A Time-Optimal Multiple Search Algorithm on Enhanced Meshes, with Applications

(Extended Abstract)

D. Bhagavathi, S. Olariu*, W. Shen, and L. Wilson
Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162

1. Introduction

The mesh-connected computer architecture has emerged as one of the more natural choices for solving a large number of computational tasks in image processing, computational geometry, and computer vision [6]. Its regular structure and simple interconnection topology makes the mesh particularly well suited for VLSI implementation [10]. However, the mesh tends to be slow when it comes to handling data transfer operations over long distances. In an attempt to overcome this problem, mesh-connected computers have recently been augmented by the addition of various types of bus systems [1,4,10]. One such system has been adopted by the DAP family of computers [8,10], and involves enhancing the mesh architecture by the addition of row and column buses. In [5] an abstraction of such a system is referred to as mesh with multiple broadcasting.

A $\sqrt{n} \times \sqrt{n}$ mesh with multiple broadcasting, consists of n identical processors positioned on a square array. The processor located in row i and column j $(1 \leq i, j \leq \sqrt{n})$ is referred to as $P(i,j)$. Throughout this paper we assume that the enhanced mesh operates in SIMD mode; each processor is assumed to know its own coordinates within the mesh and to have a constant number of registers of size $O(\log n)$; in unit time every processor can handle at most $O(\log n)$ bits of information by performing an arithmetic/boolean operation, communicating with one of its neighbors using a local link, broadcasting a value on a bus or reading a value from a specified bus. Only one processor is allowed to broadcast on a given bus at any one time. By contrast, all the processors on the bus can simultaneously read the bus. In accord with other researchers [4,5,7,8], we assume that communications along buses take $O(1)$ time. Recent experiments with the YUPPIE multiprocessor array system seem to indicate that this is a reasonable working hypothesis [7].

In this paper we propose a time-optimal algorithm for the multiple search problem on enhanced meshes and show that a number of problems in computer graphics, image processing, robotics, and computational geometry reduce to the multiple search problem or a variant thereof. The *multiple search* problem [2,12] is defined as follows: given a sorted sequence $A = a_1, a_2, \ldots, a_n$ of items from a totally ordered universe along with a sequence $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ of *queries* from the same universe, determine for every q_j $(1 \leq j \leq m)$ the unique a_i for which $a_{i-1} \leq q_j < a_i$. To handle boundary conditions, we augment the sequence A by the addition of two "dummy" elements, namely $a_0 = -\infty$ and $a_{n+1} = \infty$. Recently, Akl and Meijer [2] as well as Wen [12] have studied the multiple search problem in the PRAM model of computation. To the best of our knowledge, this important problem has found no solution in the context of meshes with multiple broadcasting.

Our algorithm runs in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh with multiple broadcasting. We also show that in this model $\Omega(\sqrt{m})$ is a lower bound for the multiple search problem. An interesting feature of our algorithm is that for any given n and m $(1 \leq m \leq n)$, the running time is independent

*Support by the National Science Foundation under grant CCR-8909996 is gratefully acknowledged

of n, thus the algorithm is adaptive.

2. The Algorithm

Consider a $\sqrt{n} \times \sqrt{n}$ mesh $\mathcal{R}$ with multiple broadcasting. The elements of the sequence A are stored in $\mathcal{R}$, one item per processor, in row-major order. The queries q_j $(1 \leq j \leq m)$ are stored initially in the first $\frac{m}{\sqrt{n}}$ columns[1] of $\mathcal{R}$ one query per processor.
The *solution* of q_j is defined to be the item a_i such that $a_{i-1} \leq q_j < a_i$ $(1 \leq i \leq n)$ or if no such a_i exists the solution is taken to be $+\infty$. For each query q_j smaller than a_n we define its *row rank* to be the row in the mesh $\mathcal{R}$ that contains the solution of q_j, otherwise the row rank is $\sqrt{n}+1$.

