Reducing I/O Complexity by Simulating Coarse Grained Parallel Algorithms [(1)]

— Extended Abstract —

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

July 30, 1998

Abstract

Block-wise access to data is a central theme in the design of efficient *external memory* (EM) algorithms. A second important issue, when more than one disk is present, is fully parallel disk I/O. In this paper, we present a *deterministic* simulation technique which transforms *Coarse Grained Multicomputer* (CGM) algorithms into parallel external memory algorithms. It optimizes block-wise data access and parallel disk I/O and, at the same time, utilizes *multiple processors* connected via a communication network or shared memory. We obtain new improved parallel external memory algorithms for a large number of problems including sorting, permutation, matrix transpose, several geometric and GIS problems including 3D convex hulls (2D Voronoi diagrams), and various graph problems.

We show that parallel algorithms known for the CGM model can be used to obtain external memory algorithms that seem to have better I/O complexity than the well known *lower bounds* for various problems, including sorting. In certain cases, a similar phenomenon can also occur for BSP algorithms. We explain this apparent contradiction by examining the parameter values which permit the ubiquitous $\log_{M/B}(N/B)$ term in the I/O complexity to become a constant. We show that such a parameter constellation arises naturally in coarse grained parallel algorithms and the external memory domain.

The practicality of our methods is demonstrated in a prototype implementation on a network of Pentium processors connected via a 2 GB Ethernet switch and with multiple disks per processor.

1 Introduction

1.1 Motivation

Some of the key applications of parallel computing include astrophysical models, genetic sequencing, geographic information systems, ecological models, weather prediction, telecommunications applications, commercial digital video and audio, digital libraries, government information systems, and biological models for medical applications. Researchers in all of these applications currently face data sets of terabyte size (perhaps increasing to petabytes in the foreseeable future). If parallel computing is to succeed in these areas, it needs to solve the problem of how to obtain efficient parallel disk I/O. Research in *external memory* (EM) algorithms has recently received considerable attention. Primary references are the report of the ACM workshop on strategic directions in computing research, ed. by Gibson, Vitter and Wilkes [30] and Vitter's survey [47]. The main questions are, how to optimize block-wise and simultaneous access to multiple disks, and how to combine this with a parallel processing environment where multiple processors (each with multiple disks) are connected via a communication network or shared memory. Closely related problems are how to include the effects of network caching and multi level memory hierarchies in general.

[1]Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

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

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

[4]Research partially supported by Almerco, Inc.

1.2 Review: Parallel Disk Model and Previous Results

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

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

Floyd [29] studied sorting (and matrix transpose) in a single-disk single-processor model, where $B = M/2 = \Theta(N^c)$, for some constant $c > 0$, and provided upper and lower I/O bounds. Agarwal and Vitter [3] generalized Floyd's model and provided matching upper and lower I/O bounds for several problems, and these bounds apply to the PDM model. The lower bound for sorting states that the worst-case number of I/O's required for sorting is $\Theta(\frac{N}{BD}\log_{\frac{M}{B}}\frac{N}{B})$[1] [3, 47]. Several EM algorithms exist for sorting, including [2, 3, 4, 36, 37, 49, 50, 38]. Surprisingly, it turns out that performing a permutation requires $\Theta(\min\{\frac{N}{D}, \frac{N}{BD}\log_{\frac{M}{B}}\frac{N}{B}\})$ I/Os [3, 47], while the same can be performed in linear time in the RAM model. Similarly, the worst-case number of I/Os required to transpose a $p \times q$ matrix from row-major order to column-major order is $\Theta(\frac{N}{BD}\log_{\frac{M}{B}}\min(M, p, q, \frac{N}{B}))$ [3, 47]. Cormen et al. [21] have studied the optimal number of I/Os required to perform several special classes of permutations. This includes permutations arising in matrix transpose, FFTs, hypercubes, matrix reblocking. Arge et al. [7] show that any problem which requires $\Omega(N \log N)$ comparisions in the comparision model, requires $\Omega(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B})$ I/Os in the PDM model.

EM algorithms have been proposed for a number of problems arising in computational geometry [8, 6, 23, 32], geographical information systems [8, 43], and graphs [5, 15, 34, 41]. Over the last few years, comprehensive computing and cost models, that incorporate multi-disks and multi-processors have been proposed [17, 25, 28, 35]. Several suggestions have been made regarding the simulation of parallel algorithms as EM algorithms. This includes the results of Chiang et al. [15] on simulating PRAM algorithms and the results of Dehne et al. [25] and Dittrich et al [28] on simulating BSP, CGM and BSP* algorithms (see also [9, 39]).

