In *Proc. 3rd Symp. Frontiers Massively Parallel Computation* (1990), pp. 75–78.

Practical Hypercube Algorithms for Computational Geometry

Preliminary Version

Philip D. MacKenzie[1] and Quentin F. Stout[2]

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI

Abstract

Many problems in computational geometry can be solved on the hypercube using a simple and practical technique, which we call *cross-stitching*. Given n inputs distributed one per processor on a hypercube with n processors, the cross-stitching paradigm runs in $\Theta(\log^2 n)$ time with very low constants. We illustrate this form of 2-dimensional divide-and-conquer, consider some of its many applications, and show its practicality by computing exact communication constants for our algorithms.

1 Introduction

The field of parallel computational geometry has seen rapid growth in recent years, especially on regular architectures, such as the mesh and the hypercube, and on Parallel Random Access Machines (PRAMs). Most of this has been theoretical work, in which asymptotic running times have been lowered at the expense of enormous constant factors and extremely complex algorithms. Even if someone were able to implement these algorithms, the constant factors would make them impractical except in the case of a staggeringly huge input set, much larger than the size of any parallel machine in the foreseeable future. In this paper, we focus on simple, practical algorithms, which can be implemented on parallel hypercube computers which are available today, such as the Connection Machine.

These algorithms are all able to a technique which we call *cross-stitching*. This is a form of 2-dimensional divide-and-conquer which runs in the following manner. We take n objects (points or line segments) as input and sort them by x coordinate. The sorted set is divided in half, and the problem is recursively solved on both halves. Then the solutions to the halves are stitched together to solve the whole problem. One can think of this as stitching the solution together along the vertical dividing line between the halves. By keeping the recursive solutions sorted by y coordinate, solutions can be stitched together by using prefix and merge operations, instead of slower sort operations. We call this cross-stitching because problems are split horizontally, but solutions are stitched together vertically.

The cross-stitching technique has been used in limited cases previously, but we wish to exhibit its generality and practicality through a multitude of examples and exact communication time analyses. Cross-stitching is well matched to the communication capabilities of the hypercube, and provides a systematic solution to many 2-dimensional geometric problems.

[1]Research supported by an AT&T Fellowship.
[2]Research partially supported by a joint NSF-DARPA grant.

Given two points p and q in d-dimensional space, p is said to *dominate* q if each coordinate of p is larger than the corresponding coordinate of q. We will solve the following computational geometry problems using the cross-stitching technique.

2-Dimensional Dominance Counting Given a set S of n points in the plane, find for each point $p \in S$ the number of points in S dominated by p.

2-Set Dominance Counting Given a set S of m points and a set T of k points, where $n = m + k$, find for each point $p \in S$ the number of points in T dominated by p. Similarly, find for each point $q \in T$ the number of points in S dominated by q.

3-Dimensional Maxima Given a set S of n points in 3-dimensional space, determine all points $p \in S$ such that no other point of S dominates p.

Closest Pair Given a set S of n points in the plane, determine a pair of points in S which are closest to each other in the Euclidean metric.

All Points Nearest Neighbors Given a set S of n points in the plane, for each point $p \in S$ find a point in $S - \{p\}$ that is closest to p in the Euclidean metric.

Iso-oriented Segment Intersection Given a set S of n iso-oriented segments, determine for each segment in S whether or not it is intersected by another segment in S.

Rectangle Area Given a set S of n iso-oriented planar rectangles, find the area covered by the union of these rectangles.

Rectangle Intersection Given a set S of n iso-oriented planar rectangles, determine for each rectangle in S whether or not it is intersected by another rectangle in S.

The following problem can be solved with the same general method, except that solutions are not stitched using a vertical merge.

Visibility from a Point Given n non-intersecting line segments and a point p, determine that part of the plane visible from p if all segments are opaque. An interesting special case of this problem is to determine the **Skyline** of a set of n rectangles resting on a horizontal line. In this case, the line segments are the upper edges of the rectangles and the point p is $(0, \infty)$.

Stojmenovic [6] gives algorithms which use the cross-stitching technique for Convex Hull, Closest Pair, 2-Set Dominance Counting, and 3-Dimensional Maxima. Boxer and Miller [2] give an algorithm similar to cross-stitching which can be used to solve Visibility from a Point with intersecting line segments in $\Theta(\log^2 n)$ time using $n\alpha(n)$ processors, where $\alpha(n)$ is the inverse Ackermann function.