Our algorithm proceeds in four stages: in Stage 0 the queries are moved into a square submesh of size $\sqrt{m} \times \sqrt{m}$; in Stage 1 we determine the row rank of every query in Q; Stage 2 has two goals: after sorting the queries by row rank we solve a subset of the queries and extract further information that will be used in Stage 3; finally, the goal of Stage 3 is to use the information computed in Stages 1 and 2 to solve all the remaining queries.

Stage 0

The purpose of this stage is to move the sequence Q of queries into the $\sqrt{m} \times \sqrt{m}$ submesh $\mathcal{R}'$ consisting of the intersection of the first $\sqrt{m}$ rows and columns of $\mathcal{R}$. In the remainder of this paper, the columns of $\mathcal{R}'$ will be referred to as *query-columns*. Let B denote the submesh of $\mathcal{R}$ consisting of processors $P(i,j)$ with $\sqrt{m}+1 \leq i \leq \sqrt{n}$ and $1 \leq j \leq \frac{m}{\sqrt{n}}$; similarly, let C be the submesh of $\mathcal{R}$ consisting of processors $P(i',j')$ with $1 \leq i' \leq \sqrt{m}$ and $\frac{m}{\sqrt{n}}+1 \leq j' \leq \sqrt{m}$. To achieve the goal of this stage we need only move the queries stored by processors in the submesh B to the processors in the submesh C, one query per processor. Due to space limitations, the details of this data movement are omitted. They can be found in [3]. We have the following result.

Lemma 2.1. The sequence Q of queries initially stored in the first $\frac{m}{\sqrt{n}}$ columns of $\mathcal{R}$ can be moved into a $\sqrt{m} \times \sqrt{m}$ submesh in $O(\sqrt{m})$ time. □

Stage 1.

The purpose of this stage is to determine the row rank of every query in Q. To accomplish the task specific to this stage, we let every processor $P(i,\sqrt{n})$ $(1 \leq i \leq \sqrt{n})$ broadcast the item $a_{i\sqrt{n}}$ horizontally to processor $P(i,i)$. Next, $P(i,i)$, broadcasts the item $a_{i\sqrt{n}}$ vertically to the whole column i of the mesh. As a result of this data movement, every processor in column i $(1 \leq i \leq \sqrt{n})$, becomes aware of the last item in row i.

To determine the row rank of every query in Q, the query-columns of $\mathcal{R}'$ are handled sequentially. For every j $(1 \leq j \leq \sqrt{m})$, each query in column j of $\mathcal{R}'$ is broadcast horizontally along its row bus. For every i $(1 \leq i \leq \sqrt{m})$, every processor $P(i,k)$ in row i of $\mathcal{R}$ compares the query it receives with $a_{k\sqrt{n}}$ (i.e. the last item in row k), and marks itself 0 or 1 depending on whether or not $a_{k\sqrt{n}}$ is strictly larger than the query.

In case processor $P(i,\sqrt{n})$ is marked 1, the row rank of the corresponding query is $\sqrt{n}+1$ and $P(i,\sqrt{n})$ will broadcast this information to $P(i,j)$ (i.e. the processor that has broadcast the query). Otherwise, let $P(i,k)$ be the leftmost processor in row i that is marked 0. Now it is easy to confirm that k must be the row rank of the query and $P(i,k)$ sends the appropriate information to processor $P(i,j)$. We now summarize our findings as follows.

Lemma 2.2. The row ranks of all queries in Q can be determined in $O(\sqrt{m})$ time. □

Stage 2

Stage 2 has two basic tasks: the first one involves solving a part of the queries in Q; the second involves further processing the query-submesh $\mathcal{R}'$ to extract information that will be needed to

[1]To avoid tedious but inconsequential details we assume that $\frac{m}{\sqrt{n}}$ is an integer.

solve the remaining queries in the next stage.

