the cost of aligning them is part of the cost of evaluating the expression tree. The choice of where to perform the operation then affects this cost.

We describe the communication cost of the parallel machine theoretically as a metric space; we model the alignment problem as that of finding a minimum-cost embedding of the expression tree into this space. We present algorithms based on dynamic programming that solve the embedding problem optimally for several communication cost metrics: multidimensional grids and rings, hypercubes, fat-trees, and the discrete metric. We also extend our approach to handle operations that change the shape of the arrays.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*interconnection networks*; *SIMD*; *MIMD*; D.3.4 [**Programming Languages**]: Processors—*compilers*; *optimization*; E.1 [**Data**]: Data Structures—*arrays*; G.2.2 [**Discrete mathematics**]: Graph Theory—*trees*

General Terms: Algorithms, Languages, Theory

This work was done while Chatterjee was a postdoctoral scientist at RIACS and Teng was a postdoctoral scientist at Xerox PARC. The work of Chatterjee and Schreiber was supported by the NAS Systems Division via Cooperative Agrement NCC2-387 between NASA and the Universities Space Research Association (USRA). An extended abstract of this article was presented at the *Second Workshop on Languages, Compilers, and Runtime Environments for Distributed Memory Multiprocessors*, Boulder, Colorado, September 30–October 2, 1992.
Authors' addresses: S. Chatterjee, Department of Computer Science, Campus Box 3175, Sitterson Hall, The University of North Carolina, Chapel Hill, NC 27599-3175; email: sc@cs.unc.edu; J. R. Gilbert, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304-1314; email: gilbert@parc.xerox.com; R. Schreiber, RIACS, Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035-1000; email: schreiber@riacs.edu; S.-H. Teng, Department of Computer Science, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55455; email: steng@cs.umn.edu.

Additional Key Words and Phrases: Array alignment, compact dynamic programming, data-parallel programming, distributed-memory parallel processors, fixed-topology Steiner tree, Fortran 90

1. INTRODUCTION

As parallel computer architectures proliferate, the need for a stable and portable software base becomes critical. Data parallelism has emerged as a suitable model for portable parallel software for a wide variety of problems in many application domains. Such languages as Fortran 90 [American National Standards Institute 1991], Fortran D [Fox et al. 1990], Vienna Fortran [Chapman et al. 1992], and High Performance Fortran [High Performance Fortran Forum 1993] standardize array-based data-parallel programming. Other languages, like C* [Rose and Steele 1987] and NESL [Blelloch 1992; Blelloch et al. 1994], extend the data-parallel model to nested and irregular collections of data.

In implementing a data-parallel language on a massively parallel distributed-memory machine, the two major issues are the alignment and layout of data, and the mapping and scheduling of tasks. In current compilers, data mappings are supplied either by the user or by simple canonical defaults; and task mappings are usually derived from data mappings by the "owner-computes" rule [Hiranandani et al. 1991]. This has several disadvantages:

—The best mapping for a problem depends on the communication architecture, the machine size, and the problem size [Wholey 1991]. Thus the user may have to write new mappings for each new machine or problem size, or be satisfied with poor performance.

—Even if the programmer maps named variables, the compiler must still map temporaries. Mapping according to the owner-computes rule can cause unnecessary communication [Knobe et al. 1992].

—The best data and task mappings may be different in different portions of a program [Chatterjee et al. 1994b; Kremer et al. 1993].

We believe that compilers should address the problem of data and computation mapping. Some early work in this direction is the following:

—Knobe et al. [1990] developed a heuristic for data distribution based on alignment preferences for arrays. Li and Chen [1991] and Gupta [1992] have developed similar algorithms. Wholey [1991] showed that optimal mappings are often not obvious. He further showed that a compiler can derive good mappings by heuristic clustering and hill-climbing techniques.

—Anderson and Lam [1993] used a linear algebra framework to determine alignments. They determine alignments that remove general communication, possibly at the cost of reduced parallelism.

—Gilbert and Schreiber [1991] developed a theoretically optimal algorithm for the alignment of temporaries in an expression tree involving array sections by describing interconnection networks as metric spaces and by modeling the alignment problem as that of finding a minimum-cost embedding of the expression

tree into this space. Their algorithm applies to a class of so-called "robust" metrics, including multidimensional grids, hypercubes, and the discrete metric. They showed that the presence of common subexpressions makes the problem NP-complete; this makes heuristic solutions necessary for basic blocks.

Here we extend the theoretical work of Gilbert and Schreiber [1991] in two different directions. First, we use more powerful dynamic programming techniques to handle two additional metrics for communication cost: multidimensional rings and fat-trees [Leiserson 1985]. These metrics are not robust and therefore cannot be handled by their algorithm. We also show that their algorithms for grids, hypercubes, and the discrete metric can be recreated in our formulation. Second, we relax their restriction that all operands must be the same size, by extending our algorithms to allow weights on the edges of the expression tree.

1.1 Theoretical Model and Assumptions

An alignment problem has two components: an array expression to be evaluated and a parallel machine on which to evalute it. We model the array expression as an expression tree, and we describe the communication cost of the parallel machine as a metric space. We then model the alignment problem as that of finding a minimum-cost embedding of the expression tree into this space.

We now list the assumptions in our theoretical model of the alignment problem. Our algorithms find solutions that are optimal in the sense of this model.

1.1.1 *Expression Tree.* We are evaluating a single data-parallel expression or statement as given in the program; commutative or associative rearrangement is not allowed, and there are no common subexpressions. Thus the data flow of the computation can be described by a tree. Gilbert and Schreiber [1991] proved that if commutative and associative rearrangement is allowed, the alignment problem in two or more dimensions is the same as the Geometric Steiner Tree problem, which is NP-hard for the grid and discrete metrics. If common subexpressions are allowed, the exact optimization problem is again NP-complete [Gilbert and Schreiber 1991; Mace 1987]. In a companion paper [Chatterjee et al. 1993b], we extend our algorithms (heuristically) to basic blocks, which can be described as directed acyclic graphs.

1.1.2 *Communication Cost.* We assume that the cost of communicating data between two different positions p and q can be described as $w \cdot d(p, q)$, where w is the amount of data communicated and $d(p, q)$ is the cost of moving one element from position p to position q. The function d is a metric (symmetric, nonnegative, and satisfying the triangle inequality). Initially, we deal with the case where all w's are equal (so that $w = 1$ without loss of generality); in Section 7 we discuss the more general case where different operands can have different weights.

Our general dynamic programming algorithm handles any cost metric, but we give particularly efficient *compact dynamic programming* algorithms for four specific metrics.

The *discrete metric* (Section 5) models architectures in which a fixed startup cost dominates the cost of a communication action, as on many MIMD machines. It can also be used to model the cost of changing contexts on SIMD machines. The *grid*

metric (Section 3) models architectures in which distance in a grid or hypercube dominates communication cost; the grid metric is also appropriate when data is distributed by blocks to an array of processors, since (for small shifts) the amount of data that must cross a physical processor boundary is proportional to the length of the shift.

Communication cost in a real machine is usually a combination of a startup cost term and a distance term. However, in most real machines, one or the other components of cost is dominant; thus one or the other of our algorithms (for the discrete metric or the grid metric) is appropriate. Efficiently optimizing a model that incorporates both startup and distance remains an open problem.

The final two metrics for which we give compact algorithms are the *ring* (Section 4) and *fat-tree* (Section 6). These metrics do model some real architectures, but we expect that in practice the discrete and grid metrics will be the most useful.

1.1.3 *Overlap.* In our model, only one parallel computation or communication action occurs at a time. Thus we do not model the possibility of overlap between computation and communication, or between different communication actions or computations in different parts of the expression tree. In the multidimensional grid and ring cases, we also assume that communication occurs only in one dimension at a time. On some architectures, a compiler might further reduce the completion time of the solutions found in our model by such overlapping or by various other machine-specific communication optimizations [Gerndt 1989; Tseng 1993]. These optimizations are beyond the scope of this work.

1.1.4 *Alignment and Distribution.* High Performance Fortran [High Performance Fortran Forum 1993] maps data to processors in two phases. The first is an *alignment* of data objects to an abstract Cartesian grid called a template, with one data element per template cell; the second is a *distribution* of the template to the processors, typically in a block-cyclic fashion. We intend the algorithms in this article to be used for the alignment phase. Thus we consider a virtual processor array, with one data element allocated per virtual processor.

Clearly, these phases interact; the true cost of communication at the template level depends on the distribution parameters. We do not model this interaction. Iterating between the alignment and distribution phases is one possible approach.

We note, incidentally, that the grid metric on the template does measure real communication cost in some situations. Consider a shift communication of an array with a block-cyclic distribution. If the shift distance is small, then both the volume of data moved between processors and the load on the most congested link are proportional to the shift distance [Chatterjee et al. 1992].

1.1.5 *Storage Optimization.* Our model makes all intermediate values explicit, as does (for example) three-address code [Aho et al. 1986]. We expect that a later phase of compilation will perform storage optimization. Cytron et al. [1991] describe such optimizations in the context of static single-assignment form; Chatterjee [1993] discusses similar optimizations for compiling fine-grained data-parallel programs for shared-memory multiprocessors. We believe that such techniques can be adapted to the storage optimization problem for array expressions.

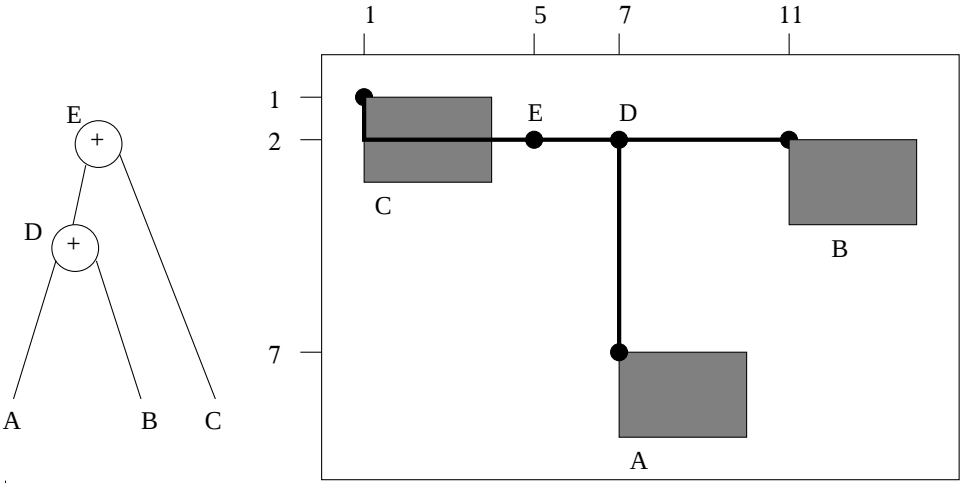

Fig. 1. Optimal evaluation of an expression tree. The tree T for the expression $(A + B) + C$ is on the left. The positions of the leaf arrays A, B, and C are the shaded rectangles. The function π encodes the position as the location of the upper left-hand corner of the array. Thus, $\pi(A) = (7, 7)$, $\pi(B) = (2, 11)$, and $\pi(C) = (1, 1)$. With the grid metric, an optimal placement of intermediate nodes is $\pi(D) = (2, 7)$ and $\pi(E) = (2, 5)$, as shown by the heavy dots. The total cost of the operation with these optimal placements of temporaries is 16, as indicated by the heavy lines.

1.2 Definitions

Throughout the article, T is a binary expression tree that represents a data-parallel expression: the leaves correspond to arrays or array sections (possibly of different shapes), and the internal nodes correspond to binary array operations. Figure 1 contains an example. The height of the expression tree is h; it has N leaves (and therefore $2N - 1$ nodes in all); and its root node is denoted ROOT. For any node x, we write T_x for the subtree of T rooted at x, and write N_x for the number of leaves in that subtree.

A *position space* is a pair $(\mathcal{P}, d)$, where $\mathcal{P}$ is the set of possible positions, and d is a *distance function* from $\mathcal{P} \times \mathcal{P}$ to the nonnegative real numbers. The size of $\mathcal{P}$, or the number of possible positions, is P. The distance function abstracts the interconnection scheme of the parallel machine: $d(p, q)$ is the cost of moving one data element from position p to position q. We assume that the distance function is a metric. In general, the cost of moving an array of w elements from position p to position q is $w \cdot d(p, q)$. In the initial part of this article, we will examine the case where all arrays have the same number of elements (so that $w = 1$ without loss of generality); we will return to the general case in Section 7.

In this framework, the expression-mapping problem is to assign a position in $\mathcal{P}$ to each operation (that is, each internal node of T), assuming that positions have been specified for the operands (that is, for the leaves of T). Let $\pi : \text{NODES}(T) \to \mathcal{P}$ be a *mapping function* from tree nodes to positions. The value of $\pi(x)$ encodes the position at which to compute the data-parallel operation represented by node x.

Figure 1 gives a (somewhat contrived) example of an embedding of a tree into a position space that is a two-dimensional grid. Here $\pi(x)$ is the position of the upper

left-hand corner of the array. The distances in the figure are measured according to the grid or "Manhattan" metric, which is appropriate if shift distance is the main component of communication cost. (If startup cost is the main component, the discrete metric would be used as in Section 5.)

