Theory Comput. Systems **35**, 567–597 (2002)
DOI: 10.1007/s00224-002-1066-2

Theory of Computing Systems

Bulk Synchronous Parallel Algorithms for the External Memory Model*

Frank Dehne,[1] Wolfgang Dittrich,[2] David Hutchinson,[1,3] and Anil Maheshwari[1]

[1]School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6
{dehne,maheshwa}@scs.carleton.ca

[2]Bosch Telecom GmbH, UC-ON/ERS, Gerberstraße 33, 71522 Backnang, Germany

[3]Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129, USA
hutchins@cs.duke.edu

Abstract. Blockwise access to data is a central theme in the design of efficient *external memory* (EM) algorithms. A second important issue, when more than one disk is present, is fully parallel disk I/O. In this paper we present a simple, *deterministic* simulation technique which transforms certain Bulk Synchronous Parallel (BSP) algorithms into efficient parallel EM algorithms. It optimizes blockwise data access and parallel disk I/O and, at the same time, utilizes *multiple processors* connected via a communication network or shared memory. We obtain new improved parallel EM algorithms for a large number of problems including sorting, permutation, matrix transpose, several geometric and GIS problems including three-dimensional convex hulls (two-dimensional Voronoi diagrams), and various graph problems. We show that certain parallel algorithms known for the BSP model can be used to obtain EM algorithms that meet well known *I/O complexity lower bounds* for various problems, including sorting.

* A preliminary version appeared in *IEEE IPPS* 1999 and *ACM–SIAM SODA* 1999. The research of F. Dehne and A. Maheshwari was partially supported by the Natural Sciences and Engineering Research Council of Canada. A. Maheshwari was also partially supported by NCE GEOIDE.

1. Introduction

1.1. *Motivation*

Some of the key applications of parallel computing include astrophysical models, genetic sequencing, geographic information systems, ecological models, weather prediction, telecommunications applications, commercial digital video and audio, digital libraries, government information systems, and biological models for medical applications. Researchers in all of these applications currently face data sets of terabyte size, perhaps increasing to petabytes in the foreseeable future. If parallel computing is to succeed in these areas, it needs to solve the problem of how to obtain efficient parallel disk I/O. Research in *external memory* (EM) algorithms has recently received considerable attention. Primary references are the report of the ACM workshop on strategic directions in computing research, edited by Gibson et al. [27] and Vitter's survey [47]. The main questions are how to optimize blockwise and simultaneous access to multiple disks, and how to combine this with a parallel processing environment where multiple processors (each with multiple disks) are connected via a communication network or shared memory. Closely related problems are how to include the effects of network caching and multilevel memory hierarchies in general.

1.2. *Review*: *Parallel Disk Model and Previous Results*

We outline a few results on EM algorithms which relate directly to our work. A more complete survey can be found in [47].

A well-studied model of computation for EM algorithms is the *Parallel Disk Model* (PDM) introduced by Vitter and Shriver [49]. It is used to model the two-level memory hierarchy consisting of parallel disks connected to one or more processors which communicate via a shared internal memory or a hypercube like network. The PDM uses the following parameters: N = problem size, M = internal memory size, B = block transfer size, D = number of disk drives, and p = number of processors, where $M < N$, and $1 \leq DB \leq M/2$. All sizes are in units of *application data items*. The PDM cost measure is the number of I/O operations required by an algorithm, where DB items can be transferred between the internal memory and the disk system in a single I/O operation.

Floyd [24] studied sorting (and matrix transpose) in a single-disk single-processor model, where $B = M/2 = \Theta(N^c)$, for some constant $c > 0$, and provided upper and lower I/O bounds. Aggarwal and Vitter [2] generalized Floyd's model and provided matching upper and lower I/O bounds for several problems, and these bounds apply to the PDM. In the worst case, the number of I/Os required for sorting is $\Theta((N/BD) \log_{M/B} (N/B))$[1] [2], [47]. Several EM algorithms exist for sorting, including [1]–[3], [37], [38], [49], [50], and [39]. Surprisingly, it turns out that performing a permutation requires $\Theta(\min\{N/D, (N/BD) \log_{M/B}(N/B)\})$ I/Os [2], [47], although permutation can be performed in linear time in the RAM model. Similarly, the worst-case number of I/Os required to transpose an $N = p \times q$ matrix from row-major order to column-major order is $\Theta((N/BD) \log_{M/B} \min(M, p, q, N/B))$ [2], [47]. Cormen et al. [17] have studied the

[1] $\mathrm{Log}_{M/B}(N/B)$ is defined to mean $\max\{1, \log_{M/B}(N/B)\}$.

optimal number of I/Os required to perform several special classes of permutations. This includes permutations arising in matrix transpose, FFTs, hypercubes, matrix reblocking. Arge et al. [6] show that any problem which requires $\Omega(N \log N)$ comparisons in the comparison model, requires $\Omega((N/B)\ \log_{M/B}(N/B))$ I/Os in the PDM.

EM algorithms have been proposed for a number of problems arising in computational geometry [7], [5], [18], [30], geographical information systems [7], [46], and graphs [4], [13], [31], [33], [44]. Over the last few years, comprehensive computing and cost models, that incorporate multiple disks and multiple processors have been proposed [15], [21], [23], [35].

The Parallel Random Access Memory (PRAM) model [25] has been used for many years as a model of parallel computation and has supported the development of a rich theory of parallel computation; see, for example, [32] and [34]. The PRAM assumption of a large shared random access memory with uniform cost access to every cell by each processor has the advantage of simplicity but has also prompted alternative proposals intended to be more representative of practical machines. These include the Bulk Synchronous Parallel (BSP) model of Valiant [45], the Coarse-Grained Multicomputer (CGM) of Dehne et al. [22], the LogP model of Culler et al. [19], and the Extended BSP (BSP*) model of Bäumker et al. [10].

Several suggestions have been made regarding the simulation of parallel algorithms as EM algorithms. This includes the work of Atallah and Tsay [8], the results of Chiang et al. [13] on simulating PRAM algorithms, and the results of Sibeyn and Kaufmann [42], Dehne et al. [21], and Dittrich et al. [23] on simulating BSP, CGM, and BSP* algorithms.

1.3. *Review*: *Parallel Models of Computation and Related Work*

The BSP model was proposed by Valiant [45] as a "bridging model for parallel computation"; a standard model on which both hardware and software designs can agree. The stated objective of the BSP model is to provide a model that is simultaneously useful to hardware designers, algorithm designers, and programmers of parallel computers. The BSP model has parameters N, v, g, and L. It consists of v processor/memory components, a *router* that delivers messages in a point to point fashion, and a facility to synchronize all processors. Each processor has a unique label in the range $0, 1, \ldots, v-1$. Computation proceeds in a succession of *supersteps* separated by synchronizations, usually divided into *communication* and *computation supersteps*, see Figure 1. In computation supersteps processors perform local computations on data that is available locally at the beginning of the superstep and issue send operations. Between computation supersteps, a communication superstep is performed, where each processor exchanges data with its peers, via the router. An *h-relation* is a superstep where $O(h)$ data are sent and received by every processor. The parameter g is the time required to send a single word of data between two processors, where time is measured in number of CPU operations, and the parameter L is the minimum setup time or latency of a superstep, measured in CPU operations.

The cost of a computation on the BSP model is represented by the sum of three quantities, one for computation time, one for communication time, and one for the time required by the processors to synchronize. Let $\mathcal{A}$ be a BSP algorithm, operating on

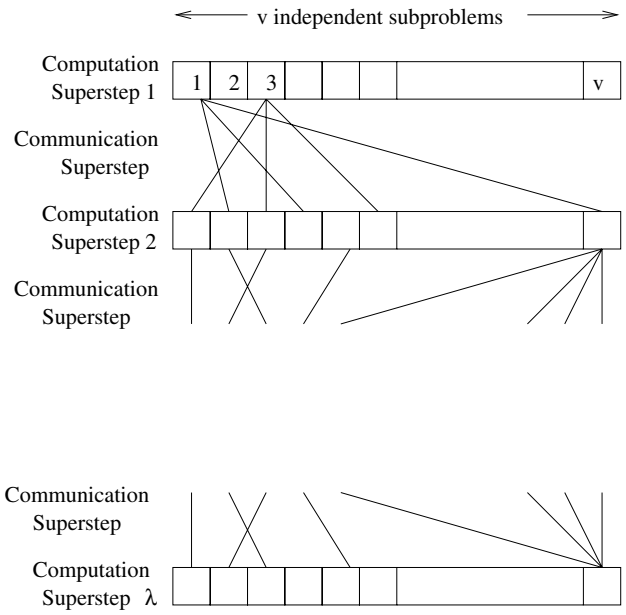

Fig. 1. A BSP computation.

input data of size N items. Let β be its parallel computation time, let α be the number of words of data sent between processors during the execution of $\mathcal{A}$, and let λ be the number of supersteps required by $\mathcal{A}$ to solve a problem of size N. Then the cost $T(\mathcal{A})$ of the computation is

$$T(\mathcal{A}) = \beta + g \cdot \alpha + \lambda \cdot L.$$

A *CGM algorithm* [22] is a particular kind of BSP algorithm where the communication part of each superstep consists of exactly one h-relation with $h = \Theta(N/v)$. In contrast to [22], however, we use the cost model of BSP [45] for analyzing CGM algorithms.

The BSP* model [10] is a special case of the BSP model where each message is assessed a minimum cost equivalent to a message of size b items. The BSP* model therefore gives incentives to send messages of at least b in size.

While the PDM captures computation and I/O costs, it is designed for a specific type of communication network, where a communication operation is expected to take a single unit of time, comparable with a single CPU instruction. BSP and similar parallel models capture communication and computational costs for a more general class of interconnection networks, but do not capture I/O costs. Cormen and Goodrich [15] posed the challenge of combining BSP-like parallel algorithms with the requirements for parallel disk I/O, and, specifically, the need for a combined model. Such combined models, namely *EM-BSP* and its variants, were proposed in [21].

The EM-BSP model [21] is an extension of the BSP model to include secondary local memories (see Figure 2). In addition to its local memory, each processor has an EM in the form of a set of hard disks. This model has four additional parameters, namely

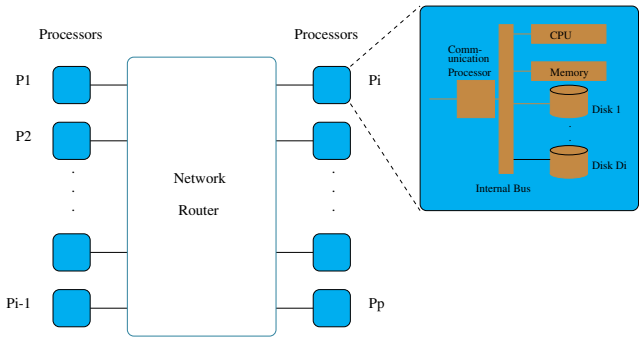

Fig. 2. Illustration of an EM-CGM model.

the size of local memory of each processor M, the number of disks at each processor D, the transfer block size B, and the ratio G of local computational capacity (number of local computation operations) divided by local I/O capacity (number of blocks of size B that can be transferred between the local disks and memory) per unit time. Like a computation on the BSP model, a computation on the EM-BSP model proceeds in a succession of supersteps. Communication and computation supersteps occur as in the BSP model and multiple I/O operations are permitted during a single computation superstep. For the EM-BSP model, the computation cost, t_{comp}, and communication cost, t_{comm}, are the same as for the BSP model. For each local operation the RAM uniform cost measure is used. For an h-relation, i.e., a routing request where each processor sends and receives at most h messages of size b, $g \cdot h + L$ time units are charged per communication superstep. The *I/O cost* (or *I/O time*) of a computation superstep is $t_{\text{I/O}} = \max_{j=1}^{v}\{w_{\text{I/O}}^{j}\}$ where $w_{\text{I/O}}^{j}$ is the I/O cost incurred by processor j. Each I/O operation costs G time steps. For a computation superstep with at most t_{comp} local operations on each processor, $t_{\text{comp}} + t_{\text{I/O}} + L$ time units are charged. The total cost of each superstep is therefore $t_{\text{comp}} + t_{\text{comm}} + t_{\text{I/O}} + L$.

