Proc 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'97) June 1997

Efficient External Memory Algorithms by Simulating Coarse-Grained Parallel Algorithms

Frank Dehne *
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
dehne@scs.carleton.ca

Wolfgang Dittrich †
Department of Computer Science
University Of Paderborn
33095 Paderborn, Germany
dittrich@uni-paderborn.de

David Hutchinson ‡
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
hutchins@scs.carleton.ca

Abstract

External memory (EM) algorithms are designed for computational problems in which the size of the internal memory of the computer is only a small fraction of the problem size. For certain large scale applications this is necessarily true. Typically, the cost models proposed for external memory algorithms have measured only the number of I/O operations, and the algorithms have been specially crafted for the EM situation. In the past, several attempts have been made to relate the large body of work based on parallel algorithms to EM, but with limited success.

In this paper we provide simulation techniques which produce efficient EM algorithms from efficient algorithms developed under BSP-like parallel computing models. Our techniques 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. In addition to the main simulation result we obtain a more comprehensive cost model for EM algorithms, which considers the total costs incurred by the algorithm including computation, I/O and communication costs.

1 Introduction

1.1 Overview

External memory (EM) algorithms are designed for computational problems in which the size of the internal memory of the computer is only a small fraction of the size of the problem. For certain large scale applications this is necessarily true. Important applications in Geographic Information Systems (GIS), Virtual Reality, VLSI Design, Weather Prediction, Computerized Medical Treatment, 3D Simulation and Modelling, Visualization and Computational Geometry fall naturally into this category.

With few exceptions, previous authors focussed on a uniprocessor EM model. When parallelism was introduced, it was either in terms of a PRAM-like interconnection, or involved parallel disks rather than parallel processors. We believe that parallel processing should be 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.

In many cases, previous EM algorithms were "new", carefully hand-crafted to work optimally in the EM environment. Existing "internal memory" algorithmic techniques and data structures were often found to be unsuitable for EM. We believe that this is 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. 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 external memory algorithms and BSP-like parallel models such as BSP[23], BSP*[7, 8, 6] and CGM[13, 14, 15]. We provide simulation techniques that map BSP-like algorithms to EM algorithms, and we further show how, using a randomized approach, an EM machine can take full advantage of parallel disk I/O and multiple processors.

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 in practice by carefully ensuring that data on disk is accessed in a block-wise fashion. Thus the overheads of rotational delay and disk arm movement are amortized over the number of items in a disk block. Ensuring that I/O is fully blocked is therefore an important issue in reducing the runtime of an algorithm. 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 at the cost of a single I/O operation. These two issues are fundamental to our approach, since if I/O is not fully blocked, the runtime can typically be up to a factor of 10^3 (the block-

---

*Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

†Research partially supported by DFG-Sonderforschungsbereich 376 "Massive Parallelität" and by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT). This work was started while the second author was visiting Carleton University.

‡Research partially supported by Almerco, Inc. and by the Natural Sciences and Engineering Research Council of Canada.

ing 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 [27].

We identify *coarse-grained communication* as a desirable property of parallel algorithms for our simulation. Coarse grained communication occurs when the average message size of the parallel algorithm is $\Omega(B)$.[1] This permits the EM algorithm to take advantage of the disk block size on the target machine, a fundamental requirement of efficient disk I/O. The BSP* model is an extension of the BSP model which rewards blockwise communication. This feature of BSP* ensures that the I/O operations necessary to simulate message passing can also be done in a blockwise fashion. In Section 4 we describe how BSP and CGM algorithms can be made into BSP* algorithms, and as a result inherit the blockwise I/O property during our simulation.

Much of the EM work to date has used the criterion of I/O-optimality to discriminate between competing EM algorithms. An EM algorithm A is *I/O-optimal* iff the number of I/O operations used matches the lower bound for the number of I/Os required to solve the problem. This is a rather strong criterion and it ignores internal computation cost completely. However, work efficiency is a crucial factor for the efficiency of algorithms.

The criterion of c-optimality was introduced in [16] for the BSP model. It gives an incentive to devise algorithms for which communication time is asymptotically smaller than computation time and the computation time is only by a factor $c+o(1)$ larger than that of an optimal RAM algorithm. For large enough problems the communication overhead is small compared to the computation time which guarantees reasonable speed-ups, around p/c, on an actual p processor machine.

In Section 3.1 we propose an extension to the definition of c-optimality. The new definition additionally provides incentives to devise algorithms for which I/O costs are asymptotically smaller than the computation time. We propose the extended c-optimality criterion as an alternative to the old EM criterion of I/O-optimality.

The introduction of communication cost and computation time makes the new EM goodness criterion more practical on one side, as it measures the run time directly, but on the other side it is a bit weaker than I/O-optimality, as we do not insist on the optimality of either communication or I/O. We only require that they be asymptotically smaller than the computation time. We feel that our criterion is useful, because if the communication and I/O cost have been controlled such that they do not influence the overall runtime, it is the computation overhead that controls the efficiency of the algorithm.

We extend the BSP* parallel computing model to include a component which measures I/O as well as communication cost. We shall refer to the new model as EM-BSP*. Similar extensions to BSP (EM-BSP) and CGM (EM-CGM) can clearly be made, but we restrict ourselves to the BSP* case in the interests of brevity and ease of exposition. Using our new model we prove that our simulation techniques transform certain c-optimal BSP* algorithms to c-optimal EM algorithms running on a single or multiple processor machine. One use of these techniques is as an efficient "parallel virtual memory" implementation. An algorithm utilizing v processors can be efficiently executed using our techniques on a real machine with $p = \lfloor v/r \rfloor$ real processors each one with multiple disks for a large range of integers r.

