Algorithmica (2003) 36: 97–122
DOI: 10.1007/s00453-002-1009-y

Algorithmica

Efficient External Memory Algorithms by Simulating Coarse-Grained Parallel Algorithms[1]

Frank Dehne,[2] Wolfgang Dittrich,[3] and David Hutchinson[4]

Abstract. External memory (EM) algorithms are designed for large-scale computational problems in which the size of the internal memory of the computer is only a small fraction of the problem size. Typical EM algorithms are specially crafted for the EM situation. In the past, several attempts have been made to relate the large body of work on parallel algorithms to EM, but with limited success. The combination of EM computing, on multiple disks, with multiprocessor parallelism has been posted as a challenge by the ACM Working Group on Storage I/O for Large-Scale Computing.

In this paper we provide a simulation technique which produces efficient *parallel* EM algorithms from efficient BSP-like parallel algorithms. The techniques obtained can accommodate one or multiple processors on the EM target machine, each with one or more disks, and they also adapt to the disk blocking factor of the target machine. When applied to existing BSP-like algorithms, our simulation technique produces improved *parallel* EM algorithms for a large number of problems.

Key Words. Parallel algorithms, Coarse grained parallel computing, External memory algorithms, Parallel I/O.

1. Introduction.

External memory (EM) algorithms are designed for large computational problems in which the size of the internal memory of the computer is only a small fraction of the size of the problem. Important applications in Geographic Information Systems (GIS), Virtual Reality, VLSI design, weather prediction, computerized medical treatment, 3D simulation and modeling, visualization, and Computational Geometry fall into this category.

With few exceptions (e.g., [33]), previous authors focussed on a uniprocessor EM model. However, parallel processing is an important issue for EM algorithms (extremely large problems) for the same reasons that parallel processing is of practical interest in non-EM algorithm design. Previous parallel EM algorithms (e.g., [33]), as well as most sequential EM algorithms, were "new" and carefully hand-crafted to work optimally

[1] A preliminary version of this paper was presented at the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97), Newport, RI, June 22–25, 1997. The research by the first author was partially supported by the Natural Sciences and Engineering Research Council of Canada. The second author's research was partially supported by DFG-Sonderforschungsbereich 376 "Massive Parallelität" and by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT), and was started while he was visiting Carleton University. The research by the third author was partially supported by Almerco, Inc. and by the Natural Sciences and Engineering Research Council of Canada.
[2] School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. frank@dehne.net, http://www.dehne.net.
[3] Bosch Telecom GmbH, Backnang, Germany.
[4] Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada K1S 5B6.

Received September 6, 1998; revised July 21, 2002. Communicated by D. T. Lee.
Online publication March 20, 2003.

in the EM environment. Existing "internal memory" algorithmic techniques and data structures were often found to be unsuitable for EM. This is largely due to the need for locality on data references, which is not generally present when algorithms are designed for internal memory, due to the permissive nature of the RAM and PRAM models. However, there are some obvious similarities between formulating an efficient algorithm for a parallel computer and formulating one for EM (see, e.g., [6]). The possibility of using the vast body of algorithms developed for parallel computers instead of reinventing new algorithms for EM is intriguing. In this paper we exploit a natural correspondence between EM algorithms and BSP-like parallel models such as BSP [28], BSP* [9], [10], [8] and CGM [19], [17], [16]. We provide simulation techniques that map BSP-like algorithms to EM algorithms and we show how, using a randomized approach, an EM machine can take full advantage of parallel disk I/O and multiple processors.

Ensuring that I/O is fully blocked is an important issue in reducing the runtime of an EM algorithm. Accessing the main memory of a computer can be orders of magnitude faster than accessing an element of data in secondary memory such as a hard disk. This large difference is typically made less significant by carefully ensuring that data on disk is accessed in a blockwise fashion. Thus, the overheads of rotational delay and disk arm movement are amortized over the number of items in a disk block. A second important issue, when more than one disk is present, is fully parallel disk I/O. If there are D disks present, and the disk block size is B, DB data items can be transferred in a single I/O operation. These two issues are fundamental, since if I/O is not fully blocked, the runtime can typically be up to a factor of 10^3 (the blocking factor) too high, and if parallel disks are not properly utilized, the runtime can be a factor of D too high. More discussion of the traditional EM system model and related issues can be found in [33].

The main contribution of this paper is to exploit a natural correspondence between EM algorithms and batch-synchronous parallel models such as BSP, BSP*, and CGM. We identify the blockwise communication characteristic of the BSP* and CGM model as an example of a type of *coarse-grained communication* which permits the generated EM algorithm to take full advantage of the disk block size. We provide a simulation technique that implements this mapping from BSP-like algorithms to EM algorithms, and we further show how, using a randomized technique, an EM machine can take full advantage of parallel disk I/O and multiple processors. Furthermore, our technique can take full advantage of the physical memory available by concurrently simulating a superstep of more than one virtual processor. Our result addresses a challenge posted by the ACM Working Group on Storage I/O for Large-Scale Computing [20].

The remainder of this paper is organized as follows. In Section 2 we review previous work on EM algorithms as well as the BSP/BSP*/CGM models. In Section 3 we introduce the *EM-BSP* model consisting of one or multiple processors, each with one or multiple disks. Section 4 gives an overview of our simulation techniques, and Section 5 presents the details. Section 5.1 describes the simulation for the single-processor case and Section 5.2 extends the result to the general multiprocessor case. Section 6 presents parallel EM algorithms obtained by applying our simulation technique to existing BSP-like parallel algorithms and Section 7 concludes the paper.

2. Review

2.1. *Previous Work For EM Algorithms.* Sorting, permutation and related problems in EM have been extensively studied [1], [15], [33], [32], [25]. I/O-optimal approaches to many computational geometry problems [22] and graph problems [14] have also been described. Data structures [3], [4], [27] and a number of applications [5] have been examined in this context. Implementation results have been presented in [29], [30], and [13].

The classical EM model is described in [1]. More complex models have been proposed as well, incorporating a hierarchy of memory layers rather than the two-level memory model of [1]. One such model is described in [2], and sorting for this model is studied in [32]. Such models are interesting because modern computers typically have several layers of memory which include main memory and caches as well as disks. We restrict ourselves to the two-level model because the speed difference between disk and main memory is much more significant than between the other layers of memory.

Chiang et al. [14] explored simulation of PRAM algorithms as a source of new EM techniques. Their approach involves an EM sort with every PRAM step. They showed that certain PRAM algorithms with a "geometrically decreasing size" property could be simulated as EM algorithms in an I/O-optimal way. However, most problems do not have a geometrically decreasing size, including problems like sorting, matrix multiplication, convex hull, and Voronoi diagram construction (see [26]).

Concurrent to the work presented in this paper, Sibeyn and Kaufmann [26] have developed a technique for simulating 1-optimal BSP algorithms to produce efficient EM algorithms. Their work is presented in the context of a single disk uniprocessor EM machine, but they suggest that the concept can be extended to multiple disks. They simulate a superstep of one virtual processor at a time, saving the context and generated messages in a $v \times v$ array on disk, where each cell is of size 3μ (v is the number of virtual (BSP) processors, and μ is the size of the context of a processor). However, the paper does not include techniques to accommodate the blocking factor, which is an intrinsic issue in efficient I/O design, nor does it provide mechanisms for handling multiple disks or multiple physical processors on the target EM machine.

2.2. *The BSP/BSP*/CGM Models.* The **BSP** (Bulk Synchronous Parallel) model was introduced in 1990 [28]. A BSP computer is a collection of processor/memory modules connected by a router that can deliver messages in a point to point fashion between the processors. A BSP-style computation is divided into a sequence of *supersteps* separated by barrier synchronizations. Each superstep consists of a computation superstep and a communication superstep. In a *computation superstep* the processors perform computations on data that was present locally at the beginning of the superstep. In a *communication superstep* data is exchanged among the processors via the router.

A BSP computer has the following parameters:

- p is the number of processors,
- $\hat{L}$ is the minimum time between synchronization steps (measured in basic computation units), and
- $\hat{g}$ is the ratio of computational capacity (number of local computation operations per

unit time) divided by the communication capacity (number of messages of unit size that can be sent to the router per unit time).

A BSP algorithm with a total of λ supersteps has the following computation and communication costs: The computation cost of the algorithm is $T_{\text{comp}} = \sum_{i=1}^{\lambda} w_{\text{comp}}^i$. The time, w_{comp}^i, expended in the ith computation superstep is $\max\{\hat{L}, t_1, \ldots, t_p\}$, where t_j is the number of basic computation operations performed by processor j in the ith superstep. The communication cost is $T_{\text{comm}} = \sum_{i=1}^{\lambda} w_{\text{comm}}^i$, where the ith communication superstep is assigned cost $w_{\text{comm}}^i = \max_{j=1}^{p}\{w_{\text{comm},j}^i\}$. Here, $w_{\text{comm},j}^i$ is the communication cost incurred by processor j in the ith superstep. Assuming that processor j receives messages of lengths $r_1, \ldots, r_{j'}$ and sends messages of lengths $\{s_1, \ldots, s_{j''}\}$ during the ith superstep, $w_{\text{comm},j}^i = \max\{\hat{L}, \hat{g}(\sum_{u=1}^{j'} r_i + \sum_{u=1}^{j''} s_i)\}$.