[1] $\log_{\frac{M}{B}}\frac{N}{B}$ is defined to mean $\max\{1, \log_{\frac{M}{B}}\frac{N}{B}\}$.

1.3 New Results

Cormen and Goodrich [17] posed the *Challenge* of combining BSP like parallel algorithms with the the requirements for parallel disk I/O. Solutions based on probablistic methods were presented in [25] and [28]. In this paper, we present *deterministic* solutions which are based on a deterministic simulation of parallel algorithms for the *Coarse Grained Multicomputer* (CGM) model and answer the challenge. The analysis of the I/O complexity of our algorithms is done as in the PDM model. (In addition, we also analyze the running time and communication time.) The obtained results not only improve on the previous bounds but also have the interesting property of being *better* than the I/O complexity lower bounds listed in Section 1.2. We explain the latter by pointing out that the I/O complexity lower bounds were proven for arbitrary ranges over the various parameters involved and do not hold if one restricts them to a particular parameter range. We show however, that our parameter range is both interesting and useful in the EM domain.

We first review some definitions for the BSP and CGM model; consult [25, 42, 27] for more details. A *BSP algorithm* $\mathcal{A}$ on a fixed problem instance $\mathcal{P}$ and computer configuration $\mathcal{C}$ can be characterized by the parameters (N, v, λ, g, L), where N is the problem size (in problem items), v is the number of processors, g is the time required for a communication operation, L is the minimum time required for the processors to synchronize, and λ is the number of BSP supersteps required by $\mathcal{A}$ on $\mathcal{P}$ and $\mathcal{C}$. (The times are in number of processor cycles.) A *CGM algorithm* is a special case of a BSP algorithm where the communication part of each superstep consists of exactly one h-relation with $h = \Theta(\frac{N}{v})$. Such a superstep is called a *round*. An algorithm for a CGM with multiple disks attached to each processor (see Figure 9) is referred to as an *EM-CGM algorithm.*

The following outlines the results obtained.

1. We show that any v processor CGM algorithm $\mathcal{A}$ with λ supersteps/rounds, local memory size μ, computation time $\beta + \lambda L$, communication time $g\alpha + \lambda L$ and message size $\Theta(\frac{N}{v^2})$ can be simulated, deterministically, as a p-processor EM-CGM algorithm $\mathcal{A}'$ with computation time $\frac{v}{p}(\beta + O(\lambda\mu)) + \frac{v}{p}\lambda L$, communication time $\frac{v}{p}g\alpha + \frac{v}{p}\lambda L$, and I/O time $\frac{v}{p}G \cdot O(\lambda\frac{\mu}{DB}) + \frac{v}{p}\lambda L$ for $M = \Theta(\mu)$, $N = \Omega(vDB)$, $B = O(\frac{N}{v^2})$.

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

   While our parameter space is constrained to a coarse grained scenario which is typical of the CGM constraints for parallel computation, we show that this parameter space is both interesting and appropriate for EM computation. We show that this constraint permits several fundamental problems to be solved in lower I/O complexity than is permitted by the general lower bounds reported by [3, 47, 7]. This answers questions of Cormen [16] and Vitter [48] on the apparent contradictions between the results of [25] and the previously stated lower bounds.

2. We obtain new, simple, parallel EM algorithms for *sorting*, *permutation*, and *matrix transpose* with I/O complexity $O(\frac{N}{pDB})$.

3. We obtain parallel EM algorithms with I/O complexity $O(\frac{N}{pDB})$ for the following *computational geometry/GIS problems* : (a) 3-dimensional convex hull and planar Voronoi diagram (these results are probabilistic since the underlying CGM algorithms are probabilistic), (b) lower envelope of line segments (here, N denotes the size of the input plus output), (c) area of union of rectangles, (d) $3D$-maxima, (e) nearest neighbour problem for planar point set, (f) weighted dominance counting for planar point set, (g) unidirectional and multi-directional separability.