1.2 Previous Work

Sorting, permutation and related problems in EM have been extensively studied [1, 12, 27, 26, 20]. I/O-optimal approaches to many computational geometry problems [17] and graph problems [11] have also been described. Data structures [3, 5, 22] and a number of applications [4] have been examined in this context. Some implementation work has also been done [24, 25, 10]. A recent survey is included in [5].

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 [26]. 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, and the multilayer models are more complex.

Chiang et al.[11] 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. Examples include problems like sorting, matrix multiplication, convex hull and Voronoi diagram construction; see [21].

Concurrent to the work presented in this paper, Sibeyn and Kaufmann [21] 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.[2] They do not explain, however, how to accommodate the blocking factor, which is an intrinsic issue in efficient I/O design, nor do they provide mechanisms for handling multiple disks or multiple physical processors on the target EM machine.

2 New Results

In this section we highlight the contributions of this paper.

A major contribution of our work is to exploit a natural correspondence between external memory algorithms and parallel models such as BSP, BSP* and CGM whose cost metric is sensitive to communication costs. We identify the blockwise communication characteristic of the BSP* model as an example of a general requirement for *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

[1] In this paper we treat the parameters of the model, including the blocksize B as growing functions of the problem size n. While a given machine will typically have a fixed blocksize, it is reasonable to assume that larger blocks can be written without a further overhead penalty.

[2] We use our notation here: v is the number of virtual (BSP) processors, and μ is the size of the context of a processor.

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. Further, our technique can take full advantage of the physical memory available by simulating a superstep of more than one virtual processor concurrently.

We extend the parallel computation model BSP* [7] to include a component which measures I/O as well as communication cost, and we propose a corresponding extension to the notion of c-optimality as an optimality criterion. Our new c-optimality definition additionally provides incentives to devise algorithms for which I/O time is asymptotically smaller than the computation time.

Using our new model we prove (Theorems 1 and 2 respectively) that our simulation techniques transform certain c-optimal BSP* algorithms to c-optimal EM algorithms running on a single or multiple processor machine.

We stress that the proposed extensions to c-optimality and BSP* apply equally well to the BSP and CGM models, and that our simulation technique preserves the property of c-optimality in these cases.

3 A New EM Model

Due to lack of space we only describe here the BSP* and its EM version in detail, however the corresponding extensions of BSP and CGM are analogous. We begin with a short description of the BSP*, see also [7]. The BSP* model consists of p processor/memory components, a router that delivers messages in a point to point fashion, and a facility to synchronize all processors in a barrier style. Each processor has a unique label in the range $0, 1, \ldots, p-1$, where the processor with label i is denoted by P_i.

A computation proceeds in a succession of *supersteps* separated by synchronizations, usually divided into *communication* and *computation supersteps*. In computation supersteps processors perform local computations on data that is available locally at the beginning of the superstep and issue send operations. In communication supersteps the send operations are implemented, i.e., the exchange of data between the processors is done by the router.

An instance of the p processor BSP* model is characterized by the parameters g, b and L. L is the minimum time (in number of operations) between successive synchronization operations. Thus L is the minimum time for a superstep. The parameter g is the time (in number of operations) the router needs to deliver a packet of size b (in number of machine words) when in continuous use. The parameter b reflects the "optimal" packet size that has to be send in order to achieve almost optimal throughput. There is a natural upper bound for b namely the size of the processor's memory. Further, we assume, $L \geq 1$ and $g \geq 1$.

We now modify the BSP* model to include secondary local memories. The basic idea is illustrated in Figure 1. Each processor has in addition to its local memory an external memory in the form of a set of hard disks. We apply this idea to extend the BSP* model to its external memory version **EM-BSP***, with the following additional 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 local disk drive, and

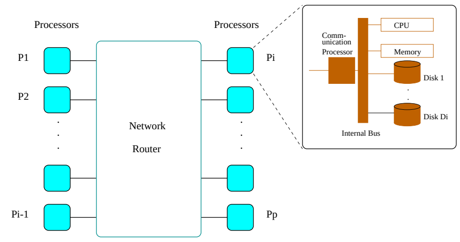

Figure 1: Illustration of a Parallel Machine with External Memory

- G is the ratio of local computational capacity (number of local computation operations) divided by the 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, thus we restrict ourselves to that case, although the model does not forbid explicitly different numbers of drives and memory sizes for each processor. We denote the disk drives of each processor by $D_0, D_1, \ldots D_{D-1}$. 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.

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. We assume further only one track per disk can be accessed without posing any restriction on which track is accessed on each disk. It takes roughly the same amount of time to access and transfer one block or one word. This reflects the fact that the seek time for a record dominates the time to transmit a record, the transfer delay. We also assume that a processor can store in its local memory at least one block from each local disk at the same time, i.e., $M \geq DB$. We allow a single processor version of the EM-BSP* model, which resembles the one processor version of the model found in [27] when $L = g = b = 1$.