Throughout we assume that we are using an n processor hypercube with the following characteristics. Each processor has a unique $\log_2(n)$-bit ID (a number from 0 to $n-1$), and two processors are neighbors (share a communication link) if their IDs differ in exactly one bit. We will consider a communication time step as a processor sending a short message to its neighbor, where short will mean less than 12 words. We will assume that this is the dominant cost in our system and thus calculate the constants on communication time. If we considered time steps, instead of communication time steps, then the computed constants would be slightly larger.

We also assume each processor has a constant amount of memory, except in the algorithm for Visibility from a Point with intersecting line segments, in which we assume each processor has a memory of size $\Omega(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann's function, a very slow growing function of n. This is necessary because the output in this case can be of size $\Theta(n\alpha(n))$.

2 Cross-Stitching

The cross-stitching technique is outlined in Figure 1. It is assumed that initially each processor has at most one data item. Because the stitching is a combination of prefix and merge operations, it takes $\Theta(\log n)$ time on a hypercube and therefore the whole algorithm takes $\Theta(\log^2 n)$ time.

In a more careful analysis, if we assume that the constant factor in the stitch is k, then we see that at level i, the stitching operation takes $k(\log n - i)$ time. Thus the total communication time is

$$\sum_{i=0}^{\log n - 1} k(\log n - i) = (k/2)\log^2 n + (k/2)\log n \; .$$

3 Useful Hypercube Algorithms

The following are some useful algorithms and their communication times. See [5] for detailed implementations.

Prefix Assume each processor i contains the value A_i. Then the *prefix* operation (*parallel prefix, scan*) results in processor i containing value $S_i = A_0 \oplus A_1 \oplus \ldots \oplus A_i$, where $\oplus$ is any associative or Π-quasi-valid (defined later) operator. This can be done in $2\log n$ communication time. Using a prefix operation one can count the number of marked items stored at processors of lower rank, find the maximum of the values stored at processors of lower rank, or perform a segmented prefix (a prefix within disjoint consecutive groups of processors). A prefix operation is called by $S_i = \mathrm{Prefix}(\oplus, A_i)$.

Compress Given m out of n marked records stored one per processor on an n processor hypercube, compress sends these records to the first m processors in the hypercube, keeping their relative order, in $3\log n$ communication time, where m need not be known in advance. This involves a prefix to count the number of marked records in front of each record, followed by a monotonic route. A compress is called by $\mathrm{Compress}(B, R)$, where B is a boolean value which is true if the record R is marked.

Increment Route Given a record at each processor in a hypercube, for all i such that $0 \leq i < n-1$, incremental route sends a copy of the record at processor i to processor $i+1$. This can be performed in $\log n$ time, but can also be performed using the prefix operation $\mathrm{Prefix}(\max_p, R)$, where $\max_p$ is given two records which came from different processors and returns the record from the larger numbered processor. Thus it usually will be combined with a previous prefix operation to save communication time. An increment route (decrement route) is called by $\mathrm{IncrementRoute}(R)$ ($\mathrm{DecrementRoute}(R)$), where R is the record to be sent to the next higher (lower) processor.

Merge Two sorted lists of size $n/2$ stored one item per processor can be merged in $2\log n$ communication time. If one of the lists is stored in reverse order, they can be merged in $\log n$ communication time [1]. In our algorithms, we assume that at each level of recursion, the lists in the odd subcubes are kept

in reversed order, so all the merges we use will take $\log n$ communication time. A merge is called by $\mathrm{Merge}(\prec, K, R)$, where $\prec$ is the relative operator used to compare keys K of the records R.

Sort A list of n items stored one per processor can be sorted in $0.5\log^2 n + 0.5\log n$ time using Bitonic Sort [1]. A sort is called by $\mathrm{Sort}(\prec, K, R)$ where $\prec$ is the relative operator used to compare keys K of the records R.

4 Examples

We will assume each processor starts with a record p which contains the fields p_x (the x-cooordinate), p_y (the y-coordinate), p_m (which is false (0) for records in the lower half of the processors, otherwise true (1)) and some others depending on the problem. Then, in accordance with the cross-stitching idea of sorting by x coordinate and merging by y coordinate, each procedure will first call Sort$(\leq, p_x, p)$, and then call CrossStitch(n). We will assume that during the recursive cross-stitch procedure, each processor knows the x coordinates of the left, middle, and right vertical lines of the two sets of points to be merged. These will be denoted x_l, x_m, and x_r, respectively.