To begin, using an optimal sorting algorithm for meshes [11], the sequence Q of queries is sorted in column-major order by row rank. A sorted query-column of $\mathcal{R}'$ is called *pure* if all the queries in the column have the same row rank. Otherwise, the query-column is termed *impure*. We begin by identifying in $O(1)$ time every query-column as pure or impure.

First, we note that the solution is $+\infty$ for all queries whose row rank is $\sqrt{n}+1$. Next, we process pure query-columns with row ranks at most $\sqrt{n}$, one by one. Specifically, let query-column j be pure and assume that the row rank of every query in j is r. Using column buses, row r is replicated through the entire mesh $\mathcal{R}$. Further, each query in column j is broadcast horizontally using its row bus. Note that in every row i $(1 \leq i \leq \sqrt{m})$ of $\mathcal{R}$ there exists precisely one processor that determines and returns the solution of the corresponding query.

Impure query-columns of $\mathcal{R}'$ are handled differently. In preparation for this, we determine which rows of the mesh $\mathcal{R}$ are dense and which ones are sparse. A row r of $\mathcal{R}$ is *sparse* if the number of queries in impure columns of $\mathcal{R}'$ whose row rank is r is not greater than $\sqrt{m}$. Otherwise, row r is termed *dense*. To complete Stage 2, we identify sparse and dense rows of $\mathcal{R}$; in addition, we build a linked list consisting of the dense rows.

Let j be an arbitrary impure query-column of $\mathcal{R}'$ and let $r_1, r_2, \ldots, r_t$ $(t \geq 2)$ be the row ranks of the queries in column j. Since Q was sorted in column-major order, the queries having the same row rank occur consecutively in j. For further reference, such a set of queries is termed a *run*. It is important to note that for any r $(1 \leq r \leq \sqrt{n})$, at most two impure query-columns contain queries whose row rank is r. Note further that if $t \geq 3$ then all rows $r_2, \ldots, r_{t-1}$ must be sparse, since queries having such row ranks cannot occur in a different query-column.

Consequently, any potential dense row r must "straddle" two impure columns. By the above observation, a dense row can only correspond to the bottomost run in some impure column and the topmost run in the closest impure column to its right. It is an easy matter to compute the number of queries in every such run, and using column buses to send to processor $P(1,j)$ in every impure column j ordered pairs consisting of the row number and the number of queries in the topmost and bottomost runs in column j. Finally, traversing the first row of $\mathcal{R}'$ sequentially, dense rows can be identified and a linked list containing all the dense rows in increasing order can be built. It is easy to see that Stage 2 can be done in $O(\sqrt{m})$ time. Consequently, we have the following result.

Lemma 2.3. The task of solving all the queries with row ranks $\sqrt{n}+1$ and those in pure columns of $\mathcal{R}'$, as well as that of identifying sparse and dense rows of $\mathcal{R}$ can be performed in $O(\sqrt{m})$ time. □

Stage 3. The goal of this stage is to solve the remaining queries in Q. In a first step, every query in an impure column is moved to the row of the mesh that equals its row rank. Again, due to space limitations the details are ommited; the reader can find the details in [3].

Our algorithm proceeds by first solving all queries in sparse rows, and then in all the dense rows. In each sparse row, the solution for each query is determined one by one, by broadcasting its value across the row, and having a unique processor identify the corresponding solution. Let r be a dense row of $\mathcal{R}$. Begin by replicating row r throughout the mesh $\mathcal{R}$ by using vertical buses. Note that every processor in row r will broadcast the item and any possible queries it holds. Consider the diagonal processors $P(i,i)$ of the mesh. It is easy to see that each of them contains at most two queries. For all values of i $(1 \leq i \leq \sqrt{n})$ if processor $P(i,i)$ contains queries it will use row i to solve them in $O(1)$ time. Consequently, we have

Lemma 2.4 The task of solving all the queries in impure columns of $\mathcal{R}'$ can be performed in $O(\sqrt{m})$ time. □

Theorem 2.5 Given a sorted sequences $A = a_1, a_2, ..., a_n$ of items from a totally ordered universe and a sequence $Q = q_1, q_2, ..., q_m$ $(1 \leq m \leq n)$ of queries, the corresponding multiple search problem can be solved in $O(\sqrt{m})$ time on a mesh with multiple broadcasting of size $\sqrt{n} \times \sqrt{n}$. Furthermore, this is time-optimal. □(For the proof see [3].)