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 and communication cost are the same as for the BSP* model. For each local operation we use the RAM uniform cost measure. For an h-relation, i.e., a routing request where each processor sends and receives at most h messages of size b, we charge $g \cdot h + L$ time units in a communication superstep. The *I/O-cost* (or *I/O-time*) of a computation superstep is $t_{I/O} = \max_{j=1}^{p}\{w_{I/O}^{j}\}$ where $w_{I/O}^{j}$ is the I/O-cost incurred by processor j, remember each I/O operation costs G time steps. For a computation superstep with at most t_{comp} local operations on each processor we charge $t_{comp} + t_{I/O} + L$ time units. As in the BSP* model, it is worthwhile to send messages of size at least b, and further the model gives incentives to access all disk drives using block transfers. For instance, a single processor EM-BSP* with D disks is capable of transferring a block of B items to or from each disk in a single I/O operation. An operation involving fewer elements incurs the same cost.

3.1 The c-optimality Criterion

Since even small multiplicative constant factors in runtime are important, we characterize the performance of an EM-BSP* algorithm by comparing its run time with an optimal sequential or PRAM algorithm. We measure ratios between runtimes on pairs of models that have the same set of local instructions and adapt the optimality criteria proposed in [16].

Definition 1 *Let $\mathcal{A}$ be the best sequential algorithm on the RAM for the problem under consideration, and let $T(\mathcal{A})$ be its worst case runtime. Let c be a constant with $c \geq 1$. A* c-optimal *EM algorithm $\mathcal{A}^*$ meets the following criteria:*

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

All asymptotic bounds refer to the problem size n as $n \to \infty$.

This definition applies equally well to EM-BSP, EM-BSP* and EM-CGM algorithms. In the interests of brevity and ease of exposition, we restrict ourselves to the EM-BSP* case. Conditions on the input size n and the EM-BSP* parameters, p, b, g, B, G, L and M are specified that are sufficient for the algorithm to make sense and the bounds on ϕ, ξ and η to hold. The restrictions on the parameters p, b, g, B, G, L and M are functions that grow with the input size n. This guarantees that the algorithms run efficiently on real parallel machines, from those with large parameter values, i.e., large bandwidth and large latency which require large messages to operate efficiently, to those with small network bandwidth and small latency. We shall say that a EM algorithm is *one-optimal* if it fulfills the requirements of the above definition for $c = 1$.

4 The Simulation

4.1 Overview of the Technique

We describe here 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, a computation, and a sending superstep.

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.

Outine 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 secondary storage.

The Fetching Phase (Computation Phase, Writing Phase) performs the operations necessary to simulate the sending superstep (computation superstep, receiving 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 are more difficult, as we do not know the communication pattern for a particular compound superstep. We describe in Section 4.2 a randomized approach which allows us to efficiently write the messages to disk and efficiently retrieve them in the next compound superstep. For communication of predetermined size, such as occurs in a CGM or a network of processors, we can devise a deterministic approach. We show in Theorems 1 and 2 that we if we perform our simulation on a c-optimal BSP* algorithm the resulting EM algorithm is c-optimal with high probability.

We will describe the BSP* simulation in detail. However, we first show how to ensure that BSP and CGM algorithms have the blockwise communication property required for efficient, blocked I/O during their simulation as EM algorithms.

Lemma 1 *A v-processor BSP algorithm $\mathcal{A}$ with communication time $\hat{g}\alpha + \lambda\hat{L}$, computation time $\beta + \lambda\hat{L}$ and context size μ can be simulated on a p-processor BSP* in communication time $O(g\frac{v}{p}\frac{\alpha}{b}+\lambda L)$, computation time $\frac{v}{p}\beta+O(\frac{v\alpha}{p}+\lambda L)$ and context size $O(\frac{v}{p}\mu)$ for $b \leq (v/p)^{\xi}$, suitable constant $\xi > 0$, and $v \geq p^{1+\epsilon}$ for constant $\epsilon > 0$, .*

In the case $\mathcal{A}$ is c-optimal on the BSP for the conditions $\hat{g} \leq g(n)$, $\hat{L} \leq L(n)$ and $v \leq p(n)$ the simulation results in a c-optimal BSP algorithm for the conditions $g \leq b \cdot g(n)$, $L \leq L(n)$, $b \leq (v/p)^{\xi}$ and $p \leq \sqrt[1+\epsilon]{p(n)}$ for suitable constant $\xi > 0$.*

We can simulate a CGM using the same approach as for the BSP* model, which results in the following lemma.

Lemma 2 *A v-processor CGM algorithm $\mathcal{A}$ with λ communication rounds, computation time τ and context size μ can be simulated on a p-processor BSP* in communication time $O(g\frac{v}{p}\frac{\mu}{b} + \lambda L)$, computation time $\frac{v}{p}\beta + O(\frac{v\mu}{p} + \lambda L)$ and context size $O(\frac{v}{p}\mu)$ for $b \leq (v/p)^{\xi}$, suitable constant $\xi > 0$, and $v \geq p^{1+\epsilon}$ for constant $\epsilon > 0$.*

4.2 Details of the Simulation

We now describe the BSP* to EM simulation in more detail.

Definition 2 *Given a collection of records, we say that the records are in* blocked format *iff they are assembled into units of a minimum size B. B is called the* block size.

