Alcom-FT Technical Report Series
ALCOMFT-TR-01-14

Parallel convex hull computation by generalised regular sampling

Alexandre Tiskin *

Department of Computer Science
University of Warwick, Coventry CV4 7AL, UK
email: tiskin@dcs.warwick.ac.uk

Abstract

The model of bulk-synchronous parallel (BSP) computation is an emerging paradigm of general-purpose parallel computing. We propose the first optimal deterministic BSP algorithm for computing the convex hull of a set of points in three-dimensional Euclidean space. Our algorithm is based on known fundamental results from combinatorial geometry, concerning small-sized, efficiently constructible ϵ-*nets* and ϵ-*approximations* of a given point set. The algorithm generalises the technique of *regular sampling*, used previously for sorting and two-dimensional convex hull computation. The BSP cost of the algorithm is optimal only for extremely large inputs, unless the number of processors is very small. However, in future it might be possible to extend the algorithm to deal with smaller inputs.

1 Introduction

The model of *bulk-synchronous parallel (BSP) computation* (see [18, 10, 11]) provides a simple and practical framework for general-purpose parallel computing. Its main goal is to support the creation of architecture-independent and scalable parallel software. Key features of BSP are its treatment of the communication medium as an abstract fully connected network, and strict separation of all interaction between processors into point-to-point asynchronous data communication and barrier synchronisation. This separation allows an explicit and independent cost analysis of local computation, communication and synchronisation. A slightly restricted version of the BSP model is known in literature as *CGM (coarse grained multicomputer)*.

In this work we propose a new deterministic BSP algorithm for computing the convex hull of a set of points in two- and three-dimensional

*Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

Euclidean space. This is a classic problem of computational geometry, extensively studied in the context of both sequential and parallel computation (see e.g. [15, 5]).

Consider computing the convex hull of n points in 2D or 3D. The lower bound on sequential computation time is $\Omega(n \log n)$ in both cases. This lower bound is provided by reduction of the sorting problem, and is attained by many existing $O(n \log n)$-time algorithms (see [15] for references). A parallel (EREW PRAM), processor-optimal, deterministic 2D convex hull algorithm with time complexity $O(\log n)$ was given in [12]. For 3D convex hulls, papers [1, 6] describe respectively a deterministic $O(\alpha^{-1} \log n)$-time CREW PRAM algorithm using $O(n^{1+\alpha})$ processors, and a randomised $O(\log n)$-time processor-optimal EREW PRAM algorithm.

For the BSP/CGM model, optimal deterministic 2D convex hull algorithms are given in [6, 3]. Assuming that the input size n is sufficiently large with respect to the number of processors p, these algorithms perform $O(n \log n/p)$ computation work, $O(n/p)$ communication work and $O(1)$ synchronisation steps. For 3D convex hulls, papers [6, 2] describe randomised BSP/CGM algorithms, achieving the same optimal cost values with high probability. To the author's knowledge, no efficient deterministic BSP/CGM algorithms for 3D convex hull computation have been known previously.

This work attempts to fill this gap by presenting an optimal deterministic BSP algorithm for convex hull computation. The algorithm works both for 2D and 3D, and is based on known fundamental results from combinatorial geometry, concerning small-sized, efficiently constructible *ϵ-nets* and *ϵ-approximations* of a given point set. The algorithm generalises the technique of *regular sampling*, used in [16] for sorting, and in [3] for two-dimensional convex hull computation.

The rest of this work is organised as follows. Section 2 gives a brief introduction to the BSP computation model; this section may be skipped by a reader already familiar with the model. Section 3 describes previously known regular sampling algorithms for sorting and 2D convex hull computation, adapted to the BSP model. Section 4 introduces the necessary concepts from combinatorial geometry, describes the proposed generalisation of the regular sampling technique, and presents the new BSP convex hull algorithm. Section 5 draws conclusions and identifies directions for future research.