We define $C(T, \mathcal{P}, \pi)$ to be the total cost of the communication required to evaluate expression T in position space $\mathcal{P}$ with node placements π.

$$C(T, \mathcal{P}, \pi) = \sum_{(x,y) \in \text{EDGES}(T)} d(\pi(x), \pi(y)).$$

The total cost in Figure 1 is $C(T, \mathcal{P}, \pi) = 16$. Given the values of π at the leaves of the tree, we wish to extend π to the internal nodes so as to minimize $C(T, \mathcal{P}, \pi)$. We call such an embedding a *minimum-cost* embedding. The embedding in Figure 1 is a minimum-cost embedding.

Finding a minimum-cost embedding is an instance of the *fixed-topology Steiner tree* problem [Winter 1987]. Efficient solutions to this problem are known for some metrics, including grids [Farris 1970; Gilbert and Schreiber 1991], Euclidean distance in the plane [Hwang 1986], and the discrete metric [Fitch 1971; Gilbert and Schreiber 1991; Hartigan 1973]. Our algorithms subsume all these results except the Euclidean, and extend them to metrics based on rings, multidimensional tori, and fat-trees.

1.3 Explicit Dynamic Programming

All our algorithms originate from a simple dynamic programming approach first used by Sankoff and Rousseau [1975] in a biological setting. Mace [1987] later developed a related algorithm for determining array layouts in memories of vector processors. The explicit dynamic programming algorithm exploits the structure of the tree but not of the metric; it can find the best embedding of any tree, for any metric, but it is far too expensive in the general case if the number P of possible positions is large. We will show that for each of the metrics we are interested in—discrete, grids, rings, tori, fat-trees—we can exploit the structure of the metric to reduce the complexity of the dynamic programming algorithm. The running times of our algorithms depend only on the size of the expression and not on the size of the position space.

The following definitions are fundamental and will be used throughout the article. Consider an expression tree T and a position space $\mathcal{P}$. We define the *subtree cost function* $C_x(p)$ to be the minimum cost of evaluating the subtree T_x rooted at an internal node x, subject to the constraint that the subtree's root x be placed at position p.

$$C_x(p) = \min_{\pi} \{C(T_x, \mathcal{P}, \pi) | \pi(x) = p\}.$$

We can express the subtree cost function at a node x in terms of the subtree cost functions at its children y and z. It is just a minimum over all possible placements of y and z.

$$C_x(p) = \min_r \Big(C_y(r) + d(p, r)\Big) + \min_r \Big(C_z(r) + d(p, r)\Big). \tag{1}$$

The key observation is that $C_x(p)$ is the sum of two independent terms, one for each subtree below x. To emphasize this, we rewrite the recurrence in terms of the

contribution function $g_{xy}(p)$, which is the contribution to $C_x(p)$ from the subtree T_y and the edge (x, y). This gives us the following master recurrence

$$\begin{aligned} C_x(p) &= g_{xy}(p) + g_{xz}(p) \\ g_{xy}(p) &= \min_r(C_y(r) + d(p, r)) \\ g_{xz}(p) &= \min_r(C_z(r) + d(p, r)) \end{aligned} \tag{2}$$

for an internal node x with children y and z, and

$$C_x(p) = d(p, \pi(x)) \tag{3}$$

for a leaf node x. Equation (3) reflects the fact that supplying the value of the fixed leaf x at a given position p requires communicating the value from $\pi(x)$ to p.

Finally, we define a position for y that realizes the minimum contribution $g_{xy}(p)$.

$$r^*_{xy}(p) = \text{some } r \text{ at which } C_y(r) + d(p, r) \text{ is minimized.} \tag{4}$$

Algorithm 1.3.1 (*Optimal Embedding Using Explicit Dynamic Programming*)
Input: An expression tree T, a position space $(\mathcal{P}, d)$, leaf positions $\pi(x)$.
Output: A minimum-cost embedding π of T in $\mathcal{P}$.
Method:

(1) Traverse T in postorder, using Eqns. (2) and (4) to tabulate $C_x(p)$ and $r^*_{xy}(p)$ at every nonleaf node x, for all $p \in \mathcal{P}$.

(2) Let $\pi(\text{ROOT}) =$ some position at which C_{ROOT} is minimized.

(3) Traverse T in preorder, setting $\pi(y) = r^*_{xy}(\pi(x))$ for each internal node y, where $x = \text{PARENT}(y)$.

The explicit dynamic programming algorithm (shown as Algorithm 1.3.1) simply tabulates $C_x(p)$ and $r^*_{xy}(p)$ for all positions p and all nodes x, working from the leaves up to the root; it then assigns positions to the nodes, working from the root down to the leaves. Algorithm 1.3.1 works for any distance function. It takes $O(NP^2)$ time and uses $O(NP)$ space.

We are interested in massively parallel machines, for which the number P of positions is large (much larger than the size N or height h of the expression tree). Thus the explicit approach is too expensive. In the rest of this article, we show how to use the properties of some specific distance functions to generate compact representations of subtree cost functions and reduce the cost of dynamic programming to $O(N)$ for grids, $O(Nh)$ for rings and for the discrete metric, and $O(N^3)$ for fat-trees. We call such algorithms *compact dynamic programming* algorithms.

2. PRELIMINARIES

This section discusses three topics that arise in both the grid and ring metrics. First, we show that multidimensional grids and tori can be treated as simple combinations of independent one-dimensional grids and rings. Second, we show that for one-dimensional grids and rings there is always an optimal embedding in which every internal node is placed at the same position as some leaf. Third, we develop a theory of piecewise-linear continuous functions on one-dimensional grids and rings.

2.1 Direct Sums of Distance Functions

Complex distance functions can sometimes be decomposed into direct sums of simpler functions. For example, a k-dimensional grid is the direct sum of 1-dimensional grids (that is, linear arrays); a toroidal mesh is the direct sum of two rings; and a k-dimensional hypercube is the direct sum of k two-point discrete metrics. We will see in Section 7 that the direct sum of a discrete metric and a grid metric is also useful.

The formal definition of direct-sum metrics is as follows. Let d_i be a real-valued distance function on $\mathcal{P}_i$, for each i from 1 to k. The *direct sum* of the d_i is the distance function d on $\mathcal{P}_1 \times \cdots \times \mathcal{P}_k$ defined by

$$d((p_1, \ldots, p_k), (q_1, \ldots, q_k)) = \sum_{i=1}^{k} d_i(p_i, q_i).$$

We write $d = d_1 \oplus \cdots \oplus d_k = \oplus_{i=1}^{k} d_i$.

The significance of direct-sum metrics is that the mapping problem can be decomposed into an independent problem in each summand metric. Thus, multidimensional grids are no harder than linear arrays, tori are no harder than rings, and so forth.

LEMMA 2.1.1. *Let $(\mathcal{P}_i, d_i)$ be a position space for each i from 1 to k. Let $(\mathcal{P}, d)$ be the position space in which $\mathcal{P} = \times_{i=1}^{k} \mathcal{P}_i$ and $d = \oplus_{i=1}^{k} d_i$. Let T be an expression tree, π an embedding of T in $\mathcal{P}$, and $\pi_i : \text{NODES}(T) \to \mathcal{P}_i$ the projection of π in dimension i. Then π is a minimum-cost embedding of T in $\mathcal{P}$ if and only if every π_i is a minimum-cost embedding of T in $\mathcal{P}_i$.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

2.2 Continuous Grids and Rings

For grids and rings, the algorithm is easier to describe if we take the set of positions not to be the integers 0 through $P-1$, but rather the closed real interval $[0, P-1]$ (for one-dimensional grids) or the half-open real interval $[0, P)$ (for rings). The continuous one-dimensional grid metric is $d(p, q) = |p - q|$; the continuous one-dimensional ring metric is $d(p, q) = \min(|p - q|, P - |p - q|)$. Equations (2), (3), and (4) still hold, but are interpreted in real numbers. The basic theory is the same for rings as for grids with one exception: for rings, we will think of the half-open interval $[0, P)$ as being the real numbers modulo P, so that P is identified with 0. Thus, in the context of rings, addition or subtraction of positions is to be interpreted modulo P.

Of course, for our results to be meaningful on a discrete grid or ring, we need to ensure that all the positions we compute are in fact integers. In this section we prove that any expression tree (even a weighted one as defined in Section 7) can be embedded optimally in a continuous one-dimensional grid or ring with every internal node placed at the same position as a leaf. The significance of this is twofold. First, if leaves are placed at integer locations, we are guaranteed that we can find an optimal position all of whose locations are integers. Second, this restriction reduces the time complexity of Algorithm 1.3.1 from $O(NP^2)$ to $O(N^3)$.

We will see in Section 7 that the same result holds for the discrete and fat-tree

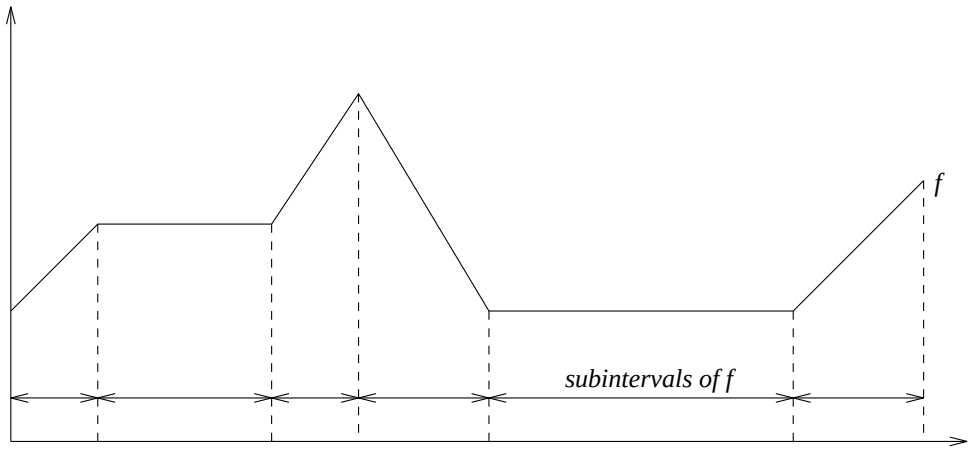

Fig. 2. A piecewise-linear continuous (PLC) function.

metrics, and the reader may be tempted to conclude that every metric admits optimal embeddings with all nodes at leaf positions. This is not true, however, even for a two-dimensional grid. In Figure 1 node D cannot be optimally placed at any leaf position. All projections of node positions are projections of leaf positions, which is always possible in a direct sum metric.

THEOREM 2.2.1. *Let T be an expression tree, d be either the one-dimensional continuous grid metric on $[0, P-1]$ or the one-dimensional continuous ring metric on $[0, P)$, and π be an embedding of the leaves of T. Then there is a minimum-cost embedding of the internal nodes of T that agrees with π on the leaves, and that embeds every internal node at the position of some leaf.*

PROOF. See the Appendix. □

Theorem 2.2.1 says that if leaf positions are integers, then there exist integer optimal positions for the internal nodes. Most of our algorithms will always find integer positions. Where this is not the case, the construction above can be applied in $O(N)$ time to make all the optimal solutions integral. We will not explicitly refer to this cleanup phase in the rest of the article.

We remark that Theorem 2.2.1 holds (with the same proof) even if we label the edges of T with nonnegative real weights, provided we consider an embedding to be optimal if it minimizes the weighted sum of the edge lengths. We will use this version of the theorem in Section 7.