An algorithm for a BSP/BSP*/CGM model with multiple disks attached to each processor (see Figure 2) is referred to as an *EM-BSP/EM-BSP*/EM-CGM algorithm*. Since the BSP, BSP*, and CGM models are very similar, algorithms which are efficient on the BSP* and CGM models are also BSP algorithms. Because of this similarity, we refer to BSP* and CGM algorithms as BSP algorithms, and to EM-BSP* and EM-CGM algorithms as EM-BSP algorithms.

Let $\mathcal{A}^*$ be the best sequential algorithm on the RAM for a problem $\mathcal{P}$ of size N items, and let $T(\mathcal{A}^*)$ be its worst case runtime. Let $c \geq 1$ be a constant. A *c-optimal* EM-BSP algorithm $\mathcal{A}$ for v processors is c-optimal in communication and I/O if it meets the following criteria (all asymptotic bounds are with respect to the problem size $N \to \infty$):

- The ratio φ between the computation times of $\mathcal{A}$ and $T(\mathcal{A}^*)/v$ is in $c + o(1)$.
- The ratio ξ between the communication time of $\mathcal{A}$ and the computation time $T(\mathcal{A}^*)/v$ is in $o(1)$.
- The ratio η between the I/O time of $\mathcal{A}$ and the computation time $T(\mathcal{A}^*)/v$ is in $o(1)$.

Note that the constraint on η differentiates this definition from the corresponding one for non-external memory parallel algorithms proposed in [26]. The simultaneous satisfaction of the constraints $\varphi \in c + o(1)$, $\xi \in o(1)$, and $\eta \in o(1)$ is a stringent requirement to place on an EM-BSP algorithm. It is not always possible to show that the running time of an algorithm is dominated by its computation time, for instance. Various relaxations of the requirements on φ, ξ, and η are possible. One such relaxation comes from the notion of balancing the costs of computation, communication, and I/O, so that none of them dominates the running time, and the work done in each is asymptotically the same as the work of an optimal sequential algorithm. We use the terms *work-optimal*, *communication-efficient*, and *I/O-efficient* to describe an algorithm for which $\varphi \in O(1)$, $\xi \in O(1)$, and $\eta \in O(1)$, respectively. An algorithm which is work-optimal, communication-efficient, and I/O-efficient, therefore, is one whose running time complexity is no worse than the complexity $T(\mathcal{A}^*)/v$. Constant factors are ignored.

1.4. *New Results*

In this paper we show that any v processor BSP algorithm $\mathcal{A}'$ which communicates via h-relations of size $h = O(N/v)$ with λ supersteps/rounds, local memory size μ, computation time $\beta + \lambda L$, and communication time $g\alpha + \lambda L$ can be simulated, deterministically, as a p-processor EM-BSP algorithm $\mathcal{A}$ with $(v/p)\lambda$ rounds, computation time $(v/p)(\beta + O(\lambda\mu)) + (v/p)\lambda L$, communication time $(v/p)g\alpha + (v/p)\lambda L$, and I/O time $(v/p)G \cdot O(\lambda(\mu/DB)) + (v/p)\lambda L$ for $M = \Theta(\mu)$, $N = \Omega(vDB)$, and $B = O(N/v^2)$.

Let $g(N)$, $L(N)$, and $v(N)$ be increasing functions of N. If $\mathcal{A}'$ is c-optimal on the BSP model for $g \leq g(N)$, $L \leq L(N)$, and $v \leq v(N)$, then $\mathcal{A}$ is c-optimal for $\beta = \omega(\lambda\mu)$, $g \leq g(N)$, $G = BD \cdot o(\beta/\mu\lambda)$, and $L \leq L(N) \cdot p/v$. $\mathcal{A}$ is work-optimal, communication-efficient, and I/O-efficient if $\mathcal{A}'$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$, $g \leq g(N)$, $G = BD \cdot O(\beta/\mu\lambda)$, and $L \leq L(N) \cdot p/v$.

The simulation technique described in this paper obtains tight upper and lower bounds on the message size, and this attribute consequently makes the technique very simple compared with the randomized simulation technique described in [21]. The analysis of the I/O complexity of our algorithms is done as in the PDM. In addition, however, we analyze the communication and computational complexities and hence the overall asymptotic running time.

In Section 3 we present new EM algorithms for a large number of problems by applying our simulation technique to parallel algorithms for these problems. Tables 1 and 2 summarize the performance of a selection of new EM-BSP algorithms obtained via the simulation techniques presented in this paper. In contrast to some previous work, these results are scalable with respect to the number of processors. The new results improve on the previous upper bounds on I/O complexity for many problems, and also meet I/O complexity lower bounds for certain problems. In some cases, our results may at first appear to improve on known lower bounds. We explain the latter in Section 3.2 by pointing out that the I/O complexity lower bounds were proven for arbitrary ranges over the various parameters involved and take a simplified form if one restricts them to our particular range of parameter values. This answers questions of Cormen [14] and Vitter [48] on the apparent contradictions between the results of [21] and previously

Table 1. Summary of new EM algorithms in comparison with previous results.

Problem description	PDM I/O complexity[a,b]	BSP complexity[c]	EM-BSP complexity[d]
	Group A: Fundamental algorithms[e]		
1. Sorting	$\Theta\left(\frac{N}{BD}\log_{M/B}\frac{N}{B}\right)$ [2], [47]	$\tau = O\left(\frac{N\log N}{v}\right)$ [29]	$\alpha = O\left(\frac{N}{p}\right)$, $\beta = O\left(\frac{N}{p}\log N\right)$
		$\lambda = O(1)$, $M = O\left(\frac{N}{v}\right)$	$\kappa = O\left(\frac{N}{pDB}\right)$, $\lambda = O\left(\frac{v}{pk}\right)$
2. Permutation	$\Theta\left(\min\left(\frac{N}{D}, \frac{N}{DB}\log_{M/B}\frac{N}{B}\right)\right)$ [2], [47]	$\tau = O\left(\frac{N\log N}{v}\right)$	$\alpha = O\left(\frac{N}{p}\right)$, $\beta = O\left(\frac{N}{p}\right)$
		$\lambda = O(1)$, $M = O\left(\frac{N}{v}\right)$	$\kappa = O\left(\frac{N}{pDB}\right)$, $\lambda = O\left(\frac{v}{pk}\right)$
3. Matrix transpose[f]	$\Theta\left(\frac{N}{BD}\frac{\log\min(M, r, c, N/B)}{\log(M/B)}\right)$ [2], [47]	$\tau = O\left(\frac{N\log N}{v}\right)$	$\alpha = O\left(\frac{N}{p}\right)$, $\beta = O\left(\frac{N}{p}\right)$
		$\lambda = O(1)$, $M = O\left(\frac{N}{v}\right)$	$\kappa = O\left(\frac{N}{pDB}\right)$, $\lambda = O\left(\frac{v}{pk}\right)$

[a] PDM I/O complexities as listed apply to general values for N, M, D, and B. If the constraints required by our techniques are applied to these results, the term $\log_{M/B}(N/B)$ becomes a constant.
[b] When the parameter D is quoted in the PDM I/O complexity column, this parameter refers to the number of disks overall, whereas the parameter D used in the EM-BSP I/O complexity column refers to the number of disks on each of the p processors.
[c] The BSP running time is $\tau + g\lambda M + \lambda L$, where τ is the parallel computation time, λ is the number of BSP rounds, and M is the peak local memory used by a processor of the BSP machine.
[d] The EM-BSP running time is $\beta + g\alpha + G\kappa + \lambda L$, where λ is the number of EM-BSP rounds, α is the amount of data communicated, β is the parallel computation time, and κ is the number of parallel I/O operations. We restrict each real processor to $O(kN/v)$ memory, for integer $1 \leq k \leq v/p$, and assume that $N \geq v^2B$, $B \geq v/2$, and $v \geq D$.
[e] The PDM I/O complexities listed for Group A apply also to multiple processors when the interconnection method is a shared RAM, hypercubic network, or cube-connected cycles.
[f] r, c are the number of columns and rows, respectively, where $N = r \cdot c$.

Table 2. Summary of new EM algorithms in comparison with previous results.

Problem description	PDM I/O complexity[a,b]	BSP complexity[c]	EM-BSP complexity[d]
		Group B: GIS and computational geometry algorithms[e]	
4. Polygon triangulation, Trapezoidal decomposition, Segment tree construction, Next element search on line segments	$O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ [7]	$\tau = O\left(\frac{N\log N}{v}\right)$ [12] $\lambda = O(1), \quad M = O\left(\frac{N\log N}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O\left(\frac{N\log N}{p}\right)$ $\kappa = O\left(\frac{N\log N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$
5. Batched planar point location	$O\left(\left(\frac{N}{B}+k\right)\log_{M/B}\frac{N}{B}\right)$ [7]	$\tau = O\left(\frac{N\log N}{v}\right)$ [12] $\lambda = O(1), \quad M = O\left(\frac{N\log N}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O\left(\frac{N\log N}{p}\right)$ $\kappa = O\left(\frac{N\log N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$
6. 3D convex hull, 2D Voronoi diagram, Delaunay triangulation[f]	$O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ [30]	$\tau = \tilde{O}\left(\frac{N\log N}{v}\right)$ [20] $\lambda = \tilde{O}(1) \quad M = O\left(\frac{N}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O\left(\frac{N\log N}{p}\right)$ $\kappa = \tilde{O}\left(\frac{N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$
7. Lower envelope of non-intersecting line segments		$\tau = O\left(\frac{N\log N}{v}\right)$ [22] $\lambda = O(1), \quad M = O\left(\frac{N}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O(N/p)$ $\kappa = O\left(\frac{N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$
8. Generalized lower envelope of line segments		$\tau = O\left(\frac{N\log N}{v}\right)$ [22] $\lambda = O(1), \quad M = O\left(\frac{N\alpha(N)}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O(N/p)$ $\kappa = O\left(\frac{N\alpha(N)}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$
9. Area of union of rectangles, 3D-maxima, 2D nearest neighbors	$O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ [30]	$\tau = O\left(\frac{N\log N}{v}\right)$ [22] $\lambda = O(1), \quad M = O\left(\frac{N}{v}\right)$	$\alpha = O\left(\frac{N\log N}{p}\right), \quad \beta = O(N/p)$ $\kappa = O\left(\frac{N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$