3. Applications

The purpose of this section is to show that the multiple search problem finds a number of surprising applications to problems in computer graphics, image processing, robotics, and computational geometry. In what follows we assume an underlying convex polygon $P = p_1, p_2, \ldots, p_n$ in standard form stored in row-major order in a $\sqrt{n} \times \sqrt{n}$ mesh $\mathcal{R}$ with multiple broadcasting.

Let $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ be a sequence of points in the plane. The *multiple point inclusion* problem asks for determining for every subscript j $(1 \leq j \leq m)$ whether the query point q_j is inside P. As it turns out, the problem at hand can be solved by reducing it to the multiple search problem. To begin, choose an arbitrary point ω inside P and convert the vertices of P as well as the query points in Q to polar coordinates with pole ω and polar axis ωp_1.

It is well-known that the vertices of P occur in sorted angular order about ω [9]. Now solving the corresponding instance of the multiple search problem, we determine for every query point q_j the wedge $p_{i-1}\omega p_i$ within which q_j lies. Finally, in one more comparison, it can be decided whether p_j and ω lie on the same side of the segment $p_{i-1}p_i$. Consequently, we have the following result.

Theorem 3.1. Given a convex polygon $P = p_1, p_2, \ldots, p_n$ and a sequence $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ of points in the plane, the multiple point inclusion problem can be solved in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh with multiple broadcasting. □

Note that Theorem 3.1 holds for star-shaped polygons, provided that a point ω in the kernel is known. A related problem is known as the *convex polygon containment* problem. Here, we are given convex polygons $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ and we are interested in finding out whether Q is contained in P. Note that this problem can be easily solved by reducing it to the multiple point inclusion problem.

Corollary 3.2. The convex polygon containment problem for $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ can be solved in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ enhanced mesh. □

Next, consider a convex polygon $P = p_1, p_2, \ldots, p_n$ and a sequence $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ of points in the plane. For a point q_j *exterior* to P the two supporting rays for P emanating from q_j are referred to as *right* and *left* depending on whether the interior of P lies to the left or to the right of the ray. Our task is to compute for every point q_j exterior to P the corresponding left and right rays. For definiteness we refer to this as the *multiple ray* problem. Surprisingly, the solution to the multiple ray problem reduces to a variant of the multiple search problem.

Recall that, by assumption, the polygon P is stored in row-major order in a $\sqrt{n} \times \sqrt{n}$ mesh $\mathcal{R}$ with multiple broadcasting. We assume that the points in Q are stored in the $\sqrt{m} \times \sqrt{m}$ submesh $\mathcal{R}'$ consisting of the first $\sqrt{m}$ rows and columns of $\mathcal{R}$. By using the solution to the multiple point inclusion problem we determine which points of Q are exterior to P. Assume, without loss of generality, that all vertices of Q are exterior to P. We now show how to reduce the problem of computing the *left* supporting ray for every point in Q to a variant of the multiple search problem. (Handling right supporting rays is done by a symmetric argument.)