2.3 Piecewise-Linear Continuous Functions

Let f be a *piecewise-linear continuous function* (or PLC function for short), defined on either $[0, P-1]$ or $[0, P)$. Then f is linear except possibly at a sequence $p_1 \leq \cdots \leq p_k$ of *breakpoints*, which we take to include 0 and $P-1$ in the grid case. We define the *subintervals* of f as the intervals $S_i = [p_i, p_{i+1}]$, for $1 \leq i < k$. In the ring case, there is an additional subinterval $S_k = [p_k, p_1]$. The *slope* m_i of f is constant on the interior of each subinterval. We define $\mathcal{B}(f) = (p_1, \ldots, p_k)$ as the ordered sequence of breakpoints, $\mathcal{S}(f)$ as the ordered sequence of subintervals, and $\mathcal{M}(f)$ as

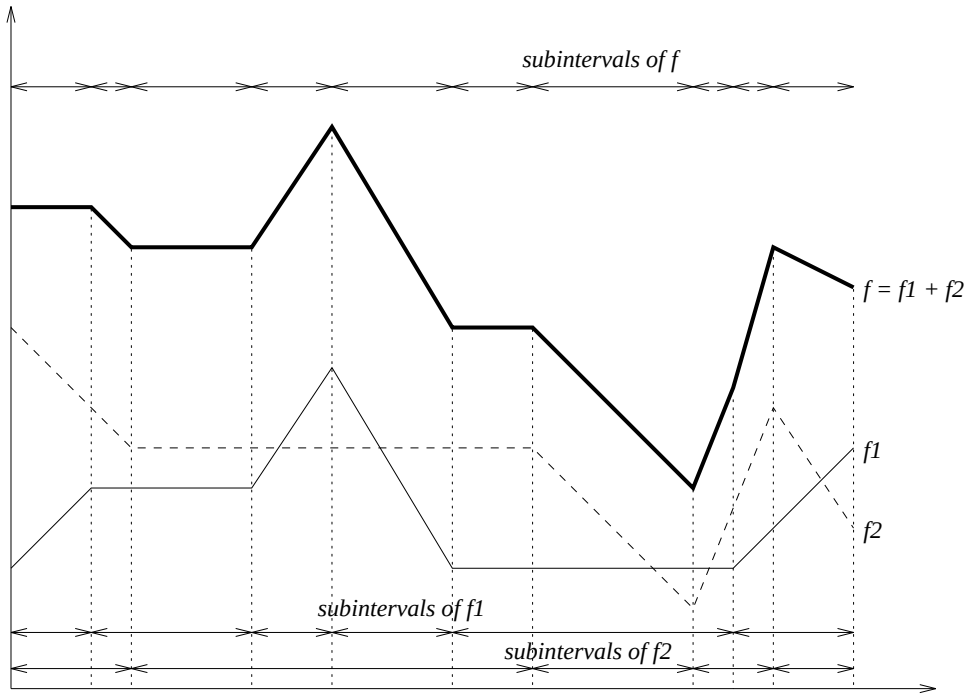

Fig. 3. The sum of two PLC functions.

the ordered sequence of slopes. We include the possibility that $p_i = p_{i+1}$, in which case we call the subinterval S_i *degenerate*. We define the slope of a degenerate subinterval as any convenient value. Figure 2 is an example of a PLC function.

The following lemma states some elementary facts about sums of PLC functions; see Figure 3.

LEMMA 2.3.1. *Let $f = f_1 + f_2$, where f_1 and f_2 are PLC functions defined on the same interval. Then breakpoints can be chosen for f so that:*

(1) f is PLC.

(2) $|\mathcal{S}(f)| \leq |\mathcal{S}(f_1)| + |\mathcal{S}(f_2)|$.

(3) $\mathcal{B}(f) \subseteq \mathcal{B}(f_1) \cup \mathcal{B}(f_2)$.

(4) If $m \in \mathcal{M}(f)$ is the slope of f on subinterval $S \in \mathcal{S}(f)$, then $m = m_1 + m_2$ where m_i is the slope of f_i on the unique subinterval $S_i \in \mathcal{S}(f_i)$ such that $S \subseteq S_i, (i = 1, 2)$.

(5) $\max(\mathcal{M}(f)) \leq \max(\mathcal{M}(f_1)) + \max(\mathcal{M}(f_2))$.

(6) $\min(\mathcal{M}(f)) \geq \min(\mathcal{M}(f_1)) + \min(\mathcal{M}(f_2))$.

PROOF. See the technical report [Chatterjee et al. 1992]. □

For each fixed position q, the grid or ring distance function $d(p, q)$ is a real-valued PLC function of one variable p, defined on $[0, P-1]$ for a grid or $[0, P)$ for a ring. Its slopes are $+1$ and -1. For a grid, its breakpoints are 0, q, and $P-1$; for a ring,

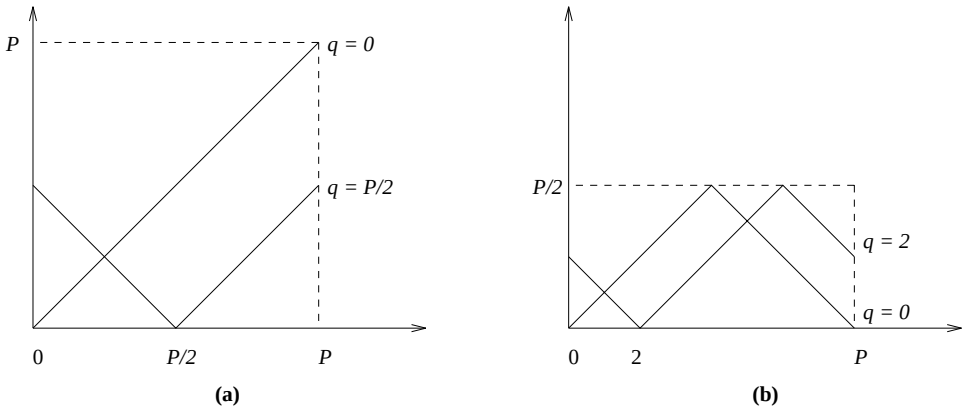

Fig. 4. (a) Grid distance $d(p,0)$ and $d(p,P/2)$. (b) Ring distance $d(p,0)$ and $d(p,2)$.

it has one breakpoint at q and another at $\Phi(q) = (q + P/2) \bmod P$, which we call the *antipode* of q. Figure 4 shows the two distance functions.

If the distance function has slopes of magnitude at most one, then the subtree cost function has slopes of magnitude at most two.

LEMMA 2.3.2. *Let d be a PLC distance function with slopes between -1 and $+1$. Let a binary expression tree be given. Then the subtree cost function C_x has slopes of magnitude at most 2.*

PROOF. We show that $C_x(p \pm 1) \leq C_x(p) + 2$. Let the children of node x be y and z. Embed T_x optimally subject to $\pi(x) = p$, with cost $C_x(p)$. Move x one place to the left or right without moving any other node. This is a (possibly suboptimal) placement of T_x subject to $\pi(x) = p \pm 1$, with cost no less than $C_x(p \pm 1)$. Its cost clearly exceeds $C_x(p)$ by at most 2. □

Finally, we show that $r^*_{xy}(p)$, the best position for node y given that its parent node x is placed at position p, can always be chosen to be either p or a breakpoint of y's subtree cost function C_y.

LEMMA 2.3.3. *Let d be the distance function of a one-dimensional grid or ring. Let p be a position, and y an internal node of an expression tree. Then there exists an $r^*_{xy}(p) \in \mathcal{B}(C_y) \cup \{p\}$.*

PROOF. Lemma 2.3.1 implies that $C_y(r) + d(p,r)$ is PLC as a function of r, with breakpoints in $\mathcal{B}(C_y) \cup \mathcal{B}(d(p,\cdot))$. Thus a minimum of $C_y(r) + d(p,r)$ occurs at the endpoint of a subinterval, that is, at a breakpoint. In the case of a ring, the breakpoint at $\Phi(p)$ is a maximum of $d(p,\cdot)$ and can be a minimum of the sum only if it is a minimum of C_y. Thus a minimum of the sum occurs either somewhere in $\mathcal{B}(C_y)$ or at p. □

3. GRIDS AND HYPERCUBES

For a k-dimensional grid, positions are k-tuples of integers, and the distance function d is defined as

$$d((p_1, \ldots, p_k), (q_1, \ldots, q_k)) = \sum_{i=1}^{k} |p_i - q_i|. \tag{5}$$

Since the distance is the direct sum of the distances along the k directions, Lemma 2.1.1 applies, and we can treat each dimension independently. A hypercube is a special case of a grid (with $k = \log_2 P$), so everything in this section applies to hypercubes as well.

We will simplify Mace's algorithm for a one-dimensional grid by showing that every C_x is a PLC function of position p, with at most five pieces. Thus, for each x, we can represent C_x by only a constant amount of information, instead of by tabulating all P of its values. We will actually show that all we need to compute are the endpoints of the interval on which C_x is minimum. Our algorithm requires $O(N)$ time and space, which is the same complexity as the algorithm of Gilbert and Schreiber [1991] that used the robustness of the grid metric; indeed, the compact dynamic programming algorithm turns out to be the same as their algorithm for this case.

3.1 Compact Dynamic Programming for Grids

Throughout this section we fix a one-dimensional grid $\mathcal{P}$ and a binary expression tree T, and define the functions C_x, g_{xy}, and r^*_{xy} as in Section 1.3.

We first show by induction that the subtree cost functions C_x have a simple form: they are PLC with slopes $(-2, -1, 0, 1, 2)$. The induction step uses the following lemma to get from the C's to the g's.

LEMMA 3.1.1. *Let C_y be PLC with slopes $(-2, -1, 0, 1, 2)$ and breakpoints $(0, p_1, p_2, p_3, p_4, P-1)$. Then g_{xy} is PLC with slopes $(-1, 0, 1)$ and breakpoints $(0, p_2, p_3, P-1)$. Furthermore, $r^*_{xy}(p)$ may always be chosen in $[p_2, p_3]$.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

The other half of the induction is the step from the g's to the C's:

LEMMA 3.1.2. *Let g_{xy} and g_{xz} be PLC functions, both with slopes $(-1, 0, 1)$, and with breakpoints $\mathcal{B}(g_{xy})$ and $\mathcal{B}(g_{xz})$. Then $C_x = g_{xy} + g_{xz}$ is also PLC, with slopes $(-2, -1, 0, 1, 2)$ and breakpoints $\mathcal{B}(g_{xy}) \cup \mathcal{B}(g_{xz})$.*

PROOF. This follows from points 3 and 4 of Lemma 2.3.1. □

This is all we need for the induction.

THEOREM 3.1.3. *The subtree cost function C_x for every node x of the expression tree T has exactly five subintervals (some of which may be degenerate), with slopes $\mathcal{M}(C_x) = (-2, -1, 0, 1, 2)$.*

PROOF. We use induction on the height of T_x.

The base case is a tree of height 0, whose single node is a leaf. By Equation (3), the subtree cost function is $C_x(p) = d(\pi(x), p)$, which is PLC with three subintervals

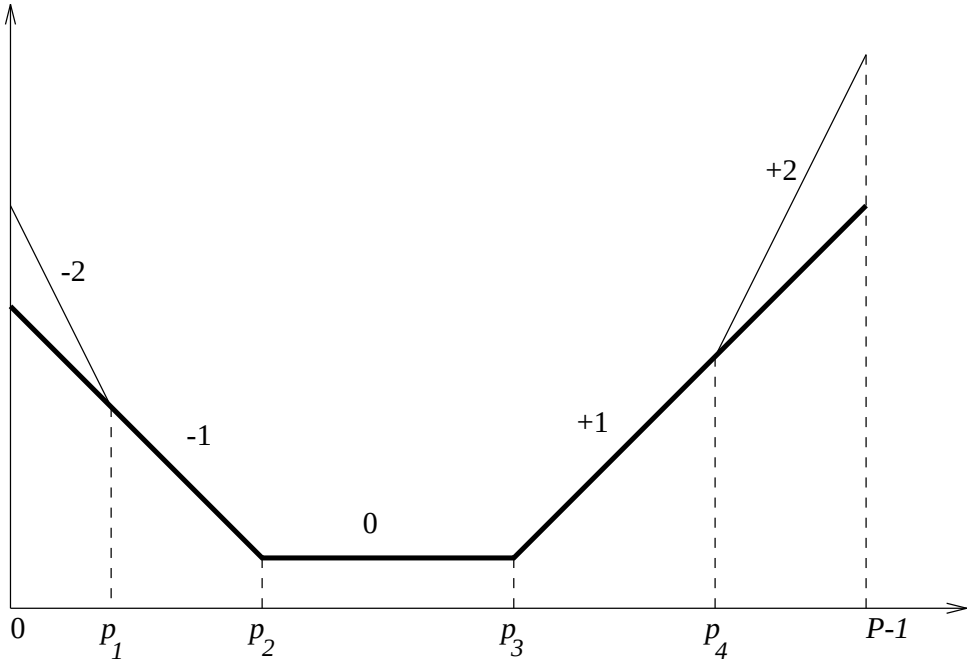

Fig. 5. The contribution function for the grid metric. The light lines show the subtree cost function C_y, and the heavy lines show the contribution function g_{xy}.

of slopes $(-1, 0, 1)$, the middle one being degenerate. We consider this to have two more degenerate subintervals, of slopes -2 and 2, at positions 0 and $P-1$.

The induction step is then just Lemmas 3.1.1 and 3.1.2. □

The efficient algorithm for grids (Algorithm 3.1.4) follows the explicit dynamic programming outline, but instead of enumerating all the values of C_x at each node x it just computes the breakpoints. Indeed, we see from Lemma 3.1.1 that only the two middle breakpoints of C_y contribute to g_{xy} or r^*_{xy}. Thus the algorithm just keeps track of the two middle breakpoints, which are the endpoints of the central flat subinterval of C_y. This turns out to be the same computation as in Gilbert and Schreiber's algorithm; the flat subinterval is a "generalized intersection" in their terminology. The algorithm uses only a constant amount of time and space at each node.

Algorithm 3.1.4 (*Optimal Embedding in a Grid*)

Input: A position space $(\mathcal{P}, d)$ and an expression tree T with an embedding π for its leaves.

Output: A minimum-cost embedding π of T in $\mathcal{P}$.

Method: For each dimension of the grid, carry out the following to compute one dimension of the embedding.