4. We obtain parallel EM algorithms with I/O complexity $O(\frac{N \log N}{pDB})$ for the following *computational geometry/GIS and graph problems*: (a) trapezoidal decomposition (b) triangulation (c) segment tree construction, (d) batched planar point location,

5. We obtain parallel EM algorithms with I/O complexity $O(\frac{N \log v}{pDB})$ for the following *computational geometry/GIS and graph problems*: (a) list ranking, (b) Euler tour of a tree, (c) connected components, (d) spanning forest, (e) lowest common ancestor in a tree, (f) tree contraction, (g) expression tree evaluation, (h) open ear decomposition, (i) biconnected components.

6. In contrast to previous work, all of our methods are also scalable with respect to the number of processors.

7. We have implemented a prototype application of our method on a network of Pentium processors connected via a 2 GB Ethernet switch and with multiple disks per processor. Experimental evidence shows that our approach is practical.

Items (2), (3), (4) and (5) above are described in more detail in Figure 5. These results are subject to the conditions $N = \Omega(vDB)$, $N \geq v^2B + v^2(v-1)/2$, and $N > v^\kappa$, where $\kappa > 1$ is a constant that depends on the problem. The latter constraint arises in the CGM algorithm which we simulate. For the problems examined in this paper, $\kappa \leq 3$.

Our results show that the EM-CGM is a good generic programming model that facilitates the design of I/O-efficient algorithms in the presence of multi-processors and multi-disks. It has relatively few parameters, generalizes the PDM model, and answers the challenge of [17].

By generating programs for a single processor computer from coarse grained parallel algorithms, our approach can also be used to control cache memory faults. This supports a suggestion of Vishkin [45, 46].

1.4 Practicality of Our Parameter Bounds

The parameter space for EM problems which we are proposing in this paper is both practical and interesting. The logarithmic term in the I/O complexity of sorting is bounded by a constant c if $(\frac{M}{B})^c \geq \frac{N}{B}$, where $M = \frac{N}{v}$. Since this constraint involves the parameters v, B, N, c, we have a four-dimensional constraint space. For practical purposes, the parameter B can be fixed at about 10^3 for disk I/O (see Figure 8) [40]. This reduces the parameter space to three dimensions. We plot the surface $N^{c-1} = v^c B^{c-1}$ in Figure 6. Any point on or above the surface represents a valid set of parameters for the elimination of the logarithmic factor. It can be seen from Figure 6 that the logarithmic factor can be replaced by a constant $c = 2$ for as many as $v = 10000$ processors, provided that the problem size is approximately 100 giga-items or more. For a larger constant, say $c = 3$, the problem size need only be 1 giga-item for $v = 10000$. It can also be seen from Figure 6 that for a smaller numbers of processors the necessary problem size for $c = 2$ is much smaller. This can be seen more clearly in Figure 7 which represents the same data as Figure 6, but for fixed $c = 2$. For 100 processors or less, for instance, we see from Figure 7 that any problem size greater than about 10 mega-items is sufficient.

2 Deterministic Simulation of CGM Algorithms as EM-CGM Algorithms

In this section we describe a deterministic simulation for CGM and other BSP-like algorithms whose communication can be characterized by h-relations. For ease of exposition, we focus on the CGM case. In Section 5 we consider other BSP-like algorithms. Due to lack of space, proofs are omitted in most cases. Some are provided in the Appendix.

Each communication superstep of the underlying CGM algorithm 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 CGM algorithm proceeds as a series of compound supersteps, and can therefore be simulated by repeated application of the simulation steps for a single compound superstep.

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

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

2.1 Single Processor Target Machine

In this section we describe a deterministic simulation technique that permits a CGM algorithm to be simulated as an external memory algorithm on a single processor target machine. We first consider the simulation of a single compound superstep, and in particular, how the contexts and messages of the virtual processors can be stored on disk and retrieved efficiently in the next superstep. The management of the contexts is straightforward. Since we know the size of the contexts of the processors, we can distribute the contexts deterministically.

The main issue is how to organize the generated messages on the D disks so that they can be accessed using blocked and fully parallel I/O operations. This task is simpler if the messages have a fixed length. Although a CGM algorithm has the property that $\Theta(\frac{N}{v})$ data is deemed to be exchanged by each processor in every superstep, there is no guarantee on the size of individual messages. Algorithm BalancedRouting gives us a technique for achieving fixed size messages.