10. 2D-weighted dominance counting, Uni- and multidirectional separability		$\tau = O\left(\frac{N \log N}{v}\right)$ [22] $\lambda = O(1), \quad M = O\left(\frac{N}{v}\right)$	$\alpha = O\left(\frac{N \log N}{p}\right), \quad \beta = O(N/p)$ $\kappa = O\left(\frac{N}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk}\right)$	
Group C: Graph algorithms				
11. List ranking, Euler tour of tree, Lowest common ancestor, Tree contraction, Expression tree evaluation	$O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ [13]	$\tau = O\left(\frac{N}{v} \log v\right)$ [11] $\lambda = O(\log v), \quad M = O\left(\frac{N}{v}\right)$	$\alpha = O\left(\frac{N \log v}{p}\right), \quad \beta = O\left(\frac{N \log v}{p}\right)$ $\kappa = O\left(\frac{N \log v}{pDB}\right), \quad \lambda = O\left(\frac{v}{pk} \log v\right)$	
12. Connected components, Spanning forest, Ear and open ear decomposition, Biconnected components[g]	$O\left(\frac{E}{DB} \log_{M/B} \frac{V}{B} \cdot \max\left\{1, \log\log \frac{VBD}{E}\right\}\right)$ [36]	$\tau = O\left(\frac{V+E}{v} \log v\right)$ [11] $\lambda = O(\log v), \quad M = O\left(\frac{V+E}{v}\right)$	$\alpha = O\left(\frac{N \log v}{p}\right), \quad \beta = O\left(\frac{N \log v}{p}\right)$ $\kappa = O\left(\frac{(V+E) \log v}{pDB}\right), \lambda = O\left(\frac{v}{pk} \log v\right)$	

[a] PDM I/O complexities as listed apply to general values for N, M, D, and B. If the constraints required by our techniques are applied to these results, the term $\log_{M/B}(N/B)$ becomes a constant.
[b] When the parameter D is quoted in the PDM I/O complexity column, this parameter refers to the number of disks overall, whereas the parameter D used in the EM-BSP I/O complexity column refers to the number of disks on each of the p processors.
[c] The BSP running time is $\tau + g\lambda M + \lambda L$, where τ is the parallel computation time, λ is the number of BSP rounds, and M is the peak local memory used by a processor of the BSP machine.
[d] The EM-BSP running time is $\beta + g\alpha + G\kappa + \lambda L$, where λ is the number of EM-BSP rounds, α is the amount of data communicated, β is the parallel computation time, and κ is the number of parallel I/O operations. We restrict each real processor to $O(kN/v)$ memory, for integer $1 \le k \le v/p$, and assume that $N \ge v^2 B$, $B \ge v/2$, and $v \ge D$.
[e] The PDM I/O complexities for problems 4, 5, 6, 9, and 11 are documented for the single processor, single disk case. We are not aware of multidisk or multiprocessor extensions. Similarly, we are not aware of any previous multiprocessor algorithm on the PDM for problem 12.
[f] The PDM I/O complexities reported for three-dimensional convex hull, etc., apply to PRAM-like parallel machines.
[g] For a graph of V vertices and E edges.

known lower bounds for sorting, permutation, and matrix transpose in EM. We argue in Section 3.1 that our restricted parameter range is both interesting and useful in the EM domain.

The remainder of this paper is organized as follows. Section 2 describes our deterministic simulation technique for single and multiple processor machines. Section 3 describes a large number of new EM algorithms obtained by simulating existing parallel BSP algorithms for these problems. Additional summary information is included at the beginning of these sections.

2. Deterministic Simulation of BSP Algorithms as EM-BSP Algorithms

In this section we describe a deterministic simulation for BSP-like algorithms whose communication can be characterized by h-relations.

Each communication superstep of the underlying BSP algorithm is divided into a *sending superstep* and a *receiving superstep*. During a sending superstep, messages are generated, and during a receiving superstep they are received. A *compound superstep* is composed of a receiving, a computation, and a sending superstep. The execution of a BSP algorithm proceeds as a series of compound supersteps, and can therefore be simulated by repeated application of the simulation steps for a single compound superstep.

The processors of the BSP machine are called *virtual processors*, and v denotes their number. The *context* of a virtual processor is the local memory it uses, and the *context size* of a virtual processor is the maximum size of its context used during the computation. The maximum context size of all virtual processors is $\mu = \Omega(N/v)$. We denote the maximum size of the data sent or received by any virtual processor over all supersteps by $\gamma = \Theta(N/v)$.

Suppose we have a problem $\mathcal{P}$ with N problem items, for which a BSP algorithm $\mathcal{A}'$ is known on a machine $\mathcal{C}'$ with v processors. We describe a simulation of $\mathcal{A}'$ on a real machine $\mathcal{C}$ with p processors, each having D local disks. The simulation models message transmissions of $\mathcal{A}'$ on $\mathcal{C}'$ by disk I/O on $\mathcal{C}$. The resulting algorithm $\mathcal{A}$ for $\mathcal{P}$ on $\mathcal{C}$ can be characterized by the parameters $(N, p, M, D, B, G, \lambda, g, L)$, where $p \leq v$ is the number of real processors, $M = \Omega(N/v)$ is the size of the local memory on each of the real processors, D is the number of disk drives on each real processor, B is the transfer block size to the disks, g is the time required for a communication operation on the real machine, G is the time for a parallel I/O operation of DB items of $\mathcal{P}$ to the D disks of a local processor, L is the time required for the real processors to synchronize, and λ is the number of supersteps performed by $\mathcal{A}$ on $\mathcal{P}$ and $\mathcal{C}$.

In Section 2.2 we describe a deterministic simulation technique that permits a BSP algorithm with fixed-size messages to be simulated as an EM algorithm on a single processor target machine. Section 2.3 deals with the multiple processor case. Our simulation techniques replace communication between processors by disk I/O, and so an important issue is how to organize the generated messages on the D disks so that they can be accessed using blocked and fully parallel I/O operations. Both Sections 2.2 and 2.3 assume that the BSP algorithm in question communicates using fixed-size messages. This leads to a simple simulation approach. A common issue in both cases is how to handle irregular h-relations. A method for handling this issue is the topic of Section 2.1.

Section 2.4 presents our generalized simulation result, incorporating the simulation ideas of Section 2.3 and the balancing of message sizes from Section 2.1.

2.1. *Balancing Communication*

Consider a single compound superstep of the original parallel algorithm. The communication in such a superstep may be highly irregular, meaning that a given processor may send messages of widely differing sizes to the other processors. In simulating the message traffic we intend to reserve space for each message on disk and replace message sending/receiving by disk writes/reads. We cannot, however, simply use the upper bound on message length as an estimate of the message length, as the maximum length may be $O(N/v)$. If we reserve $O(N/v)$ space on disk for each message, we will need $O(vN)$ space in total, which would be prohibitive.

We therefore borrow a parallel algorithm due to Bader et al. [9] which balances the message sizes at the cost of two rounds of communication, and gives us a much more usable upper bound on the message size. We will show that it also gives us a useful lower bound on message size that permits blocked disk I/O in the EM domain.

Algorithm: BalancedRouting (from [9])

Input: Each of the v processors has $\bar{n}/v$ elements, which are divided into v messages, each of arbitrary length $\leq \bar{n}/v$. Let msg_{ij} denote the message to be sent from processor i to processor j, and let $|msg_{ij}|$ be the length of such a message.

Output: The v messages in each processor are delivered to their final destinations in two balanced rounds of communication, and each processor then contains at most $\bar{h}$ data.

A. For $i = 0$ to $(v - 1)$ in parallel
   1. Processor i allocates v local bins, one for each processor
   2. For $j = 0$ to $(v - 1)$
   3. For $\ell = 0$ to $|msg_{ij}|$
      Processor i allocates the ℓth word of msg_{ij} to local bin $(i + j + \ell) \bmod v$
   4. Processor i sends bin j to processor j

Superstep Boundary

B. For $j = 0$ to $(v - 1)$ in parallel
   1. Processor j reorganizes the messages it received in Step 2 (of Superstep A) into bins according to each element's final destination
   2. Processor j routes the contents of bin k to processor k, for $0 \leq k \leq v - 1$

Algorithm BalancedRouting gives us a technique for acquiring messages of nearly uniform size. We will show upper and lower bounds on the message size, which allow us to save and retrieve messages on the disks in an efficient and convenient manner.

A total of $\bar{n}$ data is divided evenly among v processors. Each item is labeled with the identifier of a destination processor in such a way that no more than $\bar{h}$ items have the same destination, where $\bar{h} \geq \bar{n}/v$. Algorithm BalancedRouting delivers these items to their destination in two rounds of communication, where no message differs in size from any other by more than $v - 1$ items.

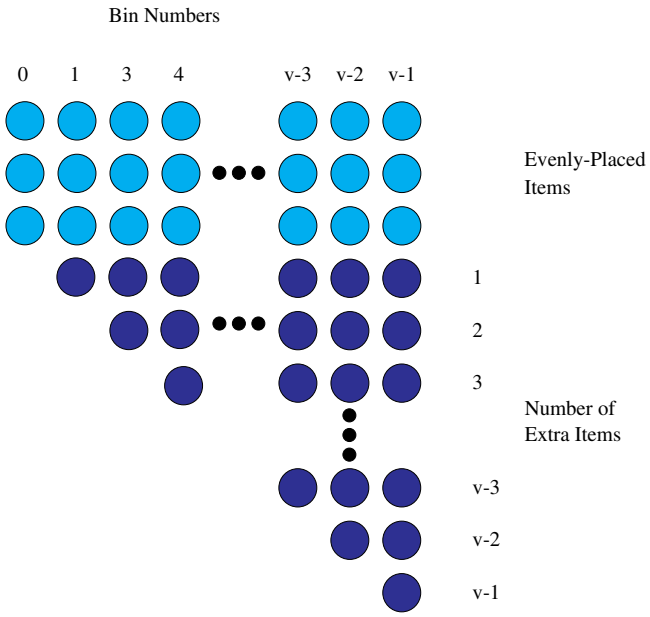

Fig. 3. Illustration of maximum imbalance in the bin sizes of a processor during Superstep A. The light circles represent evenly placed elements and the dark circles are unevenly placed, or "extra," ones. The maximum bin size (bin $v-1$ in the diagram) is at most $(v-1)/2$ more than the average, and the minimum bin size (bin 0 in the diagram) is at most $(v-1)/2$ less than the average.

Theorem 1. *We are given v processors and $\bar{n}$ data items. Each processor has exactly $\bar{n}/v$ data to be redistributed among the processors, and no processor is to be the recipient of more than $\bar{h}$ data. The redistribution can be accomplished in two communication rounds of balanced communication*: (*A*) *messages in the first round are at least $\bar{n}/v^2 - (v-1)/2$, and at most $\bar{n}/v^2 + (v-1)/2$ in size, and* (*B*) *messages in the second round are at least $\bar{h}/v - (v-1)/2$, and at most $\bar{h}/v + (v-1)/2$ in size.*

Proof. The proof of the maximum message sizes is given by Bader et al. [9]. The proof of the minimum message sizes relies on the following observation (see Figure 3): if $bin_{\min}$ is the smallest bin created at a processor in Step 1 of Superstep A, then the other $(v-1)$ bins can contain at most $1+2+\cdots+(v-1) = v(v-1)/2$ more elements than does $bin_{\min}$ (see Figure 3).

In round A each processor initially has $\bar{n}/v$ data. At the end of Superstep B, each processor will have at most $\bar{h}$ data.