(1) Traverse T in postorder, computing two numbers L and R for each node. $L(x) = R(x) = \pi(x)$ at a leaf node x. At a nonleaf node x with children y and z, $L(x)$ is the second smallest and $R(x)$ the second largest of the four positions $L(y)$, $R(y)$, $L(z)$, and $R(z)$.

(2) Set $\pi(\text{ROOT})$ to any integer in $[L(\text{ROOT}), R(\text{ROOT})]$.

(3) Traverse T in preorder. For each internal node y with parent x, set $\pi(y)$ to the point $p \in [L(y), R(y)]$ that is closest to $\pi(x)$. Thus,

$$\pi(y) = \begin{cases} L(y), & \text{if } \pi(x) \le L(y) \\ \pi(x), & \text{if } L(y) \le \pi(x) \le R(y) \\ R(y), & \text{if } R(y) \le \pi(x). \end{cases}$$

THEOREM 3.1.5. *For an expression tree T with N leaves, Algorithm 3.1.4 solves the optimal-placement problem correctly for a k-dimensional grid in $O(kN)$ time.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

4. RINGS

For a k-dimensional ring of size $P_1 \times \cdots \times P_k$, the distance function d is defined as

$$d((p_1, \ldots, p_k), (q_1, \ldots, q_k)) = \sum_{i=1}^{k} \min(|p_i - q_i|, P_i - |p_i - q_i|). \tag{6}$$

This distance function is the direct sum of the distances in the k orthogonal directions. Lemma 2.1.1 says that we can treat each dimension independently. We will therefore take the position space $\mathcal{P}$ to be a one-dimensional ring of size P for the rest of this section.

The ring metric is not robust in the sense of Gilbert and Schreiber [1991]; thus their methods do not apply. Our approach will be similar to that for the grid metric: we will show that the function C_x has a simple form that can be computed much more easily than by tabulating all P values. As in the grid case, C_x will turn out to be PLC for each node x. Now, however, the number of linear pieces is proportional to the number of leaves in subtree T_x, not constant as in the grid case. Thus the algorithm will take $O(Nh)$ time in all, where h is the height of the expression tree.

4.1 Compact Dynamic Programming for Rings: Theory

We extend the subtree cost function C_x and the contribution function g_{xy} to the half-open real interval $[0, P)$. As described in Section 2.3, we consider functions on the real numbers modulo P, so position P is identified with 0. The optimal placements that our algorithm derives will be integers.

Our goal is to prove that the subtree cost function C_x is PLC, with a number of subintervals linear in the size of the subtree. We will show that C_x is PLC with slopes $0, \pm 1$, and ± 2, and with at most $2N$ subintervals if the subtree has N leaves. The proof is by induction on the height of the tree. The induction hypothesis also describes the contribution function g_{xy} for a child y of x. We will show that g_{xy} is PLC with slopes 0 and ± 1.

For the induction step, we consider a child y of x and prove a series of lemmas about g_{xy} and C_x, starting from the hypothesis that C_y is PLC with integral slopes. The details of the induction are messy but not difficult; the squeamish reader may wish to skip directly to Section 4.2.

If p, q, and r are positions, an inequality like $p \le q$ is meaningless for rings, since one may proceed from p to q in either direction around the ring. However, we define

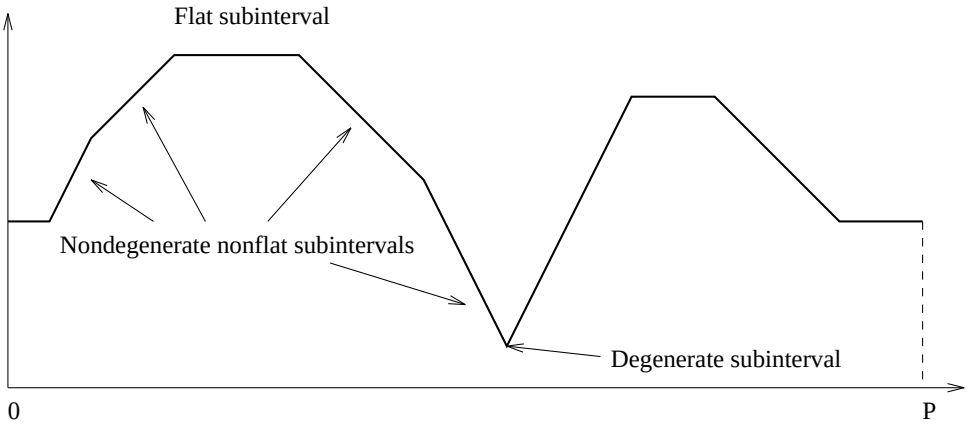

Fig. 6. Various kinds of intervals in the cost function for the ring metric.

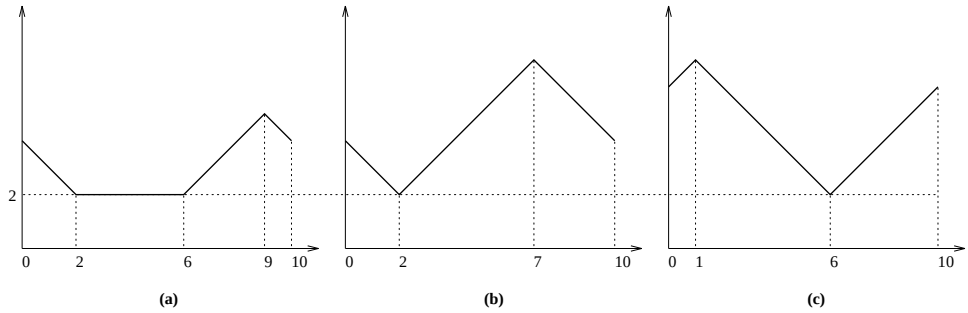

Fig. 7. The possible forms of the function g^i_{xy} for the ring metric. In all cases, S_i is the interval $(2, 6)$, the number of positions is 10, and the smallest value of C_y on the interval is 2. (a) S_i has zero slope. (b) S_i has positive slope. (c) S_i has negative slope.

a double inequality like $p \leq q \leq r$ to mean that $(q - p) \bmod P \leq (r - p) \bmod P$; in other words, as one proceeds around the ring in the increasing direction from p, one encounters q before r. For example, if $P = 8$, then $2 \leq 4 \leq 7$ and $4 \leq 7 \leq 2$, but not $4 \leq 2 \leq 7$. Double inequalities on ring positions using $\geq$, $<$, and $>$ are defined similarly. Recall also that the *antipode* of a position p on the ring is $\Phi(p) = (p + P/2) \bmod P$.

Let y be a child of x in the expression tree. Suppose, for the purposes of induction, that C_y is PLC with integral slopes. We represent C_y by a sequence $\mathcal{B}(C_y)$ of breakpoints, with associated subintervals $\mathcal{S}(C_y)$ having slopes $\mathcal{M}(C_y)$. A *flat subinterval* of C_y is one on which C_y is constant. For purposes of this proof only, we will posit that no two adjacent subintervals of C_y have slopes of opposite sign; if necessary, we introduce a *degenerate flat subinterval* of zero length between any two such subintervals. We take the PLC representation of C_y to have no zero-length subintervals other than these, and to have no two adjacent intervals of the same slope. Figure 6 is an example of a subtree cost function for the ring metric, showing the various kinds of subintervals.

Now consider the contribution function g_{xy}. It is defined (in Equation (2)) as a

minimum over positions r for node y. If we break that minimum up according to the subintervals of C_y, we get

$$g_{xy}(p) = \min_{S_i \in \mathcal{S}(C_y)} g^i_{xy}(p), \tag{7}$$

where

$$g^i_{xy}(p) = \min_{r \in S_i} (C_y(r) + d(p, r)) \tag{8}$$

is the "best" g_{xy} among those with y positioned in the ith subinterval of C_y.

We can find a closed form for g^i_{xy} as follows. Let $S_i = [p_i, p_{i+1}]$ be the ith subinterval of C_y. If S_i is flat, then $g^i_{xy}(p) = C_y(p_i) + \min_{r \in S_i} d(p, r)$. If p is within S_i, then the smallest $d(p, r)$ is 0 at $r = p$. If p is outside S_i, then the distance $d(p, r)$ is smallest when r is the endpoint of S_i closer to p. The closer endpoint changes at the antipode of the midpoint of the subinterval. Thus, for a flat subinterval S_i,

$$g^i_{xy}(p) = \begin{cases} C_y(p_i) + (p_i - p), & \text{if } \Phi((p_i + p_{i+1})/2) \le p \le p_i \\ C_y(p_i), & \text{if } p_i \le p \le p_{i+1} \\ C_y(p_i) + (p - p_{i+1}), & \text{if } p_{i+1} \le p \le \Phi((p_i + p_{i+1})/2). \end{cases} \tag{9}$$

Remember that arithmetic on positions, like $(p - p_{i+1})$, is done modulo P, yielding a number in $[0, P)$. Now suppose that C_y has positive slope on S_i. The slope is an integer, so it is at least 1. This implies that, for any position p, the minimum value of $C_y(r) + d(p, r)$ over all positions r in S_j occurs when r is the left endpoint p_i of S_i; moving r away from p_i increases $C_y(r)$ at least as fast as it decreases $d(p, r)$. Substituting $r = p_i$ into the definition of g^i_{xy} and using the definition of the distance d, we see that, in the case that C_y has positive slope on subinterval S_i,

$$g^i_{xy}(p) = \begin{cases} C_y(p_i) + (p_i - p), & \text{if } \Phi(p_i) \le p \le p_i \\ C_y(p_i) + (p - p_i), & \text{if } p_i \le p \le \Phi(p_i). \end{cases} \tag{10}$$

If C_y has negative slope, $g^i_{xy}(p)$ is as above but with p_{i+1} substituted for p_i.

Therefore, in every case (still under the inductive assumption that C_y is PLC with integral slopes), g^i_{xy} is PLC with either two or three subintervals, with slopes -1, 1, and possibly 0. Figure 7 shows the three possible forms of g^i_{xy}.

Now g_{xy} is the minimum of the g^i_{xy} functions, and therefore g_{xy} is also PLC with slopes -1, 1, and 0. We proceed to bound the number of subintervals of g_{xy}. First we show that the minimization in equation (7) can omit those g^j_{xy} that correspond to nonflat subintervals S_j of C_y, because they are dominated by g^i_{xy} corresponding to flat subintervals.

LEMMA 4.1.1. *Let C_y be the subtree cost function for node y, and suppose C_y is PLC as above. Let S_j be a subinterval of C_y with nonzero slope. Then there is some flat subinterval S_i such that $g^i_{xy}(p) \le g^j_{xy}(p)$ for all p.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

The remaining terms in the minimization (Equation (7)) are the g^i_{xy} that correspond to flat subintervals S_i of C_y. Equation (9) gives a closed form for those functions. The next lemma says that their minimum g_{xy} is itself PLC with at most two subintervals per term in the minimization.

LEMMA 4.1.2. *Let C_y be the subtree cost function for node y, and suppose C_y is PLC with a representation as described above. Let the PLC representation for C_y have f nondegenerate flat subintervals and d degenerate flat intervals. Then g_{xy} is PLC with slopes 0 and ± 1, with at most $d + 2f$ nondegenerate subintervals.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

The argument above takes the induction step from the subtree cost function C_y for the child node y to the contribution function g_{xy}. To complete the induction, we go from g_{xy} to C_x. This is the main theorem of this section.

THEOREM 4.1.3. *For a binary expression tree T with N leaves, the subtree cost function C_x for any node x of T has a PLC representation with slopes 0, ± 1, and ± 2, and with at most $2N$ nondegenerate subintervals.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

4.2 Compact Dynamic Programming for Rings: Algorithm

We now present an $O(Nh)$-time algorithm (Algorithm 4.2.1) for computing an optimal placement of an expression tree on a ring. The algorithm hinges on a compressed representation of the subtree cost function $C_x(p)$. Each function in the algorithm is PLC and is represented as a set of breakpoints, values, and slopes, sorted around the ring.

Algorithm 4.2.1 (*Optimal Embedding in a Ring*)
Input: A position space $(\mathcal{P}, d)$ and an expression tree T with positions π for its leaves.
Output: A minimal-cost embedding π of T in $\mathcal{P}$.
Method:

(1) Traverse T in postorder. If x is a leaf, do nothing. If x is an internal node with children y and z:
   (a) Call Convolve(C_y) to compute g_{xy};
   (b) Call Convolve(C_z) to compute g_{xz};
   (c) Call Sum(g_{xy}, g_{xz}) to compute $C_x = g_{xy} + g_{xz}$.
(2) Let π(ROOT) be some integer r at which $C_{\text{ROOT}}(r)$ is minimized.
(3) Traverse T in preorder. For each internal node y with parent x, set $\pi(y) = r^*_{xy}(\pi(x))$.