Algorithm 1 BalancedRouting (from [10])
Input: Each of the v processors has $\frac{\bar{n}}{v}$ elements, which are divided into v messages, each of arbitrary length $\leq \frac{\bar{n}}{v}$. Let msg_{ij} denote the message to be sent from processor i to processor j, and let $|msg_{ij}|$ be the length of such a message.
Output: The v messages in each processor are delivered to their final destinations in two balanced rounds of communication, and each processor then contains at most $\bar{h}$ data.

```
Superstep A: For $i = 0$ to $(v-1)$ in parallel
    Processor $i$ allocates $v$ local bins, one for each processor
    For $j = 0$ to $(v-1)$
      (1) For $\ell = 0$ to $|msg_{ij}|$
              Processor $i$ allocates the $\ell^{th}$ word of $msg_{ij}$ to local bin $(i+j+\ell)$ mod $v$
      (2) Processor $i$ sends bin $j$ to processor $j$

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

Observation 1 *If bin_{min} is the smallest bin created at a processor in step (1) of Superstep A, then the other $(v-1)$ bins can contain at most $1+2+...+(v-1) = \frac{v(v-1)}{2}$ more elements than does bin_{min} (see Figure 1).*

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

Proof Sketch. The maximum message sizes were shown in [10]. The proof of the minimum message sizes relies on Observation 1; see Figure 1. □

The notion of an h-relation is often used in the analysis of parallel algorithms based on BSP-like models (e.g. BSP, BSP*, CGM). An h-relation is

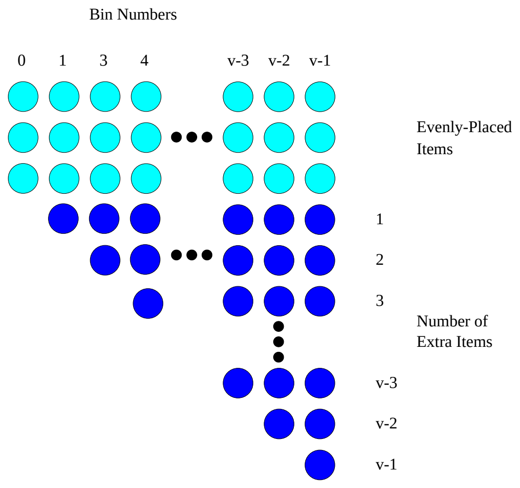

Figure 1: Illustration of maximum imbalance in the bin sizes of a processor during superstep A. The light circles represent evenly placed elements and the dark circles are unevenly placed, or "extra" ones. The maximum bin size (bin $v-1$ in the diagram) is at most $\frac{v-1}{2}$ more than the average,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

a communication superstep in which each of the v processors sends and receives at most h data items. It is typically used in bounding the communication complexity in an asymptotic analysis. Based on this usage of an h-relation, we have:

Corollary 1 *An arbitrary $h-$relation can be replaced by two "balanced" $h-$relations whose message size is bounded by $\frac{h}{v} - \frac{v-1}{2}$ and $\frac{h}{v} + \frac{v-1}{2}$.*

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

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

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

Assurances regarding the minimum message size are particularly relevant to the BSP* model. In Section 5 we outline the use of Theorem 1 for creating BSP* algorithms from BSP algorithms. First, however, we look at the deterministic simulation of CGM algorithms as EM-CGM algorithms. Not every CGM algorithm will require balancing, but Lemma 2 ensures that we can obtain balanced message sizes when necessary by increasing the number of supersteps by a factor of 2.

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

We will now turn to the actual simulation results, which rely on a message size of $k\frac{N}{v^2}$, for a known constant $k \geq 1$. As we have seen, this is guaranteed by Lemma 2. Not every CGM algorithm will require Lemma 2; see Matrix Transpose in Section 3 for an example.

Lemma 3 *A compound superstep of a v-processor CGM algorithm $\mathcal{A}$ with computation time $\tau + L$, communication time $g \cdot O(\frac{N}{v}) + L$, message size $k\frac{N}{v^2}$, for a known constant $k \geq 1$, and local memory size μ can be simulated in a compound superstep of a single processor EM-CGM algorithm in computation time $v\tau + O(v\mu)$ and I/O time $G \cdot O(\frac{N}{BD} + \frac{v\mu}{DB})$ provided that $M \geq \mu$, $D = O(\frac{N}{vB})$, and $B = O(\frac{N}{v^2})$.*

Algorithm 2 simulates a compound superstep of a v-processor CGM on a single processor EM-CGM with D disks. Due to lack of space we omit many of the details; see Appendix.

Algorithm 2 : SeqCompoundSuperstep
Input: For each $i \epsilon \{0, \ldots, v-1\}$ the blocks of the context are stored on the disks in consecutive format[2], and the arriving messages of virtual processor i are spread over the D disks consecutive format.
