Distribution Sweeping on Clustered Machines with Hierarchical Memories

Frank Dehne
School of Computer Science
Carleton University
Ottawa, Canada
frank@dehne.net
http://www.dehne.net

Stefano Mardegan and
Andrea Pietracaprina
DEI, Università di Padova
Padova, Italy
stemarde@tin.it
andrea@artemide.dei.unipd.it

Giuseppe Prencipe
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
prencipe@di.unipi.it

Abstract

This paper investigates the design of parallel algorithmic strategies that address the efficient use of both, memory hierarchies within each processor and a multilevel clustered structure of the interconnection between processors. In the past, these phenomena have usually been addressed separately. This paper is a first step towards parallel algorithmic strategies which address both at the same time. As a case study, we investigate the distribution sweeping method which has been very effective for the design of external memory algorithms for computational geometry problems. We present a novel method for parallel distribution sweeping on a clustered parallel machine with hierarchical local memories, showing that it yields optimal computation, communication and memory access times for a number of geometry problems.

1 Introduction

Typical high-performance computing platforms are characterized by a large number of processing nodes communicating through some interconnection, and by a considerable amount of storage. The topology of the interconnection often induces a multilevel clustered structure of the machine, and the cost of communication between nodes may vary considerably depending on the respective clusters they belong to. For these machines, the entire storage is usually distributed among the nodes. Furthermore, the local memories at each node are typically organized in a hierarchical fashion (cache, RAM, disk). The main factors that affect the performance of programs running on these platforms are the amount of *computation*, measured by the number of operations performed at individual nodes, and the data movement between nodes (*communication*) or between levels of a node's memory hierarchy (*memory access*). While the first factor depends on the specific algorithmic strategy chosen for the application, once such a strategy is fixed, performance can be considerably enhanced by scheduling the operations and choosing the data layout so that expensive communications (e.g., communication among "distant" nodes) and accesses to slow memory levels (e.g., disks) are minimized. This property is usually referred to as *locality*.

Over the last two decades, the development of efficient algorithms for large-scale problems requiring high performance computing, and the definition of suitable computational models to support the design and analysis of these algorithms, have addressed the issues related to memory hierarchy and parallelism separately. However, as recent studies point out [1, 4], in order to attain high performance it is crucial to deal with these aspects in an integrated fashion, both at the modelling and at the algorithm design level.

The main goal of this paper is to initiate the study of general algorithmic strategies for high-performance parallel platforms, which take advantage of parallelism while exhibiting high locality, so to effectively exploit the hierarchical structure of both the memory and the communication network. As a first case study, we investigate the distribution sweep paradigm introduced in [9] to develop efficient external memory algorithms for a family of well-known computational geometry problems, namely those for which the well-known *plane sweep* paradigm [10] yields optimal $O(n \log n)$ solutions in the standard sequential setting. Classical examples of such problems are *measure of the union of rectangles*, *all nearest neighbors*, *3D-maxima*, *2D-weighted dominance counting*, and *lower envelope*. We present a novel implementation of distribution sweeping for clustered parallel machines with hierarchical memories, and show that it affords the development of parallel algorithms which exhibit maximum locality with respect to both, the interconnection topology and the memory hierarchy, thus attaining optimal performance.

Previous Work. For some of the classical geometric problems mentioned above, parallel algorithms were presented in [2] which run on a particular parallel architecture consisting of a d-dimensional array of p processors, each provided with constant storage, connected to a single conventional RAM. On such a machine, these algorithms run in $O\left(n \log n/(p^{1-1/d} \log p)\right)$ time, thus attaining a speed-up of $\Theta\left(p^{1-1/d} \log p\right)$ over the standard sequential algorithms. In [6], parallel algorithms for the same problems are developed for the *Coarse Grained Multicomputer* (CGM). This model, introduced in the same paper, consists of p processors each provided with a large memory, which communicate through an arbitrary network. Communication is achieved exclusively by means of h-relations where each processor sends/receives h messages. For inputs of size $n = \Omega\left(p^2\right)$ the algorithms run optimally in time proportional to the time required to sort n keys evenly distributed among the processors. However, the performance of these algorithms becomes suboptimal if $n = o(n^2)$. We also note that in the CGM model the memory available at each processor is flat.