The **BSP*** model was introduced in 1995 [9] as an extension of BSP to account for the increased performance that can often be achieved if communication between processors is performed in a blockwise fashion. It introduces an additional parameter b which is the minimum size that messages must have in order to take full advantage of the bandwidth of the router. Messages of length smaller than b are charged the same cost as messages of length b. We refer to messages of size b as *packets* and call b the *packet size*.

In this paper we use the following notation to refer to the parameters of a BSP* computer:

- p is the number of processors,
- b is the packet size,
- g is the time (measured in basic computation units) to transport a packet of size b between processors, and
- L is the minimum time (measured in basic computation units) to perform a barrier synchronization between the processors.

The BSP* model assigns the same cost to an algorithm as the BSP model except that $w_{\text{comm},j}^i = \max\{L, g(\sum_{u=1}^{j'} \lceil r_i/b \rceil + \sum_{u=1}^{j''} \lceil s_i/b \rceil)\}$.

The **CGM** (Coarse Grained Multicomputer) model was introduced in 1993 [19], [17], [16]. It uses only two parameters, n and p, and assumes a collection of p processors, each with n/p local memory, connected by a router that can deliver messages in a point to point fashion. A CGM algorithm consists of an alternating sequence of computation and communication rounds separated by barrier synchronizations. A computation round is equivalent to a computation superstep in the BSP model, and the total computation cost T_{comp} is defined analogously. A communication round consists of a single h-relation with $h \le n/p$. An *h-relation* is defined as a communication step in which p messages of length at most $O(h)$ bytes are sent and received by every processor [18]. The cost w_{comm}^i of every communication round is bounded by the same value, $H_{n,p}$. Therefore, the total communication cost T_{comm} of a CGM algorithm with λ communication rounds is simply $T_{\text{comm}} = \lambda H_{n,p}$. Algorithms do usually require a lower bound on $n/p \gg 1$, e.g., $n/p \ge p$ or $n/p \ge p^{\varepsilon}$ [19], [17], [16]. The CGM model works particularly well in the case where the overall computation speed is considerably larger than the overall communication speed.

The h-relation simulation presented in [7] leads to the following observation.

OBSERVATION 1. *A CGM algorithm $\mathcal{A}$ with communication time $T_{\text{comm}} = \lambda H_{n,p}$, computation time T_{comp}, and local memory M can be simulated by a BSP* algorithm $\mathcal{A}'$ with communication time $O(g\lambda(n/pb) + \lambda L)$, computation time $O(T_{\text{comp}} + \lambda L)$, and local memory $O(M)$.*

3. The EM-BSP/EM-BSP*/EM-CGM Models.

We now extend the BSP model to include secondary local memories. The basic idea is very simple and is illustrated in Figure 1. Each processor has in addition to its local memory an EM in the form of a set of hard disks. We apply this idea to extend the BSP model to its EM version **EM-BSP** by adding the following to the standard BSP parameters:

- M is the local *memory size* of each processor,
- D is the *number of disk drives* of each processor,
- B is the *transfer block size* of a disk drive, and
- G is the ratio 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.

In many practical cases, all processors have the same number of disks. We restrict ourselves to that case, although the model does not forbid different numbers of drives and memory sizes for each processor. We denote the disk drives of each processor by $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{D-1}$. Each drive consists of a sequence of *tracks* (consecutively numbered starting with 0) which can be accessed by direct random access using their unique track number. A track stores exactly one block of B records.

Each processor can use all of its D disk drives concurrently, and transfer $D \times B$ items from the local disks to its local memory in a single I/O operation and at cost G. In such an operation, we permit only one track per disk to be accessed without any

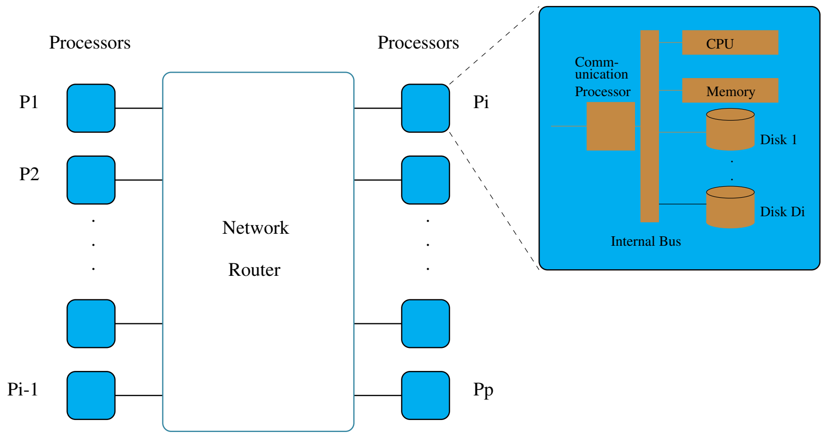

Fig. 1. Illustration of a parallel machine with EM.

restriction on which track is accessed on each disk. We assume that a processor can store in its local memory at least one block from each local disk at the same time, i.e., $M \geq DB$.

Like a computation on the BSP model, the computation on the EM-BSP model proceeds in a succession of supersteps. We adapt communication and computation supersteps from the BSP model and allow multiple I/O operations 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. The total cost of each superstep is defined as $t_{\text{comp}} + t_{\text{comm}} + t_{\text{I/O}} + L$. The term $t_{\text{I/O}}$ is the additional I/O cost charged for the superstep, where $t_{\text{I/O}} = \max_{j=1}^{p}\{w_{\text{I/O}}^{j}\}$, and $w_{\text{I/O}}^{j}$ is the I/O cost incurred by processor j. Recall that each I/O operation costs G time units.

Note that the model gives incentives to access all disk drives concurrently. For instance, a single processor EM-BSP with D disks is capable of transferring a block of B items to or from every disk in a single I/O operation. An operation involving fewer disk drives incurs the same cost.

The **EM-BSP*** and **EM-CGM** models are defined analogously to the EM-BSP model. We add to the BSP* and CGM models the additional parameters M (local memory size at each processor), D (number of disk drives at each processor), B (transfer block size for a local disk drive), and G (ratio of local computational capacity to local I/O capacity).

The cost of each EM-BSP* superstep or CGM round is defined as $t_{\text{comp}} + t_{\text{comm}} + t_{\text{I/O}} + L$ where t_{comp} and t_{comm} refer to the computation time and communication time as defined for the BSP* model, and $t_{\text{I/O}}$ is defined as indicated above for the EM-BSP model. Note that the I/O cost of each CGM round is identical, and represents the I/O cost for simulating an h-relation with $h = n/p$.

For the remainder of this paper, please refer to the table in Appendix A.2 to look up the various parameters used.

4. Overview of the Simulation of Parallel Algorithms as EM Algorithms.

We first describe in general terms how a BSP-like algorithm can be executed as an EM algorithm on a single-processor machine with multiple disks.

We adopt the following terminology: The processors of the BSP-like machine will be called *virtual processors*, and v will denote their number. Each communication superstep will be 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, computation, and sending superstep. 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 denoted as μ.

The execution of a BSP-like 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. Message send/receive operations are modeled by disk read/write operations.

Outline of the simulation for a compound superstep. A compound superstep for the v virtual processors of a BSP-like machine is simulated by performing the following steps in a round-robin fashion for $k \geq 1$ virtual processors at a time:

1. *Fetching Phase*: read the context(s) and the messages to be received by the current virtual processors from disk into memory.
2. *Computation Phase*: perform the computations indicated by the BSP* algorithm for these k processors in this compound superstep.
3. *Writing Phase*: save the current contexts and the messages sent by the current virtual processors on disk.

The Fetching Phase (Computation Phase, Writing Phase) performs the operations necessary to simulate the receiving superstep (computation superstep, sending superstep) of a compound superstep. A compound superstep produces messages which are received in the following compound superstep. The simulation must store the generated messages on disk in such a way that they can be fetched efficiently during the Fetching Phase of the next compound superstep. By efficiently we mean in this context that both input and output operations are fully blocked to the disk block size B and that if parallel disks are present they are utilized in parallel, i.e., for D parallel disks, input and output operations are performed D blocks at a time. These requirements can easily be met for the contexts, as we know their maximum size and can preallocate a dedicated area for each, spread across the D disks. For the generated message traffic, however, these requirements may be more difficult, as we may not know the communication pattern for a particular compound superstep.

In the following section we describe a randomized approach which allows us to write the messages to disk efficiently and efficiently retrieve them in the next compound superstep. We note that for communication of predetermined size, such as occurs in a CGM, our simluation result can be made deterministic.

5. BSP* to EM-BSP* Simulation.

We now describe the details of a BSP* to EM-BSP* simulation. A CGM to EM-CGM simulation follows by Observation 1.

5.1. *Simulation Algorithm for the Single-Processor Case $p = 1$*

DEFINITION 1. A collection of records is in *blocked format* if its records are grouped into blocks of size B.

DEFINITION 2. A collection of records, stored on D disks, is in *standard consecutive format* if (i) the records are in blocked format, (ii) the number of blocks on each disk differs by at most one, and (iii) on each disk, the blocks are stored in consecutive tracks.