SUBROUTINE: (CONVOLVE)
Input: A PLC function C_y and an integer P, the size of the position space.
Output: A PLC representation of $g_{xy}(p) = \min_{0 \le r < P}(C_x(r) + d(r, p))$.
Method:

(1) Let S_1, S_2, ..., S_k, $S_{k+1} = S_1$ be the sequence of intervals (both degenerate and nondegenerate) of C_y; let $x = \text{PARENT}(y)$.
(2) Initialize $g_{xy} = +\infty$ as a constant function on $[0, P)$.
(3) For $i = 1$ to $k + 1$
   (a) If S_i is nonflat, then do nothing.
   (b) If S_i is flat, first compute g^i_{xy} from Equation (9).
   Then compute $g_{xy} = \min(g_{xy}, g^i_{xy})$ by simultaneously traversing the lists of breakpoints for g^i_{xy} and g_{xy}. A breakpoint for the new g_{xy} may come either at a breakpoint of g^i_{xy} or g_{xy}, or at an intersection of the graphs of g^i_{xy} and g_{xy}. Between breakpoints, the new g_{xy} takes its slope and value from the smaller of g^i_{xy} and the old g_{xy}.

SUBROUTINE: (SUM)
Input: Two PLC functions g_{xy} and g_{xz} and an integer P, the size of the position space.
Output: A PLC representation of $C_x = g_{xy} + g_{xz}$.
Method: Simultaneously traverse the lists of breakpoints for g_{xy} and g_{xz}. A breakpoint for C_x may come only at a breakpoint for g_{xy} or g_{xz}. Between breakpoints, the slope and value of C_x are the sum of those for g_{xy} and g_{xz}.

If node y has N_y leaves and node z has N_z leaves, then their parent x has $N_x = N_y + N_z$ leaves. By Theorem 4.1.3, C_y and C_z have at most $2N_y$ and $2N_z$ nondegenerate subintervals. Furthermore, by Lemma 4.1.2, g_{xy} and g_{xz} also have at most $2N_y$ and $2N_z$ nondegenerate subintervals.

LEMMA 4.2.2. *Suppose g_{xy} and g_{xz} are PLC and have at most $2N_y$ and $2N_z$ subintervals, respectively, given in order around the ring. Then Sum(g_{xy}, g_{xz}) runs in $O(N_y + N_z)$ time.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

LEMMA 4.2.3. *If C_y is PLC with at most $2N_y$ subintervals given in a sorted order along the ring, then Convolve(C_y) computes g_{xy} in $O(N_y)$ time.*

PROOF. Equation (7) says that the output g_{xy} can be written as the minimum of one function g^i_{xy} for each subinterval S_i of C_y. Lemma 4.1.1 says we need only consider those g^i_{xy} for which C_y is flat on S_i. Thus Convolve computes the correct result.

Each g^i_{xy} has at most three subintervals, by Equation (9). Thus, computing all the g^i_{xy} takes $O(N_y)$ total time. The total time for all the merge steps is also $O(N_y)$: each step takes constant time, and each step either computes a new breakpoint for g_{xy} or consumes a breakpoint of one of the g^i_{xy}'s. Since the total number of breakpoints of g_{xy} and C_x is at most $8N_y$, the total number of steps is $O(N_y)$. □

THEOREM 4.2.4. *For an expression tree T with N leaves and height h, Algorithm 4.2.1 solves the optimal-placement problem correctly for a k-dimensional ring in $O(kNh)$ time.*

PROOF. For a node x with children y and z, the previous two lemmas show that it takes $O(N_y)$ time to compute g_{xy}, $O(N_z)$ time to compute g_{xz}, and $O(N_x)$ time to compute C_x from g_{xy} and g_{xz}. Thus the running time for the complete algorithm is on the order of

$$\sum_{x \in \text{NODES}} |\text{LEAVES}(T_x)|. \tag{11}$$

This sum counts every leaf in T once for each of its ancestors. Thus, each of the N leaves is counted at most h times, giving $O(Nh)$ in each of the k dimensions, for a total running time of $O(kNh)$. □

The ring algorithm runs in $O(N^2)$ time for any expression tree; if the height of T is $O(\log N)$ (for example, if T is a complete binary tree), then the running time is $O(N \log N)$.

5. THE DISCRETE METRIC

The discrete metric d is defined as follows:

$$d(p,q) = \begin{cases} 0, & \text{if } p = q \\ 1, & \text{if } p \neq q. \end{cases} \tag{12}$$

This is a reasonable model for the communication cost of converting a data structure from one representation to another, for example, transposing arrays or operating on array sections with different strides. The discrete metric is also a good approximation to the cost of changing context flags in a SIMD environment, the communication cost for message-passing architectures with large message startup times, and the communication cost in architectures with wormhole routing [Dally and Seitz 1986], where a single message is pipelined so that the distance it has to travel does not contribute significantly to its cost.

Gilbert and Schreiber [1991] gave an optimal algorithm for the discrete metric. In their terminology, the discrete metric is "robust,"[1] and therefore, for each node x, one need only record $\textsc{opt}(x)$, the set of positions p for x that minimize $C_x(p)$, instead of C_x itself. Rephrasing their Theorem 2 in our terminology yields:

—$C_x(p) \geq \text{mincost}(T_x) + d(p, \textsc{opt}(x))$.

—For every $p \in \textsc{opt}(x)$, we can choose $r^*_{xy}(p) \in \textsc{opt}(y)$ and $r^*_{xz}(p) \in \textsc{opt}(z)$.

They show that the OPT set of a parent is either the union or the intersection of the OPT sets of its children. We reproduce their algorithm here in our notation, and show that it runs in $O(Nh)$ time.

Algorithm 5.1 (*Optimal Embedding in the Discrete Metric*)
Input: A position space $(\mathcal{P}, d)$ and an expression tree T with positions π for its leaves.
Output: A minimum-cost embedding π of T in $\mathcal{P}$.
Method:

(1) Traverse T in postorder. For each internal node x with children y and z, compute the set $\textsc{opt}(x)$ of possible placements of x that will result in a minimum-cost embedding for T_x. Thus

$$\textsc{opt}(x) = \begin{cases} \textsc{opt}(y) \cup \textsc{opt}(z), & \text{if } \textsc{opt}(y) \cap \textsc{opt}(z) = \emptyset \\ \textsc{opt}(y) \cap \textsc{opt}(z), & \text{otherwise.} \end{cases}$$

(2) Let $\pi(\textsc{root}) = $ some p in $\textsc{opt}(\textsc{root})$.

(3) Traverse T in preorder. For each internal node x, set $\pi(x)$ as follows:

$$\pi(x) = \begin{cases} \pi(\textsc{parent}(x)), & \text{if } \pi(\textsc{parent}(x)) \in \textsc{opt}(x) \\ \text{some } p \in \textsc{opt}(x), & \text{otherwise.} \end{cases}$$

THEOREM 5.2. *Algorithm 5.1 computes an optimal placement of an expression tree with N leaves and height h for the discrete metric in $O(Nh)$ time.*

PROOF. We omit the proof of correctness; see Gilbert and Schreiber [1991]. To get the time bound, we represent sets of positions as sorted lists. Computing a

[1] Robustness amounts to a strengthening of the triangle inequality. The property of robust metrics that is relevant here is that an embedding π of an (unweighted) expression tree T in a robust metric space is optimal if and only if π also embeds every subtree of T optimally.

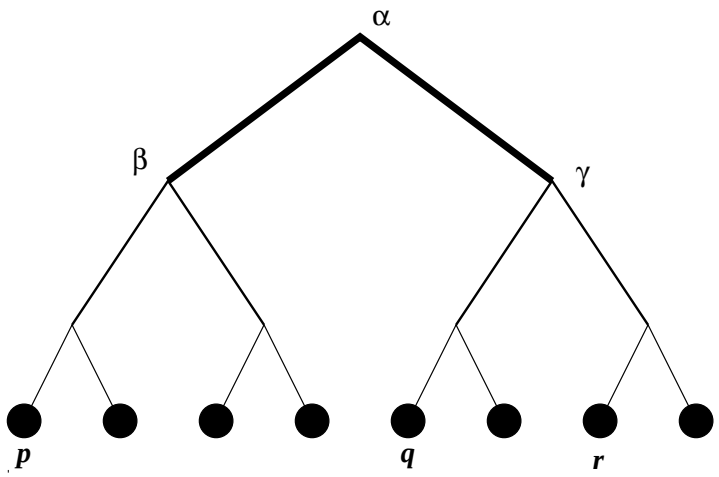

Fig. 8. An 8-node fat-tree. The bandwidth of the links doubles at every level of the tree.

union or intersection in step 2 takes time proportional to the sum of the lengths of the lists, as does the membership test in step 3. Since the set $\textsc{opt}(x)$ contains only leaves of the subtree T_x, the total time is on the order of

$$\sum_{x \in \textsc{nodes}} |\textsc{leaves}(T_x)|. \tag{13}$$

As in Theorem 4.2.4, this sum is $O(Nh)$. □

Like the ring algorithm in Section 4.2, this algorithm runs in $O(N^2)$ time for any expression tree, and $O(N \log N)$ if the tree is balanced.

6. FAT-TREES

The fat-tree machine architecture is an attempt to provide communication that behaves according to the discrete metric, with all communication actions having the same cost [Leiserson 1985]. The processors are located at the leaves of a binary tree of height h, so that $P = 2^h$. (This connection tree is not to be confused with the expression tree T.) The bandwidth of the links doubles at every level as we move upward in the tree, which prevents link congestion at any level. This argues that the discrete-metric algorithm from Section 5 is the best one to use for fat-trees, which is what we advocate in practice.

However, the discrete metric is not a completely accurate model for two reasons. First, the point-to-point communication time depends on how many levels of the tree the route traverses. This is important only when communicating small data aggregates, since moving a large enough distributed array always involves at least one route through the root. Second, real implementations of fat-trees (such as the network architecture of the CM-5 [Leiserson et al. 1992]) do not increase link bandwidths at the higher levels of the tree fast enough to meet the theoretical bounds, thereby making congestion an issue.

A complete analysis of this "nearly fat-tree" metric is an open question. We have analyzed the two extremes.

—*The "large-aggregate" case*: Data objects are distributed uniformly over all the processors, and communication cost depends on link congestion rather than path length. In this case, which we do not discuss in detail here, the cost of moving an object by distance d in the machine is $O(d^{1-\log_2 f})$, where f is the rate at which "fatness" grows up the tree. For $f = 2$, a true fat-tree, this reduces to the discrete metric.

—*The "small-aggregate" case*: All communication is point-to-point, and cost is proportional to the number of tree levels traversed by the route. We discuss this case in the remainder of this section.

We emphasize that a realistic complete model would need to combine elements of both extremes; the discrete metric is probably the best simple model for this architecture.

6.1 The Small-Aggregate Case

For any two positions p and q (that is, leaves of the fat-tree), we let $\textsc{lca}(p, q)$ denote the least common ancestor of p and q in the tree. The distance between positions p and q is defined to be the distance in the fat-tree, so

$$d(p, q) = 2 \cdot \textsc{height}(\textsc{lca}(p, q)), \tag{14}$$

where $\textsc{height}(t)$ is the height of the subtree of the connection tree rooted at t. We call this the *fat-tree metric*. (If p and q are integers from 0 to $P - 1$, this distance is equal to twice the index of the highest bit in which their binary representations differ.)

The triangle inequality relates the distances among any three points in an arbitrary metric. A curious fact about the fat-tree metric is that every triangle is isosceles, in the sense that its two longest sides are equal.

Lemma 6.1.1. (Isosceles Triangle Inequality) *Let p, q, and r be any three distinct positions in the fat-tree metric. Then exactly two of the distances $d(p, q)$, $d(q, r)$, and $d(p, r)$ are equal, and the third distance is smaller than the other two.*

Proof. See the technical report [Chatterjee et al. 1992]. □

The labeled nodes in Figure 8 are an example of Lemma 6.1.1, with $d(q, r) = 4$ and $d(p, q) = d(p, r) = 6$.

The key fact that gives us an $O(N^3)$-time algorithm for the fat-tree metric is that there is always an optimal embedding that places every internal node is at the same position as one of its children. This fact follows from the isosceles triangle inequality.

Theorem 6.1.2. *There is a minimum-cost embedding of the expression tree T for the fat-tree metric in which each internal node x is placed at one of the leaf positions of T_x.*

Proof. See the Appendix. □

Algorithm 6.1.3 embeds an expression tree in a fat-tree. The algorithm is the explicit dynamic programming algorithm of Section 1.3, but instead of tabulating the various cost functions for all positions in the position space it only tabulates them for the positions of the leaves of the current subtree.

Algorithm 6.1.3 (*Optimal Embedding in a Fat-Tree*)
Input: A position space $(\mathcal{P}, d)$ and an expression tree T with positions π for its leaves.
Output: A minimal-cost embedding π of T in $\mathcal{P}$.
Method:

(1) Traverse T in postorder. At each nonleaf node x, use equations (2) and (4) to tabulate $C_x(p)$ and $r^*_{xy}(p)$ explicitly for all positions p ranging over the positions of the leaves of T_x.
(2) Let $\pi(\text{ROOT})$ be some leaf position that minimizes $C_{\text{ROOT}}(p)$.
(3) Traverse T in preorder. For each internal node y with parent x, set $\pi(y) = r^*_{xy}(\pi(x))$.