The following definition applies to a collection of records on a machine with multiple disks:

Definition 3 *We say that the records are in* standard consecutive format *on the disks iff (i) the records are in blocked format, and (ii) the number of blocks on each disk is $\leq 1+$ $min\{$the number of blocks on any disk$\}$, and (iii) the blocks on a disk are stored in consecutive tracks of the disk.*

In general, the communication time of a BSP* superstep is $O(g\frac{\gamma}{b} + L)$. We use γ to refer to the size of the communicated data, and μ to the size of the *context* of a processor (the number of words in its memory image). In our analysis, we assume $\gamma = O(\mu)$. In the following, the context and messages generated by a virtual processor are divided into *packets* (we may use the term *blocks*) and the packets are spread in standard consecutive format (evenly) over the available disks.

Algorithm CompoundSuperstep simulates a single compound superstep of a v processor BSP* on a uniprocessor EM-BSP* machine. We simulate $k \geq 1$ BSP* virtual processors at a time. 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 $k = \lfloor \frac{M}{\mu} \rfloor$, so $M \geq \mu$.

Algorithm 1 : CompoundSuperstep
Input: For each i the packets of the contexts and arriving messages of the currently simulated processors $ik, ..., (i+1)k-1$ are spread over the D disks in standard consecutive format, so that they can be accessed in parallel. See the right hand side of Figure 2 for an example.
Output: (i) The (changed) contexts of the k simulated processors spread across the disks in standard consecutive format (ii) The messages generated during the compound superstep are grouped by destination into $\frac{v}{k}$ groups, and each group is stored in standard consecutive format on the disks.

(1) for $i = 0$ to $\frac{v}{k} - 1$

 (a) Read the contexts V_{ik} to $V_{(i+1)k-1}$ from the disks into memory.

 (b) Read the packets received by the k virtual processors (in the receiving phase of the current compound superstep) from the disks.

 (c) Simulate the local computation of the k virtual processors.

 (d) Write the packets which were sent by the k virtual processors to the D disks. (Details below.)

 (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 compound superstep.

— End of Algorithm —

Algorithm CompoundSuperstep 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, $\frac{v\mu}{DB}$ blocks on each disk, where we store the contexts. We split the context of each virtual processor into blocks of size B and store the i-th block of V_j on disk $\left(i + j\frac{\mu}{B}\right) \, mod \, D$ using track $\left\lfloor \frac{i+j\frac{\mu}{B}}{D} \right\rfloor$. 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 $k\frac{\gamma}{B}/D$ 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 $\{0, 1, ..., (D-1)\}$ 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 $\frac{v}{D}$ consecutive virtual processors. In order to maintain the buckets, the simulation uses a table of D pointers on each disk. The i^{th} 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.

Definition 4 *For convenience, we shall 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 $O(\frac{k\gamma}{DB})$ parallel I/O operations, provided each disk contains the same number of blocks.

Since we write one block in each round to each disk, and there are a total of $v\frac{\gamma}{B}/D$ rounds, the additional space required to store the table can be ignored because $v\frac{\gamma}{B}/D \geq \frac{\gamma}{B}\log\frac{M}{B} = O(D\log\frac{M}{B})$. In Lemma 4 we show that with high probability the blocks of each bucket are uniformly distributed over the disks.

Algorithm SimulateRouting provides the details of Step 2 in Algorithm CompoundSuperstep)

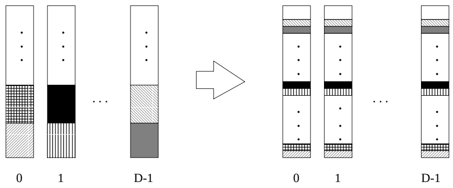

Figure 2: Reorganization of the blocks

Algorithm 2 : SimulateRouting
Input: The D buckets (of messages) stored on the disks in standard linked format.
Output: The $\frac{v}{k}$ 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 blocks b_d belonging to buckets d from disks $((d+j) \bmod D)$. Write blocks b_d to disks 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 $\frac{v\gamma}{DB}$

for $d = 0$ to $D-1$ in parallel do

read the j^{th} block from disks d

write them to disks $(d+j) \bmod D$ on track $i\lceil \frac{v\gamma}{D^2B} \rceil + \lfloor j/D \rfloor$

— End of Algorithm —

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 3 *Steps 1(a), 1(b) and 1(e) of Algorithm CompoundSuperstep have computation time $O(\gamma v)$, memory $\Theta(k\gamma)$, I/O-time $O(G\frac{v\gamma}{DB})$, and disk space $O(\frac{v\gamma}{DB})$ blocks per disk.*

Lemma 4 *The blocks representing simulated message traffic are divided into buckets by our simulation. 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 bucket j' we have the following:*

$$\Pr\left[X_{j',1} \geq l\frac{R}{D}\right] \leq \exp\left(-\Omega\left(\frac{l \log l \cdot R}{D}\right)\right)$$

Proof. The proof is similar to one described in [27]. Details can be found in the appendix. □

Lemma 5 *For $v \geq kD\log\frac{M}{B}$, the computation time of Algorithm SimulateRouting is $O(l\gamma v)$ and its I/O time is $O(Gl\frac{v\gamma}{DB})$ with probability $1 - \exp(-\Omega(l \cdot \log l \cdot \log(M/B)))$ for constant $l \geq 1$.*