2 The BSP model

A *BSP computer*, introduced in [18], consists of p *processors* connected by a communication network. Each processor has a fast *local memory*. The processors may follow different threads of computation. A BSP computation is a sequence of *supersteps*. A superstep consists of an *input phase*, a *local computation phase* and an *output phase*. In the input phase, a processor receives data that were sent to it in the previous superstep; in the output phase, it can send data to other processors, to be received in the next superstep. The processors are synchronised between supersteps. The computation within a superstep is asynchronous.

Let *cost unit* be the cost of performing a basic arithmetic operation or a local memory access. If, for a particular superstep, w is the maximum number of local operations performed by each processor, h' (respectively, h'') is the maximum number of data units received (respectively, sent) by each processor, and $h = h' + h''$ (another possible definition is $h = \max(h', h'')$), then the cost of the superstep is defined as $w + h \cdot g + l$. Here g and l are the BSP parameters of the computer. The value g is the *communication throughput ratio* (also called "bandwidth inefficiency" or "gap"), the value l is the *communication latency* (also called "synchronisation periodicity"). If a computation consists of S supersteps with costs $w_s + h_s \cdot g + l$, $1 \leq s \leq S$, then its total cost is $W + H \cdot g + S \cdot l$, where $W = \sum_{s=1}^{S} w_s$ is the local computation cost, $H = \sum_{s=1}^{S} h_s$ is the communication cost, and S is the synchronisation cost. The values of W, H and S typically depend on the number of processors p and on the problem size.

Many BSP algorithms are only defined for input sizes that are sufficiently large with respect to the number of processors. This requirement is loosely referred to as *slackness*. In this work, we are not concerned with reducing the slackness required by BSP algorithms: our goal is to obtain a simple, optimal BSP algorithm with arbitrary (polynomial) slackness.

For the sake of simplicity, we ignore small irregularities that arise from imperfect matching of integer parameters. For example, when we write "divide an array of size n equally across p processors", the value n may not be an exact multiple of p, and therefore the shares may differ in size by ± 1.

3 Regular sampling

Sorting is a classical problem of both sequential and parallel computing. Probably the simplest parallel sorting algorithm is *parallel sorting by regular sampling (PSRS)*, proposed in [16]. We consider comparison-based sorting of an array of size n, with each processor holding n/p elements of the array. Without loss of generality, we may assume that all elements of the array are distinct (otherwise, we distinguish equal elements by attaching to each a unique tag).

The PSRS algorithm proceeds as follows. First, the array is partitioned into p subarrays, each of size n/p. The subarrays are sorted independently by an optimal sequential algorithm. The problem now consists in merging the p sorted subarrays.

In the first stage of merging, $p + 1$ regularly spaced *primary samples* are

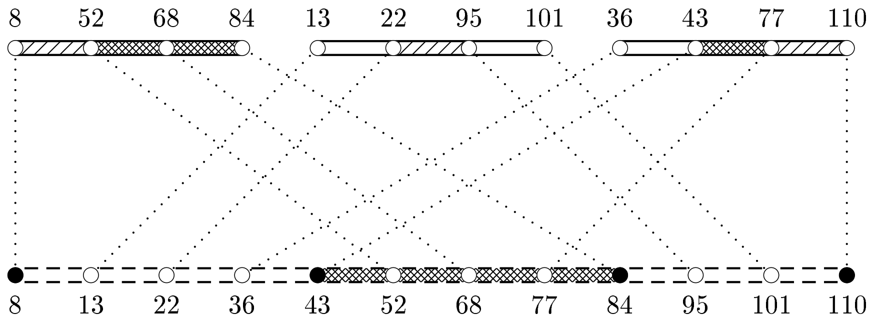

Figure 1: Parallel sorting by regular sampling

selected from each subarray (the first and the last elements of a subarray are included among the samples). The samples divide each subarray into p *primary blocks* of size at most n/p^2. Then, $p \cdot (p+1)$ primary samples are collected together and sorted by an arbitrary sequential algorithm. After that, we select $p+1$ regularly spaced *secondary samples* from the sorted array of primary samples (the first and the last elements are again included in the samples). The secondary samples partition the set of all elements into p *secondary blocks*. Each secondary block is distributed across the processors. It is easy to show that the size of a secondary block is at most $3n/p$. Now it remains to broadcast the secondary samples, to assign each secondary block to a particular processor, and then to sort the elements of this block by an efficient sequential algorithm.