Output: (i) The (changed) contexts of the v simulated processors are spread across the disks in consecutive format. (ii) The generated messages for each processor in the next superstep are stored in consectutive format on the disks.

For $i = 0$ to $v - 1$

(a) Read the context of virtual processor i from the disks into memory.

(b) Read the packets received by virtual processor i from the disks.

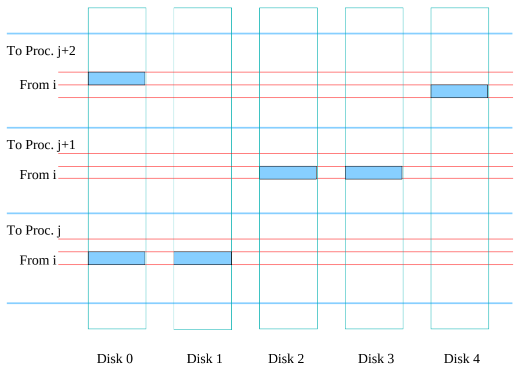

Figure 2: Illustration of the layout of message blocks on the disks. In the example we have $D = 5$ and message size $b' = 2$ blocks. Messages from processor i to processors j, $j+1$, and $j+2$ are shown as shaded rectangles. Messages to consecutively numbered processors are staggered on the disks to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

(c) Simulate the local computation of virtual processor of i.

(d) Write the packets which were sent by virtual processor i to the D disks in the staggered format illustrated in Figure 2. See appendix for details.

(e) Write the changed context of virtual processor i back to the D disks (in consecutive format).

Theorem 2 *A v processor CGM algorithm $\mathcal{A}$ with λ supersteps, local memory size μ, running time $\beta + g \cdot O(\frac{N}{v}) + \lambda L$, and message size $\Theta(\frac{N}{v^2})$ can be simulated as a single processor EM-CGM algorithm $\mathcal{A}'$ with time $v\beta + O(\lambda v\mu) + G \cdot O(\lambda\frac{v\mu}{DB})$ for $M = \Theta(\mu)$, $N = \Omega(v^2B)$, and $N = \Omega(vDB)$. In particular, algorithm $\mathcal{A}'$ is c-optimal if $\mathcal{A}$ is c-optimal, $\beta = \omega(\lambda\mu)$ and $G = DB \cdot o(\frac{\beta}{\lambda\mu})$. Furthermore, algorithm $\mathcal{A}'$ is work-optimal and I/O-efficient if $\mathcal{A}$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$ and $G = DB \cdot O(\frac{\beta}{\lambda\mu})$.*

Proof Sketch. We use the results of Lemma 3. The computation time required to simulate the computation steps of $\mathcal{A}$ is $v\beta$. The computational overhead associated with the I/O steps (Steps (a),(b),(d),(e)) is $O(\lambda v\mu) + O(\lambda N)$. Since $v\mu > N$ the total computation time is bounded by

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

$v\beta + O(\lambda v\mu)$. When c-optimality is required, we therefore need $\lambda v\mu = o(v\beta)$, or $\beta = \omega(\lambda\mu)$. Note that when $\mu = \Theta(\frac{N}{v})$, we can substitute $\beta = \omega(\frac{\lambda N}{v})$ for $\beta = \omega(\lambda\mu)$. For work-optimality, we require that $\lambda v\mu = O(v\beta)$, or $\beta = \Omega(\lambda\mu)$.

The I/O time (Steps (a),(b),(d),(e)) is $G \cdot [O(\lambda\frac{v\mu}{DB}) + O(\lambda\frac{N}{D})]$, which is bounded by $G \cdot O(\lambda\frac{v\mu}{DB})$. For c-optimality, we require the I/O time to be in $o(v\beta)$, which means that $G = DB \cdot o(\frac{\beta}{\lambda\mu})$. For I/O-efficiency, we require the I/O time to be in $O(v\beta)$, which means that $G = DB \cdot O(\frac{\beta}{\lambda\mu})$. $\square$

2.2 Multiple Processor Target Machine

For the case of $p \geq 1$ processors on the EM-CGM machine we simulate a compound superstep of a CGM algorithm $\mathcal{A}$ using the algorithm *ParCompoundSuperstep*, shown below. Unlike in the case of a single real processor, we are now forced to perform real communication between the real processors of the target machine. Each real processor i, $0 \leq i \leq p-1$, executes algorithm ParCompoundSuperstep in parallel. For ease of exposition, we assume that p divides v.

Algorithm 3 : ParCompoundSuperstep
Objective: Simulation of a compound superstep of a v-processor CGM on a p-processor EM-CGM.
Input: The message and context blocks of the virtual processors are divided among the real processors and their local disks. Each real processor i, $0 \leq i \leq (p-1)$ holds $O(\frac{N}{pB})$ blocks of messages and $\frac{v\mu}{pB}$ blocks of context, and each local disk contains $O(\frac{N}{pDB})$ blocks of messages and $O(\frac{v\mu}{pDB})$ blocks of context.
Output: The changed contexts and generated messages distributed as required for the next compound superstep.