As mentioned before, distribution sweep was proposed in [9] as a general method to efficiently implement the plane sweep paradigm on the *Parallel Disk Model* (PDM) by [12]. The PDM model, consists of p processors communicating through some arbitrary network, where each processor is provided with an internal memory of size M and one or more disks with block size B. Distribution sweep is based on a divide-and-conquer strategy that partitions the plane into $O(M)$ regions, each containing an equal number of input objects, determines the "interactions" between objects across different regions (a task that depends on the specific problem to be solved), and then recursively performs the sweep within each region. For the geometric problems mentioned above, it is shown in [9] that distribution sweep attains an optimal $O\left(n/B \log(n/B)/\log(M/B)\right)$ number of disk I/Os, on a single-processor/single-disk machine.

It must be remarked that although PDM is a multiprocessor model, parallelism is primarily introduced to account for high processing power. Communication is idealized by ignoring the (possibly hierarchical) structure of the interconnection and the costs related to its bandwidth and latency characteristics. In fact, the main objective of PDM algorithms is to minimize the number of (parallel) disk I/O's.

In [7], an analytical model is proposed to study cost/performance tradeoffs for workstation or SMP clusters with internal hierarchies relatively to given programs. Available benchmarks for FFT and image processing are employed as case studies. Howver, it must be noted that the paper is mainly concerned with the analysis of existing programs and does not address the use of the model for algorithm design.

New Results. In this paper we represent parallel and hierarchical architectures through the *Decomposable BSP* (D-BSP) model of computation, introduced in [5] as a variant of Valiant's BSP [11]. A D-BSP consists of p processor/memory pairs (*nodes*) communicating through some interconnection network. The nodes are viewed as partitionable into $\log p$ nested levels of clusters, where the nodes are able to communicate and synchronize separately within each cluster. It is argued in [3] that because of the clustered structure, the D-BSP exhibits higher effectiveness and portability than BSP (where only one cluster comprising all nodes is defined) while retaining the same generality and usability.

In order to integrate communication and memory hierarchy aspects, we extend the original D-BSP by providing each node with a two-level memory hierarchy consisting of a RAM of size M and a disk with block size B and block I/O-time G. (A detailed definition of the model is given in Section 2.) Clearly, the clustered structure of D-BSP combined with the internal memory hierarchy at each node rewards the development of algorithms with high locality.

For this extended D-BSP we devise a novel parallel implementation of the distribution sweeping method by [9]. Our implementation is a non-trivial generalization of the one in [9] aimed at maximizing both locality of reference (relatively to local memory accesses) and submachine locality (relatively inter-processor communication), whereas only the former type of locality was considered in [9].

Our strategy combines the standard distribution sweeping with an "orthogonal" partitioning method similar to the one adopted in [2, 6], organizing the computation and the data movements so to exploit the hierarchical nature of the local memories as well as the clustered structure of the machine. We show that when bandwidth and latency increase geometrically with the cluster size (this scenario captures a wide family of architectures, e.g. multidimensional arrays like Cray T3E, Intel Paragon, or Beowulf clusters with grid interconnect) and when $M/B = \Omega\left(p^{\epsilon}\right)$ for arbitrary constant $\epsilon > 0$ (as is the case for most commercially available parallel machines) then our strategy yields optimal solutions for all of the aforementioned geometric problems to which distribution sweeping applies, and for all input sizes $n > p$. Indeed, the resulting algorithms feature optimal computation, communication and memory access costs.