Proof. Each bucket contains $R = \frac{v\gamma}{DB}$ blocks since each of the D buckets contains the messages destined for $\frac{v}{D}$ virtual processors. Hence, for $M \geq DB$, and $\frac{v}{k} \geq D\log\frac{M}{B}$,

$$R = \frac{v\gamma}{DB} \geq \frac{k\gamma D\log(M/B)}{DB} \quad (1)$$

$$R \geq \frac{k\gamma\log(M/B)}{B} \quad (2)$$

By Lemma 4, each disk contains less than $l\frac{R}{D}$ records of a given bucket with probability at most $\exp(-\Omega(l \cdot \log l\frac{R}{D}))$. There are D drives and D buckets, so the probability that any disk contains more than $l\frac{R}{D}$ blocks of any bucket is at most $D^2 \cdot \exp(-\Omega(l \log l\frac{R}{D}))$. Thus, with (2) and $k\gamma \geq DB$, we have

$$\begin{aligned}
& D^2\exp(-\Omega(l\log l\frac{R}{D})) \\
\leq\; & D^2\exp(-\Omega(l\log l\frac{k\gamma\log(M/B)}{DB})) \\
\leq\; & \exp(-\Omega(l\log l\frac{DB\log(M/B)}{DB} + \log D)) \\
\leq\; & \exp\left(-\Omega(l\log l \cdot \log\frac{M}{B})\right)
\end{aligned}$$

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

In Step 2, $\frac{v\gamma}{DB}$ 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 $O(G\frac{lv\gamma}{DB})$ and the total computation time is $O(lv\gamma)$ with high probability. □

Result 1 *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 $O(Gl\frac{v\gamma}{DB})$ with probability $1 - \exp(-\Omega(l\log l \cdot \log(M/B)))$ for constant $l \geq 1$, $v \geq kD\log\frac{M}{B}$, $M = \Theta(k\mu)$, $b \geq B$, and k an arbitrary integer.*

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 4, 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 $O(D\frac{v\gamma}{DB} + k\gamma)$ and I/O time $O(\frac{k\gamma}{DB})$, giving computation time $O(v\gamma)$ and I/O time $O(G\frac{v\gamma}{DB})$ for the whole simulation.

By Lemma 3 and Lemma 5 the computation time and I/O-time respectively for Steps 2, 1(a), 1(b) and 1(d) are $O(v\gamma)$ and $O(G \cdot l\frac{v\gamma}{DB})$. Thus, overall, computation time is $v\tau + O(l\gamma v)$ and I/O-time is $O(G \cdot l\frac{v\gamma}{DB})$ with probability $1 - \exp(-\Omega(l \cdot \log l \cdot \log \frac{M}{B}))$. □

Lemma 6 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 6 *Given z independent random variables $X_1, X_2, \ldots, X_z$ with $\Pr[X_i \leq lT] \geq 1 - \exp(-l \log l \cdot m)$ and $\Pr[X_i > lT] \leq \exp(-l \log l \cdot m)$, where $m \geq \ln x$ and $T_{wc} \leq xT$, we have $\Pr[\sum_{i=1}^{z} X_i \leq e^2(l+1)zT] \geq 1 - \exp(-\Omega(l \log l \cdot m))$ for $l \geq 2$.*

Proof. Proof can be found in the appendix. □

Theorem 1 *A c-optimal 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 $O(G\lambda\frac{lv\mu}{DB})$ with probability $1 - \exp(-\Omega(l \cdot \log l \cdot \log \frac{M}{B}))$ for suitable $l \geq 1$, $\beta = \omega(\lambda\mu)$, $M = \Theta(k\mu)$, $v \geq kD \log \frac{M}{B}$, $b \geq B$, and arbitrary integer k.*

In particular, the simulation results in a c-optimal EM-BSP algorithm for the additional condition $G = BD \cdot o(\frac{\beta}{\mu\lambda})$.*

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 Result 1, 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 $O(G\frac{lv\mu}{DB})$ with probability at least $1 - \exp(-\Omega(l \log l \cdot \log \frac{k\mu}{B}))$.

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 theorem follows from Lemma 6. □

4.3 The General 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 $v \geq kpD \log \frac{M}{B}$ 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, $\frac{v}{p}$ virtual processors $i\frac{v}{p}, \ldots, (i+1)\frac{v}{p} - 1$ are assigned to each simulating processor i. As before, we perform a stepwise simulation which works in a round-robin fashion. During the j^{th} of $\frac{v}{pk}$ rounds processor i simulates the steps of the k virtual processors $i\frac{v}{p} + jk, \ldots, i\frac{v}{p} + (j+1)k - 1$. 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. Instead, therefore, we distribute the messages randomly among the processors after each round. After the last round each processor reorganizes the messages it has received so that for every j during the Fetching Phase of the j^{th} round of the next simulation it can read the messages destined to the virtual processors $jkp, \ldots, (j+1)kp - 1$ in parallel from disk and send them to the correct simulating processor.

We maintain $\frac{v}{pk}$ batches to store the generated messages. The j^{th} batch contains the messages destined to virtual processors simulated in the j^{th} round. Each processor P_i writes its share of the $\frac{v}{pk}$ batches to its local disks. The main difficulty is the efficient maintenance of the batches so that the packets which are needed during the j^{th} simulation round can be read in parallel from the disks.

Algorithm 3 : Simulation of a v-processor BSP* on a p-processor EM-BSP*
Input: Each processor holds $\frac{k\gamma}{B}$ blocks of each batch. For each j the blocks and the contexts of the virtual processors $jkp, \ldots, (j+1)kp - 1$ are distributed among the local disks such that each disk contains $\frac{k\gamma}{DB}$ blocks of each.
Output: The changed contexts and generated messages distributed as required for the next round.