The communication time of a BSP* superstep is $O(g(\gamma/b) + L)$, where γ is the maximum size of the data communicated by a virtual processor. Recall that μ is the maximum context size of a virtual processor, hence $\gamma = O(\mu)$. In the following, the context and messages generated by a virtual processor are divided into blocks, and the blocks are spread in standard consecutive format (evenly) over the available disks.

Algorithm SeqCompoundSuperstep simulates a single compound superstep of a v processor BSP* on a uniprocessor EM-BSP* machine. We simulate $k \geq 1$ virtual processors at a time, and we refer to such a collection of processors as a *group*. We also use this term to refer to the messages associated with a group of processors. To maximize the use of available memory, we choose $k = \lfloor M/\mu \rfloor$. Note that $M \geq \mu$. The following outlines the algorithm. The implementation details are presented afterwards.

Algorithm 1: SeqCompoundSuperstep

Objective: Simulation of a compound superstep of a v-processor BSP* on a single-processor EM-BSP* with D disks.
Input: For each $i \in \{0, \ldots, v/k - 1\}$ the blocks of the contexts and arriving messages of group i (i.e., virtual processors $ik, \ldots, (i+1)k - 1$) are spread over the D disks in standard consecutive format, so that they can be accessed in parallel.
Output: (i) The (changed) contexts of the k simulated processors are spread across the disks in standard consecutive format. (ii) The messages generated during the compound superstep are grouped by destination into v/k groups, and each group is stored in standard consecutive format on the disks.

```
(1) for $i = 0$ to $cv/k - 1$
    (a) Read the contexts of the $k$ virtual processors of group $i$, referred
        to as $V_{ik}, \ldots, V_{(i+1)k-1}$ from the disks into memory.
    (b) Read the packets received by the $k$ virtual processors of group $i$
        from the disks.
    (c) Simulate the local computation of the $k$ virtual processors of
        group $i$.
    (d) Write the packets which were sent by the $k$ virtual processors to
        the $D$ disks.
    (e) Write the changed contexts $V_{ik}$ to $V_{(i+1)k-1}$ back to the $D$ disks.
(2) Reorganize the blocks containing the generated messages into standard
    consecutive format for each group of $k$ processors so that they can be
    accessed in parallel from the disks in the simulation of the next com-
    pound superstep.
```

Algorithm SeqCompoundSuperstep simulates a single compound superstep of the BSP* algorithm. Steps 1(a) and 1(b) correspond to the Fetching Phase, Step 1(c) is the Computation Phase, and Steps 1(d) and 1(e) comprise the Writing Phase.

Details of Steps 1(a) *and* 1(e): Since we know the size of the contexts of the processors, and the order in which we simulate the virtual processors is static during the simulation, we can distribute the k contexts deterministically. We reserve an area of total size $v\mu$ on the disks, $v\mu/DB$ 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 standard consecutive format on the disks, we can easily read and write the contexts of k consecutive processors using D disks in parallel for every I/O operation.

Details of Step 1(b): Step (2) for the previous compound superstep guaranteed that the blocks which contain the messages destined for the current processors are stored in a reserved area evenly distributed over the disks. Therefore, we can use a similar technique to fetch the messages as we used to fetch the contexts.

Details of Step 1(d): After the Computation Phase, all messages sent by the current group of k processors in the current compound superstep have been generated and stored in internal memory. The coarse-grained nature of the BSP* algorithm results in large messages, 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 $k(\gamma/B)/D$ rounds, we write the blocks out to the disks. In each round a group of D blocks b_i, $0 \leq i \leq (D-1)$, is written in parallel to the disks by choosing a random permutation π of $\{0, 1, \ldots, (D-1)\}$ and writing block b_i to disk $\pi(i)$.

The blocks are partitioned into *D buckets* on the disks, depending on their destination address. Each bucket contains the blocks destined for v/D consecutive virtual processors. In order to maintain the buckets, the simulation uses a table of D pointers on each disk. The ith entry in the table on a disk points to the head of a list of blocks of bucket i that have been written to that disk. Whenever we write a block of bucket i to disk D_j, we allocate a free track on D_j and concatenate it to the list for bucket i. For convenience, we refer to the format just described for the blocks in a bucket as *standard linked format*.

Clearly, we can read all the blocks composing a bucket stored on D disks in standard linked format in $O(k\gamma/DB)$ parallel I/O operations, provided each disk contains the same number of blocks. In Lemma 2 we will show that with high probability the blocks of each bucket are uniformly distributed over the disks.

Algorithm SimulateRouting provides the details of Step 2 of SeqCompoundSuperstep.

Algorithm 2: SimulateRouting

Objective: Reorganize the blocks of messages from the previous computation superstep into standard consecutive format (Step 2 of Algorithm 1).
Input: The D buckets (of messages) stored on the disks in standard linked format.
Output: The v/k groups (of messages) stored on the disks in standard consecutive format.

```
(1) Allocate space for a copy of bucket $i$ on disk $i$, for $i = 0, \ldots, (D-1)$.
    Read the buckets from the disks in parallel and write them back, one
    bucket per disk. For the $j$th parallel read/write we perform the following:

    for $d = 0$ to $D-1$ in parallel do
        Read block $b_d$ belonging to bucket $d$ from disk $((d+j) \bmod D)$.
        Write block $b_d$ to disk $d$ on the next available track.

(2) From each disk, in parallel, read a block of each bucket, writing the
    blocks back to the disks so that each bucket is in standard consecutive
    format.

    for $j = 0$ to $v\gamma/DB$
        for $d = 0$ to $D-1$ in parallel do
            read the $j$th block from disk $d$ and write it to disk $(d+j) \bmod D$ on
            track $d\lceil v\gamma/D^2B\rceil + \lfloor j/D \rfloor$.
```

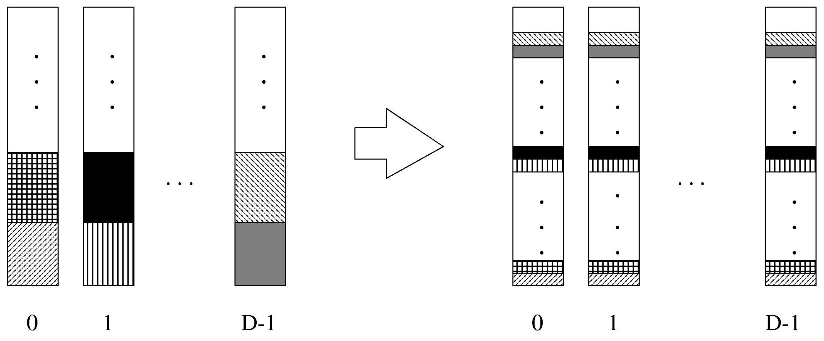

Fig. 2. Reorganization of the blocks.

After Step 1 of Algorithm SimulateRouting, all messages that will be received by a group of k processors are stored in one region on the same disk. After Step 2, the blocks are stored in standard consecutive format. In fact they are stored in fixed locations like the blocks of the context. See Figure 2. This is possible because we know that each virtual processor receives and sends messages of total size $\leq \gamma$.

LEMMA 1. *Steps* 1(a), 1(b), *and* 1(e) *of algorithm CompoundSuperstep have computation time* $O(\mu v)$, *memory* $\Theta(k\mu)$, *I/O time* $O(G(v\mu/DB))$, *and disk space* $O(v\mu/DB)$ *blocks per disk*.

PROOF. We have to read and write $k\mu/B$ blocks for the simulation of each simulated superstep. The distribution allows us to access D tracks in parallel at a time. Hence, we need the following number of I/O operations for one simulated superstep:

$$\sum_{j=1}^{v/k} \left\lceil \frac{\mu k}{DB} \right\rceil \leq \frac{v}{k}\frac{\mu k}{DB} + \frac{v}{k}.$$

Hence, we have to perform $O(v\mu/DB)$ I/O operations. Additionally, we need $O(DB(v\mu/DB)) = O(v\mu)$ local computation steps. □

The blocks representing simulated message traffic are divided into buckets by the simulation. The contents of a bucket are written to the available disks in a series of write cycles. In each write cycle, at most one block is written to any disk.