2 D-BSP With Hierarchical Local Memories

2.1 Machine Model

The D-BSP model was introduced in [5] as an extension of Valiant's BSP [11] aimed at capturing submachine locality. The following is the regular version of the model (referred to as *recursive D-BSP* in the original paper). Let p be a power of two, and let $\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log p})$ and

$\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log p})$[1]. A *D-BSP* is a collection of p processors communicating through some interconnection fabric. For $0 \leq i \leq \log p$, the p processors are partitioned into 2^i disjoint fixed *i-clusters* $C_0^{(i)}, C_1^{(i)}, \cdots, C_{2^i-1}^{(i)}$ of $p/2^i$ processors each, where the processors of a cluster are able to communicate among themselves independently of the other clusters. Parameters g_i and ℓ_i are related, respectively, to the bandwidth and latency characteristics of each i-cluster. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine: specifically, $C_j^{\log p}$ contains only processor P_j, for $0 \leq j < p$ and $C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)}$, for $0 \leq i < \log p$ and $0 \leq j < 2^i$.

We extend the D-BSP model by introducing a two-level memory hierarchy at each node comprising a RAM of size M, with unit access time, and a disk sufficiently large to hold all relevant data. Data are transferred between RAM and disk in blocks of size B, each block I/O requiring time G. Also, as typical for most systems, data injected-to or received-from the network must pass through the RAM.

A D-BSP computation consists of a sequence of labelled supersteps. In an *i-superstep*, $0 \leq i \leq \log p$, each processor executes internal computation on data held in the local memory system and sends messages exclusively to processors within its i-cluster. The superstep is terminated by a barrier, which synchronizes processors within each i-cluster. It is assumed that messages sent in one superstep are available at the destinations only at the beginning of the subsequent superstep. Consider an i-superstep where each processor performs at most w_1 operations on data held in RAM and at most w_2 disk I/Os, and the messages sent in the superstep form an h-relation (i.e., each processor is source or destination of at most h messages)[2]. Then, the cost of the superstep is upper bounded by $w_1 + w_2 G + h g_i + \ell_i$. Observe that if M is large enough so that all data fit in RAM and the disks are never used, then the model reduces to the standard D-BSP.

The above model, which we refer to as D-BSP*$(p, M, B, G, \boldsymbol{g}, \boldsymbol{\ell})$, is able to explicitly account for memory hierarchy, communication and computation costs. These are the dominant factors that impact performance on most parallel platforms [1, 4]. Moreover, although the detailed structure of the interconnection network is not specified, thus ensuring the generality of the model, the bandwidth and latency parameters used in the cost function are sufficient to ensure efficient support for the model on a wide class of architectures [3].

Although all of the algorithms developed in this paper are correct for arbitrary vectors $\boldsymbol{g}$ and $\boldsymbol{\ell}$, we will analyze their running time for a class of vectors of particular significance. Specifically, for any arbitrary constants α and β, with $0 < \alpha, \beta < 1$, the analysis will be made for a D-BSP*$(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ with $g_i^{(\alpha)} = (p/2^i)^\alpha$ and $\ell_i^{(\beta)} = (p/2^i)^\beta$, for $0 \leq i \leq \log p$. As noted in [8], these values of the parameters capture a wide family of interconnections. Also, we make the reasonable assumption that $M/B = \Omega(p^\epsilon)$ for some arbitrary constant $\epsilon > 0$, which holds for most commercially available parallel machines.

[1]Unless differently specified, all logarithms are taken to base 2.

[2]Note that if $h > M$ we must have $w_2 = \Omega(\lceil (h-M)/B \rceil)$.

2.2 Algorithmic Primitives

Sorting This primitive involves sorting $n \geq p$ keys, from a totally ordered universe, which are evenly distributed among the processors of a D-BSP*$(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$. We assume that at the beginning and at the end of the sorting the keys reside in the nodes' disks.

Proposition 1 *Sorting can be performed in optimal time* $T_{\mathrm{sort}}(n,p) = O(\frac{n}{p}(\log(\frac{n}{p}) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil \frac{\log(\frac{n}{B})}{\log(\frac{M}{B})} \right\rceil G)$.