The easiest example of a cross-stitching algorithm is the solution to **2-Dimensional Dominance Count**, given in Figure 2. Note that here and elsewhere we are actually performing merge and prefix operations on subcubes with processors from i to j, so these calls should include parameters for i and j. We leave these off to make the algorithm more readable. We also will not be concerned with the trivial change to keep the records in each odd numbered subcube in reverse order. Again this is done for clarity and readability.

This stitch operation consists of a merge and a prefix operation, so the stitch constant is 3. The total communication time of this cross-stitching is $1.5\log^2 n + 1.5\log n$. When added to the initial sorting time of $0.5\log^2 n + 0.5\log n$, we see that this algorithm takes $2\log^2 n + 2\log n$ communication steps.

The stitch procedure for the **2-Set Dominance Count** algorithm is similar, but with two prefix operations. These prefix operations can actually be performed together, however, and thus the total communication time will still be $2\log^2 n + 2\log n$. The stitch procedure for **3-Dimensional Maxima** is also similar, but instead of counting the number of points below and to the left, we find the maximum of the z coordinates of the points above and to the right. This can also be done in $2\log^2 n + 2\log n$ communication time. Due to space restrictions, these algorithms are omitted.

Willard and Wee [7] showed that **All Nearest Neighbors** can be solved with an algorithm similar to dominance counting, but using a Π-quasi-valid operator instead of plus. A Π-quasi-valid operator is defined as follows. First, a Π-*quasi-valid response* to a search query is a set which is a superset of the answer to the search query and satisfies a constraint Π. A binary operator which is Π-*quasi-valid* will take two Π-quasi-valid responses to search queries and return a Π-quasi-valid response to the union of the search queries.

The Π-quasi-valid operator used in All Nearest Neighbors is defined as follows. Let R designate a region, x_0 be the smallest x coordinate of a point in $S \cap R$, and y_0 be the smallest y coordinate of a point in $S \cap R$. Let $r = (x_0, y_0)$. Then let $V(r)$ be the points in $S \cap R$ which are as close to r as to any other points in $S \cap R$. Define a query Q_R which returns $V(r)$. The constraints Π which a quasi-valid response A to Q_R must meet will consist of $A \subset S \cap R$ and $|A| \leq 3$. The definition of the Π-quasi-valid operator follows from the definition of the Π-quasi-valid response to the query Q_R.

The Π-quasi-valid operator over searches below and to the left of a point (x, y) will be denoted $\pi_{x,y}$, and the stitch procedure is shown in Figure 3. Candidate points in all quadrants can actually be found at the same time in order to save communication time. After the candidates are found, they can be sorted, and

procedure CS(i, j)
(* Recursive procedure called to work on items $i \ldots j$ *)
begin
In parallel call CS$(i, (i+j-1)/2)$ and CS$((i+j+1)/2, j)$
Stitch(i, j) (* stitch solutions together *)
end

procedure CrossStitch(n)
(* Main procedure to solve the problem on $0 \ldots n-1$ *)
begin
Sort the data by x coordinate
Call CS$(0, n-1)$
end

Figure 1: The Cross-stitching Procedure

procedure Stitch(i, j) (* 2-d Dominance Count
Each processor contains a point record $p = (p_x, p_y, p_m, p_c)$.
At the end of the stitch, p_c will contain the correct dominance
count for the points in processor i through j, and these points
will be sorted by y coordinate. *)
begin
$p_m = id > (i + j - 1)/2$
Merge$(\leq, p_y, p)$
$s =$Prefix$(+, 1 - p_m)$ (* count points from the left half *)
if p_m **then** $p_c = p_c + s$
(* add to dominance counts in the right half *)
end

Figure 2: Stitch procedure for 2-d Dominance Count

a segmented prefix with the minimum operation within groups of candidates will find each point's nearest neighbor. Another prefix can place one record per processor containing a point and its nearest neighbor, and, using a minimum operation, find the closest pair of points. Thus both algorithms will use two sorts, a cross-stitch with a stitch constant of 3, and two prefix operations. Therefore the total communication time is $2.5\log^2 n + 6.5\log n$.

