Randomized Fully-Scalable BSP Techniques for Multi-Searching and Convex Hull Construction

(Preliminary Version)

Michael T. Goodrich*
Center for Geometric Computing
Dept. of Computer Science
Johns Hopkins Univ.
Baltimore, MD 21218
goodrich.cs.jhu.edu

Abstract

We study randomized techniques for designing efficient algorithms on a p-processor bulk-synchronous parallel (BSP) computer, which is a parallel multicomputer that allows for general processor-to-processor communication rounds provided each processor is guaranteed to send and receive at most h items in any round. The measure of efficiency we use is in terms of the internal computation time of the processors and the number of communication rounds needed to solve the problem at hand. We present techniques that achieve optimal efficiency in these bounds over all possible values for p, and we call such techniques *fully-scalable* for this reason. In particular, we address two fundamental problems: multi-searching and convex hull construction. Our methods result in algorithms that use internal time that is $O(\frac{n \log n}{p})$ and, for $h = \Theta(n/p)$, a number of communication rounds that is $O(\frac{\log n}{\log(h+1)})$, with high probability. Both of these bounds are asymptotically optimal for the BSP model.

1 Introduction.

Most of the research on parallel computational geometry in the past decade has focused on fine-grain massively-parallel models of computation (e.g., see [2, 3, 6, 28, 41]), where the ratio of memory to processors is fairly small (typically $O(1)$), and this focus has been independent of whether the model of computation was a parallel random-access machine (PRAM) or a network model, such as the hypercube. But, as more and more parallel computer systems are being built, researchers are realizing that processor-to-processor communication is a prime bottleneck in parallel computing (e.g., see Aggarwal *et al.* [1], Bilardi and Preparata [13], Culler *et al.* [15], Kruskal *et al.* [30], Mansour *et al.* [31], Mehlhorn and Vishkin [33], Papadimitriou and Yannakakis [37], and Valiant [45, 44]). The real potential of parallel computational geometry, therefore, probably lies in algorithm design for coarse-to-medium-grain parallel environments [18, 21], where the ratio of memory to processors is non-constant, for such systems allow an algorithm designer to balance communication latency with internal computation time. Indeed, this realization has given rise to a powerful algorithmic model, which Valiant [44] calls "bulk synchronous" processing (BSP). In such a model an input of size n is distributed evenly across a p-processor parallel computer, with $p < n$. In a single computation *round* (which Valiant calls a *superstep*) each processor may send and receive h messages (typically $h = \Theta(n/p)$) and then perform an internal computation on its internal memory cells using the messages it has just received. To avoid any conflicts that might be caused by asynchronies in the network (whose topology is left undefined) the messages sent out in a round t by some processor should not depend upon any messages that processor receives in round t (but, of course, they may depend upon messages received in round $t-1$).

The running time of a BSP algorithm is characterized by two parameters: T_I, the internal computation time, and T_C, the number of communication rounds. The goal in designing a BSP algorithm, of course, is to minimize both of these parameters. Alternatively, by introducing additional characterizing parameters of the BSP model, one can combine T_I and T_C into a single running time parameter, called the *combined running time*. Specifically, if we let L denote the latency of the network—that is, the worst-case time needed to send one processor-to-processor message—and we let g denote the time "gap" between consecutive messages received by a processor in a communication round, then we can characterize the total running time of a BSP computation as $O(T_I + (L+gh)T_C)$ [44] (similarly for the related LogP model [15, 29]).

The goal of this paper is to further the study of bulk-synchronous parallel algorithms by addressing two fundamental problems in parallel computational geometry: multi-searching and convex hull construction.

*This research supported by the NSF under Grants CCR-9300079 and CCR-9625289, and by ARO under Grant DAAH04-96-1-0013.

1.1 Previous related work in parallel computational geometry. There has been a significant amount of previous work on parallel computational geometry (e.g., see [2, 3, 6, 28, 41]). This work has resulted in a number of powerful techniques for solving computational geometry problems in parallel, with particular attention paid to convex hull construction, because of its wide applicability, such as the well-known reductions of planar Voronoi diagram and Delaunay triangulation constructions to 3-dimensional convex hulls. The current best fine-grain parallel solutions for convex hulls in $\mathbb{R}^2$ run in $O(\log n)$ time using n processors in the EREW PRAM model[1] [34] and in $\mathbb{R}^3$ run in $O(\log^2 n)$ time using $n/\log n$ processors in the EREW PRAM model [4] or, alternatively, in $O(\log n)$ time, with high probability, using n processors in the CREW PRAM model [39, 42].

Perhaps counter-intuitive to the notion of PRAM algorithms as extractors of maximum parallelism, these PRAM methods do not translate into efficient BSP algorithms. This is because simulating a PRAM algorithm in the BSP framework requires at least a constant number of communication rounds for each PRAM step (and even this is often quite difficult to achieve), whereas there are several known BSP solutions [17, 18, 21, 19, 20] to a number of computational geometry problems that use only $O(1)$ communication rounds in total, albeit assuming that p, the number of processors, is fairly small relative to n, the problem size. The best previous BSP algorithm for 3-dimensional convex hull construction is a method by Dehne *et al.* [17] that completes in $O(1)$ communication rounds, with high probability, assuming that $p \leq n^{1/(3+\epsilon)}$, for any fixed constant $\epsilon > 0$. Such algorithms are *scalable* [18, 21] in the sense that they are efficient over a range of values of p, but they are not *fully scalable*, in that there is a limit placed on this range of values (which in the case of 3-dimensional convex hull construction is fairly restrictive). In fact, the only fully-scalable BSP algorithm we are familiar with is a sorting algorithm of the author [26], which runs in $O(\log_h n)$ communication rounds, for $h = \Theta(n/p)$. This bound is $O(1)$, of course, when p is $O(n^{1-\epsilon})$ for some constant $\epsilon > 0$, and the author shows that $\Omega(\log_h n)$ communication rounds are in fact necessary, even for the simple problem of computing the bitwise-or of n bits distributed evenly across p processors in the BSP model.