First, we consider the minimum message size in Superstep A. Due to the round robin allocation mechanism, a given bin after Step 1 will contain at most one more element of a message to processor j than does any other bin. We fix any processor i. Consider the bin sizes after all of the messages have been distributed among the bins by processor i (see Figure 3). Clearly, all of the bins will contain at least as many elements as the smallest bin, $bin_{\min}$. Let e_j be the number of extra elements (more than this minimum) in bin j at Step 2. The crucial observation is that if $bin_{\min}$ is the smallest bin, then

the other $(v-1)$ bins can hold at most $1+2+\cdots+(v-1) = v(v-1)/2$ extra elements.

Thus,

$$\frac{\bar{n}}{v} = v|\underset{\min}{bin}| + \sum_j e_j.$$

Since $v(v-1)/2 \geq \sum_j e_j$, we have

$$\frac{\bar{n}}{v} \leq v|\underset{\min}{bin}| + \frac{v(v-1)}{2},$$

$$|\underset{\min}{bin}| \geq \frac{\bar{n}}{v^2} - \frac{v-1}{2}.$$

We now turn to the message sizes in Superstep B. The elements which arrive at processor j as a result of Step 2 are the contents of the jth local bins formed in Step 1 at each of the processors 0 through $v-1$. We can think of the jth local bin of each of the v processors as a component of a single, global superbin, which is the union of the jth local bins of all v processors. Consider only the messages destined for a fixed processor k which are held by each processor i, $0 \leq i \leq v-1$, prior to Step 1. These are allocated among the superbins, starting with superbin $(i+k)$ mod v by Step 1. Superbin j now contains the message which is to be sent from processor j to processor k in Step 4.

In a similar manner to the analysis of Superstep A, let E_j be the number of extra elements in superbin j after Step 1. Let $sbin_{\min}$ be the superbin which contains the minimum number of elements after Step 1, and hence $|sbin_{\min}|$ represents the minimum message size in Step 4. When processor k is one of the processors which receives the maximum h data elements, we have $\bar{h} = v|sbin_{\min}| + \sum_j E_j$, and since $v(v-1)/2 \geq \sum_j E_j$, we have $|sbin_{\min}| \geq \bar{h}/v - (v-1)/2$. □

The notion of an h-relation is often used in the analysis of parallel algorithms based on *BSP-like* models (e.g. BSP, BSP*, CGM). An h-relation is a communication superstep in which each of the v processors sends and receives at most h data items. It is typically used in bounding the communication complexity in an asymptotic analysis. Based on this usage of an h-relation, we have:

Corollary 1. *An arbitrary h-relation can be replaced by two balanced h-relations whose message size is bounded by $h/v - (v-1)/2$ and $h/v + (v-1)/2$.*

Proof. In Theorem 1, clearly $\bar{h} \geq \bar{n}/v$. For $h = \bar{h} \geq \bar{n}/v$ we have a message size of at most $h/v + (v-1)/2$ for each of the two rounds of communication. We can reproduce the precise conditions of Theorem 1 by adding dummy items if necessary in Superstep A to ensure that $\bar{n}/v = h$.

We can assume a minimum message size of $h/v - (v-1)/2$ in the second round because the cost of communication is bounded by the assumption of an h-relation. When every processor is the destination of h data, it does not affect the worst case complexity of the superstep. We can therefore assume that every processor receives h data (by adding

dummy items for the sake of the argument). Hence the minimum message size for any processor in Superstep B becomes $h/v - (v-1)/2$ without affecting the asymptotic communication cost of the superstep. □

Lemma 1. *An arbitrary minimum message size $b_{\min}$ can be assured provided that*

$$N \geq v^2 b_{\min} + \frac{v^2(v-1)}{2}, \tag{1}$$

where N is the total number of problem items (*summed over the v processors*).

Proof. From Corollary 1, we can achieve a minimum message size $b_{\min}$ provided that $b_{\min} \leq N/v^2 - (v-1)/2$. □

Assurances regarding the minimum message size are particularly relevant to the BSP* model. In Section 2.5 we discuss the use of algorithm BalancedRouting and Theorem 1 for creating BSP* algorithms from BSP algorithms.

In Sections 2.2 and 2.3 we describe the simulation of BSP algorithms with the additional properties that algorithm BalancedRouting provides. Not every BSP algorithm will require balancing, but Lemma 2 ensures that we can obtain balanced message sizes when necessary by increasing the number of supersteps by a factor of 2.

Lemma 2. *Let $\mathcal{A}'$ be a BSP algorithm with N data, v processors, and λ communication steps, where $h = N/v$ in every step. The λ communication steps of $\mathcal{A}'$ can be replaced by 2λ steps of balanced communication in which the minimum message size is $\Omega(B)$ and the maximum message size is $2 \cdot (N/v^2)$ provided that $N \geq v^2 B + v^2(v-1)/2$.*

Proof. The minimum and maximum message sizes follow from Corollary 1, with $h = N/v$, and the constraint that $N/v^2 + (v-1)/2 \leq 2 \cdot (N/v^2)$. This is true if $N \geq v^2(v-1)/2$, which is absorbed by (1) from Lemma 1. □

In many practical EM situations $2(N/v^2)$ will be a significant overestimate of the maximum message size, as often $v \ll N/v^2$.

2.2. *Single Processor Target Machine*

We now turn to the actual simulation results, which rely on a message size of $c \cdot B$, for a known constant $c \geq 1$. For irregular h-relations, this requirement can be met by applying algorithm BalancedRouting (see Lemma 2), for $B = N/\bar{v}^2$ and $c = 2$. (Note that not every algorithm will require balancing; see AlgorithmTranspose in Section 3.6 for an example of a BSP algorithm whose messages are already balanced.)

For the case of a single EM processor, we simulate a compound superstep of a BSP algorithm $\mathcal{A}$, using the algorithm SeqCompoundSuperstep, shown below. No real communication is required. Lemma 3 gives the resulting complexities for a single compound superstep, and Theorem 2 summarizes the overall simulation result for a single real processor. For ease of exposition, we assume that k divides v.

Lemma 3. *A compound superstep of a v-processor BSP algorithm $\mathcal{A}'$ with computation time $\tau + L$, communication time $g \cdot O(N/v) + L$, message size $c \cdot B$, for a known constant $c \geq 1$, and local memory size μ can be simulated as a single processor EM-BSP algorithm $\mathcal{A}$ in computation time $v\tau + O(v\mu)$ and I/O time $O(G(N/BD + v\mu/BD))$ provided that $M \geq k\mu + BD$, and $N = \Omega(\bar{v}BD)$ for an arbitrary integer $k \leq v$, and $\bar{v} = v/k$.*

Proof. Since messages are at most $c \cdot B$ in size we can allocate fixed-sized slots for them on the disks while preserving an $O(v\mu)$ disk space requirement. The assurance of minimum message size $\Omega(B)$ further implies that I/O operations will be blocked.

We simulate a compound superstep of BSP algorithm $\mathcal{A}'$ using algorithm SeqCompoundSuperstep, shown below. The algorithm expects the input messages to the virtual processors in the current superstep to be organized (by destination) in a parallel format on the disks, and it also writes the messages generated in the current superstep to the disks in a parallel format. We use the phrase "a *parallel format*" to mean an arrangement of the data that permits fully parallel access to the disks, both for writing the messages, and for reading them back in a different order in the next superstep. The *consecutive* and *staggered* formats, defined below, are parallel formats.

Consecutive format. We say that a disk read/write operation on D blocks is *consecutive* when the qth block, $0 \leq q \leq D$, is read/written from/to disk $(d + q) \bmod D$ on track $T_0 + \lfloor (d+q)/D \rfloor$, where T_0 is the track used for the first of the D blocks to be read/written, and d is the disk offset (from disk 0) for the first of the D blocks to be read/written.

Staggered format. We say that a disk read/write operation on D blocks involving n messages, each of size at most b' blocks, is *staggered* when the qth block, $0 \leq q \leq (b' - 1)$, of the jth message, $0 \leq j \leq (n - 1)$, is read/written from/to disk $(d + S + q) \bmod D$ on track $T_0 + \lfloor (d + S + q)/D \rfloor$, where T_0 is the track used for the blocks of the 0th message, d is the disk offset (from disk 0) for the first of the D blocks to be read/written, and $\lceil S/D \rceil$ is the number of tracks by which consecutive messages are to be staggered (separated).

Algorithm SeqCompoundSuperstep simulates a compound superstep of a v-processor BSP on a single processor EM-BSP with D disks. It simulates k processors at a time, where $1 \leq k \leq v$, provided that the simulating processor has $M = \Theta(k\mu)$ memory.

Algorithm: SeqCompoundSuperstep

Input: For each $j \in \{0, \ldots, v - 1\}$ the blocks of the context are stored on the disks in consecutive format, and the arriving messages of virtual processor j are spread over the D disks consecutive format.

Output: (i) The (changed) contexts of the v simulated processors are spread across the disks in consecutive format. (ii) The generated messages for each processor in the next superstep are stored in consecutive format on the disks.

1. For $j = 0$ to $v/k - 1$
   (a) Read the context of virtual processors jk to $(j + 1)k - 1$ from the disks into memory.
   (b) Read the packets received by virtual processors jk to $(j + 1)k - 1$ from the disks.

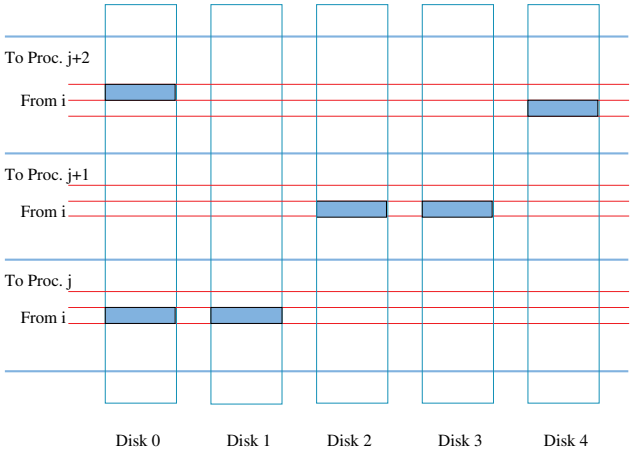

Fig. 4. llustration of the layout of message blocks on the D disks. In the example we have $D = 5$ and message size $b' = 2$ blocks. Messages from processor i to processors j, $j+1$, and $j+2$ are shown as shaded rectangles. Messages to consecutively numbered processors are staggered on the disks to permit D blocks to be written in parallel.

(c) Simulate the local computation of virtual processors jk to $(j+1)k-1$.
(d) Write the packets which were sent by virtual processors jk to $(j+1)k-1$ to the D disks in the staggered format illustrated in Figure 4. See below for details.
(e) Write the changed context of virtual processors jk to $(j+1)k-1$ back to the D disks (in consecutive format).

Details of Steps (*a*) *and* (*e*): We reserve an area of total size $v\mu$ on the disks, $v\mu/BD$ blocks on each disk, where we store the contexts. We split the context V_j of virtual processor j into blocks of size B and store the ith block of V_j on disk $(i + j(\mu/B)) \bmod D$ using track $\lfloor (i + j(\mu/B))/D \rfloor$. Since the context of each processor is now in striped format on the disks, we can read and write the contexts of k consecutive virtual processors using D disks in parallel for every I/O operation.

Details of Step (*b*): The previous compound superstep guaranteed that the blocks which contain the messages destined for the current processor are stored in consecutive format on the disks. Therefore, we can use a similar technique to fetch the messages as we used to fetch the contexts.