The method is illustrated in Figure 1 for $p = 3$. The state of the array after local sorting of the subarrays is represented by three horizontal bars at the top. Primary samples are shown as white dots. Dotted lines show the rearrangement of primary samples into a sorted array at the bottom; note that the order of primary samples from each subarray is preserved, but the samples from different subarrays may be interleaved. The dashed bars at the bottom show the elements of the array assumed to lie between the samples; the number of elements between adjacent primary samples is not necessarily equal. Black dots indicate the secondary samples. The hatching of primary blocks relates to the proof of the $3n/p$ bound on the size of a secondary block (see [17] for details).

The sorting algorithm based on regular sampling can be easily implemented in the BSP model. The computation cost of the algorithm is $O(n \log n/p)$. In the worst case, a processor sends and receives at most $3n/p$ values, therefore the communication cost is $O(n/p)$. The algorithm performs $O(1)$ global synchronisations. The total BSP cost of the algorithm is

$$W = O(n \log n/p) \qquad H = O(n/p) \qquad S = O(1)$$

For the above resource bounds to hold, it is necessary that the parallel part of the algorithm dominate the sequential part. In particular, we need

$n/p \geq p^2$, therefore $n \geq p^3$.

The method of regular sampling is relying on the principle of representing the processor's local data by a set of regular primary samples, the number of which does not depend on the size of the input. From the union of all primary samples we can select a subset of regular secondary samples, which represent the global data structure, and can be used for efficient data repartitioning intended to reflect this global structure.

We will return to this principle of two-level regular sampling in the next section. Here, for the sake of comparison, we describe a known algorithm for 2D convex hull computation. The algorithm is based on an idea similar to PSRS, but has a somewhat different structure. The rest of this section may be skipped at first reading.

We consider the problem of computing the convex hull for an array of n points in 2D. Initially, every processor holds n/p points of the array. Without loss of generality, we may assume that the points are in general position (otherwise, we achieve this by a slight perturbation of the points). By symmetry, we only need to compute the upper hull (i.e. the part of the convex hull visible from $(0, +\infty)$, assuming that the edges of the hull are opaque).

The algorithm, which is a simplified version of the algorithm from [3], proceeds as follows. Initially, the array is sorted by x-coordinate with an optimal BSP algorithm, such as PSRS. Each processor now contains a sorted subarray of points of size n/p. Independently from each other, the processors compute upper hulls of their subarrays by an optimal sequential algorithm. The problem now consists in merging the p local hulls.

In the first stage of merging, $p+1$ regularly spaced (primary) samples are selected from each local hull (the first and the last elements of a local hull are included among the samples). The samples divide each local hull into p (primary) blocks of size at most n/p^2. Then, $p \cdot (p+1)$ samples are collected together, and their upper hull is found by an arbitrary sequential algorithm. This global sample hull is broadcast to all processors.

Upon receipt of the global sample hull, each processor compares it with the local sample hull. Two cases are possible.

Case 1: the local and the global sample hulls are disjoint. This means that the local sample hull lies completely below a single edge of the global sample hull. We consider the local sample closest to this edge, and call it *the critical sample.*

Case 2: the local and the global sample hulls have a non-empty intersection. This means that a consecutive set of local samples lies on the global hull. We consider the initial and the final samples in this set, and call them *critical samples.* In other words, the critical samples are the ones at which the global sample hull detaches from the local sample hull. The two critical samples may coincide, if the global sample hull touches the local sample hull in exactly one sample point.