While convex hull construction is a well-known "self-contained" problem that is often studied in parallel computational geometry, a general problem that often arises as a subproblem in solutions to other problems is the multi-searching problem (e.g., see [7, 8, 9, 11, 12, 23]). In this problem one is given a collection S of "generic" searches that need to simultaneously access a data structure T (which in the context of this paper will always be a binary tree) to solve the problem at hand. What makes this problem interesting is that comparing searches to each other yields no useful information (so, for example, the searches cannot be sorted by any "key" value). The only previous efficient BSP we know of for this problem is a method of Devillers and Fabri [21] that uses $O(1)$ communication rounds if $p \leq n^{1/2}$ and the communication network allows for segmented broadcasts to be performed in one round, where $n = |S| + |T|$. We refer to this version of the BSP model that allows for segmented broadcasts as the weak-CREW BSP model [26]); we call the (standard) version of the BSP model, which requires that each communication packet have a unique destination, the EREW BSP model. There is also some work by Bäumker *et al.* [11, 12] on multi-searching for another variant of the BSP that allows for very long messages, and methods by Gerbessiotis and Siniolakis for multi-searching level graphs. These methods do not translate into communication-optimal BSP algorithms for any range of values of p, however. Indeed, we are not aware of any fully-scalable algorithms for the multi-searching or 3-dimensional convex hull problems.

1.2 Our results. In this paper we give the first fully-scalable method for multi-searching in the (standard) BSP model. Our algorithm uses $O(\log_h n)$ communication rounds and internal computation time of $O((n \log n)/p)$, with high probability, for $h = \lceil n/p \rceil + 1$. Thus, the number of communication rounds is $O(1)$ any time $p \leq n^{1-\epsilon}$ for some constant $\epsilon > 0$. We demonstrate the utility of our multi-searching algorithm in the full version of this paper by applying it to several well-known parallel computational geometry problems, including searching in arrangements, 2D-all-nearest-neighbor searching, and 3D-maxima. We also describe the first fully-scalable BSP algorithm for 3-dimensional convex hull construction to illustrate the natural way multi-searching arises in other problems. Our convex hull algorithm also uses $O(\log_h n)$ communication rounds and internal computation time of $O((n \log n)/p)$, with high probability.

We begin with some preliminaries about the BSP model.

[1] The PRAM is a synchronous shared-memory model, with the EREW version not allowing for concurrent memory accesses, the CREW version allowing concurrent reads, and the CRCW allowing for concurrent reads and writes (assuming some reasonable conflict resolution protocol).

2 Some Preliminary Observations.

There is a rich body of knowledge that exists for performing basic operations on fine-grain parallel models, but the knowledge base for fully-scalable coarse-grain techniques is not as rich. Thus, before we give our methods for multi-searching and convex hull construction, let us discuss a few basic BSP primitives. The primitives we discuss have been studied by others in bulk-synchronous contexts (e.g., see [24, 44]), but we describe them here in the fully-scalable framework for the sake of completeness.

2.1 Generalized Broadcast and Combine. Let S be a set of m items stored on a single processor. The *generalized broadcasting* problem is to distribute these items to all the other processors.

Lemma 2.1: *The items in S can be broadcast to k other processors on an EREW BSP computer in $O(\log_d k + \lceil m/h \rceil)$ communication rounds, for $d = 2\lceil h/m \rceil$, where h is the maximum number of items that can be sent by a processor in a communication round.*

Proof: Let us consider two cases:

1. $m \leq h$. The idea in this case is fairly straightforward. Processor 1 sends the items in S to $2\lceil h/m \rceil$ other processors in $O(1)$ communication rounds, then these processors send S to $2\lceil h/m \rceil$ other processors, and so on. The total number of communication rounds is $O(\log_d k)$, for $d = 2\lceil h/m \rceil$.

2. $m > h$. In this case we divide S into $\lceil m/h \rceil$ subsets of size at most h each, and we broadcast each of them as in the previous case in a pipelined fashion. The total number of communication rounds is $O(\log_2 k + \lceil m/h \rceil)$.

■

Of course, if we can perform generalized broadcast in this many communication rounds, we can also perform the inverse operation in this many rounds. The "inverse" problem, which we call *generalized combine*, involves computing the value of an associative function on the items in each row of a $k \times m$ matrix, A, where each column is stored on a different processor. This is actually a special case of an even more general problem, which we describe next.

2.2 Generalized Parallel Prefix. Suppose we are given a $k \times m$ array A, with each column stored on a different processor, together with an associative summation operator defined on each row. The *generalized parallel prefix* problem is to determine for each i and l, the value of the partial sum $s_{i,l} = \sum_{j=1}^{l} A[i,j]$, where the summation operator is the one defined for row i.