Proof. Omitted due to space restrictions. □

Merging Consider $K \leq p$ sorted sequences of n/K keys each, with each sequence evenly distributed among the p/K nodes of a distinct $(\log K)$-cluster. For the purposes of this paper, the complexity of merging can be bound from above by that of sorting. However, when all data fit in the nodes' RAMs, hence $n/p = O(M/B)$, the following tighter bound is easily established using bitonic merging.

Proposition 2 *If $n/p = O(M/B)$ and all relevant data are kept in the nodes' RAMs, then the K sorted sequence can be merged on a D-BSP*$(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ in optimal time $T_{\mathrm{merge}}(n, K, p) = O(\frac{n}{p}p^\alpha + p^\beta)$.*

Broadcast Consider $K \leq M$ values stored at the RAM of a D-BSP*$(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ node, to be broadcast to all other nodes. By broadcasting the values to larger and larger clusters we can obtain the following optimal performance.

Proposition 3 *$K \leq M$ values stored in the RAM of a D-BSP*$(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$ node can be broadcast to all other nodes in time $T_{\mathrm{broadcast}}(K, p) = O(Kp^\alpha + p^\beta)$.*

3 Parallel Distribution Sweeping On A D-BSP With Hierarchical Local Memories

Distribution sweeping was introduced in [9] as an efficient strategy to implement the well-known *plane sweep* paradigm [10] in the external memory setting. When applied to a geometric problem, the standard sequential plane sweep involves scanning the input objects in sorted order for a given dimension and updating a suitable data structure (e.g., a search tree) when a new object is encountered. The main idea in [9] is to define $O(M)$ regions that evenly partition the input space, recursively solve the problem in each region, and account for the interaction of objects across different regions. The latter task can be accomplished by simply scanning the input objects and using suitable data structures whose size is proportional to the number of regions, so that they fit in RAM. As discussed in [9] this strategy optimizes the number of disk I/Os.

In this section we present a novel parallel distribution sweeping method for a D-BSP with hierarchical local memories. Our method minimizes both the number of disk I/Os and, at the same time, the communication costs for an assumed hierarchical interconnection fabric. The main idea is the following. Consider a geometric problem, to be solved via parallel distribution sweeping on a D-BSP$^*(p, M, B, G, \boldsymbol{g}, \boldsymbol{\ell})$. We partition the input objects into $K = \min\{n/p, M/B\}$ subsets (e.g., through vertical slabs in the plane), and recursively solve the subproblem associated with each subset. If $K < p$ we assign each subproblem to a distinct $(\log K)$-cluster (with p/K processors); otherwise we assign p/K subproblems to each processor. In this fashion, the clustered structure of the machine is suitably exploited. Then, in order to account for the interactions among objects across different subsets, we employ an "orthogonal" partitioning of the inputs and of data resulting from the recursive calls into new subsets of size $\Theta(K)$ (e.g., through horizontal slabs in the plane), in such a way that all interactions be computable independently, hence in parallel, for the new subsets. This partitioning is similar to the one adopted in [2, 6]. The choice of K ensures that the work on each new subset can be entirely done in a node's RAM, and that at any time no processor holds (hence sends or receives) more than $O(n/p)$ data. The following case studies explain out methodology in more detail.

3.1 Measure Of The Union Of Rectangles

Let $\mathcal{R}$ be a set of n isothetic rectangles in the (x, y)-plane, evenly distributed among the nodes of a D-BSP$^*(p, M, B, G, \boldsymbol{g}, \boldsymbol{\ell})$, and initially stored at the nodes' disks. The *Measure of Union of Rectangles* (MUR) problem requires finding the area covered by the union of these rectangles. For convenience, we make the standard assumption that no two rectangles have vertices at the same abscissa or at the same ordinate.