(1) In the j^{th} round, $1 \leq j \leq \frac{v}{pk}$, for each processor i, $0 \leq i \leq p-1$, do in parallel:

  (a) (Fetching Phase): P_i reads in parallel the blocks for its share of batch j from disk and sends them to the appropriate simulating processors.

  (b) (Computing Phase): P_i simulates the computation supersteps of its current virtual processors and collects all generated messages in its local memory.

  (c) (Writing Phase): P_i splits all generated messages into packets of size b (the message size) and sends each one to a randomly chosen processor.

(2) P_i reorganizes the received batches using *Simulate-Routing* so that each one is evenly distributed over the disks.

— End of Algorithm —

Many of the details of Algorithm 3 are similar to ones described for Algorithm 1. In each round a processor receives $\frac{k\gamma}{b}$ blocks with high probability. In Step 1(c), each processor cuts the messages 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 distributed on the disks as in Step 1(d) of the single-processor simulation. Each bucket contains the blocks destined to $\frac{v}{pk}/D$ batches. The i^{th} bucket on each disk contains packets belonging to the same batch. As before, a table of D pointers is maintained on each disk. As before, we can show that each disk will receive the same number of blocks for every bucket with high probability.

Result 2 *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 $\tau\frac{v}{p} + O(l\frac{v}{p}(\gamma + \mu) + L\frac{v}{pk})$, communication time $O(gl\frac{v}{p}(\frac{\gamma}{b} + \frac{\log(M/B)}{k}) + L\frac{v}{pk})$ and I/O-time $O(G \cdot l\frac{v}{p}\frac{\mu}{BD})$ with probability $1 - \exp(-\Omega(l \log l \cdot \log(M/B)))$ for suitable $l \geq 1$, $M = \Theta(k\mu)$, $b\log(M/B) = O(M)$, $v = \Omega(kpD \cdot \log\frac{M}{B})$, $M/B \geq p^{\epsilon}$, and arbitrary constants k, $\epsilon > 0$.*

Theorem 2 states that a c-optimal BSP* algorithm is transformed into a c-optimal EM algorithm by our simulation for the general case of $p \geq 1$ simulating processors.

Theorem 2 *A c-optimal 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 $(1 + o(1))\frac{v}{p}\beta + O(L\lambda\frac{v}{pk})$, communication time $O(gl(\frac{v}{p}\frac{\alpha}{b} + \lambda\frac{\log(M/B)}{k}) + L\lambda\frac{v}{pk})$, and I/O time $O(Gl\frac{v}{p}(\frac{\mu\lambda}{BD}))$ with probability $1 - \exp(-\Omega(l\log l \cdot \log\frac{M}{B}))$ for suitable $l \geq 1$, $\beta = \omega(\lambda\mu)$, $M = \Theta(k\mu)$, $v = \Omega(pkD \cdot \log\frac{M}{B})$, $M/B \geq p^{\epsilon}$, $b\log(M/B) = O(M)$, and arbitrary constants k, $\epsilon > 0$.*

In the case $\mathcal{A}$ is c-optimal on the BSP for the additional conditions $g \leq g(n)$, $b \leq b(n)$, $L \leq L(n)$ and $v \leq p(n)$ the simulation results in a c-optimal EM-BSP* algorithm for the conditions $\lambda\frac{\log(M/B)}{k} = O(\frac{\alpha}{b})$, $g \leq g(n)$, $b \leq b(n)$, $G = BD \cdot o(\frac{\beta}{\mu\lambda})$ and $L \leq L(n) \cdot \frac{pk}{v}$.*

The slackness $\frac{v}{p}$ 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 $M/B \geq p^{\epsilon}$ is usually fulfilled for actual machines. The simulation increases the number of supersteps by a factor of $\frac{v}{p\log p}$. However we inherit the good communication time from the BSP* algorithm which results in a c-optimal multiple processor EM-BSP* algorithm.

5 Conclusion

In this paper we described simulation techniques which produce efficient EM algorithms from efficient algorithms developed under BSP-like parallel computing models. Our techniques 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. In addition to the main simulation result, we obtained a more comprehensive cost model for EM algorithms which considers the total costs incurred by the algorithm including computation, I/O and communication costs.

We note that our techniques apply to BSP-like algorithms for which $\tau = \omega(\lambda\mu)$, where τ is the computation time, λ is the number of supersteps, and μ is the size of the local context of a processor. This is a large class of algorithms. Examples include computational geometry problems such as area of the union of rectangles, $3D$-maxima, $2D$-nearest neighbors of a point set, lower envelope problems, $2D$-weighted dominance counting, and $3D$-convex hull [13, 14, 15].

Algorithms which do not fall into this category are typically algorithms that maintain very large data structures. An example of such an application is multisearch [7]. In such cases, we suggest that classical EM techniques like distribution sweeping[17], batch filtering[17], and buffer tree[5] to name a few must be employed by each virtual processor in order to achieve efficiency by any cost measure that includes I/O costs. In these cases, the techniques in this paper provide a basis for pursuing efficient parallel implementations of such algorithms. We are currently exploring these ideas in the context of multisearch.

Our results show that for certain efficient BSP-like algorithms, a corresponding efficient EM algorithm exists, and that such algorithms can, in fact, be logically used in both environments. This creates a scenario where an application equipped with such an algorithm could adapt dynamically to available resources such as processors, memory, and disks in an efficient way, as dictated by the problem size. We are in the process of implementing this idea on ASP, a Linux-based multiprocessor with multiple disks recently built at Carleton University.