Lemma 2.2: *The generalized parallel prefix problem can be solved in computer in $O(\log_d k + \lceil m/h \rceil)$ communication rounds, for $d = 2\lceil h/m \rceil$, where h is the maximum number of items that can be sent by a processor in a communication round.*

Proof: The proof is essentially a generalized combine followed by a generalized broadcast, and is left to the reader. ■

The final preliminary result we discuss is that of computing a random permutation bulk-synchronously.

2.3 Computing a random permutation on a BSP computer. Suppose we are given a set S of n elements, distributed evenly across p processors on a BSP computer. An important primitive-level computation that must often be performed in randomized parallel algorithms is to produce a random permutation of the elements of S. The method we use in this paper is an adaptation of a strategy due to Reif [40] (see also Hagerup [27]):

1. For each element s_i in S we select a random integer key s'_i in the range $[1, n^2]$, and we sort these random keys using the comparison-based optimal bulk-synchronous sorting algorithm of the author [26]. This takes $O(\log_h n)$ communication steps and $O((n \log n)/p)$ internal computation time.

2. Sequentially, we perform a random permutation for each group of s'_i elements that are given the same key. Assuming that the total number of elements in any group is at most some constant c, this step can easily be implemented in $O(1)$ communication rounds and $O(n/p)$ internal computation time. If the number of elements in some group is more than c, we repeat the process, starting with Step 1.

3. Finally, we store for each element s_i in S the position of s'_i in the sorted list. The mapping of s_i's to their respective s'_i values defines a random permutation.

This algorithm produces a random permutation of the elements in S, and all permutations are equally likely. Moreover, as is formalized in the following lemma, this procedure will terminate after just one iteration, with high probability.

Lemma 2.3: *If $c \geq 3$, then the probability that the above random permutation algorithm will not terminate in a given iteration is at most $1/n^{c-2}$.*

Proof: We prove this by an application of a Chernoff bound (see [35], p. 68). Let $X_{i,j}$ be an indicator random variable that is 1 if processor i chooses value j. Clearly, $\mathbf{Pr}(X_{i,j} = 1) = 1/n^2$. Then, by a slight abuse of notation, define X_j to be the number of processors that choose value j, so $X_j = \sum_{i=1}^{n} X_{i,j}$ and $E(X_j) = 1/n$. Therefore, since, once j is fixed, all the $X_{i,j}$'s are mutually independent,

$$\begin{aligned}
\mathbf{Pr}(X_j > c) &= \mathbf{Pr}(X_j > (1 + (cn-1))(1/n)) \\
&\leq \left[\frac{e^{cn-1}}{(cn)^{cn}}\right]^{1/n} \\
&\leq (e^c/c^c)\frac{1}{n^c} \\
&\leq \frac{1}{n^c}.
\end{aligned}$$

Thus, the probability that X_j is more than c for any j is at most $1/n^{c-2}$. ■

Having presented these preliminary results, let us now give our method for randomized BSP multi-searching.

3 Fully-Scalable BSP Multi-Searching.

Let S be a set of query items distributed evenly across p processors in a bulk-synchronous parallel computer. Also, let T be a binary search tree. The *multi-searching* problem is to determine, for each query q in S, the leaf node in T where a root-to-leaf search in T for q would result. We assume that for any element q in S and any node v in T a *comparison* for q at v yields one the following results:

- **terminate**: the node v is the leaf in T that terminates the search for s_i.
- **child** u: the search procedure for q should proceed to v's child u.
- **incomparable**: the search procedure for q should not visit node v in T.

3.1 A simple partially-scalable solution. Before we give our fully-scalable solution to this multi-searching problem, let us observe that there is a fairly simple BSP method for solving this problem that scalable but not fully-scalable. This simple method begins by stratifying T into subtrees, which we call *packet trees*, by defining every $l = (1/2)\log h$ level in T a distinguished level. A node v on a distinguished level in T defines a packet tree of $O(h^{1/2})$ nodes consisting of descendents of v down to the next distinguished level in T. We assume that these packet trees are distributed evenly across the set of processors, as are the queries for S.

To solve the multi-searching problem we begin with the packet tree t rooted at the root of T. We can apply the generalized broadcast procedure to broadcast this entire packet tree to all the other processors in $O(\log_h p)$ time. Then each processor j performs the comparisons for all the nodes in this packet tree and determines for each query q at j the leaf in t where the search for q should continue. By then performing a generalized parallel prefix we can collect together all the queries that should proceed at the same node in T. This allows us to then repeat this procedure for all those groups in parallel. We can balance the broadcast costs against the queries, so that the total number of communication rounds is $O(\log_h p \log_h n)$, for $h = \lceil n/p \rceil + 1$, where $n = |S| + |T|$. This is $O(1)$ if $n/p \geq n^{\epsilon}$, for some constant $\epsilon > 0$, but it is not optimal for all values of p; hence, it is not fully-scalable.

3.2 Our approach for fully-scalable multi-searching. Our fully-scalable method for efficiently answering the queries in S for all values of p is based upon a recursive strategy for searching T. We first concentrate on routing the searches through the subtree T' consisting of the top-most $n^{1/4}$ nodes in T (i.e., the nodes on the first $(\log n)/4$ levels of T), where $n = |S| + |T|$. Once we have performed all the searches in S through T', we then subdivide the searches to the subtrees rooted at the leaves of T' and recurse on each one (assuming $|T| > |T'|$, of course). Once we have determined all the search paths through T' we can then subdivide the multi-search problem into subproblems of size $\lceil n^{3/4} \rceil$ each through parallel prefix and broadcasting steps that run in $O(\log_h n)$ time. Assuming we can route the searches in S through T' in an expected $O(\log_h n)$ number of rounds, then, with high probability, the expected total number of communication rounds, then, is bounded by the following recurrence relation:

$$\mathcal{T}(n) \leq \mathcal{T}(n^{3/4}) + O(\log_h n),$$

which is $O(\log_h n)$.