Let $Y_{\mathcal{R}}$ denote the set of $2n$ ordinates of the vertices of the rectangles in $\mathcal{R}$. Let also ℓ' and ℓ'' denote the leftmost and rightmost abscissae, respectively, of the vertices of these rectangles. The triple $(Y_{\mathcal{R}}, \ell', \ell'')$ defines $2n - 1$ rectangular *stripes* on the plane, each stripe delimited by pairs of consecutive ordinates in $Y_{\mathcal{R}}$ and by ℓ' and ℓ''. Let $S(Y_{\mathcal{R}}, \ell', \ell'')$ denote the set of these stripes. For $s \in S(Y_{\mathcal{R}}, \ell', \ell'')$, we denote by $A(s, \mathcal{R})$ the *cover of s relative to* $\mathcal{R}$, that is, the area of the portion of s covered by rectangles in $\mathcal{R}$. As shown in [2, 6], the overall area A of the union of the rectangles in $\mathcal{R}$ is given by

$$A = \sum_{s \in S(Y_{\mathcal{R}}, \ell', \ell'')} A(s, \mathcal{R}). \qquad (1)$$

Hence, the computation of A reduces essentially to the computation of the stripe covers, which we describe below.

Computation of the stripe covers

We present the computation of the stripe covers within a slightly more general setting. Let $\ell' < \ell''$ be two arbitrary abscissae on the plane and let $\mathcal{R}$ be a set of n rectangles, where each rectangle has at least one vertical edge in the portion of the plane delimited by ℓ' and ℓ''. Procedure STRIPE-COVERS$(\mathcal{R}, p, \ell', \ell'')$, described below, computes the covers relative to $\mathcal{R}$ for all stripes in $S(Y_{\mathcal{R}}, \ell', \ell'')$, on a D-BSP$^*(p, M, B, G, \boldsymbol{g}, \boldsymbol{\ell})$.

For convenience, we assume that for every vertical edge e of a rectangle $R \in \mathcal{R}$ at abscissa $x_e \in [\ell', \ell'']$, the pair (x_e, R) is provided in the input to the procedure (hence, we may have two pairs in the input for a rectangle). Moreover, we assume that the pairs are given in sorted order of their first components. This requires a sorting step prior to the invocation of the procedure. The output of the procedure consists of the set of stripes in $S(Y_{\mathcal{R}}, \ell', \ell'')$, sorted according to their top ordinate, each stripe s associated with its cover $A(s, \mathcal{R})$.

Procedure STRIPE-COVERS$(\mathcal{R}, p, \ell', \ell'')$

If ($p = 1$ and $n = O(M)$) compute the stripe covers in RAM through a standard plane sweep. Else, do the following:

Step 1: Let $K = \gamma \min\{n/p, M/B\}$, for a suitable constant $0 < \gamma < 1$. Partition the portion of the plane between ℓ' and ℓ'' into K *vertical slabs* $V_1, V_2, \ldots, V_K$, such that the vertices of the rectangles in $\mathcal{R}$ falling in such a portion are evenly distributed among these slabs. Let $\mathcal{R}_{V_i}$, $1 \le i \le K$, denote the subset of rectangles of $\mathcal{R}$ with vertices in V_i. (Note that a rectangle may have vertices in two slabs.) Let $\ell' = \ell_1 < \ell_2 < \cdots < \ell_{K+1} = \ell''$ be such that slab V_i is delimited by the lines at abscissae ℓ_i and ℓ_{i+1}, for $1 \le i \le K$. Send the set of abscissae $\{\ell_i : 1 \le i \le K + 1\}$ to the first DBSP node and then, from this node, broadcast the set to all other nodes.

Step 2: (recursive step) If $K \le p$, then, for $1 \le i \le K$ in parallel, recursively execute STRIPE-COVERS$(\mathcal{R}_{V_i}, p/K, \ell_i, \ell_{i+1})$ on the i-th $(\log K)$-cluster. Else ($K > p$), for $1 \le j \le p$ in parallel, recursively execute STRIPE-COVERS$(\mathcal{R}_{V_i}, 1, \ell_i, \ell_{i+1})$, at the j-th DBSP node, for every $(j-1)K/p < i \le jK/p$. Note that in either case, each DBSP node deals with $O(n/p)$ data.