THEOREM 6.1.4. *For an expression tree T with N leaves, Algorithm 6.1.3 solves the optimal-placement problem correctly on a fat-tree in $O(N^3)$ time.*

PROOF. Correctness follows from Theorem 6.1.2, which says that it is sufficient to consider only leaf positions of the current subtree.

For an internal node x whose subtree T_x has N_x leaves, the functions r^*_{xy} and C_x are tabulations of N_x values. Computing each value by Equations (2) and (4) takes $O(N_x)$ time. Thus computing the functions r^*_{xy} and C_x for one node x takes $O(N_x^2)$ time, and the running time for the complete algorithm is on the order of

$$\sum_{x \in \text{NODES}} |\text{LEAVES}(T_x)|^2. \tag{15}$$

This is a sum of N terms each of size at most N^2. □

If T is well balanced (for example, if T is a complete binary tree) then the sum in Equation (15) and the running time are $O(N^2)$.

7. WEIGHTED TREES

Up to now we have assumed that the cost of communication between positions p and q depended only on the distance $d(p, q)$ and not on the size of the data aggregate being communicated. This is valid in data-parallel expressions all of whose variables and intermediate results have the same size.

We now consider expressions that include data-parallel operations whose outputs can have different shapes or sizes than their inputs. For example, a fundamental building block of numerical linear algebra is the *rank-one update*

$$A := A + ab^T, \tag{16}$$

where A is an $m \times n$ matrix, a an $m \times 1$ column vector, and b an $n \times 1$ column vector. Figure 9 shows an example in a metric which is the direct sum of a two-dimensional grid and a discrete metric whose two positions represent "rowwise" and "columnwise." If communication cost depends only on distance, the cheapest solution is to form the product ab^T without moving a or b, and then to move that intermediate result to the position of A for the addition. However, a better solution is to move a and b^T to the position of A before forming ab^T; this performs two moves instead of one, but moves only $m + n$ elements instead of mn.

We model different-sized operands by labeling the edges of the expression tree with nonnegative real-valued *weights*. The weight of edge (x, y) is written w_{xy},

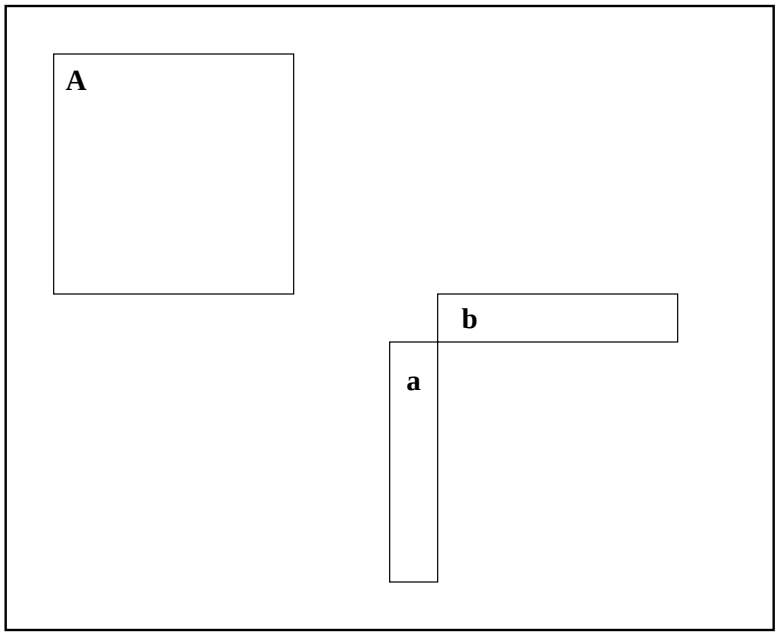

Fig. 9. Alignment alternatives for a rank-1 update operation. One choice is to compute the product ab^T in place, align the resulting matrix with A, and perform the addition. The other choice is to move a and b before multiplying, so that the product matrix ab^T is aligned with A. The first choice communicates mn elements; the second choice communicates only $m+n$ elements.

and represents the size of the object that is computed at node y. Now the total communication cost of evaluating an expression tree is

$$\sum_{(x,y)\in \mathrm{EDGES(T)}} w_{xy} \cdot d(\pi(x), \pi(y)),$$

and the master recurrences (2), (3), and (4) become

$$\begin{aligned} C_x(p) &= g_{xy}(p) + g_{xz}(p) \\ g_{xy}(p) &= \min_r (C_y(r) + w_{xy} \cdot d(p,r)) \\ g_{xz}(p) &= \min_r (C_z(r) + w_{xz} \cdot d(p,r)) \end{aligned} \tag{17}$$

for an internal node x with children y and z,

$$C_x(p) = w_{\mathrm{PARENT}(x),x} \cdot d(p, \pi(x)) \tag{18}$$

for a leaf node x, and

$$r^*_{xy}(p) = \text{some } r \text{ at which } C_y(r) + w_{xy} \cdot d(p,r) \text{ is minimized.} \tag{19}$$

Explicit dynamic programming (Algorithm 1.3.1) once again produces a minimum-cost embedding of a weighted tree in an arbitrary metric in $O(NP^2)$ time, just as in the unweighted case. In this section, we show how to use compact dynamic programming to remove the dependence on P for the grid, ring, discrete, and fat-tree metrics.

7.1 Grids and Hypercubes

Lemma 2.1.1 on decomposing direct sums of metrics still holds in the weighted case, so we need only consider one-dimensional grids. As in the unweighted case, the key idea is to compute PLC representations of the functions C_x and g_{xy}. However, the number of subintervals will turn out to be linear (it was constant in the unweighted case), and hence the running time of the weighted algorithm will be $O(N \log^2 N)$ (instead of $O(N)$ in the unweighted case).

Here are the details of the analysis. A PLC function is *convex* if the sequence $\mathcal{M} = (m_1, \ldots, m_k)$ of slopes of its subintervals is strictly increasing. We partition this sequence into three sections around the two values $-w_{xy}$ and w_{xy}:

$$m_1 < \cdots < m_{s-1} < -w_{xy} \le m_s < \cdots < m_{t-1} \le w_{xy} < m_t < \cdots < m_k . \qquad (20)$$

(Any of the three sections may be empty, in which case $s = 1$, $s = t$, or $t = k + 1$.)

THEOREM 7.1.1. *Let T be a weighted expression tree, and let d be the grid metric in one dimension. Then for every node x, the function C_x is PLC and convex with at most $2N_x$ subintervals, where the subtree T_x rooted at x has N_x leaves.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

A straightforward implementation of the dynamic programming algorithm, using PLC representations of all the functions with $O(N)$ intervals, would use $O(N)$ time at each node and $O(N^2)$ time in all. We now show how to reduce this to $O(N \log^2 N)$. The idea is to compute $g_{xy} + g_{xz}$ not by merging the two lists of breakpoints, but by using binary search to insert breakpoints from the smaller list into the larger one. Algorithm 7.1.2 gives details.

Algorithm 7.1.2 (*Weighted Optimal Embedding in a Grid*)

Input: A position space $(\mathcal{P}, d)$ and a weighted expression tree T with an embedding π for its leaves.

Output: A minimum-cost embedding π of T in $\mathcal{P}$.

Method: This algorithm represents a PLC function as a search tree of subintervals; splay trees [Sleator and Tarjan 1983] are a suitable choice of search tree. The breakpoint positions are stored with the subintervals in the search tree. Each internal node of the search tree stores a "partial slope." The slope of a subinterval is represented as the sum of the partial slopes along the search path leading to the subinterval. This allows the algorithm to modify the slopes of an entire range of subintervals at once by changing one partial slope.

(1) Form the search trees representing C_y for all $y \in \text{LEAVES}(T)$.

(2) Traverse T in postorder. If x is a leaf, do nothing. If x is an internal node with children y and z:
   (a) Call Convolve(C_y, w_{xy}) to compute g_{xy};
   (b) Call Convolve(C_z, w_{xz}) to compute g_{xz};
   (c) Call Sum(g_{xy}, g_{xz}) to compute $C_x = g_{xy} + g_{xz}$.

(3) Let $\pi(\text{ROOT})$ be some integer r at which $C_{\text{ROOT}}(r)$ is minimized.

(4) Traverse T in preorder. For each internal node y with parent x, set $\pi(y) = r^*_{xy}(\pi(x))$.

SUBROUTINE: (CONVOLVE)

Input: A PLC function C_y and a slope w_{xy}.

Output: A PLC representation of $g_{xy}(p) = \min_r(C_x(r) + w_{xy} \cdot d(p,r))$.
Method:

(1) Use binary search with $-w_{xy}$ and w_{xy} in the monotone list of slopes for C_y to find the two positions p_s and p_t defined in Equation (20).
(2) Convert the PLC representation of C_y into one for g_{xy} by replacing all subintervals to the left of p_s (if any) with a single subinterval of slope $-w_{xy}$, and replacing all subintervals to the right of p_t (if any) with a single subinterval of slope w_{xy}.

SUBROUTINE: (SUM)
Input: Two PLC functions g_{xy} and g_{xz}.
Output: A PLC representation of $C_x = g_{xy} + g_{xz}$.
Method: Suppose g_{xy} has at least as many breakpoints as g_{xz}; if not, reverse their roles in the following. Convert the PLC representation of g_{xy} into one for C_x by traversing the breakpoints for g_{xz}, inserting each one into the list of breakpoints for g_{xy}, and updating the slopes.

It remains to analyze the running time of this algorithm. Each search tree operation on a PLC function takes time proportional to the logarithm of the number of subintervals of the function (in an amortized sense).

LEMMA 7.1.3. *Convolve(C_y,w_{xy}) computes g_{xy} in $O(\log N_y)$ amortized time.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

LEMMA 7.1.4. *Sum(g_{xy},g_{xz}) computes C_x in $O(N_z \log N_y)$ amortized time provided $N_y \geq N_z$.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

LEMMA 7.1.5. *The solution to the recurrence*

$$f(m+n) = n \log m + f(m) + f(n), \quad m > n$$

is $f(n) = O(n \log^2 n)$.

PROOF. See the technical report [Chatterjee et al. 1992]. □

THEOREM 7.1.6. *For a weighted expression tree T with N leaves, Algorithm 7.1.2 solves the optimal-placement problem for the grid metric in $O(N \log^2 N)$ time.*

PROOF. The running time at node x is dominated by the $O(N_z \log N_y)$ time to compute $C_x = g_{xy} + g_{xz}$. Thus the recurrence in Lemma 7.1.5 describes the total running time. □

7.2 The Ring Metric

Lemma 2.1.1 on decomposing direct sums of metrics still holds in the weighted case, so we need only consider one-dimensional rings.

In the unweighted case, we showed that the cost function at a node x could be represented as a PLC function with $O(N_x)$ subintervals, and thus derived an $O(Nh)$ optimal embedding algorithm. This is no longer true for the weighted case. We can still get a compact dynamic programming algorithm whose running time does not depend on P, however, by using the fact that every internal node can be embedded at the position of some leaf of T. The algorithm (Algorithm 7.2.2) is much simpler

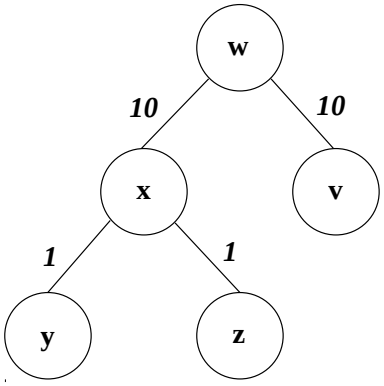

Fig. 10. A weighted tree in which the internal node x cannot be embedded at the position of any leaf of its subtree.

than the unweighted ring algorithm; it just traverses the nodes of T, tabulating the subtree cost functions for every leaf position at every node. This takes $O(N^3)$ time in the worst case, as compared to the $O(N^2)$ unweighted ring algorithm.

The following theorem is a minor variation of Theorem 2.2.1.

THEOREM 7.2.1. *Let T be a weighted expression tree, and let d be the ring metric. Then any minimum-cost embedding of T places each node at the position of a leaf of T.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

Algorithm 7.2.2 (*Weighted Optimal Embedding in a Ring, the Discrete Metric, or a Fat-Tree*)

Input: A position space $(\mathcal{P}, d)$ and a weighted expression tree T with an embedding π for its leaves.

Output: A minimum-cost embedding π of T in $\mathcal{P}$.

Method:

(1) Traverse T in postorder. At each nonleaf node x, use Eqs. (17) and (19) to tabulate $C_x(p)$ and $r^*_{xy}(p)$ explicitly for all positions p ranging over the positions of the leaves of T.
(2) Let $\pi(\text{ROOT})$ be some leaf position that minimizes $C_{\text{ROOT}}(p)$.
(3) Traverse T in preorder. For each internal node y with parent x, set $\pi(y) = r^*_{xy}(\pi(x))$.