Let us therefore concentrate on how to route the searches of S for the case when $|T| = \lceil n^{1/4} \rceil$. Our approach in this case is based upon a randomized two-phase strategy for searching in such a tree T, which is in turn based upon randomized searching techniques of Reif and Sen [39, 42, 43]. This strategy alone is not sufficient, however, to achieve the high-probability

bound in the typical case when $h \geq n^{\epsilon}$, for some constant $\epsilon > 0$. To achieve a high probability bound for all values of h we augment our strategy with a *failure sweeping* technique [25, 32].

In the first phase we build a layered network C from T and in the second phase we route the searches in S through C using a simple BSP packet routing protocol. For each node v in T, let $n(v)$ denote the number of searches in S that pass through v (or terminate at v if v is a leaf). The specific goal of the phase-one computation is to create the network C so that the total number of nodes on each level is $O(n)$, the in- and out-degree of every node is at most h, and such that, for each node $v \in T$, there are at least $\Omega(n(v))$ nodes in C associated with v (whose job it is to process the searches in S that pass through v in T). Intuitively, each node v in C is to process approximately $O(1)$ searches through a corresponding node in T, although our BSP routing strategy will actually allow more than a constant number of searches to pass through v in any round in some cases.

In the beginning of the second phase the queries in S are distributed at most $\lceil n/p \rceil$ per "root" node of C, each of which is associated with a distinct processor. In a generic phase-two step each element of S will be associated with a node in C, which in turn is associated with a distinct processor. The phase-two computation proceeds by then having each processor perform the comparisons for each search element it contains. This will determine, for each element s stored at a processor i, a processor j that i needs to route s to. We develop a protocol, then, so that we do not violate the communication constraints of the BSP model and, with high probability, we complete the entire computation in $O(\log_h n)$ communication steps.

3.3 Phase One: Building the Search Network.

In this subsection we describe a method for constructing the network C that will allow us to perform the searches in T for all elements of S in $O(\log_h n)$ communication rounds. We begin by compressing T into an h-ary tree $\hat{T}$ using the packet-tree stratification technique described in Section 3.1, with $l = \log h$. Thus, each node v in $\hat{T}$ is associated with an h-node subtree in T, with each leaf in this subtree corresponding to the roots of the subtrees associated with v's children in $\hat{T}$.

We construct a circuit C that will allow us to process the searches in S through $\hat{T}$, then, as follows:

1. We choose a random sample $S' \subseteq S$ of size $\lceil n^{1/2} \rceil$. For each node v in $\hat{T}$ we then determine $n'(v)$, the number of searches in S' that pass through or terminate at v, by a "brute force" quadratic comparison (which requires at most $O(n^{3/4})$ comparisons in total). We then let $\hat{n}(v) = n'(v)|S(v)|/|S'(v)|$, which we will use as an estimate for $n(v)$. This step takes $O(\log_h n)$ communication steps and $O(n/p)$ internal computation time.

2. If $|S| \leq \lceil n^{1/2} \rceil$, then we are done. So, let us now assume that $|S| > \lceil n^{1/2} \rceil$. We recursively define a replication parameter $r(v)$ for each node v in $\hat{T}$. We initially define $r(root(\hat{T})) = \tau(\alpha n)$, where $\tau(x)$ denotes the smallest power of 2 larger than x (i.e., $\tau(x) = 2^{\lceil \log x \rceil}$), and $\alpha \geq 4$ is a constant, called the *dilution parameter*, which we set in the analysis. Then, for each non-root node v in $\hat{T}$, with parent w, we define

$$r(v) = \tau(\max\{\alpha \hat{n}(v), r(w)/h\}).$$

   Note that α determines that there will be excess capacity for sending elements from w to v. This step can easily be implemented in $O(\log_h n)$ communication rounds.

3. For each non-root node v in $\hat{T}$, with parent w, we create a set $C(v)$ of $r(v)$ copies of v and we connect each copy of node v to $r(w)/r(v)$ distinct copies of node w (in $C(w)$). If this ratio is not integral, then we approximate this as best as possible, connecting each copy of node v to either $\lfloor r(w)/r(v) \rfloor$ or $\lceil r(w)/r(v) \rceil$ copies of node w. We refer to these added edges as the *down* edges in C. This step takes $O(\log_h n)$ communication steps and $O(n/p)$ internal computation time.

This completes the construction of the network C, and gives us the following:

Lemma 3.1: *The above computation creates a layered network C such that the in- and out-degree of the down edges for any node is at most h. Moreover, with probability at least $1 - 1/e^{n^c}$ (for some fixed constant $c > 0$), for each node $v \in \hat{T}$, with parent w in $\hat{T}$, there are $r(v)$ nodes in $C(v)$, where $\max\{n(v)/2, r(w)/h\} \leq r(v) \leq \max\{3n(v), r(w)/h\}$.*

Proof: Let v be a node with parent w in $\hat{T}$. The bound on the in- and out-degree of v follows immediately from the recursive definition of the $r(v)$'s. Let us therefore consider the probability that the size bound $r(v)$ is too far off the mark, beginning with the probability that it significantly exceeds the upper bound, which we quantify as $r(v) > \max\{3n(v), r(w)/h\}$.