LEMMA 2. *Let R be the number of blocks in each bucket. Let $X_{j,k}$ be a random variable representing the number of tracks of disk k that belong to bucket j* (*a track belongs to bucket j, when it contains a record of bucket j*). *Then*, *for any fixed j, k we have the following*:

$$\Pr\left[X_{j,k} \geq l\frac{R}{D}\right] \leq \exp\left(-\Omega\left(\frac{l \log l \cdot R}{D}\right)\right).$$

PROOF. The proof is similar to one described in [33]. Let g_t denote the number of disks written to from bucket j during write cycle t, for $1 \le t \le C$, where C is the total number of write cycles used. We have

(1) $$\sum_{1 \le i \le C} g_t \le R.$$

Let G_t be the number of tracks belonging to bucket j that are assigned to disk k in write cycle t. Since only one track can be written to any disk in a write cycle, G_t is restricted to the values 0 and 1. We have $\Pr[G_t = 1] = g_t/D$ and $\Pr[G_t = 0] = 1 - g_t/D$. Let $\mathcal{G}_{G_t}(z)$ be the probability generating function for G_t:

(2) $$\begin{aligned} \mathcal{G}_{G_t}(z) &= \Pr[G_t = 0]z^0 + \Pr[G_t = 1]z^1 \\ &= 1 - \frac{g_t}{D} + \frac{g_t}{D}z \\ &= 1 + \frac{g_t}{D}(z-1). \end{aligned}$$

Let $\mathcal{G}_{X_{j,k}}(z)$ be the probability generating function for $X_{j,k}$. We can bound $X_{j,k}$ by the sum of independent random variables: $X_{j,k} \le G_1 + G_2 + \cdots + G_C$. For purpose of bounding, we consider that $X_{j,k} = G_1 + G_2 + \cdots + G_C$. Since the sum of independent random variables corresponds to the product of the corresponding probability generating functions and using (2), we have

(3) $$\begin{aligned} \mathcal{G}_{X_{j,k}}(z) &= \mathcal{G}_{G_1+G_2+\cdots+G_C}(z) \\ &= \mathcal{G}_{G_1}(z) \times \mathcal{G}_{G_2}(z) \times \cdots \times \mathcal{G}_{G_C}(z) \\ &= \prod_{1 \le t \le C} \left(1 + \frac{g_t}{D}(z-1)\right). \end{aligned}$$

By the tail estimate Lemma 8 we have

(4) $$\Pr\left[X_{j,k} \ge l\frac{R}{D}\right] \le \frac{E[\exp(rX_{j,k})]}{\exp(rlR/D)}$$

for each $r \ge 0$. We can express the numerator in (4), using (3) and the definitions of expected value and probability generating function, as

(5) $$\begin{aligned} E[e^{rX_{j,k}}] &= \sum_{t \ge 0} \Pr[e^{rX_{j,k}} = e^{rt}]e^{rt} \\ &= \sum_{t \ge 0} \Pr[X_{j,k} = t]e^{rt} \\ &= \mathcal{G}_{X_{j,k}}(e^r) \\ &= \prod_{1 \le t \le C} \left(1 + \frac{g_t}{D}(e^r - 1)\right). \end{aligned}$$

By (1) and convexity arguments, we can maximize (5) by setting $g_t = R/C$ for each t. Thus

$$\begin{aligned} E[e^{rX_{j,k}}] &\le \prod_{1\le t\le C}\left(1+\frac{R\cdot(e^r-1)}{DC}\right) \\ &= \left(1+\frac{R\cdot(e^r-1)}{DC}\right)^C. \end{aligned}$$

Substituting this bound into (4), we get

$$\text{(6)} \qquad \Pr\left[X_{j,k}\ge l\frac{R}{D}\right] \le \frac{(1+R(e^r-1)/DC)^C}{\exp(rlR/D)}.$$

From the bound $(1+a)^b \le e^{ab}$, for $a > -1$, we can approximate the numerator in (6) and get for $r = \ln l$,

$$\begin{aligned} \Pr\left[X_{j,k}\ge l\frac{R}{D}\right] &\le \frac{\exp(R\cdot(e^r-1)/D)}{\exp(rlR/D)} \\ &= \exp\left(\frac{R\cdot(e^r-1)-rlR}{D}\right) \\ &= \exp\left(-\Omega\left(\frac{l\log lR}{D}\right)\right). \end{aligned}$$

□

LEMMA 3. *The computation time of algorithm SimulateRouting is* $O(l\gamma v)$ *and its I/O time is* $O(Gl(v\gamma/DB))$ *with probability* $1-e^{-\Omega(l\cdot\log l\cdot\log(M/B))}$ *for* $v \ge kD\log(M/B)$ *and constant* $l \ge 1$.

PROOF. For the purposes of the proof we assume that the maximum amount of communication is required. This can be accomplished by the introduction of dummy blocks if necessary. Each bucket contains $R = v\gamma/DB$ blocks since each of the D buckets contains the messages destined for v/D virtual processors. Hence, for $M \ge DB$, and $v/k \ge D\log(M/B)$,

$$\begin{aligned} &\text{(7)} \qquad R = \frac{v\gamma}{DB} \ge \frac{k\gamma D\log(M/B)}{DB} \\ &\text{(8)} \qquad R \ge \frac{k\gamma\log(M/B)}{B}. \end{aligned}$$

By Lemma 2, a given disk contains more than $l(R/D)$ records of a given bucket with probability at most $\exp(-\Omega(l\cdot\log l(R/D)))$. Let X denote the event that any disk contains more than $l(R/D)$ blocks of any bucket. There are D drives and D buckets, so with (8) and $k\gamma \ge DB$, we have

$$\begin{aligned} \Pr[X] &\le D^2\exp\left(-\Omega\left(l\log l\frac{R}{D}\right)\right) \\ &\le D^2\exp\left(-\Omega\left(l\log l\frac{k\gamma\log(M/B)}{DB}\right)\right) \end{aligned}$$

$$\leq \exp\left(-\Omega\left(l \log l \frac{DB\log(M/B)}{DB} + \log D\right)\right)$$
$$\leq \exp\left(-\Omega\left(l \log l \cdot \log \frac{M}{B}\right)\right).$$

After D iterations of Step 1 in algorithm SimulateRouting, D blocks per bucket have been moved. Each disk contains less than $v\gamma/D^2B$ blocks of each bucket with high probability. Thus, after $D(v\gamma/D^2B)$ iterations, all blocks have been moved.

In Step 2 $(v\gamma/DB)$ iterations are performed. During each iteration, a parallel read and a parallel write operation are performed.

Thus, the total I/O time of algorithm SimulateRouting is $O(G(lv\gamma/DB))$ and the total computation time is $O(lv\gamma)$ with high probability. $\square$

LEMMA 4. *A compound superstep of a v-processor BSP* with computation time $\tau + L$, communication time $g\gamma/b + L$, and local memory size μ can be simulated on a single-processor EM-BSP* in computation time $v\tau + O(lv\gamma)$ and I/O time $O(Gl(v\gamma/DB))$ with probability* $1 - \exp(-\Omega(l \log l \cdot \log(M/B)))$ *for constant* $l \geq 1$, $v \geq kD\log(M/B)$, $M = \Theta(k\mu)$, $b \geq B$, *and arbitrary integer k.*

PROOF. Since the local memory of a virtual processor is large enough to store the incoming messages and we need μ memory to store the context, we need $M \geq k\mu$ memory in the EM-BSP* machine.

The disk space needed by the simulation is the total context size $v\mu$, which includes space for incoming messages. By Lemma 2, the communicated data is evenly distributed over the disks with high probability. Therefore we need in total $O(v\mu/(DB))$ space on each disk.

Step 1(c) of algorithm CompoundSuperstep consumes $v\tau$ computation time. For each batch of k virtual processors, $k\gamma/b$ messages are generated. This adds $O(v\gamma)$ computation time overall.

During Step 1(d) of algorithm CompoundSuperstep, a permutation can be generated in $O(D)$ time, so the computation time for each batch is $O(D(v\gamma/DB) + k\gamma)$ and I/O time $O(k\gamma/DB)$, giving computation time $O(v\gamma)$ and I/O time $O(G(v\gamma/DB))$ for the whole simulation.

By Lemmas 1 and 3 the computation time and I/O time, respectively, for Steps 2, 1(a), 1(b), and 1(d) are $O(v\gamma)$ and $O(G \cdot l(v\gamma/DB))$. Thus, overall, the computation time is $v\tau + O(l\gamma v)$ and I/O time is $O(G \cdot l(v\gamma/DB))$ with probability $1 - \exp(-\Omega$ $(l \cdot \log l \cdot \log(M/B)))$. $\square$

Lemma 5 allows us to exploit the independence of the random experiments performed during each compound superstep in order to prove that the success probability for the entire simulation is as large as for the simulation of a single compound superstep.

LEMMA 5. *Let $X_1, X_2, \ldots, X_z$ be independent random variables such that* $\Pr[X_i > lT] \leq \exp(-l \log l \cdot m)$ *and* $\Pr[X_i \leq lT] \geq 1 - \exp(-l \log l \cdot m)$, *where* $m \geq \ln x$ *and*

$l \geq 2$. Let $T_{wc} \leq xT$ be the worst-case size of X_i for any i, that is, $X_i \leq T_{wc}$. Then

$$\Pr\left[\sum_{i=1}^{z} X_i \leq e^2(l+1)zT\right] \geq 1 - e^{-\Omega(l \log l \cdot m)}.$$

PROOF. We identify two cases: (1) $z \geq xm^c$ and (2) $z \leq xm^c$ for $c = 1 + (\log\log l)/\log m$.