Step 3: Partition the plane into $2n/K$ *horizontal slabs* $H_1, H_2, \ldots, H_{2n/K}$ such that each slab contains exactly K ordinates of $Y_{\mathcal{R}}$. For $1 \le j \le 2n/K$, let $\mathcal{R}_{H_j}$ denote the subset of rectangles of $\mathcal{R}$ with vertices in H_j, and note that $|\mathcal{R}_{H_j}| \le K$. Let $h_1 > h_2 > \cdots > h_{2n/K+1}$ be such that slab H_j is delimited by the lines at ordinates h_j and h_{j+1}, for $1 \le j \le 2n/K$. The set $\bar{Y} = \{h_j : 1 \le j \le 2n/K + 1\}$, which defines the horizontal slabs, can be determined by merging the sorted sequences of stripes resulting from the recursive calls of the previous step, so that each node determines $(2n/K)/p$ ordinates of $\bar{Y}$. (Note that the choice of K implies $2n/K > p$.) The horizontal and vertical slabs define $2n$ *boxes*, namely $B_{i,j} = V_i \cap H_j$, for $1 \le i \le K$ and $1 \le j \le 2n/K$.

Step 4: If $K \le p$, then broadcast $\bar{Y}$ to every $(\log K)$-cluster, each node of the cluster receiving $\Theta((n/K)/(p/K)) = \Theta(n/p)$ ordinates in sorted order. If instead $K > p$, broadcast $\bar{Y}$ to every node. Note that, in the latter case each node receives $\Theta(n/K) = O(n/p)$ ordinates.

Step 5: For $1 \le i \le K$, determine $A(s, \mathcal{R}_{V_i})$ for every stripe $s \in S(Y_{\mathcal{R}_{V_i}} \cup \bar{Y}, \ell_i, \ell_{i+1})$. To do so, the stripes and the corresponding covers produced by the recursive calls of Step 2 are first moved back to their positions at the end of that step. Then, for every $1 \le i \le K$ the new stripes induced by $Y_{\mathcal{R}_{V_i}} \cup \bar{Y}$ are determined by scanning concurrently the stripes in $S(Y_{\mathcal{R}_{V_i}}, \ell_i, \ell_{i+1})$ and the set $\bar{Y}$. Computing the new covers is straightforward. If $K \le p$, each $(\log K)$-cluster, in parallel, will do the computation for a distinct i, while if $K > p$, each node, in parallel, will do the computation for K/p values of i.

Step 6: For every $1 \le i \le K$ and $1 \le j \le 2n/K$, determine the number $b_{i,j}$ of boxes $B_{i',j}$, with $i' < i$, which are entirely covered by rectangles in $\mathcal{R}_{V_i}$. This is accomplished by a vertical sweep of the rectangles of $\mathcal{R}_{V_i}$, employing a segment tree with $i-1$ leaves associated with the vertical slabs to the left of V_i (see [10] for the definition and the properties of segment trees). Note that this requires the rectangles in $\mathcal{R}_{V_i}$ to be sorted by the ordinate of their top edges. The sorted order can be easily derived by that of the stripes in $S(Y_{\mathcal{R}_{V_i}}, \ell_i, \ell_{i+1})$. As in the previous step, if $K \le p$, each $(\log K)$-cluster, in parallel, will compute all of the $b_{i,j}$'s for a distinct i, while if $K > p$, each node, in parallel, will compute all of the $b_{i,j}$'s for K/p consecutive values of i.

Step 7: For $1 \le i \le K$ and $1 \le j \le 2n/K$, let $S_{i,j}$ be the set of stripes in $S(Y_{\mathcal{R}_{V_i}} \cup \bar{Y}, \ell_i, \ell_{i+1})$ which fall within H_j. For $1 \le j \le 2n/K$, let the $\lceil j/(2n/(Kp)) \rceil$-th D-BSP node be in charge of H_j, and send the following three sets of data to such a node: $\mathcal{R}_{H_j}$, $\{b_{i,j} : 1 \le i \le K\}$, and $\bigcup_{i=1}^{K} S_{i,j}$. Note that each of the three sets has size $O(K)$, hence each node receives $O(n/p)$ data, overall.