Details of Step (*d*): After the Computation Phase, all messages sent by the current k virtual processors have been generated and stored in internal memory. The coarse-grained nature of the underlying BSP algorithm results in large messages (see Lemma 2) which are as long or longer than the block size B. We cut the messages into blocks of size B. Each block inherits the destination address from its original message. In $k\gamma/BD$ rounds, we write the blocks out to the disks, as described in detail below. Recall that γ is the maximum size of the data sent or received by any virtual processor over all supersteps.

Let b represent the maximum message size, and let b' represent the maximum number of disk blocks per message. Hence, $b' = \lceil b/B \rceil$. Let msg_{ij} represent the message sent

from processor v_i to processor v_j in one communication superstep. We store the messages destined for a fixed processor j in consecutive format, beginning with msg_{0j} and ending with $msg_{v-1,j}$. We ensure that the first block of $msg_{i,j+1}$ is assigned to disk $(b_0 + b') \bmod D$, for $0 \leq j \leq v-2$, where b_0 is the disk number of the first block for msg_{ij}. In other words, the starting block positions for messages to consecutive processors are appropriately staggered on the disks to ensure that we can write blocks of messages to consecutively numbered processors in a single parallel I/O when $b' \bmod D \neq 0$. Let $T_j = j \cdot \lceil vb'/D \rceil$ be the track offset for msg_{0j} (the first such message destined for processor v_j). Let $d_j = jb' \bmod D$ be the disk offset (from disk 0) for the first block of msg_{0j}. The qth block of msg_{ij} is assigned to disk $(d_j + ib' + q) \bmod D$ on track $T_j + \lfloor (d_j + ib' + q)/D \rfloor$. This storage scheme maintains what we call the *messaging matrix* across the parallel disks. The messages destined to a particular virtual processor are stored in a band, or stripe, of consecutive parallel tracks.

Outgoing message blocks are placed in a FIFO queue for servicing by procedure DiskWrite. DiskWrite removes at most D blocks from the queue in each write cycle and writes them to the disks in a single write operation. Blocks are serviced strictly in FIFO order. Blocks will be added to the current write cycle and removed from the queue until a block is encountered whose disk number conflicts with that of an earlier block in the current write cycle.

Since the messages destined for any given processor are stored in consecutive format on the disks, we can read the messages received by a virtual processor using D disks in parallel for every I/O operation. Except possibly for the last, each parallel read performed by the simulation of processor v_j will obtain D message blocks. By staggering the first message blocks for consecutive virtual processors across the disks, we can achieve fully parallel writes to the disks.

The scheme just described requires two copies of the messaging matrix because the messages generated by virtual processor i in compound superstep ℓ must be stored on disk before virtual processor $i+1$ can process the messages generated for it in compound superstep $\ell - 1$.

We can avoid this extra space requirement, however, by alternating between {consecutive reads, staggered writes} and {staggered reads, consecutive writes} in successive compound supersteps. This allows the simulation to achieve fully parallel I/O on all message blocks with a single copy of the messaging matrix.

Algorithm SeqCompoundSuperstep loads k virtual processors into the real memory at once, requiring that $M \geq k\mu$. Since the messages sent or received in a superstep by a virtual processor are $h = O(N/v)$ in total size, we require that $kN/vB = \Omega(D)$ to ensure that our I/O scheme for messages is efficient. This means that $N = \Omega(\bar{v}BD)$, where $\bar{v} = v/k$.

Computation time. Steps (a) and (e) of algorithm SeqCompoundSuperstep take $O(v\mu)$ computation time overall. In Steps (b) and (d), $O(N/v)$ message data is routed for each virtual processor. Over all v processors, this adds $O(N)$ computation time overall, which can be ignored. Step (c) consumes $v\tau$ computation time.

I/O Time. Steps (a) and (e) consume $O(G(v\mu/BD))$ time, and Steps (b) and (d) consume $O(G(N/BD))$ time overall. Since $O(N/p)$ message data is sent in each superstep, and $N/p \leq \mu$ we have time $O(G(v\mu/BD))$ due to I/O, overall.

Thus, overall, the computation time is $v\tau + O(v\mu)$ and the I/O time is $O(G(v\mu/BD))$. □

Theorem 2. *A v processor BSP algorithm $\mathcal{A}'$ with λ supersteps, local memory size μ, running time $\beta + g \cdot O(N/v) + \lambda L$, and message size $c \cdot B$ for some known constant $c \geq 1$, can be simulated as a single processor EM-BSP algorithm $\mathcal{A}$ with time $v\beta + O(\lambda v\mu) + G \cdot O(\lambda(v\mu/BD))$ for $M \geq k\mu + BD$, $N = \Omega(\bar{v}^2 B)$, and $v = \Omega(D)$, for an arbitrary integer $1 \leq k \leq v$, and $\bar{v} = v/k$.*

In particular, algorithm $\mathcal{A}$ is c-optimal if $\mathcal{A}'$ is c-optimal, $\beta = \omega(\lambda\mu)$, and $G = o(\beta BD/\lambda\mu)$. Furthermore, algorithm $\mathcal{A}$ is work-optimal and I/O-efficient if $\mathcal{A}'$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$, and $G = O(\beta BD/\lambda\mu)$.

Proof. We use the results of Lemma 3. The computation time required to simulate the computation steps of $\mathcal{A}'$ is $v\beta$. The computational overhead associated with the I/O steps (Steps (a), (b), (d), (e)) is $O(\lambda v\mu) + O(\lambda N)$. Since $v\mu > N$ the total computation time is bounded by $v\beta + O(\lambda v\mu)$. When c-optimality is required, we therefore need $\lambda v\mu = o(v\beta)$, or $\beta = \omega(\lambda\mu)$. Note that when $\mu = \Theta(N/v)$, we can substitute $\beta = \omega(\lambda N/v)$ for $\beta = \omega(\lambda\mu)$. For work-optimality, we require that $\lambda v\mu = O(v\beta)$, or $\beta = \Omega(\lambda\mu)$.

The number of I/O operations (Steps (a), (b), (d), (e)) is $O(\lambda(v\mu/BD)) + O(\lambda(N/D))$, which is bounded by $O(\lambda(v\mu/BD))$. For c-optimality, we require the I/O time to be in $o(v\beta)$, which means that $G = o(\beta BD/\lambda\mu)$. For I/O-efficiency, we require the I/O time to be in $O(v\beta)$, which means that $G = O(\beta BD/\lambda\mu)$.

The messages sent by a group of k virtual processors to any other group must be large enough in total that they fill a disk block, so $N = \Omega(\bar{v}^2 B)$. Also, the messages sent or received by a group in a single superstep must fill a track of the D parallel disks, meaning that $N = \Omega(\bar{v}BD)$, where $\bar{v} = v/k$. These constraints can be combined into $N = \Omega(\bar{v}^2 B)$, for $v = \Omega(D)$. □

2.3. *Multiple Processor Target Machine*

For the case of $p \geq 1$ processors on the EM-BSP machine we simulate a compound superstep of a BSP algorithm $\mathcal{A}'$ using the algorithm ParCompoundSuperstep, shown below. Unlike in the case of a single real processor, we are now forced to perform real communication between the processors of the target machine. Fortunately, provided that the v virtual processors are evenly divided among the p real processors, and we use the technique of Lemma 2, such communication is balanced. The simulation need not introduce a randomizing round of communication for this purpose as is done in [21].

Each real processor i, $0 \leq i \leq p-1$, executes algorithm ParCompoundSuperstep in parallel. For ease of exposition, we assume that pk divides v.

Algorithm: ParCompoundSuperstep

Objective: Simulation of a compound superstep of a v-processor BSP on a p-processor EM-BSP.

Input: The message and context blocks of the virtual processors are divided among the real processors and their local disks. Each real processor i, $0 \leq i \leq (p-1)$,

holds $O(N/pB)$ blocks of messages and $v\mu/pB$ blocks of context, and each local disk contains $O(N/pBD)$ blocks of messages and $O(v\mu/pBD)$ blocks of context.

Output: The changed contexts and generated messages distributed as required for the next compound superstep.

1. For $j = 0$ to $v/pk - 1$ do
   (a) Read the contexts of virtual processors ijk to $i(j+1)k-1$ from the local disks.
   (b) Read any message blocks addressed to virtual processors ijk to $i(j+1)k-1$ from the local disks.
   (c) Simulate the computation supersteps of virtual processors ijk to $i(j+1)k-1$, collecting all generated messages in the local internal memory.
   (d) Send all generated messages to the required (real) destination processors. Upon arrival, the messages are arranged within the internal memory of the real destination processor and then written to its disks as in the single processor simulation; see algorithm SeqCompoundSuperstep.
   (e) Write the contexts for virtual processors ijk to $i(j+1)k-1$ back to the local disks; see algorithm SeqCompoundSuperstep.

Lemma 4. *A compound superstep of a v-processor BSP algorithm $\mathcal{A}'$ with computation time $\tau + L$, communication time $O(g(N/v)) + L$, message size $c \cdot B$ for some known constant $c \geq 1$, and local memory size μ can be simulated as $\bar{v}/p$ compound supersteps of a p-processor EM-BSP algorithm $\mathcal{A}$ in parallel computation time $(v/p)\tau + O((v/p)\mu) + (\bar{v}/p)L$, communication time $g \cdot O(N/p) + (\bar{v}/p)L$, and I/O time $G(v/p) \cdot O(\mu/BD) + (\bar{v}/p)L)$, for $pk \leq v$, arbitrary integer $1 \leq k \leq v/p$, $N = \Omega(\bar{v}^2 B)$, $v = \Omega(D)$, and $\bar{v} = v/k$ provided $M \geq k\mu + BD$ memory is available on each real processor.*

Proof. There are v/p virtual processors resident on each real processor. Each real processor simulates v/p virtual processors of a round of $\mathcal{A}'$ in a total of $v/pk = \bar{v}/p$ supersteps, as k virtual processors are simulated on each real processor in each round of $\mathcal{A}$.

Communication time. In Step (d), in each round of $\mathcal{A}$, each real processor receives

- p real messages, one from each real processor,
- $pk \cdot (v/p)$ virtual messages, one from each of pk virtual processors simulated in this round, to each of its own v/p resident virtual processors,
- $pk \cdot (v/p) \cdot (N/v^2) = kN/v$ data items, since each virtual message is N/v^2 items in size.

It also sends kN/v data in each round of $\mathcal{A}$. We have v/pk such rounds, so we have $(v/pk) \cdot 2k(N/v) = 2(N/p)$ data to communicate overall. Therefore, the overall communication time of $\mathcal{A}$ is $g \cdot O(N/p) + (v/pk)L$.

I/O time. The I/O time is determined by the cost of swapping contexts plus the cost of simulating the original messaging via I/O. Each group of k processors has context of size $O(k\mu)$, which requires $O(k\mu/BD)$ I/O operations to swap in or out of memory. Over v/pk supersteps, the swapping of contexts therefore costs $G(v/pk) \cdot O(k\mu/BD)$.

The I/O costs due to messaging between virtual processors are bounded by $O(kN/vBD)$ in each superstep, and so the total costs due to virtual messaging over v/pk supersteps is $G(v/pk) \cdot O(kN/vBD)$. Since the context size $\mu = \Omega(N/v)$, the total I/O cost overall is dominated by $G \cdot (v/p)(\mu/BD)$.