First, we consider Case (1). The mean of the quantity $\sum_{i=1}^{z} X_i$ can be bounded from above as follows for suitable constant $l \geq 2$ and $m \geq \ln x$:

$$\begin{aligned}
(9) \qquad E\left[\sum_{i=1}^{z} X_i\right] &\leq \sum_{i=1}^{z} (lT \cdot \Pr[X_i \leq lT] + T_{wc} \cdot \Pr[X_i > lT]) \\
&\leq zlT(1 - \exp(-l \log l \cdot m)) + zxT \exp(-l \log l \cdot m) \\
&\leq zlT + zT \exp(-l \log l \cdot m + \ln x) \\
&\leq zlT(1 - \exp(-l \log l \cdot m)) + zxT \exp(-l \log l \cdot m) \\
&\leq zlT + zT \exp(-l \log l \cdot m + \ln x) \\
&\leq (l+1)zT.
\end{aligned}$$

We can bound the mean $E[\sum_{i=1}^{z} X_i]$ from below as follows:

$$\begin{aligned}
E\left[\sum_{i=1}^{z} X_i\right] &\geq \sum_{i=1}^{z} (\Pr[X_i \leq lT] \cdot lT + \Pr[X_i > lT] \cdot T) \\
(10) \qquad &\geq (l-1)zT \\
(11) \qquad &\geq (l-1)xm^cT.
\end{aligned}$$

From Lemma 9, using $k = T_{wc} \leq xT$, $m = E[\sum_{i=1}^{z} X_i] \geq (l-1)xm^cT$, and $u = e^2$,

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{z} X_i \geq e^2 m\right] &\leq \exp\left(-e^2 \frac{(l-1)xm^cT}{xT}\right) \\
&\leq \exp(-(l-1)m^c) \\
&\leq \exp(-(l-1)\log l \cdot m).
\end{aligned}$$

Now, we consider Case (2). We repeat the experiment only $z \leq xm^c$ times, where $c = 1 + (\log\log l)/\log m$. Thus we have, with $m \geq \ln x$,

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{z} X_i \leq zlT\right] &\geq (1 - \exp(-l \log l \cdot m))^z \\
&\geq 1 - xm^c \exp(-l \log l \cdot m) \\
&\geq 1 - \exp(-l \log l \cdot m + c \ln m + \ln x). \qquad \square
\end{aligned}$$

LEMMA 6. *A v-processor BSP* algorithm $\mathcal{A}$ with communication time $g\alpha/b + \lambda L$, computation time $\beta + \lambda L$, and local memory μ can be simulated on a single-processor EM-BSP* with computation time $(1 + o(1))v\beta$ and I/O time $O(G\lambda(lv\mu/DB))$ with probability $1-\exp(-\Omega(l \log l \cdot \log(M/B)))$ for suitable $l \geq 1, \beta = \omega(\lambda\mu), M = \Theta(k\mu)$, $v \geq kD\log(M/B)$, $b \geq B$, and arbitrary integer k.*

PROOF. Since we assume that the amount of communication each BSP* processor performs per superstep is bounded by its memory size μ, we can conclude that the communication time of each compound superstep is bounded by $g\mu/b + L$.

Using Lemma 4, we can simulate on an EM-BSP* machine with D disks and local memory $\Theta(k\mu)$ a compound superstep with communication time $\tau + L$ and computation time $g\mu/b + L$ in computation time $v\tau + O(lv\mu)$ and I/O time $O(G(lv\mu/DB))$ with probability at least $1 - \exp(-\Omega(l \log l \cdot \log(k\mu/B)))$.

The computation time required to simulate the computation steps of $\mathcal{A}$ is $v\beta$. The computational overhead is $O(lv\gamma\lambda)$, which is asymptotically smaller than $v\beta\lambda$ for $\beta = \omega(\mu\lambda)$ and constant l.

Since the worst case runtime of a compound superstep is at most D times the average runtime, the claim follows from Lemma 5. $\square$

5.2. *The Multiprocessor Case $p \geq 1$.* In this section we generalize the simulation to $p \geq 1$ processors on the target EM machine.

We first describe the simulation of a compound superstep of a v-processor BSP* with communication time $g\gamma/b + L$, computation time $\tau + L$, and context size μ on a p-processor EM-BSP*, for $v \geq kpD\log(M/B)$ and $p \geq 1$. We assume further that $b \geq B$, where b is the message block size of the BSP* virtual machine.

Outline of the parallel simulation. As an initial step, v/p virtual processors $i(v/p), \ldots,$ $(i + 1)(v/p) - 1$ are assigned to each simulating processor i. As before, the simulation of each compound superstep is composed of a series of rounds. During the jth of v/pk rounds processor i simulates the steps of the k virtual processors $i(v/p) + jk, \ldots, i(v/p) + (j + 1)k - 1$. Messages sent between virtual processors on different real machines require real communication by the simulation. If these messages are sent directly to their destinations by the simulation the traffic may be unbalanced, causing inefficiencies in communication. Therefore, we distribute the messages randomly among the processors after each round. After the last round of a compound superstep, each processor reorganizes the messages it has received so that during the fetching phase of the jth round of the next compound superstep it can read the messages destined for the virtual processors $jkp, \ldots, (j + 1)kp - 1$ in parallel from disk and direct them to the correct simulating processor.

We maintain v/pk batches to store the generated messages. The jth batch contains the messages destined for virtual processors simulated in the jth round of the current compound superstep. Each processor writes its share of the v/pk batches to its local disks. The main difficulty is the efficient maintenance of the batches so that the packets which are needed during the jth simulation round can be read in parallel from the disks.

Algorithm 3: ParCompoundSuperstep

Objective: Simulation of a compound superstep of a v-processor BSP* on a p-processor EM-BSP*.
Input: For every batch j, where $0 \leq j \leq v/pk$, the message and context blocks of the virtual processors $jkp, \ldots, (j+1)kp-1$ are divided among the real processors and their local disks as follows:

- Each real processor holds $O(k\gamma/B)$ blocks of messages and $k\mu/B$ blocks of context.
- Each local disk contains $O(k\gamma/DB)$ blocks of messages and $O(k\mu/DB)$ 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
    For $i = 0$ to $p - 1$ do in parallel
    (a) (Fetching Phase): processor $i$ reads any message blocks pertaining
        to batch $j$ from its local disks (in parallel) and sends them to the
        appropriate simulating processors.
    (b) (Computing Phase): processor $i$ simulates the computation super-
        steps of its current virtual processors and collects all generated
        messages in its local memory.
    (c) (Writing Phase): processor $i$ splits all generated messages into
        packets of size $b$ (the message size) and sends each packet to a
        randomly chosen processor.
(2) For $i = 0$ to $p - 1$ do in parallel: processors $i$ reorganize the received
    batches using algorithm SimulateRouting so that each batch is evenly
    distributed over the local disks.
```

Many of the details of algorithm ParCompoundSuperstep are similar to ones previously described for Algorithm 1, so we focus on the differences. In each round a processor receives $k\gamma/b$ packets with high probability. In Step 1(c) each processor cuts the packets it receives into blocks of size B. The blocks are written to the disks, using a random permutation of disk numbers as before. Depending on their destination address, the blocks are maintained in D buckets, which are divided among the disks as in Step 1(d) of the single-processor simulation. Each bucket contains the blocks for $(v/pk)/D$ batches. As before, a table of D pointers is maintained for each disk. As before, we can show that each disk will receive the same number of blocks of every bucket with high probability.

LEMMA 7. *A compound superstep of a v-processor BSP* with computation time $\tau + L$, communication time $g\gamma/b + L$, and local memory size μ can be simulated on a p-processor EM-BSP* in computation time $\tau(v/p) + O(l(v/p)(\gamma + \mu) + L(v/pk))$, communication time $O(gl(v/p)(\gamma/b + \log(M/B)/k) + L(v/pk))$, and I/O time $O(G \cdot l(v/p)(\mu/DB))$ with probability $1 - \exp(-\Omega(l \log l \cdot \log(M/B)))$ for suitable $l \geq 1$,*

$M = \Theta(k\mu)$, $b\log(M/B) = O(M)$, $v = \Omega(kpD \cdot \log(M/B))$, $M/B \geq p^{\varepsilon}$, and arbitrary constants $k, \varepsilon > 0$.

PROOF. Each of the v/pk batches contains the packets generated by pk simulated processors. We ensure that each batch contains $pk\gamma/b$ packets by creating dummy packets if necessary.

We first consider the runtime of Step 1 for a fixed round j:

Step 1(a): Each processor reads the blocks belonging to batch j from its local disks in I/O time $O(G(k\gamma/DB))$. The blocks destined for a common processor are combined into packets of size b, and all packets are then sent to their destination in a single superstep. Each simulating processor sends $O(k\gamma/b)$ message packets, since each holds $O(k\gamma/B)$ blocks of messages. Each simulating processor receives $k\gamma/b$ packets. Thus an $O(k\gamma/b)$-relation is routed, consuming $O(g(k\gamma/b) + L)$ communication time. The processors can reassemble messages from received packets in linear time since each receives $k\gamma$ data per round and sufficient real memory exists to hold all of the received messages. Each processor then reads the contexts of its k currently simulated processors in I/O time $O(G(k\mu/DB))$ from its local disks.

Step 1(b): The steps of the currently simulated processors are performed, and the contexts are written back to the disks. Since $\gamma = O(\mu)$, the I/O time is $O(Gl(k\mu/DB))$ and the computation time is $k\alpha + O(lk\mu + L)$.

Step 1(c): During the previous step, message packets of total size $O(k\gamma)$, each one of size b, were generated and stored in the local memories of the simulating processors. Now each processor sends each of its $O(k\gamma/pb)$ packets to a randomly chosen processor. This corresponds to randomly throwing $p(k\gamma/b)$ balls into p bins. Let us assume for the sake of the proof that each processor sends at least $\log(M/B)$ messages.[5]

With Lemma 10 for $x = p(k\gamma/b + \log(M/B))$ we can show that each processor receives more than $l(k\gamma/b+\log(M/B))$ packets with probability $\exp(-\Omega(l \ln l \cdot (k\gamma/b + \log(M/B)) - \ln p))$.

Since these messages are exchanged in one superstep, an $O(l(k\gamma/b + \log(M/B)))$-relation is routed which requires communication time $O(gl(k\gamma/b + \log(M/B)) + L)$. After the packets have been received they are written to the disks. Let $b\log(M/B) = O(M)$ and $M = \Theta(k\mu)$. Thus each processor receives data of size $O(k\mu)$ which is written to its disks in I/O time $O(Gl(k\mu/DB))$.

For a single round the I/O time is $O(Gl(k\mu/DB))$, communication time is $O(gl(k\mu/b + \log(M/B)) + L)$, and computation time is $k\alpha + O(kl(\gamma + \mu) + L)$ with probability $\exp(-\Omega(l \ln l \cdot (k\gamma/b + \log(M/B)) - \ln p))$. For $M/B \geq p^{\varepsilon}$ the probability becomes $1 - \exp(-\Omega(l \log l \cdot \log(M/B)))$.

Now we consider the runtime for v/pk rounds. The rounds are independent. We introduce $z = v/pk$ independent random variables $X_1, X_2, \ldots, X_z$, where X_i represents the cost (communication, computation, and I/O time) of the ith round. Since $\Theta(\log(M/B)) = \ln p$, we can apply Lemma 5 to bound the total cost of all v/pk rounds after making the substitution in the identity $x = p$, and $m = \Theta(\log(M/B))$. In total, we have I/O time $O(G \cdot l(v\mu/pDB))$, communication time $O(gl(v\gamma/pb +$

[5] This condition can be easily achieved by the introduction of dummy packets.

$v\log(M/B)/pk) + L(v/pk))$, and computation time $\alpha(v/p) + O(l(v/p)(\gamma + \mu) + L(v/pk))$ with probability $1 - \exp(-\Omega(l\log l \cdot \log(M/B)))$.

We now examine the costs of reorganizing the batches in Step 2 of algorithm ParCompoundSuperstep. We begin by considering how the packets belonging to a fixed bucket are distributed among the processors. A bucket contains the blocks/packets of v/pkD batches and each batch contains $kp\gamma/b$ packets. In total, for each bucket, $v\gamma/Db$ packets are distributed randomly among p processors. Using Lemma 10 we can show that any processor receives more than $l(v\gamma/pDb)$ packets with probability $\exp(-\Omega(l\ln l(v\gamma/pDb) - \ln p))$. Since we have D buckets, the probability that any processor receives more than $R = l(v\gamma/pDb)$ packets for any bucket is $\exp(-\Omega(l\ln l(v\gamma/pDb) - \ln p - \ln D))$. With $v = \Omega(kpD\log(M/B))$ we have probability $1 - \exp(-\Omega(l\log l \cdot \log(M/B)))$ that each bucket of every processor contains less than $l(v\gamma/pDb)$ packets.

We know that each processor holds $R = v\gamma/pDb$ packets for each bucket. For a fixed processor we have a similar situation to the single simulating processor case. For the sake of the proof, we introduce dummy blocks so that each bucket contains $R' = v\mu/pDB$ blocks.

With $v = \Omega(kpD\log(M/B))$ and $DB = \Theta(k\gamma)$, we have

$$R' = \frac{v\mu}{pDB} \geq \frac{k\mu\log(M/B)}{B}. \tag{12}$$

By Lemma 2, we know that a fixed drive contains less than $l(R'/D)$ blocks of a fixed bucket with probability at most $\exp(-\Omega(l\log l(R'/D)))$. Let X denote the event that any disk contains more than $l(R'/D)$ blocks of any bucket.

There are D drives and D buckets, so with 12 and $R' \geq k\mu\log(M/B)/B$ we have

$$\begin{aligned}
\Pr[X] &\leq D^2\exp\left(-\Omega\left(l\log l \cdot \frac{R'}{D}\right)\right)\\
&\leq \exp\left(-\Omega\left(l\log l \cdot \frac{k\mu\log(M/B)}{DB} + \log D\right)\right)\\
&\leq \exp\left(-\Omega\left(l\log l \cdot \frac{k\mu\log(M/B)}{DB} + \log\frac{M}{B}\right)\right)\\
&\leq \exp\left(-\Omega\left(l\log l \cdot \log\frac{M}{B}\right)\right).
\end{aligned}$$

Because we have p processors, any processor needs more than $O(G \cdot l(v\gamma/pB))$ I/O time with probability at most $\exp(-\Omega(l\log l \cdot \log(M/B) - \log p))$. Again, assuming $M/B \geq p^{\varepsilon}$, this probability becomes $\exp(-\Omega(l\log l \cdot \log(M/B)))$.

Hence, in Step 2 each processor has $O(k\gamma/b)$ packets for each bucket and they can be reorganized as required by the input in algorithm ParCompoundSuperstep in I/O time $O(Gl(k\mu/BD))$ with probability $1 - \exp(-\Omega(l\log l \cdot \log(M/B)))$. □

5.3. *The Main Result*. The following theorem states our main simulation result.

THEOREM 1. *A v-processor BSP* algorithm $\mathcal{A}$ with communication time $g\alpha/b + \lambda L$, computation time $\beta + \lambda L$, and local memory μ can be simulated on a p-processor EM-BSP* with computation time $(1 + o(1))(v/p)\beta + O(L\lambda(v/pk))$, communication time $O(gl((v/p)(\alpha/b) + \lambda(\log(M/B)/k)) + L\lambda(v/pk))$, and I/O time $O(Gl(v/p)(\mu\lambda/BD))$ with probability $1 - \exp(-\Omega(l \log l \cdot \log(M/B)))$ for suitable $l \geq 1$, $\beta = \omega(\lambda\mu)$, $M = \Theta(k\mu)$, $v = \Omega(pkD \cdot \log(M/B))$, $M/B \geq p^{\varepsilon}$, $b \log(M/B) = O(M)$, and arbitrary constants k, $\varepsilon > 0$.*

PROOF. Algorithms 1–3 and Lemmas 1–7.

Since the local memory of a virtual processor is large enough to store the incoming messages and we need μ memory to store the context, we need $M \geq k\mu$ memory in the EM-BSP* machine.

The disk space needed by the simulation is the total context size $v\mu$, which includes space for incoming messages. By Lemma 2, the communicated data is evenly distributed over the disks with high probability. Therefore, in total, the space used on each disk is $O(v\mu/DB)$.

Step 1(c) of Algorithm 1 consumes $v\tau$ computation time. For each batch of k virtual processors, $k\gamma/b$ messages are generated. This adds $O(v\gamma)$ computation time overall.