For this purpose, consider the convex polygon $P' = p_{\sqrt{n}}, p_{2\sqrt{n}}, \ldots, p_n$ consisting of the vertices of P whose subscripts are multiples of $\sqrt{n}$. Note that P' partitions the boundary of P into chains $C_1, C_2, \ldots, C_{\sqrt{n}}$ such that $C_i = p_{(i-1)\sqrt{n}+1}, p_{(i-1)\sqrt{n}+2}, \ldots, p_{i\sqrt{n}}$ $(1 \leq i \leq \sqrt{n})$. It is easy to see that the vertices of P' are stored by the processors in the last column of $\mathcal{R}$, and that every chain C_i $(1 \leq i \leq \sqrt{n})$ defined above involves points stored by processors in row i of the mesh.

We note that the multiple ray problem involving P' and Q can be solved in $O(\sqrt{m})$ time. As in the multiple search problem, we let every processor $P(i, \sqrt{n})$ holding $p_{i\sqrt{n}}$ broadcast the point horizontally to processor $P(i,i)$ which, in turn, broadcasts $p_{i\sqrt{n}}$ to the entire column i of the mesh. Next, the columns of $\mathcal{R}'$ are processed sequentially as follows.

To process query-column j $(1 \le j \le \sqrt{m})$, each point in that column is broadcast horizontally to all the processors in its own row. To make the exposition easier to follow, consider a generic point q in Q stored in some processor $P(i,j)$ of $\mathcal{R}'$. Note that exactly *two* processors in row i determine that the ray emanating from q and passing through the points of P' they contain, are supporting rays for P'. However, only one of them detects that the corresponding ray is a left ray. The processor in row i that detects this condition broadcasts its identity to processor $P(i,j)$.

Observe that if the left supporting ray for P' determined by some point q in Q and some point $p_{t\sqrt{n}}$ is a supporting ray for P, then no further action is needed. Otherwise, it is easy to confirm that the ray $qp_{t\sqrt{n}}$ intersects *precisely* one of the chains C_{t-1} or C_t. Furthermore, the chain intersected by the ray $qp_{t\sqrt{n}}$ can be determined in O(1) time by checking the edges of P incident to $p_{t\sqrt{n}}$.

Consider points q in Q for which the left supporting ray for P' is not a supporting ray for P. For every such point q, we define its *chain rank* to be the subscript of the chain which the left supporting ray to P' from q intersects. Further, we sort the points in Q in column-major order by their chain ranks. Assume that the chain rank of q is t. In order to find a left supporting ray for P emanating from q we need find a left supporting ray for the convex polygon determined by $p_{(t-1)\sqrt{n}}, p_{(t-1)\sqrt{n}+1}, \ldots, p_{t\sqrt{n}}$. This can be done in $O(\sqrt{m})$ time by a slight modification of the Stages 2 and 3 of Section 2. Consequently, we have the following result.

Theorem 3.3. Given a convex polygon $P = p_1, p_2, \ldots, p_n$ and a sequence $Q=q_1, q_2, \ldots, q_m$ $(1 \le m \le n)$ of points in the plane, the corresponding multiple ray problem can be solved in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh with multiple broadcasting. □

As a further application, consider the following problem that arises frequently in computer graphics, robotics, and image processing. A convex polygon $P = p_1, p_2, \ldots, p_n$ is given along with a sequence $L=l_1, l_2, \ldots, l_m$ $(1 \le m \le n)$ of lines in the plane. The *multiple stabbing* problem involves answering queries of the type "does line l_j intersect P?".

As we are about to explain, the multiple stabbing problem reduces to the multiple ray problem. To begin, for every j $(1 \le j \le m)$ choose a point q_j on the line l_j. What results is a sequence $Q=q_1, q_2, \ldots, q_m$ $(1 \le m \le n)$ of points in the plane. In $O(\sqrt{m})$ time we solve the multiple point inclusion problem.

Clearly, if some point q_j is inside P then the line l_j intersects the polygon. For the points in Q that are outside P we solve the multiple ray problem. Finally, we determine whether l_j is within the wedge bounded by the left and right supporting rays at q_j. We have proved the following result.