References

[1] A. Aggarwal and J.S. Vitter. The Input/Output Complexity of Sorting and Related Problems. CACM V. 31 No. 9:1116-1127, 1988.

[2] B. Alpern, et al. 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. Proceedings of WADS'95, LNCS955, pp. 334-345, 1995.

[4] L. Arge, D.E. Vengroff and J.S. Vitter. External-Memory Algorithms for Processing Line Segments in Geographic Information Systems. 3rd Annual European Symposium on Algorithms, LNCS 979, pp. 295-310, 1995.

[5] L. Arge. Efficient External-Memory Data Structures and Applications, Ph.D Thesis, BRICS, University of Aarhus, 1996.

[6] A. Bäumker and W. Dittrich. Parallel Algorithms for Image Processing: Practical Algorithms with Experiments. Proc. 10th International Parallel Processing Symposium, pp. 429–433, 1996.

[7] A. Bäumker, W. Dittrich, and F. Meyer auf der Heide. Truly Efficient Parallel Algorithms: c-Optimal Multisearch for an Extension of the BSP model. Proc. European Symposium on Algorithms, pp. 17–30, 1995.

[8] A. Bäumker, W. Dittrich, and A. Pietracaprina. The Deterministic Complexity of Parallel Multisearch. Proc. of the 5th Scandinavian Workshop on Algorithm Theory, pp. 404–415, 1996.

[9] P. Callahan, M.T. Goodrich and K. Ramaiyer. Topology B-Trees and Their Applications. Proc. WADS'94, LNCS 955, pp. 381-392, 1995.

[10] Y.-J. Chiang. Dynamic and I/O Efficient Algorithms for Computational Geometry and Graph Problems: Theoretical and Experimental Results. PhD Thesis,Brown University 1995.

[11] Y.-J. Chiang, et al. External Memory Graph Algorithms. Proc. SODA, pp.139-149, 1995.

[12] T.H. Cormen. Virtual Memory for Data Parallel Computing. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[13] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers. Proc. ACM 9th Annual Computational Geometry, pp. 298–307, 1993.

[14] F. Dehne, A. Fabri, and C. Kenyon. Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time. Proc. 6th IEEE Symposium on Parallel and Distributed Processing, pp. 586–593, 1994.

[15] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Kokhar. A randomized parallel 3D convex hull algorithm for coarse grained multicomputers. Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA'95), pp. 27–33, 1995.

[16] A.V. Gerbessiotis and L. Valiant. Direct Bulk-Synchronous Parallel Algorithms. Journal of Parallel and Distributed Computing, 1994.

[17] M.T. Goodrich, et al. External Memory Computational Geometry. Proc. FOCS, 714-723, 1993.

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

[19] M.H. Nodine and J.S. Vitter. Paradigms for Optimal Sorting with Multiple Disks. Proc. Hawaii International Conference on System Sciences 1:50-59, 1993.

[20] M.H. Nodine and J.S. Vitter. Greed Sort: Optimal Deterministic Sorting on Parallel Disks. JACM 42 No. 4:919-933, 1995.

[21] J.F. Sibeyn and M. Kaufmann. BSP-like External-Memory Computation. To appear in Proc. CIAC '97.

[22] S. Subramanian and S. Ramaswamy. The P-range Tree: A New Data Structure for Range Searching in Secondary memory. Proc. SODA 378-387, 1995.

[23] L.G. Valiant. A Bridging Model for Parallel Computation. CACM 33 No. 8:103-111, 1990.

[24] D.E. Vengroff. TPIE User Manual and Reference. Tech Report 1995.

[25] D.E. Vengroff and J.S. Vitter. I/O-Efficient Scientific Computation Using TPIE. Proc. IEEE Symp. on Parallel and Distributed Computing, 1995.

[26] J.S. Vitter and M.H. Nodine. Large Scale Sorting in Uniform Memory Hierarchies. J.Par.Distr.Comp. 17:107-114, 1993.

[27] J.S. Vitter and E.A.M. Shriver. Algorithms for Parallel Memory, I: Two Level Memories. Algorithmica 12:110-147, 1994.

6 Appendix

We use the following tail estimates:

Lemma 7 *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,

$$\Pr[X \geq u] = \Pr[\exp(rX) \geq \exp(ru)].$$

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

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

□

Lemma 4 *The blocks representing simulated message traffic are divided into buckets by our simulation. 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 if it contains a record of bucket j). Then, for any fixed bucket j' we have the following:*

$$\Pr\left[X_{j',1} \geq l\frac{R}{D}\right] \leq \exp\left(-\Omega\left(\frac{l \log l \cdot R}{D}\right)\right)$$

Proof. The proof is similar to one described in [27]. Let g_t denote the number of disks (i.e. the number of records) written to from bucket j' during write cycle t, for $1 \leq t \leq C$, where C is the total number of write cycles used. We have

$$\sum_{1 \leq i \leq C} g_t \leq R \tag{3}$$

We define G_t to be the number of tracks belonging to bucket j' that are assigned to track 1 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:

$$\begin{aligned} \mathcal{G}_{G_t}(z) &= \Pr[G_t = 0]z^0 + \Pr[G_t = 1]z^1 && (4)\\ &= 1 - \frac{g_t}{D} + \frac{g_t}{D}z && (5)\\ &= 1 + \frac{g_t}{D}(z-1) && (6)\end{aligned}$$