For **Iso-oriented Segment Intersection**, we create a record for each endpoint, and initially sort these endpoints. The cross-stitching procedure will determine which horizontal line segments cross a vertical line segment. For each that does, at least one of its endpoint records will be marked. Then the same procedure is performed with the coordinates reversed to determine which vertical line segments cross horizontal segments. These can be done at the same time to reduce communication time. The stitch procedure is given in Figure 4. Each stitch consists of a merge and a prefix, and thus the stitch constant will be 3. Thus cross-stitching will take $1.5\log^2 n + 1.5\log n$ communication time. With the initial sort, the total communication time will be $2\log^2 n + 2\log n$.

For **Rectangle Intersection** and **Rectangle Area** (see Figure 5) we create a record for each corner of the rectangle, and use these to solve the problems. By combining prefix operations and increment routes wherever possible, Rectangle Area can be solved in $5\log^2 n + 3\log n$ communication time, and Rectangle Intersection can be solved in $2\log^2 n + 2\log n$ communication time. Note however that Rectangle Area does not exactly follow the cross-stitch outline since it must perform some calculations before making the recursive calls to the procedure $CS()$. Therefore we have given the whole $CS()$ procedure for Rectangle Area. The algorithm for Rectangle Intersection has been omitted due to space restrictions.

Visibility from a Point can be solved by merging lists of visible line segments together in $1.5\log^2 n + 1.5\log n$ total communication time. The stitch procedureis not exactly a cross-stitch, since the segments need not be initially sorted and the stitch is not a vertical merge, but it follows the basic cross-stitch outline. Due to space limitations, the algorithm is omitted.

From the above examples, we obtain the following theorem.

Theorem 4.1 *The following problems can all be solved using cross-stitching in* $\Theta(\log^2 n)$ *time with the following communication times:*

2-*Dimensional Dom. Counting*	$2.0\log^2 n + 2.0\log n$
2-*Set Dominance Counting*	$2.0\log^2 n + 2.0\log n$
3-*Dimensional Maxima*	$2.0\log^2 n + 2.0\log n$
Closest Pair	$2.5\log^2 n + 6.5\log n$
All Points Nearest Neighbors	$2.5\log^2 n + 6.5\log n$
Iso-oriented Segment Intersec.	$2.0\log^2 n + 2.0\log n$
Rectangle Intersection	$2.0\log^2 n + 2.0\log n$
Rectangle Area	$5.0\log^2 n + 3.0\log n$
Visibility from a Point	$1.5\log^2 n + 1.5\log n$

Additional problems which can be solved using cross-stitching techniques on the hypercube include:

Convex Hull Given a set S of n planar points, find vertices of the smallest convex polygon containing S.

All Iso-Oriented Rectangles Nearest Neighbors Given a set S of n iso-oriented planar rectangles, for each rectangle $r \in S$ find a rectangle in $S - \{r\}$ that is closest to r in the Euclidean metric.

procedure Stitch(i, j) (* Nearest Neighbors and Closest Pair
Each processor contains a point record $(p_x, p_y, p_m, p_a, p_b, p_c, p_d)$.
p_a, p_b, p_c, and p_d are sets of points which could have p
as a nearest neighbor. They are split by quadrant. *)
begin
$p_m = id > (i + j - 1)/2$
Merge$(\geq, p_y, p)$
if p_m **then** $s =$Prefix(π_{x_m, p_y}, p_a)
else $s =$Prefix$(\pi_{x_m, p_y}, NULL)$
(* find candidates in the left half *)
if p_m **then** $p_a = \pi_{p_x, p_y}(p_a, s)$
(* check if points are candidates for points on right*)
(* Now compute similarly for the other 3 quadrants *)
end

Figure 3: All Nearest Neighbors and Closest Pair

procedure Stitch(i, j) (* Iso-Oriented Segment Intersection
Each processor contains a segment record $(p_x, p_y, p_{x'}, p_{y'}, p_m, p_i)$.
(p_x, p_y) is this endpoint, $(p_{x'}, p_{y'})$ is the other, of a segment.
p_i will be set to TRUE if and only if this segment intersects
another segment. Initially p_i is FALSE. *)
begin
$p_m = id > (i + j - 1)/2$
Merge$(\leq, p_y, p)$
(* First set lower endpoints of vertical lines in the left half
to $+1$, and their upper endpoints to -1 *)
$k = 0$
if $p_x = p_{x'}$ **and** (**not** p_m **and** $p_y < p_{y'}$ **then** $k = +1$
if $p_x = p_{x'}$ **and** (**not** p_m **and** $p_{y'} < p_y$ **then** $k = -1$
$s =$Prefix$(+, k)$
if $s > 0$ **and** p_m **and** $p_{x'} < x_l$ **then** $p_i = TRUE$
(* If a horizontal line crosses the left half and
intersects a vertical line, mark it *)
(* Perform a similar procedure for the right half *)
(* Now repeat the whole procedure with coordinates reversed *)
end