Theorem 3.4. Given a convex polygon $P = p_1, p_2, \ldots, p_n$ and a sequence $L=l_1, l_2, \ldots, l_m$ $(1 \le m \le n)$ of lines in the plane, the corresponding multiple stabbing problem can be solved in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh with multiple broadcasting. □

Finally, consider again two convex polygons $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$ $(1 \le m \le n)$ both in standard form. An important problem in computational geometry is to determine whether P and Q are linearly separable and, if so, to construct a line that separates them.

As it turns out, the solution to the multiple ray problem affords us an efficient solution to the problem at hand. Specifically, assume that a solution to the multiple ray problem is available. For definiteness, we assume that the polygon Q is stored in column-major in the submesh $\mathcal{R}'$ consisting of the first $\sqrt{m}$ rows and columns of $\mathcal{R}$. Note that the polygons are separable if and only if some common supporting line for the two polygons is separating.

We proceed as follows: every processor in $\mathcal{R}'$ determines whether any of its supporting rays for P is also a supporting ray for Q. This, of course can be done in O(1) time by verifying whether or not the left and right neighbors of a vertex in Q are to the same side of the supporting ray. Note that either the polygons are found not to be separable or else rays that are common supporting lines for P and Q will be identified in the process described above. It is now an easy matter to identify those supporting rays that are separating. Therefore, we have the following result.

Theorem 3.5. Given convex polygons $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ in the plane, the task of determining whether P and Q are linearly separable can be performed in $O(\sqrt{m})$ time on a $\sqrt{n} \times \sqrt{n}$ enhanced mesh. Furthermore, in case the polygons are separable a separating line can be identified in the same time bounds. □

Note also that with a trivial modification, the algorithm for computing a separating line for two convex polygons can determine common tangents in case neither polygon contains the other. The following result summarizes this finding.

Corollary 3.6. Given convex polygons $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$ $(1 \leq m \leq n)$ in the plane, the common tangents to P and Q, if any, can be computed in $O(\sqrt{m})$ time on a mesh with multiple broadcasting of size $\sqrt{n} \times \sqrt{n}$. □

References

1. A. Aggarwal, Optimal bounds for finding maximum on array of processors with k global buses, *IEEE Trans. on Computers*, C-35, 1986, 62-64.
2. S. G. Akl and J. Meijer, Parallel binary search, *IEEE Transactions on Parallel and Distributed Systems*, 1, 1990, 247-250.
3. D. Bhagavathi, S. Olariu, W. Shen, and L. Wilson, A Time-Optimal Multiple Search Algorithm on Enhanced Meshes, with Applications, Department of Computer Science, Old Dominion University, Tech. Report TR-092-08, 1992.
4. S. H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. on Computers* vol. C-33, no. 2, Feb. 1984. 133-139.
5. V. P. Kumar and C. S. Raghavendra, Array processor with multiple broadcasting, *Journal of Parallel and Distributed Computing,* vol 2, 1987, 173-190.
6. F. Thomson Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Publishers, San Mateo, California, 1992.
7. M. Maresca and H. Li, Connection autonomy and SIMD computers: a VLSI implementation, *Journal of Parallel and Distributed Computing,* vol. 7, 1989, 302-320.
8. D. Parkinson, D. J. Hunt, and K. S. MacQueen, The AMT DAP 500, 33rd *IEEE Comp. Soc. International Conf.*, Feb. 1988, 196-199.
9. F. P. Preparata and M. I. Shamos, Computational Geometry, An Introduction, Springer-Verlag, New York, Berlin, 1988.
10. D. Tabak, Multiprocessors, Prentice-Hall, Englewood Cliffs, New Jersey, 1990
11. C. D. Thompson and H. T. Kung, Sorting on a mesh-connected parallel computer, *Communications of the ACM*, vol. 20, 1977, 263-271.
12. Z. Wen, Parallel Multiple Search, *Information Processing Letters*, vol. 37, February 1991, 181-186.