Step 8: Do the following at every D-BSP node, in parallel. For every H_j assigned to the node and for every $s \in S(Y_{\mathcal{R}}, \ell', \ell'')$ falling within H_j, compute $A(s, \mathcal{R})$, in sorted order by top ordinate. This is accomplished as follows. For $1 \le i \le K$, the node first determines whether box $B_{i,j}$ is entirely covered by some rectangle. This requires a scan of the values $\{b_{i,j} : 1 \le i \le K\}$ available at the node. Corresponding to every box entirely covered by some rectangle, a special stripe is created whose cover coincides with the entire area of the box. Also, for every rectangle in $\mathcal{R}_{H_j}$ spanning entirely one or more vertical slabs, a special stripe is created corresponding to the portion relative to H_j and to the vertical slabs entirely spanned by the rectangle, whose cover coincides with the area of this portion. Then, a sweep is performed through the stripes $s \in \bigcup_{1 \le i \le K} S_{i,j}$ and the special stripes (all sorted by ordinate). At each sweep step, a suitable segment tree of size $O(K)$, whose leaves correspond to the vertical slabs, is updated based on the cover associated with the stripe being processed. Note that $O(K)$ data are required, overall, for the computation relative to H_j. Hence, if the constant γ in the definition of K is suitably chosen, such a computation can be entirely done in the node's RAM.

Analysis

Let $T_{\mathrm{MUR}}(n,p)$ denote the time required by the algorithm to compute the area of the union of n rectangles on a D-BSP$^*(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$. Let $T_{\mathrm{SC}}(m,q)$ denote the running time of procedure STRIPE-COVERS when invoked for a set of m rectangles, on a cluster of $q \le p$ processors of a D-BSP$^*(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$. Based on Equation 1 and recalling that procedure STRIPE-COVERS requires the input rectangles sorted by the abscissae of their vertical edges, we have that

$$T_{\mathrm{MUR}}(n,p) = O\left(T_{\mathrm{sort}}(n,p) + T_{\mathrm{SC}}(n,p)\right). \qquad (2)$$

Lemma 1 *If $M/B = \Omega(p^\epsilon)$, for some arbitrary constant $\epsilon > 0$, then for every $n > p$, we have $T_{\mathrm{SC}}(n,p) = O(T_{\mathrm{sort}}(n,p))$.*

Proof. Omitted due to space restrictions. □

The following theorem is an immediate consequence of the above lemma and Equation 2.

Theorem 1 *If $M/B = \Omega(p^\epsilon)$, for some arbitrary constant $\epsilon > 0$, then for every $n > p$ $T_{\mathrm{MUR}}(n,p) = O\left(\frac{n}{p}(\log(n/p) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil \frac{\log(n/B)}{\log(M/B)} \right\rceil G\right)$.*

The optimality of our method follows from Proposition 1.

3.2 Other Problems

Due to space limitations, we can only **list** our results for other problems to which our new parallel distribution sweep method applies. For the remainder of this subsection, we assume that the n inputs are evenly distributed among the nodes of a D-BSP$^*(p, M, B, G, \boldsymbol{g}^{(\alpha)}, \boldsymbol{\ell}^{(\beta)})$, and that $M/B = \Omega(p^\epsilon)$, for some arbitrary constant $\epsilon > 0$.

All Nearest Neighbors Let S be a set of n points on the plane. For each $q \in S$ the problem requires to find the closest point NN$(q) \in S \setminus \{q\}$.

Theorem 2 *For every $n > p$ the all nearest neighbors problem can be solved in time* $O\left(\frac{n}{p}(\log(n/p) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil\frac{\log(n/B)}{\log(M/B)}\right\rceil G\right)$.