Figure 4: Iso-oriented Segment Intersection

procedure CS(i, j)
(* CrossStitch procedure for Rectangle Area.
Each processor contains a record $p = (p_x, p_y, p_{x'}, p_e, p_m)$.
(p_x, p_y) is this endpoint, $(p_{x'}, p_y)$ is the other endpoint of the
horizontal edge.
p_e will be TRUE if the edge is a lower edge, else FALSE. *)
begin
$p_m = id > (i + j - 1)/2$
Merge$(\leq, p_y, p)$
$k = 0$
if $p_{x'} \leq x_l$ **and** p_m **and** p_e **then** $k = +1$
if $p_{x'} \leq x_l$ **and** p_m **and not** p_e **then** $k = -1$
$s =$Prefix$(+, k)$
$q =$IncrementRoute(p)
if $s > 0$ **then** $k = p_y - q_y$
else $k = 0$
$s =$Prefix$(+, k)$
$k' = 0$
if $p_{x'} \geq x_r$ **and not** p_m **and** p_e **then** $k' = +1$
if $p_{x'} \geq x_r$ **and not** p_m **and not** p_e **then** $k' = -1$
$s' =$Prefix$(+, k')$
$q =$IncrementRoute(p)
if $s' > 0$ **then** $k' = p_y - q_y$
else $k' = 0$
$s' =$Prefix$(+, k')$
if p_m **then** $p_y = p_y - s'$
else $p_y = p_y - s$
$a = k(x_m - x_l) + k'(x_r - x_m)$
$s =$GlobalPrefix$(+, a)$ (* over all n processors *)
if $id = n - 1$ **then** $T = T + s$ (* Total in processor $n-1$ *)
In parallel call CS$(i, (i+j-1)/2)$ and CS$((i+j+1)/2, j)$
end

Figure 5: CrossStitch procedure for Rectangle Area

Visibility from a Point with Intersections Given n possibly intersecting line segments and a point p, determine that part of the plane visible from p if all segments are opaque.

Convex Hull can be solved using a stitch which merges line segments of two convex hulls by slopes to obtain support lines [6]. All Rectangles Nearest Neighbors can be solving using a stitch that is similar to the stitches of All Nearest Neighbors and Rectangle Intersection put together. Visibility from a Point with intersecting line segments can be done using a cross-stitching technique similar to the one for Visibility from a Point with non-intersecting line segments. More care must be taken in merging however, and the interested reader is referred to [3, 4] for details.

5 Conclusion

Cross-stitching is a systematic approach for producing simple and uniform hypercube algorithms for computational geometry that have very low communication constants and are very practical. All of these algorithms run in at most $5.0 \log^2 n + O(\log n)$ communication steps and thus will outperform the known asymptotically faster algorithms [4] on any feasible data sets. We note that while we have only given an exact analysis of communication times, the local processing times of all algorithms is also $\Theta(\log^2 n)$, and again involves small constants.

References

[1] K. Batcher, "Sorting networks and their applications", *Proc. AFIPS Spring Joint Computer Conf.* 32, 307–314, 1968.

[2] L. Boxer and R. Miller, "Dynamic computational geometry on meshes and hypercubes", *Proc. 1988 Intl. Conf. on Parallel Proc.*, 323–330.

[3] J. Hershberger, "Finding the upper envelope of n line segments in $O(n \log n)$ time", *Info. Process. Lett.*, 33(4):169–174.

[4] P.D. MacKenzie and Q.F. Stout, "Asymptotically optimal hypercube algorithms for computational geometry", *Proc. 3rd Frontiers of Massively Parallel Proc.*, 1990.

[5] R. Miller and Q.F. Stout, *Parallel Algorithms for Regular Architectures*, MIT Press, 1990.

[6] I. Stojmenovic, "Computational geometry on a hypercube", *Proc. 1986 Intl. Conf. on Parallel Proc.*, 100–103.

[7] D.E.Willard and Y.C. Wee, "Quasi-valid range querying and its implications for nearest neighbor problems", *Proc. 4th ACM Symp. on Comp. Geom.*, 34–43, 1988.