Consider all local (primary) blocks of a particular processor. Let us call a block *critical*, if at least one of its ends is a critical sample; *inner*, if it lies between two critical samples neither of which is its end; and *outer*, if it is neither inner nor critical. It is easy to show that every outer block lies strictly below the global upper hull, and therefore may be discarded. It is also easy to show that every inner block lies strictly on the global upper hull, and therefore may be included en masse in the final output.

It remains to process the critical blocks. There are at most four of them in each processor, therefore the total size of all critical blocks is at most $4p \cdot n/p^2 = 4n/p$. We collect the elements of all critical blocks in a single processor, and compute the upper hull of all critical blocks by an efficient sequential algorithm. The inner blocks and the upper hull of the critical blocks, taken together, comprise the upper hull of the input array.

When implemented in the BSP model, the above algorithm has computation cost $O(n \log n/p)$. In the worst case, $4n/p$ points are collected in a single processor, therefore the communication cost is $O(n/p)$. The algorithm performs $O(1)$ global synchronisations. The total BSP cost of the algorithm is

$$W = O(n \log n/p) \qquad H = O(n/p) \qquad S = O(1)$$

For the above resource bounds to hold, the values of n and p must satisfy the same slackness condition as in the PSRS algorithm: we need $n/p \geq p^2$, therefore $n \geq p^3$.

Initial pre-sorting of points is a crucial part of the described 2D convex hull algorithm. Therefore, it is hard to modify it for computing convex hulls in higher dimensions, where pre-sorting does not help. In the next section we develop an alternative, more general approach to geometric regular sampling.

4 Generalised regular sampling

This section presents a simple, efficient BSP algorithm for the convex hull problem both in 2D and 3D. To apply the regular sampling technique, we need to find a way of representing an arbitrary, unsorted set of points in 2D or 3D by a small-sized sample. Such a representation is provided by the concept of *ϵ-approximation* (see e.g. [13, 7, 8]).

Let X be a finite set of points in 2D (respectively, 3D). Take any ϵ, $0 \leq \epsilon \leq 1$. A subset $A \subseteq X$ is *an ϵ-approximation to X* with respect to halfplanes (respectively, halfspaces), if for any halfplane (respectively, halfspace) H, we have

$$\left| \frac{|A \cap H|}{|A|} - \frac{|X \cap H|}{|X|} \right| \leq \epsilon,$$

that is, the relative number of points of A in H approximates the relative number of points of X in H with accuracy ϵ. The above definition is a special case of a general definition from VC-dimension theory. Since we only consider this special type of ϵ-approximation, from now on we will omit the words "with respect to halfplanes (halfspaces)".

The following theorem is a corollary of a more general result on ϵ-approximation with respect to simplices, proved in [8].

Theorem 1. *Let X be a set of n points in 2D (respectively, 3D). For any $\delta > 0$, and for any r, $1 \leq r < 1/d - \delta$, an $1/r$-approximation to X of size $O(r^d)$ can be computed deterministically in time $O(n \log r)$, where $d = 3$ (respectively, 4).*

The general VC-dimension theory guarantees the existence of smaller ϵ-approximations, but they are harder to compute. The analogue of Theorem 1 holds for any dimension $d \geq 2$.

In our BSP convex hull algorithm, a $1/p$-approximation to a local point set will play the role of a primary sample. Note that for a 2D array of points of size n in the convex position, a regular sample containing every n/p-th point is an an $1/p$-approximation of size p. However, we will have to use larger approximations provided by Theorem 1, in order to deal with non-convex or three-dimensional sets.

The relatively large size of an approximation guaranteed by Theorem 1 prevents us from using ϵ-approximations as a secondary sample, since such a sample would not provide an efficient problem partitioning. Fortunately, the approximation properties required from a secondary sample are weaker compared to a primary sample. Therefore, we can trade approximation accuracy for sample size. A suitable concept here is that of *ϵ-net* (see e.g. [13, 7, 8]). As before, let X be a finite set of points in 2D (3D), and ϵ an arbitrary real number, $0 \leq \epsilon \leq 1$. A subset $A \subseteq X$ is *an ϵ-net for X*, if for any halfplane (halfspace) H, such that $|X \cap H|/|X| > \epsilon$, we have $|A \cap H| > 0$, i.e. H contains at least one point from A. In other words, any halfplane (halfspace) that does not cut off any points of A can cut off at most $\epsilon|X|$ points of X. Note that by definition, every ϵ-approximation is an ϵ-net, but not vice versa.