During Step 1(d) of Algorithm 1, a permutation can be generated in $O(D)$ time, so the computation time for each batch is $O(D(v\gamma/DB) + k\gamma)$ and the I/O time is $O(k\gamma/DB)$. Overall, the computation time is $O(v\gamma)$ and the I/O time is $O(G(v\gamma/DB))$ for the whole simulation.

By Lemmas 1 and 3 the computation time and I/O time, respectively, for Steps 2, 1(a), 1(b), and 1(d) are $O(v\gamma)$ and $O(G \cdot l(v\gamma/DB))$. Thus, overall, the computation time is $v\tau + O(l\gamma v)$ and the I/O time is $O(G \cdot l(v\gamma/DB))$ with probability $1 - \exp(-\Omega(l \cdot \log l \cdot \log(M/B)))$. □

The slackness v/p required by the simulation is controlled by the number of processors and disks we want to employ as well as the desired success probability. The condition $M/B \geq p^{\varepsilon}$ is usually fulfilled for actual machines. Combining Theorem 1 with Observation 1, we obtain the following.

COROLLARY 1. *A CGM algorithm $\mathcal{A}$ with communication time $T_{\text{comm}} = \lambda H_{n,p}$, computation time T_{comp}, and local memory M can be simulated by an EM-CGM algorithm $\mathcal{A}'$ with communication time $\tilde{O}(g\lambda(n/pb + \log(M/B)) + \lambda LD \log(M/B))$, computation time $\tilde{O}(T_{\text{comp}} + \lambda LD \log(M/B))$, and I/O time $\tilde{O}(\lambda G(M/BD))$ for $T_{\text{comp}} = \omega(\lambda M)$, $b \log(M/B) = O(M)$, and $M/B \geq p^{\varepsilon}$ (fixed constant $\varepsilon > 0$).*

5.4. *A Note On C-Optimal BSP* Algorithms*. Since even small multiplicative constant factors in runtime are important, Bäumker et al. [9] characterize the performance of a BSP* algorithm $\mathcal{A}^*$ as c-optimal if (a) the ratio between the computation times of $\mathcal{A}^*$ and $T(\mathcal{A})/p$, where $T(\mathcal{A})$ is the runtime of the best sequential algorithm for the problem under consideration, is $c + o(1)$ and (b) the ratio between the communication time of $\mathcal{A}^*$ and the computation time $T(\mathcal{A})/p$ is $o(1)$.

Consider the following natural extension of c-optimality to EM-BSP* algorithms. An EM-BSP* algorithm $\mathcal{A}^*$ is c-optimal if (a) the ratio between the computation times of $\mathcal{A}^*$ and $T(\mathcal{A})/p$, where $T(\mathcal{A})$ is the runtime of the best sequential algorithm for the problem under consideration, is $c + o(1)$, (b) the ratio between the communication time of $\mathcal{A}^*$ and the computation time $T(\mathcal{A})/p$ is $o(1)$, and (c) the ratio between the I/O time of $\mathcal{A}^*$ and the computation time $T(\mathcal{A})/p$ is $o(1)$.

We observe that Theorem 1 preserves c-optimality, i.e., a c-optimal BSP* algorithm is converted into c-optimal EM-BSP* algorithms.

OBSERVATION 2. *If algorithm $\mathcal{A}$ in Theorem* 1 *is c-optimal on the BSP* for $g \le g(n)$, $b \le b(n)$, $L \le L(n)$, and $v \le v(n)$, then the simulation according to Theorem* 1 *results in a c-optimal EM-BSP* algorithm for* $\lambda(\log(M/B)/k) = O(\alpha/b)$, $g \le g(n)$, $b \le b(n)$, $L \le L(n) \cdot pk/v$, *and* $G = BD \cdot o(\beta/\mu\lambda)$.

6. Applications. In this section we study applications of Theorem 1 and Corollary 1. Recall from Corollary 1 that the simulation of a CGM algorithm $\mathcal{A}$ with λ supersteps results in a parallel EM algorithm with I/O time $\tilde{O}(\lambda G(M/BD))$. In practice, this means that the parallel EM algorithm reads the entire disk contents λ times. Therefore, it is crucial that the underlying CGM algorithm has a *very* small λ. Fortunately, this is the case, as demonstrated in Table 1. In fact, several important problems have CGM algorithms with $\lambda = O(1)$, resulting in parallel EM algorithms with I/O time $\tilde{O}(G(M/BD))$ which is optimal.

The first column of Table 1 lists three groups of applications. (A) Fundamental algorithms: sorting, permutation, and matrix transpose. (B) GIS and computational geometry algorithms: polygon triangulation, trapezoidal decomposition, segment tree construction, next element search on line segments batched planar point location, 3D convex hull, 2D Voronoi diagram, Delaunay triangulation, lower envelope of nonintersecting line segments, area of union of rectangles, 3D-maxima, 2D-nearest neighbors, 2D-weighted dominance counting, uni- and multi-directional separability. (C) Graph algorithms: list ranking, euler tour in a tree, lowest common ancestor, tree contraction, expression tree evaluation, connected components, spanning forest, ear and open ear decomposition, and biconnected components. The second column of Table 1 shows the I/O complexity of previously known EM algorithms for these problems in Vitter's parallel disk model (PDM) [33]. The third column of Table 1 shows the complexity of known parallel CGM algorithms for these problems. Note that $\lambda = O(1)$ for the CGM algorithms for Groups A and B, and $\lambda = O(\log p)$ for the CGM algorithms for Group C. For most applications, the number of processors, p, is fixed and and not very large, and $\log p$ is a fairly small number.

The fourth column of Table 1 shows the complexity of the parallel EM algorithms obtained through our simulation technique (Theorem 1 and Corollary 1). We observe that the I/O complexity of the parallel EM algorithms obtained is $\tilde{O}(G(n/pBD))$ for the problems in Groups A and B and $\tilde{O}(G \log(p)(n/pBD))$ for the problems in Group C. In many cases this is a considerable improvement. The I/O complexity of our parallel EM

Table 1. New parallel EM algorithms obtained and comparison with previous sequential EM methods. See Appendix A.2 for notation.

Problem description	**Previous Results:** Sequential I/O complexity (one processor, multiple disks)	CGM complexity (multiple processors, no disks)	**New Results:** Parallel EM-CGM complexity (multiple processors, multiple disks)
Group A: Fundamental Algorithms			
Sorting	$\Theta\left(G \frac{n}{BD} \log_{M/B} \frac{n}{B}\right)$ [1], [31]	$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$ [21]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Permutation	$\Theta\left(G \min\left(\frac{n}{D}, \frac{n}{DB} \log_{M/B} \frac{n}{B}\right)\right)$ [1], [31]	$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Matrix transpose (r rows, c cols, $n = r \cdot c$.)	$\Theta\left(G \frac{n}{BD} \frac{\log \min(M, r, c, n/B)}{\log(M/B)}\right)$ [1], [31]	$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Group B: GIS and Computational Geometry Algorithms			
Polygon triangulation, Trapezoidal decomposition, Segment tree construction, Next element search on line segments	$O\left(G \frac{n}{B} \log_{M/B} \frac{n}{B}\right)$ [5]	$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n \log n}{p}\right)$ [12]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n \log n}{pB}\right) + LD \log \frac{n \log n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n \log n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n \log n}{pBD}\right)$
Batched planar point location	$O\left(G\left(\frac{n}{B} + k\right) \log_{M/B} \frac{n}{B}\right)$ [5]	$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n \log n}{p}\right)$ [12]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n \log n}{pB}\right) + LD \log \frac{n \log n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n \log n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n \log n}{pBD}\right)$
3D convex hull, 2D Voronoi diagram, Delaunay triangulation	$O\left(G \frac{n}{B} \log_{M/B} \frac{n}{B}\right)$ [22]	$\lambda = \tilde{O}(1)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$ [16]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Lower envelope of non-intersecting line segments		$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$ [19]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Generalized lower envelope of line segments		$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n\alpha(n)}{p}\right)$ [19]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n\alpha(n)}{pB}\right) + LD \log \frac{n\alpha(n)}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n\alpha(n)}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n\alpha(n)}{pBD}\right)$
Area of union of rectangles, 3D-maxima, 2D-nearest neighbors	$O\left(G \frac{n}{B} \log_{M/B} \frac{n}{B}\right)$ [22]	$\lambda = O(1)$ $T_{\mathrm{comm}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$ [19]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
2D-weighted dominance counting, Uni- and multi-directional separability		$\lambda = O(1)$ $T_{\mathrm{comp}} = O\left(\frac{n \log n}{p}\right)$ $M = O\left(\frac{n}{p}\right)$ [19]	$T_{\mathrm{comm}} = \tilde{O}\left(g\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log n}{p} + LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \frac{n}{pBD}\right)$
Group C: Graph Algorithms			
List ranking, Euler tour (tree), Lowest common ancestor, Tree contraction, Expression tree evaluation	$O\left(G \frac{n}{B} \log_{M/B} \frac{n}{B}\right)$ [14]	$\lambda = O(\log p)$ $T_{\mathrm{comp}} = O\left(\frac{n}{p} \log p\right)$ $M = O\left(\frac{n}{p}\right)$ [11]	$T_{\mathrm{comm}} = \tilde{O}\left(g \log(p)\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + \log(p) LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n \log p}{p} + \log(p) LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \log(p) \frac{n}{pBD}\right)$
Connected components, Spanning forest, Ear and open ear decomposition, Biconnected components (V vertices, E edges, $n = V + E$)	$O\left(G \frac{E}{DB} \log_{M/B} \frac{V}{B} \cdot \max\left\{1, \log\log \frac{VBD}{E}\right\}\right)$ [24]	$\lambda = O(\log p)$ $T_{\mathrm{comp}} = O\left(\frac{n}{p} \log p\right)$ $M = O\left(\frac{n}{p}\right)$ [11]	$T_{\mathrm{comm}} = \tilde{O}\left(g \log(p)\left(\frac{n}{pb} + \log \frac{n}{pB}\right) + \log(p) LD \log \frac{n}{pB}\right)$ $T_{\mathrm{comp}} = \tilde{O}\left(\frac{n}{p} \log p + \log(p) LD \log \frac{n}{pB}\right)$ $T_{\mathrm{I/O}} = \tilde{O}\left(G \log(p) \frac{n}{pBD}\right)$