Computation time. We have $p \leq v$ real processors, so the time to simulate Step (c) is $(v/p)\tau$. Computational overhead is contributed by $O((v/pk)(k\mu + kN/v))$, due to swapping of contexts (Steps (a), (e)) and messaging I/O (Steps (b), (d)). As before, the computational overhead is dominated by the cost of swapping contexts. □

Lemma 5 (Lower Bound on Number of Rounds). *The number of rounds of algorithm ParCompoundSuperstep per round of the client algorithm is v/pk.*

Proof. In order to pass the problem through internal memory even once, each processor of the real machine "touches" $\Omega(N/p)$ data. This can be done in no fewer than $(N/p)/M$ rounds, where M is the internal memory size of a real processor. Since we have $M = kN/v$, the number of rounds is $\Omega(v/pk)$. □

Theorem 3. *A v processor BSP algorithm $\mathcal{A}'$ with λ supersteps, computation time $\beta + \lambda L$, communication time $g\lambda(N/v) + \lambda L$, local memory size μ, and message size $b = c \cdot B$ for some known constant $c \geq 1$ can be simulated as a p-processor EM-BSP algorithm $\mathcal{A}$ with computation time $(v/p)\beta + (v/p)O(\lambda\mu) + (\bar{v}/p)\lambda L$, communication time $g\lambda O(N/p) + (\bar{v}/p)\lambda L$, and I/O time $G\lambda(v/p)O(\mu/BD) + (\bar{v}/p)\lambda L$ for $M \geq k\mu + BD$, $p \leq v$, $N = \Omega(\bar{v}^2 B)$, and $v = \Omega(D)$, for arbitrary integer $1 \leq k \leq v/p$, and $\bar{v} = v/k$.*

Let $g(N), L(N)$, and $v(N)$ be increasing functions of N. If $\mathcal{A}'$ is c-optimal on the BSP for $g \leq g(N)$, $L \leq L(N)$, and $v \leq v(N)$, then $\mathcal{A}$ is a c-optimal EM-BSP algorithm for $\beta = \omega(\lambda\mu)$, $g \leq g(N)$, $G = o(\beta BD/\mu\lambda)$, and $L \leq L(N) \cdot (p/\bar{v})$. $\mathcal{A}'$ is work-optimal, communication-efficient, and I/O-efficient if $\mathcal{A}'$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$, $g \leq g(N)$, $G = O(\beta BD/\mu\lambda)$, and $L \leq L(N) \cdot (p/\bar{v})$.

Proof. We use the results of Lemma 4. The computation time required to simulate the computation steps of $\mathcal{A}'$ is $(v/pk)k\beta$. The computational overhead associated with the I/O and communication steps (Steps (a), (b), (d), (e)) is $O((v/p)\lambda\mu + (v/pk)\lambda(N/v))$ from Lemma 4. Since $\mu \geq N/v$, the total computation time is bounded by $(v/p)\beta + O((v/pk)\lambda\mu)$. When c-optimality is required, we need $\beta = \omega(\lambda\mu)$. Note that in many cases $N/v = \Theta(\mu)$. Also, when only work-optimality is required, $\beta = \Omega(\lambda\mu)$ suffices.

From Lemma 4, the communication time of the simulation per superstep of $\mathcal{A}'$ is $O(g(N/p) + (v/pk)L)$, giving $g\lambda(N/p) + (v/pk)\lambda L$ time overall.

The I/O time (Steps (a), (b), (d), (e)) is $G\lambda(v/p) \cdot O(\mu/BD) + N/vBD))$, which is bounded by $G\lambda(v/p)O(\mu/BD)$. For c-optimality, we require the I/O time to be in $o((v/p)\beta)$, which means that $G = o(\beta BD/\lambda\mu)$. For I/O-efficiency we need only that $G = O(\beta BD/\lambda\mu)$. Since the number of supersteps increases by a factor of v/pk we require that $L \leq L(N) \cdot (pk/v)$. □

2.4. *General Simulation Result*

Theorem 4 states that Theorem 3 holds even if $\mathcal{A}'$ has unknown message size, provided that, in addition to the constraints of Theorem 3, $B \geq v/2$, $N \geq \bar{v}^2 B$, and $v \geq D$.

Theorem 4. *A v processor BSP algorithm $\mathcal{A}'$ with λ supersteps, computation time $\beta + \lambda L$, communication time $g\lambda(N/v) + \lambda L$, and local memory size μ can be simulated as a p-processor EM-BSP algorithm $\mathcal{A}$ with computation time $(v/p)\beta + (v/p)O(\lambda\mu) + (\bar{v}/p)\lambda L$, communication time $g\lambda O(N/p) + (\bar{v}/p)\lambda L$, and I/O time $G\lambda(v/p)O(\mu/BD) + (\bar{v}/p)\lambda L$ for $M \geq k\mu + BD$, $p \leq v$, $N = \Omega(\bar{v}^2 B)$, $v = \Omega(D)$, and $B \geq v/2$, for arbitrary integer $1 \leq k \leq v/p$, and $\bar{v} = v/k$.*

Let $g(N), L(N)$, and $v(N)$ be increasing functions of N. If $\mathcal{A}'$ is c-optimal on the BSP for $g \leq g(N)$, $L \leq L(N)$, and $v \leq v(N)$, then $\mathcal{A}$ is a c-optimal EM-BSP algorithm for $\beta = \omega(\lambda\mu)$, $g \leq g(N)$, $G = o(\beta BD/\mu\lambda)$, and $L \leq L(N) \cdot (p/\bar{v})$. $\mathcal{A}'$ is work-optimal, communication-efficient, and I/O-efficient if $\mathcal{A}'$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$, $g \leq g(N)$, $G = O(\beta BD/\mu\lambda)$, and $L \leq L(N) \cdot (p/\bar{v})$.

Proof. We use algorithm BalancedRouting, which ensures message size $b \leq N/v^2$, provided that $N \geq v^2 b + v(v-1)/2$ (Lemma 2). If $b = B$, and $B \geq v/2$, this condition is preserved by $N \geq v^2 B$ and $v \geq D$ which also satisfy Theorem 3. □

2.5. *Summary*

The result of Corollary 1 can be applied to any algorithm which communicates exclusively via h-relations. The concept of an h-relation is relevant primarily with respect to whether it is an assumption in the analysis of the algorithm in question. Typically, a good algorithm has been shown to be asymptotically optimal when the communication volume to and from each processor is bounded by h in each superstep. Using Lemma 1 we can additionally ensure any desired minimum communication block size of b at the cost of at most doubling the number of communication rounds for problems with sufficient slackness. Here we use the term communication to mean either I/O or conventional message passing.

The main results of our simulation technique are as follows:

1. Using Theorem 3, BSP algorithms which have both BSP* and CGM properties can be converted to EM-BSP*/EM-CGM algorithms, provided that b, B are in $\Omega(N/v^2)$, $v = \Omega(D)$.
2. Using Theorem 4, BSP algorithms which have CGM properties can be converted to EM-BSP*/EM-CGM algorithms, provided that b, B are in $\Omega(N/v^2)$, $v = \Omega(D)$, and $B \geq v/2$.
3. Using Corollary 1, CGM algorithms can be converted to BSP* algorithms with $b = O(N/v^2)$.
4. Using Corollary 1, conforming BSP algorithms can be converted to BSP* algorithms with $b \leq h_{\min}/v - (v-1)/2$, where $h_{\min}$ is the minimum value of h used in any communication superstep.

The term "conforming" in item 4 above refers to the need for the bounding concept of an h-relation to be a universal assumption in the analysis of the original BSP algorithm

for each of its communication rounds. It is convenient, but not necessary, that the same value of h be used in every round.

3. New EM Algorithms

A number of basic algorithms with applications in data structuring, computational geometry, and graph theory have been developed for BSP models by various authors. In this section we present adaptations of a selection of these algorithms for the EM-BSP models.

In many cases the I/O complexities of these EM-BSP algorithms appear at first glance to contradict known lower bounds for their problems. In Section 3.2, we explain this contradiction by discussing the effect of our constraints on the lower bounds.

In Section 3.3 we present simple BSP algorithms for the fundamental problems of sorting, permutation, and matrix transpose, and describe their adaptations to our EM-BSP models. In Tables 1 and 2 we summarize the performance of these and a number of other new EM-BSP algorithms which we obtained from known BSP algorithms using the simulation techniques of this paper.

3.1. *Practicality of Our Parameter Bounds*

The parameter space for EM problems which we propose in this paper is both practical and interesting. The logarithmic term in the I/O complexity of sorting is bounded by a constant c if $(M/B)^c \geq N/B$, where $M = N/v$. Since this constraint involves the parameters v, B, N, and c, we have a four-dimensional constraint space. For practical purposes, the parameter B can be fixed at about 10^3 for disk I/O (see Figure 5) [43]. This reduces the parameter space to three dimensions. We plot the surface $N^{c-1} = v^c B^{c-1}$ in Figure 6. Any point on or above the surface represents a valid set of parameters for the elimination of the logarithmic factor. It can be seen from Figure 6 that the logarithmic

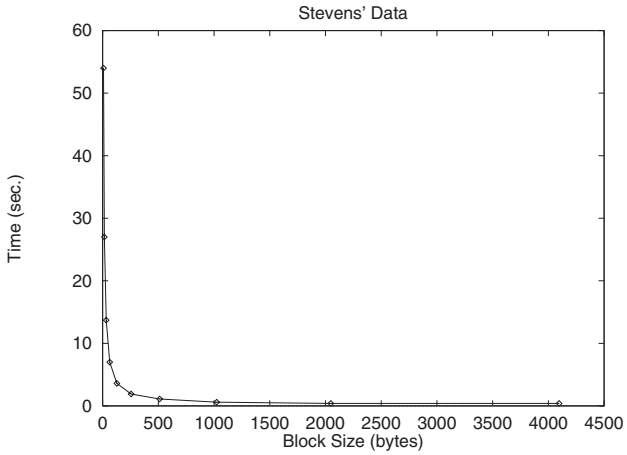

Fig. 5. Stevens' measurements on the effects of varying the block-size.

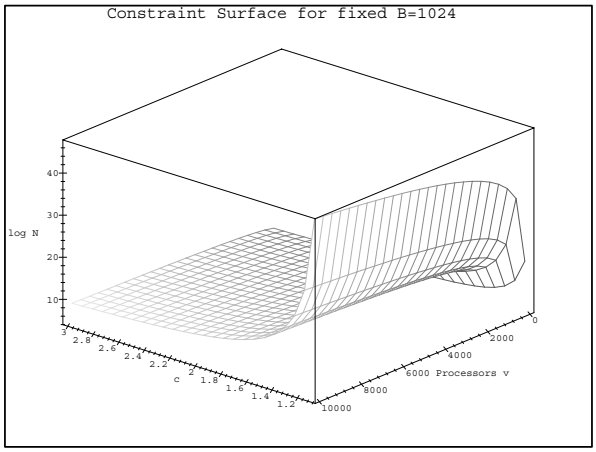

Fig. 6. The surface $N^{c-1} = v^c B^{c-1}$.

factor can be replaced by a constant $c = 2$ for as many as $v = 10{,}000$ processors, provided that the problem size is approximately 100 giga-items or more. For a larger constant, say $c = 3$, the problem size need only be 1 giga-item for $v = 10{,}000$. It can also be seen from Figure 6 that for a smaller numbers of processors the necessary problem size for $c = 2$ is much smaller. This can be seen more clearly in Figure 7 which represents the same data as Figure 6, but for fixed $c = 2$. For 100 processors or less, for instance, we see from Figure 7 that any problem size greater than about 10 mega-items is sufficient.