Proof. Omitted due to space restrictions. □

Lower Envelope Let S be a set of n non-intersecting line segments on the plane. The problem requires to find a set of $m \leq 2n$ x-monotone segments, sorted by right endpoint, representing the lower envelope formed by the segments of S visible from $(0, -\infty)$.

Theorem 3 *For every $n > p$ the lower envelope problem can be solved in time* $O\left(\frac{n}{p}(\log(n/p) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil\frac{\log(n/B)}{\log(M/B)}\right\rceil G\right)$.

Proof. Omitted due to space restrictions. □

3D-Maxima Let S be a set of n points in $\Re^3$. For any two points v and w in S, we say that v dominates w if each coordinate of v is greater than the corresponding coordinate of w. A point $v \in S$ is said to be a maximum in S if it is not dominated by any other point of S. The 3D-maxima problem requires to compute the set 3Dmax(S) of maxima in S.

Theorem 4 *For every $n > p$ the 3D-maxima problem can be solved in time* $O\left(\frac{n}{p}(\log(n/p) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil\frac{\log(n/B)}{\log(M/B)}\right\rceil G\right)$.

Proof. Omitted due to space restrictions. □

2D-Weighted Dominance Counting Let S be a set of n points on the plane, where each point q is associated with a weight $w(q)$. For each $q \in S$ the problem requires to find the total weight WD(q) of the points in S dominated by q, that is, those points belonging to the region to the left and below point q.

Theorem 5 *For every $n > p$ the 2D-weighted dominance counting problem can be solved in time*
$O\left(\frac{n}{p}(\log(n/p) + p^\alpha) + p^\beta + \frac{n}{Bp}\left\lceil\frac{\log(n/B)}{\log(M/B)}\right\rceil G\right)$.

Proof. Omitted due to space restrictions. □

4 Conclusions

We presented a novel implementation of the distribution sweeping paradigm [9] aimed at maximizing both, locality with respect to local memory reference and inter processor communication. This method yields optimal solutions for a number of geometry problems. We consider this as a first step towards the development of more algorithmic strategies that exhibit high locality with respect to both memory accesses and communication, which is necessary for efficient software on current parallel architectures.

References

[1] N. Amato, J. Perdue, A. Pietracaprina, G. Pucci, and M. Mathis. Predicting performance on smp's. a case study: The sgi power challenge. In *Proc. International Parallel and Distributed Processing Symposium*, volume IPDPS 2000, pages 729–737, Cancun, MEX, 2000.

[2] M. Atallah and J. Tsay. On parallel decomposability of geometric problems. *Algorithmica*, 8:209–231, 1992.

[3] G. Bilardi, C. Fantozzi, A. Pietracaprina, and G. Pucci. On the effectiveness of d-bsp as a bridging model of parallel computation. *Proc. of the Int. Conference on Computational Science*, LNCS 2074:579–588, 2001.

[4] G. Chaudhry, T. Cormen, and L. Wisniewski. Columnsort lives! an efficient out-of-core sorting program. In *Proc. of the 13th ACM Symp. on Parallel Algorithms and Architectures*, pages 169–178, 2001.

[5] P. De la Torre and C. Kruskal. Submachine locality in the bulk synchronous setting. In *roc. of EUROPAR 96*, volume LNCS 1124, pages 352–358, 1996.

[6] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.

[7] X. Du and X. Zhang. The impact of memory hierarchies on cluster computing. In *Proc. IPDPS*, pages 35–44, 1999.

[8] C. Fantozzi, A. Pietracaprina, and G. Pucci. Implementing shared memory on clustered machines. In *2nd International Parallel and Distributed Processing Symposium*, 2001.

[9] M. Goodrich, J. Tsay, D. Vengroff, and J. Vitter. External-memory computational geometry. In *Proc. of the 31stIEEE Symp. on Foundations of Computer Science*, pages 714–723, 1993.

[10] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

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

[12] J. Vitter and E. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2/3):110–147, 1994.