Since we defined $r(v) = \max\{\hat{n}(v), r(w)/h\}$, this can only be the case if $\hat{n}(v) > 3n(v)$ and $3n(v) > r(w)/h$. Thus, if we let $A(v)$ be the event that $r(v)$ is above the bound, and we let $C(v)$ be the event that $3n(v) > r(w)/h$, then, by a Chernoff bound analysis ([35], p. 72), we can show the following:

$$\begin{aligned}
\mathbf{Pr}(A(v)) &= \mathbf{Pr}(\hat{n}(v) > 3n(v) \mid C(v)) \\
&= \mathbf{Pr}(n'(v) > \frac{3n(v)}{n^{1/2}} \mid C(v)) \\
&\leq c^{-3n(v)/n^{1/2}} \\
&\leq c^{-r(w)/n^{1/2}h} \\
&\leq c^{-n^{1/4}},
\end{aligned}$$

where $c = 3/e$, since $r(w)/h \geq n^{3/4}$. Likewise, let $B(v)$ be the event that $r(v)$ is significantly less than its desired amount, which we quantify as the condition $r(v) < \max\{n(v)/2, r(w)/h\}$. Also, let $D(v)$ be the event that $n(v)/2 > r(w)/h$. Then, by another Chernoff bound ([35], p. 70), we can show the following:

$$\begin{aligned}
\mathbf{Pr}(B(v)) &= \mathbf{Pr}(\hat{n}(v)/p < n(v)/2 \mid D(v)) \\
&= \mathbf{Pr}(n'(v) > n(v)/2n^{1/2} \mid D(v)) \\
&\leq e^{-n(v)/8n^{1/2}} \\
&\leq e^{-r(v)/4n^{1/2}h} \\
&\leq e^{-n^{1/4}/4}.
\end{aligned}$$

Combining these two bounds establishes the lemma. ■

Thus, with very high probability, we correctly construct the network C.

3.4 The Phase Two Computation: Routing the Searches. Let us therefore next consider the problem of routing the searches of S through C. For $i = 1$ to $\lceil \log_h n \rceil$, and each v on level i of $\hat{T}$, we assign h contiguous nodes of $C(v)$ to a separate processor (so that at most $h\lceil \log_h |\hat{T}| \rceil$ nodes are assigned to each processor in total). We do not assign different $C(v)$ lists to the same processor, however. We assume that, for any node v in C, a processor i can determine in $O(1)$ time (without communication) the processor j that is associated with v. Moreover, since we assign each h continuous nodes on level i to a separate processor, there are only $O(1)$ processors holding different "in" neighbors (on level $i-1$) of down edges for a node u in a $C(v)$. Thus, for any processor P_i, the number of other processors holding nodes adjacent to nodes of C stored at P_i is $O(h)$.

Initially, all the elements of S are stored at most one per node in $C(v)$, and every h nodes of $C(v)$ are in turn stored in a unique processor, where $v = root(\hat{T})$. Before we attempt to route the searches through C we first apply a random permutation to the contents of the nodes of $C(v)$, using the method of Section 2.3. Each processor i then performs the following *transfer-step* computation:

1. Processor i determines which of the h nodes of $C(v)$ it stores actually contain search elements in S. The processor i then performs the comparison associated with $v \in \hat{T}$ (which is actually a search through a $\lceil \log h \rceil$-height subtree of T associated with v) for all the elements in S currently at $C(v)$ and stored in processor i's internal memory. Each such comparison determines a child u of v in $\hat{T}$ where a search in S should proceed.

2. Each search at $C(v)$ that wishes to go to u has at most 2 processors that it needs to be routed to (storing nodes of $C(u)$). For each child u of v in $\hat{T}$, and each processor j storing nodes of $C(u)$ reachable from the nodes stored at i, processor i determines $n_{i,j}(u)$, the number of searches currently at i in $C(v)$ that need to proceed to a node of $C(u)$ in processor j.

3. Processor i sends a message to each processor j such that $n_{i,j}(u) > 0$ informing processor j of the value of $n_{i,j}(u)$.

4. Processor j receives at most h messages, and adds up all the values it receives, producing a sum n_j. Processor j then sends back a message to each processor i informing it that it can then send $\lceil n_{i,j}(u) \min\{1, h/n_j\} \rceil$ of its searches that need to be routed to j. (So if $n_j \leq h$, then all of the elements of $n_{i,j}(u)$ can be routed to j.)

5. Processor i sends to processor j a total of $\lceil n_{i,j}(u) \min\{1, h/n_j\} \rceil$ of its searches that need to be routed to j.

This transfer-step computation can be performed in $O(1)$ communication rounds, as described above. We continue repeating this computation for $b\lceil \log_h n \rceil$ iterations, where $b \geq 1$ is a constant to be determined in the analysis. At that time all of the searches are expected to be at the leaf-level of C (and hence $\hat{T}$). At this point we check to make sure that all searches have indeed reached their final destinations in C and that no searches have been "left behind" anywhere. This condition can easily be tested in $O(\log_h n)$ communication rounds. If we have any such incomplete searches, then we repeat the entire computation described above. Otherwise, we have completed all the searches of S in the tree $\hat{T}$. The

next lemma establishes the probability that all the searches in S can be routed through C in $O(\log_h n)$ communication rounds (which, of course, is what we desire).