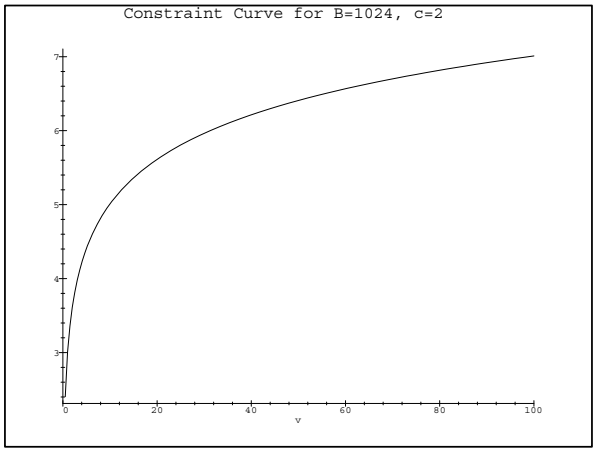

Fig. 7. Two-dimensional projection of Figure 6 for fixed $c = 2$, clipped to $v \leq 100$. The vertical axis is $\log_{10} N$.

3.2. *Apparent Reductions in I/O Complexity*

Several known lower bounds on I/O complexity contain a multiplicative factor of $\log_{M/B}(N/B)$. We obtain a number of results where this factor does not appear. This can be explained by the fact that the term $\log_{M/B}(N/B)$ is a constant when $N \geq v^{1+\varepsilon}b$, for $0 < \varepsilon \leq 1$, as shown by Lemma 6.

For instance, Lemma 6, together with Aggarwal and Vitter's lower bound of $\Omega((N/BD)\ \log_{M/B}(N/B))$ I/Os for sorting [2], [47], implies lower bounds of $\Omega(N/pBD)$ I/O operations for a number of problems such as sorting, general permutation, matrix transpose, computing FFTs, etc., when the slackness constraint $N \geq v^{1+\varepsilon}b$ is satisfied.

Lemma 6. *For $N \geq v^{1+\varepsilon}b$, where ε is a constant, $0 < \varepsilon \leq 1$, the value of $\log_{M/B}(N/B)$ is a constant whose value depends on ε, but not on N.*

Proof. For constant $0 < \varepsilon \leq 1$,

$$N \geq v^{1+\varepsilon}b \tag{2}$$

$$\Longrightarrow N^{(\varepsilon+1)/\varepsilon-1} \geq v^{(\varepsilon+1)/\varepsilon}b^{(\varepsilon+1)/\varepsilon-1}. \tag{3}$$

Now, choosing $c = (\varepsilon + 1)/\varepsilon \geq 2$, (3) becomes

$$N^{c-1} \geq v^c b^{c-1}. \tag{4}$$

Substituting $v = N/M$ and $b = B$ into (4) gives

$$\left(\frac{M}{B}\right)^c \geq \frac{N}{B}$$

$$\Longrightarrow \log_{M/B}\frac{N}{B} \leq c.$$

We can conclude that $\log_{M/B}(N/B)$ is a constant whose value depends on ε, but not on N when $N \geq v^{1+\varepsilon}b$. □

So for constant ε, where $0 < \varepsilon \leq 1$, the logarithmic term is a constant of size at most $(\varepsilon + 1)/\varepsilon$ when $N \geq v^{1+\varepsilon}b$.

3.3. *Basic Applications*

We first present new EM-BSP algorithms, obtained from BSP algorithms via Lemma 2 and Theorem 3, for the fundamental problems of sorting, permutation, and matrix transpose. In each case the BSP algorithm uses $\lambda = O(1)$ communication rounds, and $O(N/v)$ internal memory per processor. Since our simulation techniques require that $N \geq v^{1+\varepsilon}b$, the I/O complexities of the resulting parallel, EM algorithms do not exhibit the logarithmic factor known to be present in the general case for these problems. Table 1 summarizes the performance of our EM sorting, permutation, and matrix transpose algorithms.

3.4. *Sorting*

The time complexity of sorting N items is $\Theta(N \lg N)$ for a comparison-based model of computation. On the PDM, sorting has been shown to have I/O complexity $O((N/BD) \log_{M/B}(N/B))$ for general values of N, M, D, and B [2], [47].

Goodrich [29] has described a deterministic v processor BSP algorithm for sorting which has a constant number of supersteps for $N \geq v^{1+\varepsilon}$, $\varepsilon > 0$ a fixed constant. We can achieve I/O complexity $\kappa = O(N/pBD)$ by simulating this algorithm using the techniques of this paper for $p \leq v$, $N \geq v^2 B$, and $v = \Omega(D)$.

Alternatively, we can simulate the BSP algorithm SampleSort, due to Shi and Schaeffer [41]. This algorithm has the following features [41]:

1. It is asymptotically optimal in computation time for $N \geq v^3$.
2. The load balancing between processors is nearly perfect in practice, and within a factor of two in theory.
3. It causes little memory and network contention.

We use it as a basis for a parallel, EM sort.

Algorithm: SampleSort

Input: The N items to be sorted are distributed evenly among the internal memories of the v processors of a BSP machine. Each processor has a unique label between 0 and $v-1$.

1. The v processors each sort the N/v items in their local internal memories.
2. Each processor chooses v equally spaced samples from the items in its possession.
3. The processors each send the v samples to processor 0.
4. Processor 0 sorts the v^2 samples and chooses from them v equally spaced splitter elements.
5. Processor 0 sends the v splitters to each of the other processors.
6. Each processor divides its local elements into v buckets determined by the splitter elements.
7. Each processor sends the contents of bucket i to processor i for all $0 \leq i < v$.
8. Each processor merges the v sets of items now in its possession to produce the final sorted order.

Lemmas 7 and 8 give the performance of algorithm SampleSort and its external memory version algorithm EM-SampleSort, respectively, analyzed under the EM-BSP model.

Lemma 7. *BSP algorithm SampleSort uses $O((N/v)\log(N/v)+L)$ computation time, $O(g\cdot(N/v)+L)$ communication time, $O(N/v)$ local memory per processor, and $O(1)$ communication supersteps, provided that $N/v \geq v^2$.*

Lemma 8. *Let $k = O(N/p)$. BSP algorithm SampleSort on v processors can be simulated on a p processor EM-BSP machine in $O((N\log(N/v))/p+(\bar{v}/p)L)$ computation time, $O(g\cdot(N/p)+(\bar{v}/p)L)$ communication time, and $O(G\cdot(N/pBD)+(\bar{v}/p)L)$ I/O time, for $\bar{v} = v/k$, provided that $p \leq v$, $N \geq v^2 B$, $D \leq B$, and $B \geq v/2$.*

Since algorithm Samplesort requires $N/v \geq v^2$, we obtain I/O complexity of $\kappa = O(N/pBD)$ for algorithm EM-SampleSort for $N \geq v^2B$, $v \geq D$, and $B \geq v$.

3.5. *Permutation*

Permutation of N items on a RAM has time complexity $\Theta(N)$. On the PDM, this problem has I/O complexity $\Theta(\min(N/D, (N/BD)\log_{M/B}(N/B))$ (see [2] and [47]). However, we can achieve I/O complexity $\kappa = O(N/pBD)$ by simulating algorithm AlgorithmPermute, for $p \leq v$, $N \geq v^2B$, and $D \leq B$. For ease of exposition, we assume that v divides N.

Algorithm: AlgorithmPermute
$\mathcal{V}$ is an N element vector containing items to be permuted. $\mathcal{P}$ is a corresponding N element vector containing new indices for each element of $\mathcal{V}$.

Input: Each processor i, $0 \leq i \leq (v-1)$, holds an N/v element vector $\mathcal{V}_i$, containing elements $i \cdot (N/v)$ to $(i+1) \cdot (N/v) - 1$ of $\mathcal{V}$, and an N/v element vector $\mathcal{P}_i$, containing elements $i \cdot (N/v)$ to $(i+1) \cdot (N/v) - 1$ of $\mathcal{P}$.
Output: Each processor i contains items $i \cdot (N/v)$ to $(i+1) \cdot (N/v) - 1$ of the permuted vector $\mathcal{V}'$.

1. Each processor $i, 0 \leq i \leq (v-1)$, sends the items of $\mathcal{V}_i$ to the processors holding the items indicated by $\mathcal{P}_i$.
2. Each processor performs the necessary rearrangements in its local memory to complete the calculation of $\mathcal{P}$.

Algorithm AlgorithmPermute performs the indicated permutation in a single communication round consisting of an (N/v)-relation. The internal computation time is $O(N/v)$, and the memory used is $O(N/v)$ per processor.

3.6. *Matrix Transpose*

Transposing an $n \times m$ matrix, where $N = n \times m$, takes $\Theta(N)$ time on a RAM. On the PDM, this problem has I/O complexity $\Theta((N/BD)(\log \min(M, n, m, N/B)/\log(M/B)))$ [2], [47]. However, we can achieve I/O complexity $\kappa = O(N/pBD)$ by simulating algorithm AlgorithmTranspose, below, for $p \leq v$, $N \geq v^2B$, and $D \leq B$. For ease of exposition, we assume that v divides N.

Algorithm: AlgorithmTranspose
Let $\mathcal{M}$ be an $n \times m$ matrix, and let a_{ij} be the element of $\mathcal{M}$ in row i and column j. Let $\mathcal{M}_0$ be a one-dimensional array containing the elements of $\mathcal{M}$ row by row, i.e., $a_{11}, a_{12}, \ldots, a_{1m}, a_{21}, a_{22}, \ldots, a_{nm}$. Let $\mathcal{M}'$ be the transpose of $\mathcal{M}$, and let a'_{ji} be the element of $\mathcal{M}'$ in row j and column i. Let $\mathcal{M}'_0$ be a one-dimensional array containing the elements of $\mathcal{M}'$ row by row, i.e., $a'_{11}, a'_{12}, \ldots, a'_{1m}, a'_{21}, a'_{22}, \ldots, a'_{nm}$.

Input: Processor $i, 0 \leq i \leq (v-1)$, holds items $i(N/v)$ to $(i+1)(N/v) - 1$ of $\mathcal{M}_0$.
Output: Processor i holds items $i(N/v)$ to $(i+1)(N/v) - 1$ of $\mathcal{M}'_0$.

1. Each processor determines the destination processor for each of its items and sends them in a single superstep to their destination.
2. Each processor inserts the received items into the appropriate positions in its memory.

Algorithm AlgorithmTranspose transposes the matrix $\mathcal{M}$ in a single communication round consisting of an (N/v)-relation. The internal computation time is $O(N/v)$, and the memory used is $O(N/v)$ per processor.

3.7. *Summary*

Table 2 lists a number of other important problems arising in computational geometry, GIS, and graph algorithms, for which we report EM-BSP algorithms created by our technique, together with their I/O complexity and that of the previously best known algorithm for the problem. In Group B we report the same I/O complexities as previously known, after accounting for our parameter restrictions. The I/O complexities we report in Group C are not as competitive with previous results. In each case, however, our algorithm is scalable not only in terms of the number of disks per processor but also in terms of the number of processors used. Previous algorithms were often not efficient in a multiprocessor environment (particularly in a distributed memory environment), and in many cases it is not clear how they could be adapted to parallel disks.

The problems listed in Table 2 are stated briefly below. Please consult [16] and [28] for applications and more complete definitions of these problems and for related previous results on RAM and PRAM computation models.

1. Given a simple polygon S, the *triangulation problem* is to partition the interior of S into a set of triangles by joining vertices of S with non-overlapping straight line segments.