algorithms for the problems in Groups A and B is optimal. Furthermore, the EM-CGM algorithms obtained through our simulation technique are *parallel*, i.e., yield speedup through the use of multiple processors, while the algorithms in the first column are for single-processor machines only. Another important advantage of our method is that our EM-CGM algorithms are automatically generated through simulation, instead of being individually designed.

7. Conclusion. In this paper we described a simulation technique which produces efficient parallel EM algorithms from efficient BSP-like parallel algorithms. When applied to existing BSP, BSP*, or CGM algorithms, our simulation technique produces improved *parallel* EM algorithms for a large number of problems. Our technique can accommodate one or multiple processors on the EM target machine, each with one or more disks, and it also adapts to the disk blocking factor of the target machine. This allows a scenario where an application that is based on our method could adapt dynamically to the operating parameters and numbers of the available resources such as processors, memory, and disks.

Note that our technique applies only to BSP-like algorithms for which T_{comp} is at least λM; see Theorem 1 and Corollary 1. This is a large class of algorithms, including all those listed in Table 1. Typically, algorithms for problems with at least linear sequential time complexity fall into this category. Algorithms which do not fall into this category are typically for problems with sublinear time complexity. An example of such an algorithm is multisearch [9]. In general, sublinear time external memory data structure search/update is not applicable for our technique. This is a very important open problem for future research.

Appendix

A.1. *Probability Estimates*. We use the following tail estimates:

LEMMA 8. *If X is a non-negative random variable and $r \geq 0$, we have*

$$\Pr[X \geq u] \leq \frac{E[e^{rX}]}{e^{ru}}.$$

PROOF. We use the following Markov inequality. Let X by any random variable. Then, for all $t \in I\!R^{+}$,

$$\Pr[X \geq t] \leq \frac{E[X]}{t}.$$

For any positive real r,

$$\Pr[X \geq u] = \Pr[e^{rX} \geq e^{ru}].$$

Applying the above Markov inequality to the right-hand side, we have

$$\Pr[X \geq u] \leq \frac{E[e^{rX}]}{e^{ru}}. \qquad \square$$

LEMMA 9. *Let $X_1, \dots, X_n$ be independent random variables with $X_i \in [0, \dots, k]$ and $m = E[\sum_{i=1}^n X_i]$. Then for $u \geq e^2$,*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq u \cdot m\right] \leq \exp\left(-u\frac{m}{k}\right).$$

PROOF. Hoeffding [23] showed that for this situation we have

$$\Pr\left[\sum_{i=1}^{n} X_i \geq (\delta+1)\cdot m\right] \leq \left(\frac{e^{\delta}}{(\delta+1)^{(\delta+1)}}\right)^{m/k}.$$

Let $u = \delta - 1$. We have

$$\begin{aligned}\left(\frac{e^{\delta}}{(\delta+1)^{(\delta+1)}}\right)^{m/k} &= e^{(u-1)(m/k)} \cdot u^{-u(m/k)} \\ &= e^{(u-1)(m/k)} \cdot e^{\ln u^{(-u(m/k))}} \\ &= e^{-(m/k)(u\ln u - u + 1)}.\end{aligned}$$

Finally, we have $u \ln u - u + 1 \geq u$ if $\log u \geq 2$, or $u \geq e^2$. □

LEMMA 10. *Given x balls and y bins. If the balls are randomly and independently distributed to the bins, each bin contains more than lx/y balls with probability $1 - e^{-\Omega(l \ln l(x/y) - \ln y)}$.*

PROOF. The probability of the event that a bin receives exactly i balls is

$$\binom{x}{i} \cdot \left(\frac{1}{y}\right)^{i} \cdot \left(1 - \frac{1}{y}\right)^{x-i} \leq \binom{x}{i} \cdot \left(\frac{1}{y}\right)^{i}.$$

Let X be the event that a bin receives more than k balls, and let Y denote the event that at least one of the bins receives more than $k = l(x/y)$ balls. Thus,

$$\Pr[X] = \sum_{i=k+1}^{x} \binom{x}{i} \cdot \left(\frac{1}{y}\right)^{i}.$$

We can conclude that

$$\begin{aligned}\Pr[X] &= \sum_{i=k+1}^{x} \binom{x}{i} \cdot \left(\frac{1}{y}\right)^{i} \\ &\leq \sum_{i=k}^{x} \frac{x!}{i!\,(x-i)!}\frac{1}{y^i}\end{aligned}$$

$$
\begin{aligned}
&= \frac{x!}{k!\,(x-k)!\,y^k} + \frac{x!}{(k+1)!\,(x-(k+1))!\,y^{k+1}} \\
&\quad + \frac{x!}{(k+2)!\,(x-(k+2))!\,y^{k+2}} + \cdots \\
&= \frac{x!}{k!\,(x-k)!\,y^k} \cdot \left(1 + \frac{x-k}{(k+1)y} + \frac{(x-k)(x-(k+1))}{(k+1)(k+2)y^2} + \cdots\right) \\
&\le \binom{x}{k}\frac{1}{y^k} \cdot \left(1 + \frac{x-k}{(k+1)y} + \left(\frac{x-k}{(k+1)y}\right)^2 + \cdots\right) \\
&\le \left(\frac{xe}{ky}\right)^k \cdot \sum_{i\ge 0}\left(\frac{x-k}{(k+1)y}\right)^i .
\end{aligned}
$$

For $k = l(x/y)$ and $l \le y$ we note that

$$
\frac{x-k}{(k+1)y} = \frac{x - l(x/y)}{(l(x/y)+1)y} = \frac{1-l/y}{l + y/x} < 1 - \frac{l}{y} < 1
$$

and so

$$
\sum_{i\ge 0}\left(\frac{x-k}{(k+1)y}\right)^i = \frac{y}{l}.
$$

Thus, for $k = l(x/y)$ and $l > e$,

$$
\begin{aligned}
\Pr[X] &\le \left(\frac{xe}{ky}\right)^k \cdot \sum_{i\ge 0}\left(\frac{x-k}{(k+1)y}\right)^i \\
&= \left(\frac{e}{l}\right)^{lx/y} \cdot \frac{y}{l} \\
&= \frac{y}{l} \cdot e^{l(x/y) - l\ln l(x/y)}.
\end{aligned}
$$

Recall that Y denotes the event that at least one of the bins receives more than $k = l(x/y)$ balls:

$$
\begin{aligned}
\Pr[Y] = y \cdot \Pr[X] &\le \frac{y^2}{l} \cdot e^{l(x/y) - l\ln l(x/y)} \\
&= e^{l(x/y) - l\ln l(x/y) - \ln l + 2\ln y}.
\end{aligned}
$$

So we can conclude that

$$
\Pr[Y] = e^{-\Omega(l\ln l(x/y) - \ln y)}. \qquad \square
$$

A.2. Terminology

Symbol	Meaning
n	The number of data items
p	The number of real processors
b	The minimum packet size for communication
B	The disk block size
D	The number of disk drives on a (real) processor
g	The time required for the router to deliver a packet of size b
$\hat{g}$	The time required for the router to deliver a packet of unit size
G	The time required for a processor to transfer D blocks between its D local disks and its local memory
L	The time required for synchronizing the processors
M	The memory size of a (real) processor
v	The number of virtual processors
γ	The maximum total size of messages sent or received by a virtual processor in a single superstep
μ	The maximum size of the context of a virtual processor
T_{comm}	The communication time
T_{comp}	The computation time
$T_{\text{I/O}}$	The I/O time
λ	The number of supersteps

References

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

[2] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12:72–109, 1994.

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

[5] 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 a special issue of *Algorithmica*) appears as BRICS Technical Report RS-96-12, University of Aarhus.

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

[7] 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/`.

[8] A. Bäumker and W. Dittrich. Fully dynamic search trees for an extension of the BSP model. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 233–242, 1996.

[9] 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.

[10] A. Bäumker, W. Dittrich, and A. Pietracaprina. The deterministic complexity of parallel multisearch. In *Proc. Scandinavian Workshop on Algorithms Theory*, pages 404–415, 1996.

[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. International Colloquium 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. International Parallel Processing Symposium*, pages 320–325, 1997.

[13] Y.-J. Chiang. Dynamic and I/O-Efficient Algorithms for Computational Geometry and Graph Problems: Theoretical and Experimental Results. Ph.D. thesis, Brown University, August 1995.

[14] 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 Symposium on Discrete Algorithms*, pages 139–149, 1995.

[15] T. H. Cormen. Virtual Memory for Data Parallel Computing. Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992.

[16] 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 Symposium on Parallel Algorithms and Architectures*, pages 27–33, 1995.

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

[18] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. In *Proc. ACM Annual Conference on Computational Geometry*, pages 298–307, 1993.

[19] 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.

[20] G. A. Gibson, J. S. Vitter, and J. Wilkes. Report of the working group on storage I/O issues in large-scale computing. *ACM Computing Surveys*, 28(4), December 1996.

[21] M. T. Goodrich. Communication efficient parallel sorting. In *Proc. ACM Symposium on Theory of Computation*, 1996.

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

[23] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, :13–30, 1963.

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

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

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

[27] S. Subramanian and S. Ramaswamy. The p-range tree: a new data structure for range searching in secondary memory. In *Proc. ACM–SIAM Symposium on Discrete Algorithms*, pages 378–387, 1995.

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

[29] D. E. Vengroff. *TPIE User Manual and Reference*. Duke University, 1995. Available via `WWW at http://www.cs.duke.edu/ ~dev`.

[30] D. E. Vengroff and J. S. Vitter. Supporting I/O-efficient scientific computation in TPIE. In *Proc. IEEE Symposium on Parallel and Distributed Computing*, 1995. Appears also as Duke University Department of Computer Science Technical Report CS-1995-18.

[31] J. S. Vitter. External memory algorithms. *Proc. ACM Symposium on Principles of Database Systems*, pages 119–128, 1998.

[32] J. S. Vitter and M. H. Nodine. Large-scale sorting in uniform memory hierarchies. *Journal of Parallel and Distributed Computing*, 17:107–114, 1993.

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