THEOREM 7.2.3. *For a weighted expression tree T with N leaves, Algorithm 7.2.2 solves the optimal-placement problem for the ring metric in $O(N^3)$ time.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

7.3 The Discrete Metric

In the unweighted case, the robustness of the discrete metric implies the existence of an optimal embedding of a tree T that is also an optimal embedding of every subtree T_x. Therefore in Section 5 we could omit the computation of the subtree cost functions C_x, and only compute the minimum-cost sets $\text{OPT}(x)$ for each node.

All the elements of OPT(x) were leaves of T_x, and the computation was a simple matter of unions and intersections of OPT sets.

None of this works in the weighted case. Indeed, there may be no optimal embedding of T that places node x at the position of any of the leaves of the subtree T_x. For example, in the tree of Figure 10, the only minimum-cost embedding is to place both internal nodes x and w at $\pi(v)$.

We can still get an efficient compact dynamic programming algorithm, however. First we show that every internal node can be embedded at the position of some leaf of T, so that we need only tabulate $C_x(p)$ for the N leaf positions. Then we observe that $C_x(p)$ takes on only one value for all p that are not positions of leaves of T_x, which simplifies the tabulation further. Finally, we show that each entry can be computed in constant time. The resulting algorithm runs in $O(Nh)$ time. This is the same asymptotic complexity as the unweighted algorithm, though the weighted algorithm is slightly more complicated. The details follow.

THEOREM 7.3.1. *Let T be a weighted expression tree, and let d be the discrete metric. Then any minimum-cost embedding of T places every node at the position of a leaf of T.*

PROOF. See the Appendix. □

LEMMA 7.3.2. *Let T be a weighted expression tree; let d be the discrete metric; let π be a mapping of the leaves of T; and let C_x be as defined in Eqns. (17) through (19). For each node x, the subtree cost function $C_x(p)$ takes only one value for all $p \notin \pi(\text{LEAVES}(T_x))$.*

PROOF. Let p and q be positions, neither of which is in $\pi(\text{LEAVES}(T_x))$. Extend π to an embedding of the entire subtree T_x that embeds node x at position p and that has cost $C_x(p)$. Now modify this extended embedding by moving every node of T_x that is embedded at p, including node x, to be embedded at q. This modification does not move any leaves of T_x. It does not increase the cost of any edge of T_x. We conclude that $C_x(q) \leq C_x(p)$. Reversing the roles of p and q in the argument gives $C_x(p) \leq C_x(q)$. Therefore $C_x(q) = C_x(p)$. □

LEMMA 7.3.3. *Let T be a weighted expression tree; let d be the discrete metric; let π be a mapping of the leaves of T; and let C_x be as defined in Eqns. (17) through (19). For each node y with parent x, the contribution function $g_{xy}(p)$ is equal either to $C_y(p)$ or to $\min_r(C_y(r)) + w_{xy}$.*

PROOF. Equation (17) says that $g_{xy}(p) = \min_r(C_y(r) + w_{xy} \cdot d(p, r))$. If the minimum occurs at $r = p$, this is equal to $C_y(p)$. Otherwise, since the metric is discrete, this is equal to $C_y(r) + w_{xy}$ for some r; and the same r minimizes $C_y(r)$ and $C_y(r) + w_{xy}$. □

The compact dynamic programming algorithm for the weighted discrete case tabulates the subtree cost function C_x for each node x, just like the weighted ring algorithm. However, the lemmas above imply that only $O(|\text{LEAVES}(T_x)|)$ information needs to be tabulated at each node x, and that each entry can be computed in constant time.

THEOREM 7.3.4. *For a weighted expression tree T with N leaves and height h,*

in the discrete metric, Algorithm 7.2.2 solves the optimal-placement problem and can be implemented to run in $O(Nh)$ time.

PROOF. See the technical report [Chatterjee et al. 1992]. □

7.4 Fat-Trees

In the unweighted case, we showed that some optimal embedding of T places every node at the position of one of the leaves of its subtree. The example in Figure 10, if embedded in a fat-tree with 4 positions, shows that this is not true in the weighted case. As with the weighted discrete metric, however, there is always an optimal embedding with every node placed at the position of *some* leaf of T.

THEOREM 7.4.1. *Let T be a weighted expression tree, and let d be the fat-tree metric. Then some minimum-cost embedding of T places every node at the position of a leaf of T.*

PROOF. See the Appendix. □

The compact dynamic programming algorithm for this case is identical to the weighted ring algorithm (except for the definition of $d(p, q)$). It tabulates the subtree cost functions for every leaf position at every node. Its $O(N^3)$ running time is the same as the unweighted fat-tree algorithm (Algorithm 6.1.3) in the worst case. However, this algorithm is never faster than $O(N^3)$, whereas the unweighted fat-tree algorithm runs in $O(N^2)$ time on well-balanced expression trees and in $O(N^2 h)$ time on trees of height h because it can always embed nodes at leaf positions chosen from their own subtrees.

THEOREM 7.4.2. *For a weighted expression tree T with N leaves, Algorithm 7.2.2 solves the optimal-placement problem for the fat-tree metric in $O(N^3)$ time.*

PROOF. See the technical report [Chatterjee et al. 1992]. □

8. CONCLUSIONS AND OPEN PROBLEMS

We have presented a theoretical model of the cost of evaluating Fortran 90 array expressions on massively parallel distributed-memory machines. Although the model simplifies reality somewhat, we believe that it is faithful enough to be useful in practice. We have developed algorithms to solve the data alignment problem optimally in this model, for a wide range of model architectures, including multidimensional grids and rings, hypercubes, the discrete metric, and fat-trees.

Our algorithms are based on dynamic programming. Expressions containing operands of different sizes are aligned optimally using algorithms that extend the dynamic programming technique to edge-weighted trees. The running times of these algorithms range from linear to cubic in the size of the expression, and are independent of the size of the position space. In a companion paper [Chatterjee et al. 1993b], we present a unified optimization framework incorporating the ideas of this article that extends the approach heuristically to basic blocks that are represented as directed acyclic graphs.

We conclude with a list of open problems in this area of research and their status.

—It might be possible to improve the asymptotic running times of some of the slower algorithms in this article; we feel that they are already sufficiently fast

to be practical. The question of lower bounds of CDP algorithms for specific metrics remains open.

—CDP algorithms are nonuniform in that they must be derived individually for each distance function. It remains to be seen whether one can develop a uniform theory. The characterization of metrics that admit optimal embeddings with all nodes at leaf positions (Section 2.2) may be a step in this direction.

—A complete analysis of the "nearly fat-tree" metric (Section 6) remains open.

—Analysis of a metric modeling startup costs in communication remains wide open. As noted in Section 1, the problematic case is when the distance term and the startup term are comparable. For the case where the distance term is the grid metric, we have a solution involving mixed integer linear programming (MILP), in which the startup term is represented using 0-1 discrete variables. However, this solution is not particularly desirable, since MILP can take exponential time in the worst case. Developing better algorithms or proving intrinsic lower bounds even for this restricted version of the problem remains open.

—Some progress has been made in determining array replication and collapsing (aligning some array axes with memory), but a comprehensive treatment remains open. We have developed solutions to the replication problem [Chatterjee et al. 1993a] using network flow algorithms, while Anderson and Lam [1993] have worked on array collapsing using a linear algebra framework. Knobe and Dally [1994] have studied other aspects of this problem.

—We have developed extensions of the analysis to handle control flow [Chatterjee et al. 1994a], but the problem is by no means completely solved. Our extensions are based on a static single-assignment framework. Knobe et al. [1990], Gupta [1992], and Anderson and Lam [1993] use other approaches.

—The alignment phase that we study here is followed by a distribution phase that maps the position space to the processors. Distributions have been determined using heuristic search [Gupta 1992; Wholey 1991], integer programming [Bixby et al. 1993], and divide-and-conquer methods [Chatterjee et al. 1994b]. Many open questions remain.

APPENDIX

PROOFS OF SELECTED THEOREMS

THEOREM 2.2.1. *Let T be an expression tree, d be either the one-dimensional continuous grid metric on $[0, P-1]$ or the one-dimensional continuous ring metric on $[0, P)$, and π be an embedding of the leaves of T. Then there is a minimum-cost embedding of the internal nodes of T that agrees with π on the leaves, and that embeds every internal node at the position of some leaf.*

PROOF. Extend π to an arbitrary embedding of all the nodes of T. We will show how to modify this extended π so that every internal node is at a leaf position, without moving any leaves or increasing the cost of the embedding.

We ignore the parent-child relationship and consider T to be an undirected tree; two nodes are neighbors if one is the other's parent in the directed tree. A node is

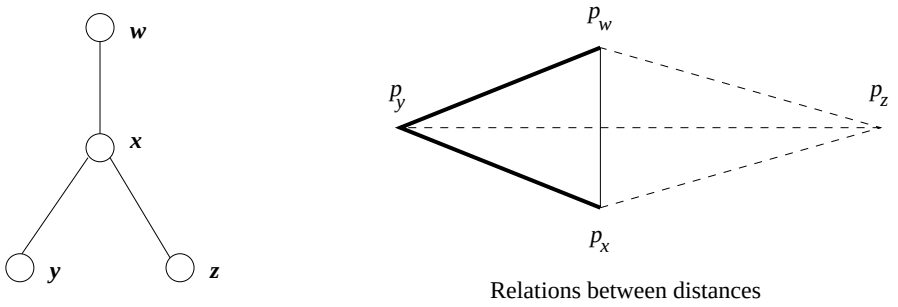

Fig. 11. Relations between distances in the fat-tree metric used in the proof of Theorem 6.1.2.

a leaf if it has one neighbor, and an internal node if it has more than one neighbor. For reasons that will shortly become clear, we will prove the result for undirected trees whose internal nodes are not restricted to three neighbors.

We proceed by induction on the number of internal nodes of T. The base case is a tree with one leaf, in which case there is nothing to prove. For the induction step, suppose T has some internal node x that is not embedded at a leaf position. There are two cases.

Case 1. Suppose x has some neighbor y with $\pi(x) = \pi(y)$. Since $\pi(x)$ is not a leaf position, y is not a leaf. Form tree T' by merging nodes x and y into a single node x', which inherits all the other neighbors of both x and y. This tree has the same leaves as T, and one fewer internal node. The embedding of T' that agrees with π everywhere, with node x' placed at $\pi(x) = \pi(y)$, costs the same as the embedding π of T. By the induction hypothesis, there is an embedding π' that agrees with π on the leaves, places all internal nodes of T' at leaf positions, and costs no more than π. We lift π' back to an embedding of T by placing both x and y at position $\pi'(x')$. The result is an embedding that agrees with π on the leaves, places all internal nodes of T at leaf positions, and costs no more than π.

Case 2. Suppose x has no neighbor embedded at $\pi(x)$. Consider moving x from $\pi(x)$ to the left until it first encounters a neighbor y_ℓ, and to the right until it first encounters a neighbor y_r. At most one of those moves increases the cost of the embedding, depending on whether x has more neighbors embedded to the left or to the right of $\pi(x)$. Move x to whichever of $\pi(y_\ell)$ and $\pi(y_r)$ does not increase the cost of the embedding. Now x is embedded at the same position as a neighbor; use the construction in Case 1 to finish the induction step. □

THEOREM 6.1.2. *There is a minimum-cost embedding of the expression tree T for the fat-tree metric in which each internal node x is placed at one of the leaf positions of T_x.*

PROOF. It suffices to prove that every internal node of T is placed at the same position as one of its children. Let π be a minimum-cost embedding of T. For brevity, we write $p_x = \pi(x)$ and $d_{xy} = d(\pi(x), \pi(y))$. Suppose there is a node x with children y and z, such that $p_x \neq p_y$ and $p_x \neq p_z$. We will show that we can

move x either to p_y or to p_z without increasing the total cost of the embedding. Therefore we can traverse the tree in postorder, moving each node to the same position as one of its children, never increasing the cost of the embedding, and eventually achieving an embedding that satisfies the statement of the lemma.

If x is the root of T, we move x to position p_y. This replaces two edges of cost d_{xy} and d_{xz} by two edges of cost 0 and d_{yz}; the triangle inequality says that the total cost does not increase. (We could as well have moved x to p_z.)

If x is not the root, let w be the parent of x in T, as shown in Figure 11. Moving x either to p_y or to p_z does not increase the sum of the costs of edges (x, y) and (x, z), by the argument in the last paragraph. If one of these moves does not increase the cost of edge (w, x), we make it, and we are done.

The only remaining possibility is that both of those moves would increase the cost of edge (w, x), that is, that $d_{wx} < d_{wy}$ and also $d_{wx} < d_{wz}$. Suppose this is so. Suppose in addition that $d_{xy} \leq d_{xz}$, as in Figure 11 (the opposite case is symmetric). We move x to p_y. The three edges (w, x), (x, y), and (x, z) change from total cost $d_{wx} + d_{xy} + d_{xz}$ to total cost $d_{wy} + 0 + d_{yz}$.

Now we apply the isosceles triangle inequality twice. First, $d_{wx} < d_{wy}$ implies $d_{xy} = d_{wy}$. Second, $d_{xy} \leq d_{xz}$ implies $d_{xz} \geq d_{yz}$. Therefore the total cost of the three edges (w, x), (x, y), and (x, z) does not increase when we move x to p_y.