```
For $j = 0$ to $\frac{v}{p} - 1$ do
    (a) Read the context for virtual processor $\frac{v}{p}i + j$ from the local disks.
    (b) Read any message blocks addressed to virtual processor $\frac{v}{p}i + j$ from the local disks.
    (c) Simulate the computation supersteps of virtual processor $\frac{v}{p}i + j$, collecting all generated messages in the local internal memory.
    (d) Send all generated messages to the required (real) destination processor. Upon arrival, the messages are arranged within the internal memory of the real destination processor and then written to its disks as in the single processor simulation; see Algorithm 2.
    (d) Write the contexts for virtual processor $\frac{v}{p}i+j$ back to the local disks; see Algorithm 2.
```

Lemma 4 *A compound superstep of a v-processor CGM algorithm $\mathcal{A}$ with computation time $\tau + L$, communication time $g \cdot O(\frac{N}{v}) + L$, message size $\Theta(\frac{N}{v^2})$, and local memory size μ can be simulated as $\frac{v}{p}$ compound supersteps of a p-processor EM-CGM algorithm $\mathcal{A}'$ in parallel computation time $\frac{v}{p}\tau + O(\frac{v}{p}\mu) + \frac{v}{p}L$ and I/O time $G \cdot O(G \cdot \frac{v}{p}\frac{\mu}{DB}) + \frac{v}{p}L$, for $p \leq v$, $N = \Omega(vDB)$, and $B = O(\frac{N}{v^2})$.*

Theorem 3 *A v processor CGM algorithm $\mathcal{A}$ with λ supersteps, computation time $\beta + \lambda L$, communication time $g\alpha + \lambda L$, local memory size μ and message size $\Theta(\frac{N}{v^2})$ can be simulated as a p-processor EM-CGM algorithm $\mathcal{A}'$ with computation time $\frac{v}{p}(\beta + O(\lambda\mu)) + \frac{v}{p}\lambda L$, communication time $\frac{v}{p}g\alpha + \frac{v}{p}\lambda L$, and I/O time $\frac{v}{p}G \cdot O(\lambda\frac{\mu}{DB}) + \frac{v}{p}\lambda L$ for $M = \Theta(\mu)$, $p \leq v$, $N = \Omega(vDB)$, and $N = \Omega(v^2B)$. Let $g(N)$, $L(N)$, and $v(N)$ be increasing functions of N. If $\mathcal{A}$ is c-optimal on the CGM for $g \leq g(N)$, $L \leq L(N)$ and $v \leq v(N)$, then $\mathcal{A}'$ is a c-optimal EM-CGM algorithm for $\beta = \omega(\lambda\mu)$, $g \leq g(N)$, $G = BD \cdot o(\frac{\beta}{\mu\lambda})$ and $L \leq L(N) \cdot \frac{p}{v}$. $\mathcal{A}'$ is work-optimal, communication-efficient, and I/O-efficient if $\mathcal{A}$ is work-optimal and communication-efficient, $\beta = \Omega(\lambda\mu)$, $g \leq g(N)$, $G = BD \cdot O(\frac{\beta}{\mu\lambda})$, and $L \leq L(N) \cdot \frac{p}{v}$.*

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

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

3 New EM Algorithms

A number of important problems have been shown to have non-linear I/O complexity [3, 7, 47]. The purpose of this section is to illustrate that, for a restricted parameter space that arises naturally in the EM domain, these problems have better I/O complexity than suggested by the lower bounds (which apply to a more general parameter constellation). Figure 5 lists a large number of problems for which we obtain new EM-CGM algorithms with lower I/O complexity by simulating CGM algorithms using Theorem 3.

3.1 Fundamental Problems

We first present new EM-CGM algorithms, obtained via Lemma 2 and Theorem 3, for the fundamental problems of sorting, permutation and matrix transpose. For each of these problems a CGM algorithm uses $\lambda = O(1)$ communication rounds, and $O(\frac{N}{v})$ internal memory per processor.