Let $\mathcal{G}_{X_{j',1}}(z)$ be the probability generating function for $X_{j',1}$. We can bound $X_{j',1}$ by the sum of independent random variables: $X_{j',1} \leq G_1 + G_2 + \cdots + G_C$. For purpose of bounding, let us consider that $X_{j',1} = G_1 + G_2 + \cdots + G_C$. Since the sum of independent random variables corresponds to the product of the corresponding probability generating functions and using (6), we have

$$\begin{aligned}\mathcal{G}_{X_{j',1}}(z) &= \mathcal{G}_{G_1+G_2+\cdots+G_C}(z) \\ &= \mathcal{G}_{G_1}(z) \times \mathcal{G}_{G_2}(z) \times \cdots \times \mathcal{G}_{G_C}(z) \\ &= \prod_{1\le t\le C} (1+\frac{g_t}{D}(z-1)) \qquad (7)\end{aligned}$$

By the tail estimate Lemma 7 we have

$$\Pr\left[X_{j',1} \ge l\frac{R}{D}\right] \le \frac{E[\exp(rX_{j',1})]}{\exp(rlR/D)} \qquad (8)$$

for each $r \ge 0$. We can express the numerator in (8), using (7) and the definitions of expected value and probability generating function, as

$$\begin{aligned}E[\exp(rX_{j',1})] &= \sum_{t\ge 0} \Pr[e^{rX_{j',1}} = e^{rt}]e^{rt} \\ &= \sum_{t\ge 0} \Pr[X_{j',1} = t]e^{rt} \\ &= \mathcal{G}_{X_{j',1}}(e^r) \\ &= \prod_{1\le t\le C} (1+\frac{g_t}{D}(e^r-1)) \qquad (9)\end{aligned}$$

By (3) and convexity arguments, we can maximize (9) by setting $g_t = R/C$ for each t. Thus

$$\begin{aligned}E[\exp(rX_{j',1})] &\le \prod_{1\le t\le C}\left(1+\frac{R\cdot(e^r-1)}{DC}\right) \\ &= \left(1+\frac{R\cdot(e^r-1)}{DC}\right)^C \qquad (10)\end{aligned}$$

Substituting this bound into (8), we get

$$\Pr\left[X_{j',1} \ge l\frac{R}{D}\right] \le \frac{(1+R(e^r-1)/DC)^C}{\exp(rlR/D)} \qquad (11)$$

From the bound $(1+a)^b \le e^{ab}$, for $a > -1$, we can approximate the numerator in (11) and get for $r = \ln l$

$$\begin{aligned}\Pr\left[X_{j',1} \ge l\tfrac{R}{D}\right] &\le \tfrac{\exp(R\cdot(e^r-1)/D)}{\exp(rlR/D)} \\ = \exp\left(\tfrac{R\cdot(e^r-1)-rlR}{D}\right) &\le O\left(\exp\left(-\tfrac{l\log lR}{D}\right)\right) \qquad (12)\end{aligned}$$

□

Fact 1 (Hoeffding [18]) *Let $X_1,\cdots,X_l$ be independent random variables with $X_i \in [0,\ldots,k]$ and $m = E[\sum_{i=1}^{l} X_i]$. Then for $u \ge e^2$:*

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

Lemma 6 *Given z independent random variables $X_1, X_2, \ldots, X_z$ with $\Pr[X_i \le lT] \ge 1-\exp(-l\log l\cdot m)$ and $\Pr[X_i > lT] \le \exp(-l\log l\cdot m)$, where $m \ge \ln x$ and $T_{wc} \le xT$. Then, we have $\Pr[\sum_{i=1}^{z} X_i \le e^2(l+1)zT] \ge 1-\exp(-\Omega(l\log l\cdot m))$ for $l \ge 2$.*

Proof. We identify two cases: (1) $z \ge xm^c$ and (2) $z \le xm^c$ for $c = 1+\frac{\log\log l}{\log m}$.

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

$$\begin{aligned}E[\sum_{i=1}^{z} X_i] &\le \sum_{i=1}^{z}(\Pr[X_i \le lT]\cdot lT + \Pr[X_i > lT]\cdot T_{wc}) \\ &\le zlT(1-e^{-l\log l\cdot m}) + zxTe^{-l\log l\cdot m} \\ &\le zlT + zTe^{-l\log l\cdot m+\ln x} \\ &\le (l+1)zT \qquad (13)\end{aligned}$$

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

$$\begin{aligned}E[\sum_{i=1}^{z} X_i] &\ge \sum_{i=1}^{z}(\Pr[X_i \le lT]\cdot lT + \Pr[X_i > lT]\cdot T) \\ &\ge (l-1)zT \qquad (14)\end{aligned}$$

We can conclude from Fact 1 with $k = T_{wc} \le xT$, $m = E[\sum_{i=1}^{z} X_i] \ge (l-1)zT$, and $z \ge xm^c$:

$$\begin{aligned}\Pr[\sum_{i=1}^{z} X_i \ge e^2 m] &\le \exp(-e^2\frac{(l-1)zT}{xT}) \\ &\le \exp(-\frac{(l-1)z}{x}) \le \exp(-(l-1)m^c) \\ &\le \exp(-(l-1)\log l\cdot m)\end{aligned}$$

Now, we consider the Case (2), remember $z \le xm^c$ with $c = 1+\frac{\log\log l}{\log m}$. We repeat the experiment only z times, thus we have with $m \ge \ln x$:

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

□