Lemma 3.2: *All the searches in S can be routed through C in $(c+1)\lceil \log_h n \rceil$ communication rounds with probability at least $1 - 1/n^{ch/4\log h}$, for any constant $c \geq 2$.*

Proof: Our proof is an adaptation of arguments used to justify hypercube packet routing strategies [35] to our BSP protocol on the search network C. Let $\hat{C}$ denote the compression of C implicitly defined by the assignment of nodes in C to processors. That is, let each node $\hat{v}$ in $\hat{C}$ correspond to h nodes in a $C(v)$ that were all assigned to the same processor. Note that, even with this compression, once two search paths separate in $\hat{C}$ they do not rejoin. Fix a particular search item $s_i \in S$ and let $\rho_i = (e_1, e_2, \ldots, e_k)$ denote the search path in $\hat{C}$ for s_i, where $k = \lfloor (\log_h n)/4 \rfloor$. Let d_i denote the total number of rounds that s_i is delayed during its routing through $\hat{C}$. Further let S_i denote the set of all searches in S whose search path passes through at least one of the edges in ρ_i. Because of our BSP protocol for routing searches through $\hat{C}$, $d_i \leq |S_i|/h$. Let $H_{i,j}$ denote the indicator random variable that is 1 if and only if the path for search s_j passes through at least one edge in ρ_i, and is 0 otherwise. Thus, $|S_i| = \sum_{j=1}^{n} H_{i,j}$. For each edge e_l in ρ_i, let $N(e_l)$ denote the number of searches in S_i that pass through e_l. Of course,

$$\sum_{j=1}^{n} H_{i,j} \leq \sum_{l=1}^{k} N(e_l);$$

hence,

$$\mathbf{E}\left[\sum_{j=1}^{n} H_{i,j}\right] \leq \sum_{l=1}^{k} \mathbf{E}[N(e_l)].$$

Moreover, if C was constructed correctly, then $\mathbf{E}[N(e_l)] \leq h/\alpha \leq h/4$ for each e_l in ρ_i, because of our initial random permutation step in the routing algorithm. This implies that

$$\mathbf{E}\left[\sum_{j=1}^{n} H_{i,j}\right] \leq \frac{kh}{4}.$$

Thus, we can apply a Chernoff bound (e.g., see [35], p. 72) to derive the following bound:

$$\begin{aligned}
\mathbf{Pr}(d_i > c\log_h n) &\leq \mathbf{Pr}(|S_i| > ch\log_h n) \\
&\leq \mathbf{Pr}\left(\sum_{j=1}^{n} H_{i,j} > 4c(kh/4)\right) \\
&\leq 2^{-ckh} \\
&\leq 2^{-ch(\log_h n)/4} = n^{-ch/4\log_2 h},
\end{aligned}$$

provided $c \geq 2$. This establishes the lemma. ■

Even though this lemma implies a high probability bound for routing the searches in S through $\hat{T}$, it is not sufficient to imply a high probability bound for our entire computation. Since it depends upon the size of the problem being solved, the probability of Lemma 3.2 degrades as we recursively solve searches using the approach of Section 3.2. In the end it only implies that the running time of routing the searches in S is *expected* to be $O(\log_h n)$. Fortunately, there is a simple way to boost this probability back to a high probability bound.

3.5 Improving the Success Probability via Failure Sweeping. In the full version we show how to apply a generalized version of the failure sweeping paradigm [25, 32] to improve the probability of success for routing all the searches in S through T in $O(\log_h n)$ communication rounds to be at least $1 - 1/n^c$ for any constant $c \geq 1$. The main idea behind this technique is to terminate recursive calls that go too long, and then replicate each of these "unlucky" subproblems at least $O(\log n)$ times and apply our expected-time computation on each of these subproblems (but now without any failure sweeping in the recursive calls). We can show inductively, that the number of such subproblems is small with high probability; hence, we will have enough resources to solve all the replicated subproblems simultaneously. Since one of the copies of each subproblem returns after $O(\log_h n)$ communication rounds, with high probability, we can establish the following:

Theorem 3.1: *Given an balanced binary search tree T, and a set S of searches defined for T, one can simultaneously perform the searches in S on T in a BSP computer using $O(\log_h n)$ communication rounds, with probability $1 - 1/n^c$ for any constant $c \geq 1$, where $n = |S| + |T|$ and $h = \lceil n/p \rceil + 1$. The combined running time is $O((n\log n)/p + (L + g(n/p))\log_h n)$, with probability $1 - 1/n^c$ for any constant $c \geq 1$.*

There are a number of immediate applications of this problem to problems in computational geometry, such as searching in arrangements, 2D-all-nearest-neighbor searching, and 3D-maxima, which we explore in the full version of this paper. We describe here

a novel application to the 3-dimensional convex hull problem.

4 BSP Convex Hull Construction.

Let S be a set of n points in $\mathbb{R}^3$. The *convex hull* of S is the polytope defined by the smallest convex set containing all the points of S. The convex hull problem, then, is to construct a representation of this polytope. In this section we show how to construct the convex hull of S in the BSP model.

4.1 2-dimensional convex hull construction. We begin by giving a deterministic algorithm for 2-dimensional convex hulls that uses $O(\log_h n)$ communication rounds and combined running time of $O((n \log n)/p + (L + gh) \log_h n)$, for $h = \lceil n/p \rceil + 1$. Our method is a BSP adaptation of the EREW PRAM algorithm of Miller and Stout [34].