2. Let S be a set of line segments in the plane. The *trapezoidation problem* is to decompose the plane into a set of trapezoids, based on the arrangement of the line segments.

3. A *segment tree* is a data structure used to organize a set of line segments and support various kinds of queries against the set.

4. Let S be a set of n non-intersecting line segments in the plane, and let Q be a set of m query points. The *next element search problem* is to find, for each query point $q_i \in Q$, the line segment directly above q_i. The *endpoint dominance problem* is a special case of the next element search problem, where the query set is composed of the endpoints of the line segments themselves.

5. Let G be an embedding of a graph in the plane. The *batched planar point location problem* is to find, for each of a set of N query points q_i, $1 \leq i \leq N$, the face of G which contains q_i.

6. Given a set S of points in two-dimensional (three-dimensional) space, the $2D$ ($3D$) *convex hull problem* is to find the convex polygon (polytope) which contains all points in S and whose vertices are points in S.

7. Given a set $S = \{s_1, \ldots, s_N\}$ of N points in two-dimensional space, the $2D$ *Voronoi diagram problem* is to find a partition of the plane into N regions $R_1, \ldots, R_N$,

such that for each i, $1 \leq i \leq N$, region R_i contains a single point s_i of S, and all the other points in R_i are closer to s_i than to any other point of S.

8. Given a set S of N points in two-dimensional space, the *Delaunay triangulation problem* is to find a triangulation of the points in S such that for each such triangle, the circle incribed through its vertices does not contain any other point of S.

9. Given a set S of n non-intersecting line segments in the plane, the *lower envelope* problem consists of computing the set of segment portions visible from the point $(0, -\infty)$.

10. The generalized lower envelope problem is similar to 9 above, except that segments in S may intersect.

11. Given a set R of isothetic rectangles, the *measure problem* is to compute the area covered by the union of R.

12. Consider a set S of n points in 3-space. For a point v let $x(v)$, $y(v)$, and $z(v)$ denote the x-coordinate, y-coordinate, and z-coordinate, respectively, of v. Point v *dominates* a point w iff $x(v) > x(w)$, $y(v) > y(w)$, and $z(v) > z(w)$. A point is *maximal* in S if it is not dominated by any other point of S. The 3*D-maxima problem* consists of determining the set of all maximal points in S.

13. Given a set S of n points in the Euclidean plane, the *all nearest neighbors problem* is to determine for each point $v \in S$ its nearest neighbor $NN_S(v)$ in S, where $NN_S(v)$ is a point $w \in S\backslash\{v\}$ such that $dist(v, w) \leq dist(v, u),\ \forall u \in S\backslash\{v\}$.

14. Let S be a set of n points in the plane with some *weight* $w(v)$ assigned to each $v \in S$. The 2*D-weighted dominance counting problem* consists of determining for each $v \in S$, the total weight of all points which are dominated by v.

15. Let S be a set of r pairwise disjoint m-vertex polygons. The *unidirectional separability problem* consists of determining all directions d such that S is separable by a sequence of r translations in direction d, one for each polygon. The *multidirectional separability problem* consists of determining if S is separable by a sequence of r translations in different directions.

16. Given a linked list ℓ of N objects, the *list ranking* problem is to compute, for each object in ℓ, its distance from the beginning of the list.

17. Given a connected graph $G = (V, E)$, the *Euler tour* problem is to find a cycle that traverses each edge of G exactly once.

18. Given a rooted tree $G = (V, E)$, and a pair of vertices (u, v) of G, the *lowest common ancestor* problem is to find the vertex w of G that is an ancestor to both u and v and is farthest from the root.

19. The *tree contraction technique* is a method for constructing parallel algorithms on trees, working from the bottom up [40].

20. The *expression tree evaluation problem* is to evaluate the result of an arithmetic expression represented as an expression tree.

21. Given an undirected graph $G = (V, E)$, a connected subset of vertices is a subset of vertices in which there is a path in G between each pair of vertices. The *connected components* problem is to find the maximal connected subsets of vertices in G.

22. Given a planar graph $G = (V, E)$, the *spanning forest problem* is to form a spanning tree for each connected component of G.

23. Given a connected undirected graph $G = (V, E)$, the *ear decomposition* problem [12] is to find an ordered partition of E into r simple paths $P_1, \ldots, P_r$ such that P_1 is

a cycle, and for each i, $2 \leq i \leq r$, P_i is a simple path whose endpoints belong to $P_1 \cup \cdots \cup P_{i-1}$, but with none of its internal vertices belonging to P_j, for $j < i$. The *open ear decomposition* problem is similar, but none of the P_i, for $i > 1$, is a cycle.

24. Given a connected undirected graph $G = (V, E)$, the *bi-connected components* problem is to find the maximal connected subsets of vertices of G which remain connected when any single edge is deleted.

References

[1] A. Aggarwal and G. Plaxton. Optimal parallel sorting in multi-level storage. In *Proc. ACM–SIAM Symp. on Discrete Algorithms*, pages 659–668, 1994.

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[3] L. Arge. The buffer tree: a new technique for optimal I/O-algorithms. In *Proc. Workshop on Algorithms and Data Structures*, pages 334–345. LNCS 955. Springer-Verlag, Berlin, 1995. A complete version appears as BRICS Technical Report RS-96-28, University of Aarhus.

[4] L. Arge. The I/O-complexity of ordered binary-decision diagram manipulation. In *Proc. Internat. Symp. on Algorithms and Computation*, pages 82–91. LNCS 1004. Springer-Verlag, Berlin, 1995. A complete version appears as BRICS Technical Report RS-96-29, University of Aarhus.

[5] L. Arge. Efficient External-Memory Data Structures and Applications. Ph.D. thesis, University of Aarhus, February/August 1996.

[6] L. Arge, M. Knudsen, and K. Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In *Proc. Workshop on Algorithms and Data Structures*, pages 83–94. LNCS 709. Springer-Verlag, Berlin, 1993.

[7] L. Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. Annual European Symposium on Algorithms*, pages 295–310. LNCS 979. Springer-Verlag, Berlin, 1995. A complete version (to appear in special issue of *Algorithmica*) appears as BRICS Technical Report RS-96-12, University of Aarhus.

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

[9] D. Bader, D. Helman, and J. Jájá. Practical parallel algorithms for personalized communication and integer sorting. *Journal of Experimental Algorithmics*, 1, 1996. http://www.jea.acm.org/1996/BaderPersonalized/.

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

[11] E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Reiping, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *Proc. Internat. Colloquium on Algorithms, Languages and Programming*, pages 390–400. LNCS 1256. Springer-Verlag, Berlin, 1997.

[12] A. Chan, F. Dehne, and A. Rau-Chaplin. Coarse grained parallel next element search. In *Proc. Internat. Parallel Processing Symp.*, pages 320–325, 1997.

[13] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. ACM–SIAM Symp. on Discrete Algorithms*, pages 139–149, 1995.

[14] T. H. Cormen. Personal communication, 1997.

[15] T. H. Cormen and M. T. Goodrich. Position statement, ACM Workshop on Strategic Directions in Computing Research: working group on storage I/O for large-scale computing. *ACM Computing Surveys*, 28A(4), December 1996.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[17] T. H. Cormen, T. Sundquist, and L. F. Wisniewski. Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. Technical Report PCS-TR94-223, Dept. of Computer Science, Dartmouth College, July 1994. *SIAM Journal on Computing*, 28(1):105–136, 1998.

[18] A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, and E. Ramos. Randomized external memory algorithms for geometric problems. In *Proc. ACM Annual Conf. on Computational Geometry*, pages 259–268, 1998.

[19] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. Erik Schauser, R. Subramonian, and T. von Eicken. Logp: a practical model of parallel computation. *Communications of the ACM*, 39(11):260–270, November 1996.

[20] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar. A randomized parallel 3d convex hull algorithm for coarse grained multicomputers. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 27–33, 1995.

[21] F. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 106–115, 1997.

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

[23] W. Dittrich, D. Hutchinson, and A. Maheshwari. Blocking in parallel multisearch problems. *Theory of Computing Systems*, 34:145–189, 2001.

[24] R. W. Floyd. Permuting information in idealized two-level storage. In *Complexity of Computer Calculations*, pages 105–109 (R. Miller and J. Thatcher, Eds.). Plenum, New York, 1972.

[25] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. ACM Symp. on Theory of Computation*, pages 114–118, 1978.

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

[27] G. A. Gibson, J. S. Vitter, and J. Wilkes. Strategic directions in storage I/O issues in large-scale computing. *ACM Computing Surveys*, 28(4):779–793, December 1996.

[28] J. E. Goodman and J. O'Rurke (editors). *Handbook of Discrete and Computational Geometry*. CRC Press, Boca Raton, FL, 1997.

[29] M. T. Goodrich. Communication efficient parallel sorting. In *Proc. ACM Symp. on Theory of Computation*, pages 247–256, 1996.

[30] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 714–723, 1993.

[31] D. A. Hutchinson, A. Maheshwari, and N. Zeh. An external memory data structure for shortest path queries. In *Proc. 5th Annual Combinatorics and Computing Conf.* (*COCOON '99*), pages 51–60. LNCS 1627. Springer-Verlag, Berlin, 1999.

[32] J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, MA, 1992.

[33] V. Kumar and E. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. IEEE Symp. on Parallel and Distributed Processing*, 1996.

[34] F. T. Leighton. *An Introduction to Parallel Algorithms and Architectures*. Morgan Kaufman, San Mateo, CA, 1992.

[35] Z. Li, P. H. Mills, and J. H. Reif. Models and resource metrics for parallel and distributed computation. *Parallel Algorithms and Applications*, 8:35–59, 1996.

[36] K. Munagala and A. Ranade. I/O complexity of graph algorithms. *Proc. ACM–SIAM Symp. on Discrete Algorithms*, pages 687–694, 1999.

[37] M. H. Nodine and J. S. Vitter. Deterministic distribution sort in shared and distributed memory multiprocessors. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 120–129, 1993.

[38] M. H. Nodine and J. S. Vitter. Paradigms for optimal sorting with multiple disks. In *Proc. 26th Hawaii Internat. Conf. on Systems Sciences*, 1993.

[39] M. H. Nodine and J. S. Vitter. Greed sort: optimal deterministic sorting on parallel disks. *Journal of the ACM*, 42(4):919–933, 1995.

[40] J. H. Reif (editor). *Synthesis of Parallel Algorithms*. Morgan Kaufman, San Mateo, CA, 1993.

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

[42] J. F. Sibeyn and M. Kaufmann. BSP-like external-memory computation. In *Proc. 3rd Italian Conf. on Algorithms and Complexity*, pages 229–240. LNCS 1203. Springer-Verlag, Berlin, 1997.

[43] R. W. Stevens. *Advanced Programming in the Unix Environment*. Addison-Wesley, Don Mills, Ont., 1995.
[44] J. D. Ullman and M. Yannakakis. The input/output complexity of transitive closure. *Annals of Mathematics and Artificial Intellegence*, 3:331–360, 1991.
[45] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.
[46] M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer (editors). *Algorithmic Foundations of Geographic Information Systems*. LNCS 1340. Springer-Verlag, Berlin, 1997.
[47] J. S. Vitter. External memory algorithms. *Proc. ACM Symp. Principles of Database Systems*, pages 119–128, 1998.
[48] J. S. Vitter. Personal communication, 1998.
[49] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.
[50] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, II: Hierarchical multilevel memories. *Algorithmica*, 12(2–3):148–169, 1994.

Received October 17, 2000. *Online publication September* 9, 2002.