The following simple, but important facts can be proved by straightforward application of the definitions (cf. [7]).

Lemma 1. *Let $X = X_1 \cup \cdots \cup X_m$, where $X_1, \ldots, X_m$ are disjoint subsets of equal cardinality. For each i, let $A_i \subseteq X_i$ be an ϵ-approximation to X_i. Then $A_1 \cup \cdots \cup A_m$ is an ϵ-approximation to X.*

Lemma 2. *Let A be an ϵ-approximation to X. Let B be a δ-approximation to (respectively, a δ-net for) A. Then B is an $(\epsilon + \delta)$-approximation to (respectively, an $(\epsilon + \delta)$-net for) X.*

The crucial property of geometric ϵ-nets is proved in [14] for 2D and in [9] for 3D.

Theorem 2. *Let X be a set of n points in 2D (respectively, 3D). For any $r \geq 1$, an $1/r$-net for X of size $O(r)$ can be computed deterministically in time $O(n(r^2 \log r)^d)$, where $d = 3$ (respectively, 4).*

In fact, paper [9] presents an $O(n^3)$ algorithm for computing an $1/r$-net; however, this time bound can be reduced to the one stated in Theorem 2 by applying Lemma 2.

The analogue of Theorem 2 is not known for higher dimensions. The general VC-dimension theory only guarantees the existence of $1/r$-nets of size $O(r \log r)$, where the constant factor depends on the dimension (see e.g. [13, 7, 8]).

We are now able to describe the optimal BSP algorithm for computing the convex hull of n points. For simplicity, we describe the algorithm for 3D; the corresponding 2D algorithm can be obtained by changing all references to planes into lines, etc. Initially, every processor holds n/p points of the set. Without loss of generality, we may assume that the points are in general position (otherwise, we achieve this by a slight perturbation of the points).

The algorithm proceeds as follows. In the first stage, each processor computes, by Theorem 1, a $1/p$-approximation of size $O(p^3)$ to its local point set. The points of this approximation are the *primary samples.* (In practice, it might be useful to compute the local convex hulls before determining the primary samples; we view the local convex hull computation as optional, since it does not affect the asymptotic complexity of the algorithm.)

Then, all $p \cdot O(p^3) = O(p^4)$ primary samples are collected together in a single processor. By Lemma 1, these primary samples form a $1/p$-approximation to the global point set. By Theorem 2, we compute sequentially a $1/p$-net of size $O(p)$ for the set of all primary samples. The points of this net are the *secondary samples.* By Lemma 2, these secondary samples form a $2/p$-net for the global point set.

We now compute sequentially the convex hull of the set of secondary samples. This secondary hull defines a convex polytope with at most p vertices, and therefore with $O(p)$ edges and $O(p)$ faces. By definition of a $2/p$-net, every tangent plane of the polytope cuts off at most $2n/p$ points from the global point set. Each edge of the polytope defines a *secondary block*, which is the set of all points visible (assuming the polytope is opaque) by an observer sitting on the external surface of the polytope, at an interior point of the defining edge. A secondary block consists of points cut off by the supporting planes of two adjacent polytope faces, therefore its size is at most $4n/p$. Now it remains to broadcast the secondary samples, to assign each secondary block to a particular processor, and then to compute the convex hull of this block by an efficient sequential algorithm. Every edge of the global convex hull must be visible from the interior of at least one edge of the secondary sample polytope, and therefore must be computed by the processor in charge of the corresponding secondary block.