We begin by sorting the input points by their x-coordinates. This can be done in $O(\log_h n)$ communication rounds and combined running time of $O((n \log n)/p + (L + gh) \log_h n)$, using the BSP sorting algorithm of the author [26]. Without loss of generality, we concentrate on the problem of computing an upper hull, i.e., those edges whose normals have positive second components. We proceed as follows:

1. If all the input points are contained on a single processor, compute the upper hull using any efficient sequential method (e.g., see [22, 36, 38]). Let us therefore assume for the remainder of this algorithm that $n > h$.

2. Divide the input into $O(n^{1/4})$ contiguous groups of size $O(n^{3/4})$ each, and recursively find the upper hull of each set.

3. Atallah and Goodrich [10] and Dadoun and Kirkpatrick [16] describe CREW PRAM methods for finding upper common tangents between two upper hulls in $O(1)$ time using $O(n^\epsilon)$ processors, for any constant $\epsilon > 0$. Let us apply a straightforward BSP simulation of one of these methods to find the common upper tangents between each pair of upper tangents. The total number of communication rounds is $O(\log_h n)$ to implement this simulation.

4. For each group i compute the maximum-slope tangent line t_r to groups $j > i$ and the minimum-slope tangent line t_l to groups $j < i$. If these two tangents cross, then no points of hull i are on the upper hull. Otherwise, all the vertices (inclusive) on the upper hull i between the tangent points for t_l and t_r, respectively, are on the upper hull.

5. Perform a parallel prefix computation to compress together all the points on the upper hull.

After the preprocessing sorting step this method will find the upper hull of the input set of points in a number of communication rounds bounded by the recurrence relation

$$\mathcal{T}(n) = \mathcal{T}(n^{3/4}) + O(\log_h n),$$

which implies that $\mathcal{T}(n)$ is $O(\log_h n)$. Thus, we can compute the convex hull of n points in the plane in $O(\log_h n)$ communication rounds and $O((n \log n)/p)$ internal computation time on a p-processor BSP computer.

4.2 3-dimensional convex hull construction. Our method for 3-dimensional convex hulls is based upon using our multi-searching method to adapt the EREW PRAM algorithm of Amato *et al.* [4] to the BSP model. An outline of our algorithm is as follows:

1. Dualize the points in S to n planes in $\mathbb{R}^3$, thereby converting the convex hull problem to that of determining the intersection polytope P of n halfspaces determined by these planes and the origin.

2. Select a random sample $S' \subset S$ of size n^ϵ of the halfspaces and construct their intersection polytope P' by "brute force," where $\epsilon > 0$ is a suitably-small constant.

3. Triangulate the faces of P' and form a triangular "cones" for each using the origin as apex (thereby constructing a simplicial cell complex Chazelle refers to as the *geode* [14]).

4. Construct a search tree T for this geode such that each leaf of T identifies for a plane h all the cells of the geode that h crosses.

5. Perform the multi-search of T using all the planes dual to points in S as queries.

6. For each tetrahedron τ in the geode, find the 2-dimensional contour of the intersection between the boundary of τ and the final intersection polytope P using our 2-dimensional convex hull algorithm.

7. Use the "pruning" strategy of Amato *et al.* [4] to eliminate from each subproblem determined by a tetrahedron τ those halfspaces that cannot contribute any vertex to P inside τ, using the 2-dimensional contours on the boundary of τ. This also reduces the total problem size to be $O(n)$.

8. Recurse on each tetrahedron τ in the geode.

In the full version we describe how to implement each of the above steps in $O(\log_h n)$ communication rounds, with high probability. This implies that the expected running time of the algorithm satisfies the recurrence equation

$$\mathcal{T}(n) \leq \mathcal{T}(n^{1-\epsilon}) + O(\log_h n),$$

which implies that the expected value of $\mathcal{T}(n)$ is $O(\log_h n)$. Moreover, in the full version we show that we can again apply failure sweeping to this expectation to derive the following theorem:

Theorem 4.1: *Given a set S of n points in $\mathbb{R}^3$, one can construct the convex hull of S in $O(\log_h n)$ communication rounds and combined expected running time of $O((n \log n)/p + (L + g(n/p)) \log_h n)$ in the BSP model, with probability $1 - 1/n^c$ for any constant $c \geq 1$, for $h = \lceil n/p \rceil + 1$.*

Incidentally, when $p = n/2$ this result implies the first $O(\log n)$-time optimal-work EREW PRAM method for 3-dimensional convex hulls, which, with high probability, improves the time bounds, work bounds, or model assumptions of several previous methods [4, 5, 39, 42].

5 Conclusion.

We have given a general algorithm for multi-searching in the BSP framework and given examples of how this method can be used to derive fully-scalable work-optimal parallel methods for several computational geometry problems, including 3-dimensional convex hull construction. Our framework is based upon satisfying a set of searches defined for a binary search tree T. There are a number of additional applications in parallel computational geometry that depend upon multi-searching directed acyclic search graphs (e.g., see [8, 23]). Thus, a possible direction for future work would be to extend our results to search dags.

Acknowledgements. We would like to thank Mikhail Atallah for several helpful comments regarding the multi-searching problem, and we would also like to thank Sandeep Sen for several helpful e-mail comments concerning this problem as well.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

[2] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3:293–327, 1988.