The opposite case $d_{xy} \geq d_{xz}$ is symmetric; we move x to p_z. □

THEOREM 7.3.1. *Let T be a weighted expression tree, and let d be the discrete metric. Then any minimum-cost embedding of T places every node at the position of a leaf of T.*

PROOF. Suppose that embedding π does not place every node at a leaf position. Then there is an internal node x such that π does not place x at a leaf position, but π does place both children y and z of x at leaf positions. Let π' be the embedding

$$\pi'(v) = \begin{cases} \pi(y), & \text{if } \pi(v) = \pi(x) \\ \pi(v), & \text{otherwise.} \end{cases}$$

Notice that $\pi'(v)$ is the same as $\pi(v)$ if v is a leaf. Changing π to π' does not increase the cost of any edge of T, and it decreases the cost of edge (x, y) from w_{xy} to 0. Therefore π was not minimum-cost. □

THEOREM 7.4.1. *Let T be a weighted expression tree, and let d be the fat-tree metric. Then some minimum-cost embedding of T places every node at the position of a leaf of T.*

PROOF. Let π be an embedding of T. For brevity, we write $p_x = \pi(x)$ and $d_{xy} = d(\pi(x), \pi(y))$.

Suppose that embedding π does not place the internal node x at a leaf position. The position p_x is a leaf of the fat-tree (not to be confused with a leaf of the expression tree). Let α be the minimum-height fat-tree vertex that is an ancestor in the fat-tree both of p_x and of p_y for some leaf y of T. In other words, walk up the fat-tree from p_x, and stop at the first vertex α that is an ancestor of some leaf position p_y. Then in fact $\alpha = \text{LCA}(p_x, p_y)$. Figure 12 is an illustration.

Suppose the two children of α in the fat-tree are β and γ, where β is an ancestor

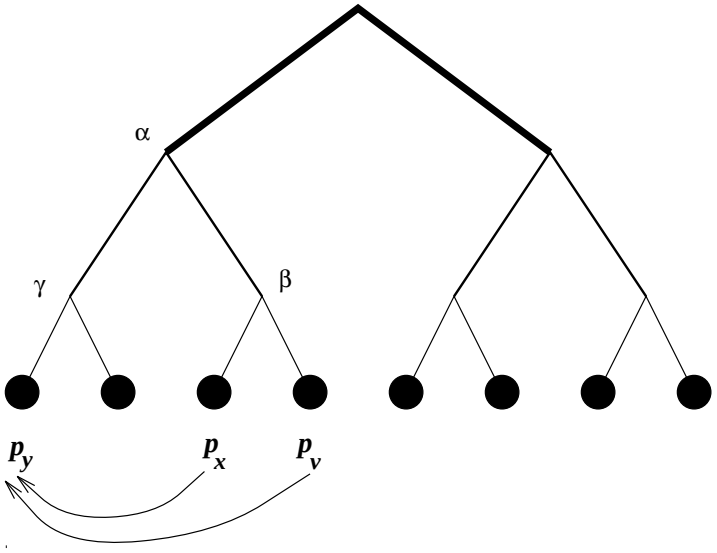

Fig. 12. Moving internal tree nodes x and v to leaf position p_y.

of p_x, and γ is an ancestor of p_y. Now let π' be such that

$$\pi'(v) = \begin{cases} p_y, & \text{if } p_v \text{ is a fat-tree descendant of } \beta \\ p_v, & \text{otherwise.} \end{cases}$$

Notice that $\pi'(v)$ is the same as $\pi(v)$ if v is a leaf of the expression tree, since by choice of α, no expression tree leaf position is a descendant of β. We claim that changing π to π' does not increase the cost of any edge of T. Consider an arbitrary edge (a, b) of T. If neither p_a nor p_b is a fat-tree descendant of β, then the cost of (a, b) does not change. If both p_a and p_b are fat-tree descendants of β, then $\pi'(a) = \pi'(b) = p_y$, and the cost of (a, b) becomes zero. Finally, suppose that p_a is a fat-tree descendant of β and p_b is not. Then the fat-tree path from p_a to p_b goes through α, and therefore $d_{ab} \geq 2 \cdot \text{HEIGHT}(\alpha) = d_{ay}$. The isosceles triangle inequality for p_a, p_b, and p_y then implies that $d_{ab} \geq d_{by}$. But d_{ab} is the cost of edge (a, b) under π, and d_{by} is the cost of edge (a, b) under π'; so the cost of (a, b) does not increase in changing π to π'.

Therefore changing π to π' preserves the leaf positions, does not increase the total cost, and decreases the number of internal nodes at nonleaf positions. We can repeat this transformation until every internal node is at a leaf position. □

REFERENCES

Aho, A. V., Sethi, R., and Ullman, J. D. 1986. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Mass.

American National Standards Institute 1991. Fortran 90: X3J3 internal document S8.118 Submitted as Text for ANSI X3.198-1991, and ISO/IEC JTC1/SC22/WG5 internal document N692 Submitted as Text for ISO/IEC 1539:1991. American National Standards Institute, New York.

Anderson, J. M. and Lam, M. S. 1993. Global optimizations for parallelism and locality on scalable parallel machines. In *Proceedings of the ACM SIGPLAN'93 Conference on*

Programming Language Design and Implementation. ACM, New York, 112–125.

Bixby, R., Kennedy, K., and Kremer, U. 1993. Automatic data layout using 0-1 integer programming. Tech. Rep. CRPC-TR93349-S (Nov.), Center for Research on Parallel Computation, Rice University, Houston, Tex.

Blelloch, G. E. 1992. Nesl: A nested data-parallel language. Tech. Rep. CMU-CS-92-103 (Jan.), School of Computer Science, Carnegie Mellon University, Pittsburgh, Pa.

Blelloch, G. E., Hardwick, J. C., Sipelstein, J., Zagha, M., and Chatterjee, S. 1994. Implementation of a portable nested data-parallel language. *J. Parallel Distrib. Comput. 21,* 1 (Apr.), 4–14.

Chapman, B., Mehrotra, P., and Zima, H. 1992. Programming in Vienna Fortran. *Sci. Program. 1,* 1 (Fall), 31–50.

Chatterjee, S. 1993. Compiling nested data-parallel programs for shared-memory multiprocessors. *ACM Trans. Program. Lang. Syst. 15,* 3 (July), 400–462.

Chatterjee, S., Gilbert, J. R., and Schreiber, R. 1994a. The alignment-distribution graph. In *Languages and Compilers for Parallel Computing*, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, Eds. Lecture Notes in Computer Science, vol. 768. Springer-Verlag, New York, 234–252.

Chatterjee, S., Gilbert, J. R., and Schreiber, R. 1993a. Mobile and replicated alignment of arrays in data-parallel programs. In *Proceedings of Supercomputing'93.* ACM, New York, 420–429. Also available as RIACS Technical Report 93.08 and Xerox PARC Technical Report CSL-93-7.

Chatterjee, S., Gilbert, J. R., Schreiber, R., and Sheffler, T. J. 1994b. Array distribution in data-parallel programs. In *Proceedings of the 7th Annual Workshop on Languages and Compilers for Parallelism*, pp. 6.1–6.17. Also available as RIACS Technical Report 94.09.

Chatterjee, S., Gilbert, J. R., Schreiber, R., and Teng, S.-H. 1993b. Automatic array alignment in data-parallel programs. In *Proceedings of the 20th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages.* ACM, New York, 16–28. Also available as RIACS Technical Report 92.18 and Xerox PARC Technical Report CSL-92-13.

Chatterjee, S., Gilbert, J. R., Schreiber, R., and Teng, S.-H. 1992. Optimal evaluation of array expressions on massively parallel machines. Tech. Rep. TR 92.17 (Sept.), Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, Calif. Also available as Xerox PARC Technical Report CSL-92-11. Available by anonymous ftp as RIACS-TR-92.17.PS from riacs.edu in the directory pub/Chatterjee.

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., and Zadeck, F. K. 1991. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst. 13,* 4 (Oct.), 451–490.

Dally, W. J. and Seitz, C. L. 1986. The torus routing chip. *Distrib. Comput. 1*, 187–196.

Farris, J. S. 1970. Methods for computing Wagner trees. *Syst. Zoolog. 19*, 83–92.

Fitch, W. M. 1971. Toward defining the course of evolution: Minimum change for a specific tree topology. *Syst. Zoolog. 20*, 406–416.

Fox, G. C., Hiranandani, S., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C.-W., and Wu, M.-Y. 1990. Fortran D language specification. Tech. Rep. Rice COMP TR90-141 (Dec.), Dept. of Computer Science, Rice University, Houston, Tex.

Gerndt, M. 1989. Automatic parallelization for distributed-memory multiprocessing systems. Ph. D. thesis, Univ. of Bonn, Bonn, Germany.

Gilbert, J. R. and Schreiber, R. 1991. Optimal expression evaluation for data parallel architectures. *J. Parallel Distrib. Comput. 13,* 1 (Sept.), 58–64.

Gupta, M. 1992. Automatic data partitioning on distributed memory multicomputers. Ph. D. thesis, Univ. of Illinois at Urbana-Champaign, Urbana, Ill. Available as Technical Reports UILU-ENG-92-2237 and CRHC-92-19.

Hartigan, J. A. 1973. Minimum mutation fits to a given tree. *Biometrics 29*, 53–65.

High Performance Fortran Forum 1993. High Performance Fortran language specification. *Sci. Program. 2,* 1–2, 1–170.

Hiranandani, S., Kennedy, K., and Tseng, C.-W. 1991. Compiler support for machine-independent parallel programming in Fortran D. Tech. Rep. Rice COMP TR90-149 (Feb.), Dept. of Computer Science, Rice University, Houston, Tex.

Hwang, F. K. 1986. A linear time algorithm for full Steiner trees. *Oper. Res. Lett. 4*, 235–237.

Knobe, K. and Dally, W. J. 1994. Subspace optimizations. In *Automatic Parallelization — New Approaches to Code Generation, Data Distribution, and Performance Prediction*, C. W. Kessler, Ed. Vieweg Advanced Studies in Computer Science, Verlag Vieweg, Wiesbaden, Germany, 153–176.

Knobe, K., Lukas, J. D., and Dally, W. J. 1992. Dynamic alignment on distributed memory systems. In *Proceedings of the 3rd Workshop on Compilers for Parallel Computers.* Austrian Center for Parallel Computation, 394–404.

Knobe, K., Lukas, J. D., and Steele, G. L., Jr. 1990. Data optimization: Allocation of arrays to reduce communication on SIMD machines. *J. Parallel Distrib. Comput. 8,* 2 (Feb.), 102–118.

Kremer, U., Mellor-Crummey, J., Kennedy, K., and Carle, A. 1993. Automatic data layout for distributed-memory machines in the D programming environment. Tech. Rep. CRPC-TR93-298-S (Feb.), Center for Research on Parallel Computation, Rice University, Houston, Tex. Appears in *Proceedings of the 1st International Workshop on Automatic Distributed Memory Parallelization, Automatic Data Distribution and Automatic Parallel Performance Prediction (AP'93)*, Vieweg Verlag, Wiesbaden, Germany.

Leiserson, C. E. 1985. Fat-Trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput. C-34,* 10 (Oct.), 892–901.

Leiserson, C. E., Abuhamdeh, Z. S., Douglas, D. C., Feynman, C. R., Ganmukhi, M. N., Hill, J. V., Hillis, W. D., Kuszmaul, B. C., St. Pierre, M. A., Wells, D. S., Wong, M. C., Yang, S.-W., and Zak, R. 1992. The network architecture of the Connection Machine CM-5. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures.* ACM, New York, 272–285.

Li, J. and Chen, M. 1991. The data alignment phase in compiling programs for distributed-memory machines. *J. Parallel Distrib. Comput. 13,* 2 (Oct.), 213–221.

Mace, M. E. 1987. *Memory Storage Patterns in Parallel Processing.* Kluwer International Series in Engineering and Computer Science. Kluwer Academic Press, Norwell, Mass.

Rose, J. R. and Steele, G. L., Jr. 1987. C*: An extended C language for data parallel programming. In *Proceedings of the 2nd International Conference on Supercomputing.* ACM, New York, 2–16.

Sankoff, D. and Rousseau, P. 1975. Locating the vertices of a Steiner tree in an arbitrary metric space. *Math. Program. 9*, 240–246.

Sleator, D. D. and Tarjan, R. E. 1983. A data structure for dynamic trees. *J. Comput. Syst. Sci. 26*, 362–391.

Tseng, C.-W. 1993. An optimizing Fortran D compiler for MIMD distributed-memory machines. Ph. D. thesis, Dept. of Computer Science, Rice University, Houston, Tex. Available as Technical Report CRPC-TR93291.

Wholey, S. 1991. Automatic data mapping for distributed-memory parallel computers. Ph. D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pa. Available as Technical Report CMU-CS-91-121.

Winter, P. 1987. Steiner problem in networks: A survey. *Networks 17*, 129–167.

Received November 1992; revised August 1993 and June 1994; accepted August 1994