When implemented in the BSP model, the above algorithm has computation cost $O(n \log n/p)$. In the worst case, $4n/p$ points are collected in a single processor, therefore the communication cost is $O(n/p)$. The algorithm performs $O(1)$ global synchronisations. The total BSP cost of the algorithm is

$$W = O(n \log n/p) \qquad H = O(n/p) \qquad S = O(1)$$

For the above resource bounds to hold, it is necessary that the parallel part of the algorithm dominate the sequential part. In particular, we need $n \log n/p \geq p^4 \cdot (p^2 \log p)^4$, therefore $n \log n \geq p^{13}(\log p)^4$. Such a high slackness assumption is only realistic for very small values of p. However, it may be possible to reduce the slackness by performing more than two levels of regular sampling, or by improving the time bound in Theorem 2 (which, for the purpose of this algorithm, may be restricted to point sets in a convex position).

5 Conclusion

We have presented the first optimal deterministic BSP algorithm for computing the convex hull of a finite set of points in three-dimensional Euclidean space. Our algorithm is based on known fundamental results from combinatorial geometry, concerning small-sized, efficiently constructible *ϵ-nets* and *ϵ-approximations* of a given point set. The algorithm generalises the technique of *regular sampling*, used previously for sorting and two-dimensional convex hull computation. The BSP cost of the algorithm is optimal only for extremely large inputs, unless the number of processors is very small. However, more sophisticated regular sampling techniques and future advances in methods for constructing geometric ϵ-nets may improve the required lower bound on the input size.

References

[1] N. M. Amato and F. P. Preparata. A time-optimal parallel algorithm for three-dimensional convex hulls. *Algorithmica*, 14:169–182, 1995.

[2] F. Dehne, X. Deng, P. Dymond, and A. Fabri. A randomized parallel 3D convex hull algorithm for coarse grained multicomputers. *Theory of Computing Systems*, 30(6):547–558, 1997.

[3] M. Diallo, A. Ferreira, A. Rau-Chaplin, and S. Ubéda. Scalable 2D convex hull and triangulation algorithms for coarse grained multicomputers. *Journal of Parallel and Distributed Computing*, 56:47–70, 1999.

[4] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. The CRC Press Series on Discrete Mathematics and Its Applications. CRC Press, 1997.

[5] M. T. Goodrich. Parallel algorithms in geometry. In Goodman and O'Rourke [4], chapter 36, pages 669–681.

[6] M. T. Goodrich. Randomized fully-scalable BSP techniques for multi-searching and convex hull construction. In *Proceedings of the 8th ACM SODA*, 1997.

[7] J. Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50:203–208, 1995.

[8] J. Matoušek. Derandomization in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 559–596. 2000.

[9] J. Matoušek, R. Seidel, and E. Welzl. How to net a lot with little: Small ϵ-nets for disks and halfspaces. In *Proceedings of 6th ACM Symposium on Computational Geometry*, pages 16–22, 1990. An updated version available from `http://www.ms.mff.cuni.cz/acad/kam/matousek/oldpaps.html`.

[10] W. F. McColl. Scalable computing. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 46–61. Springer-Verlag, 1995.

[11] W. F. McColl. Universal computing. In L. Bougé et al., editors, *Proceedings of Euro-Par '96 (Part I)*, volume 1123 of *Lecture Notes in Computer Science*, pages 25–36. Springer-Verlag, 1996.

[12] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, C–37(12):1605–1618, 1988.

[13] K. Mulmuley and O. Schwarzkopf. Randomized algorithms. In Goodman and O'Rourke [4], chapter 34, pages 633–652.

[14] J. Pach and G. Wöginger. Some new bounds for epsilon-nets. In *Proceedings of 6th ACM Symposium on Computational Geometry*, pages 10–15, 1990.

[15] R. Seidel. Convex hull computations. In Goodman and O'Rourke [4], chapter 19, pages 361–375.

[16] H. Shi and J. Schaeffer. Parallel sorting by regular sampling. *Journal of Parallel and Distributed Computing*, 14(4):361–372, 1992.

[17] A. Tiskin. The bulk-synchronous parallel random access machine. *Theoretical Computer Science*, 196(1–2):109–130, April 1998.

[18] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.