[3] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry*. Prentice-Hall, 1993.

[4] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 683–694, 1994.

[5] N. M. Amato and F. P. Preparata. The parallel 3D convex hull problem revisited. *Internat. J. Comput. Geom. Appl.*, 2(2):163–173, 1992.

[6] M. J. Atallah. Parallel techniques for computational geometry. *Proc. IEEE*, 80(9):1435–1448, Sept. 1992.

[7] M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading divide-and-conquer: A technique for designing parallel algorithms. *SIAM J. Comput.*, 18:499–532, 1989.

[8] M. J. Atallah, F. Dehne, R. Miller, A. Rau-Chaplin, and J.-J. Tsay. Multisearch techniques for implementing data structures on a mesh-connected computer. In *Proc. ACM Sympos. Parallel Algorithms Architect. (SPAA)*, pages 204–214, 1991.

[9] M. J. Atallah and A. Fabri. On the multisearching problem for hypercubes. Research Report 1990, INRIA, BP93, 06902 Sophia-Antipolis, France, June 1993.

[10] M. J. Atallah and M. T. Goodrich. Parallel algorithms for some functions of two convex polygons. *Algorithmica*, 3:535–548, 1988.

[11] A. Bäumker, W. Dittrich, and F. Meyer auf der Heide. Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model. In *Proc. 3rd European Symposium on Algorithms (ESA)*, pages 17–30, 1995.

[12] A. Bäumker, W. Dittrich, and A. Pietracaprina. The deterministic complexity of parallel multisearch. In *Proc. 1996 Scadanavian Workshop on Algorithmic Theory*, page to appear, 1996.

[13] G. Bilardi and F. P. Preparata. Lower bounds to processor-time tradeoffs under bounded-speed message propagation. In *Proc. 4th International Workshop on Algorithms and Data Structures (WADS), LNCS 955*, pages 1–12. Springer-Verlag, 1995.

[14] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10:377–409, 1993.

[15] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. 4th ACM SIGPLAN Symp. on Princ. and Practice of Parallel Programming*, pages 1–12, 1993.

[16] N. Dadoun and D. G. Kirkpatrick. Optimal parallel algorithms for convex polygon separation. Technical Report 89-21, Dept. of Computer Science, Univ. of British Columbia, 1989.

[17] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A.

Khokhar. A randomized parallel 3D convex hull algorithm for course grained multicomputers. In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 27–33, 1995.

[18] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 298–307, 1993.

[19] F. Dehne, C. Kenyon, and A. Fabri. Scalable and architecture independent parallel geometric algorithms with high probability optimal time. In *Proc. 6th IEEE Symp. on Parallel and Distributed Processing (SPDP)*, pages 586–593, 1994.

[20] X. Deng. A convex hull algorithm on course-grained multicomputer. In *Proc. 5th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 94)*, pages 634–642, 1994.

[21] O. Devillers and A. Fabri. Scalable algorithms for bichromatic line segment intersection problems on coarse grained multicomputers. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 277–288, 1993.

[22] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[23] A. Gerbessiotis and C. Siniolakis. Communication efficient data structures on the bsp model with applications in computational geometry. In *Proceedings of EUROPAR'96*, August 1996.

[24] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *J. of Parallel and Distributed Computing*, 22:251–267, 1994.

[25] M. Ghouse and M. T. Goodrich. In-place techniques for parallel convex hull algorithms. In *Proc. 3rd ACM Sympos. Parallel Algorithms Architect.*, pages 192–203, 1991.

[26] M. T. Goodrich. Communication-efficient parallel sorting. Technical Report, Dept. of Computer Science, Johns Hopkins Univeristy, 1995.

[27] T. Hagerup. Fast parallel generation of random permutations. In *Annual International Colloquium on Automata, Languages and Programming, LNCS 510*, pages 405–416. Springer-Verlag, 1991.

[28] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, Mass., 1992.

[29] R. M. Karp, A. Sahay, E. Santos, and K. E. Schauser. Optimal broadcast and summation in the LogP model. In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, pages 142–153, 1993.

[30] C. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71:95–132, 1990.

[31] Y. Mansour, N. Nisan, and U. Vishkin. Trade-offs between communication throughput and parallel time. In *Proc. 26th ACM Symposium on Theory of Computing (STOC)*, pages 372–381, 1994.

[32] Y. Matias and U. Vishkin. Converting high probability into nearly-constant time—with applications to parallel hashing. In *23rd ACM Symp. on Theory of Computing*, pages 307–316, 1991.

[33] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 9(1):29–59, 1984.

[34] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. *IEEE Trans. Comput.*, C-37(12):1605–1618, 1988.

[35] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[36] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[37] C. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *Proc. 20th ACM Symp. Theory Comp. (STOC)*, pages 510–513, 1988.

[38] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[39] J. Reif and S. Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM J. Comput.*, 21(3):466–485, 1992.

[40] J. H. Reif. An optimal parallel algorithm for integer sorting. In *Proc. 26th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 496–504, 1985.

[41] J. H. Reif. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[42] J. H. Reif and S. Sen. Erratum: Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM J. Computing*, 23(2):447–448, 1994.

[43] J. H. Reif and S. Sen. Randomized algorithms for binary search and load balancing on fixed connection networks with geometric applications. *SIAM J. Computing*, 23(3):633–651, 1994.

[44] L. G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, 1990.

[45] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 943–972. Elsevier/The MIT Press, Amsterdam, 1990.