Sorting: The time complexity of sorting N items is $\Theta(N \lg N)$ on a RAM. On the PDM, sorting has been shown to have I/O complexity $O(\frac{N}{DB} \log_{\frac{M}{B}} \frac{N}{B})$ for general values of N, M, D, and B [3, 47]. However, for $\frac{N}{v} \geq v^\epsilon$, $\epsilon > 0$ a fixed constant, we can achieve I/O complexity $O(\frac{N}{pBD})$ by simulating the deterministic CGM sorting algorithm of Goodrich [31] for $N \geq max\{v^{1+\epsilon}, vDB\}$.

Permutation: Permutation of N items on a RAM has time complexity $\Theta(N)$. On the PDM, this problem has I/O complexity $\Theta(min(\frac{N}{D}, \frac{N}{DB} \log_{M/B} \frac{N}{B})$ (see [3, 47]). However, we can achieve I/O complexity $O(\frac{N}{pBD})$ by simulating algorithm CGMPermute, for $N \geq max\{v^{1+\epsilon}, vDB\}$.

Algorithm 4 CGMPermute
$\mathcal{V}$ is an N element vector containing items to be permuted. $\mathcal{P}$ is a corresponding N element vector containing new indices for each element of $\mathcal{V}$.
Input: Each processor i, $0 \leq i \leq (v-1)$ holds an $\frac{N}{v}$ element vector $\mathcal{V}_i$, containing elements $i \cdot \frac{N}{v}$ to $(i+1) \cdot \frac{N}{v} - 1$ of $\mathcal{V}$, and an $\frac{N}{v}$ element vector $\mathcal{P}_i$, containing elements $i \cdot \frac{N}{v}$ to $(i+1) \cdot \frac{N}{v} - 1$ of $\mathcal{P}$.
Output: Each processor i contains items $i \cdot \frac{N}{v}$ to $(i+1) \cdot \frac{N}{v} - 1$ of the permuted vector $\mathcal{V}'$.
Assumption: v divides N evenly.

1. Each processor i, $0 \leq i \leq (v-1)$ sends the items of $\mathcal{V}_i$ to the processors holding the items indicated by $\mathcal{P}_i$.

2. Each processor performs the necessary rearrangements in its local memory to complete the calculation of $\mathcal{P}$.

Transpose: Transposing a $k \times \ell$ matrix, where $N = k\ell$ takes $\Theta(N)$ time on a RAM. On the Parallel Disk Model, this problem has I/O complexity $\Theta(\frac{N}{BD}\frac{\log min(M,k,\ell,N/B)}{\log(M/B)})$ [3, 47]. However, we can achieve I/O complexity $O(\frac{N}{pBD})$ by simulating an algorithm CGMTranspose, similar to CGMPermute, for $N \geq max\{v^2, vDB\}$.

Theorem 4 *Sorting, permutation, and matrix transpose can be performed on a p-processor EM-CGM in $O(\frac{N}{pBD})$ I/O operations, provided $N = \Omega(vBD)$, $N \geq v^2B + \frac{v^2(v-1)}{2}$, and $N \geq v^\kappa$, where constant $\kappa > 1$ depends on the problem.*

3.2 External Memory Algorithms Made Available by the Simulation

Figure 5 lists a number of important problems arising in computational geometry, GIS, and graph algorithms, for which we report EM-CGM algorithms created by our technique, together with their I/O complexity and that of the previously best known algorithm for the problem. In some cases, we obtain a better I/O complexity, and in other cases we obtain the same I/O complexities as previously known. In each case, however, our algorithm is scalable not only in terms of the number of disks per processor but also in terms of the number of processors used. Previous algorithms were often not efficient in a multiprocessor environment (particularly in a distributed memory environment), and in many cases it is not clear how they could be adapted to parallel disks.

4 Experiments

In recent years there have been a small but growing number of implementation projects which focused

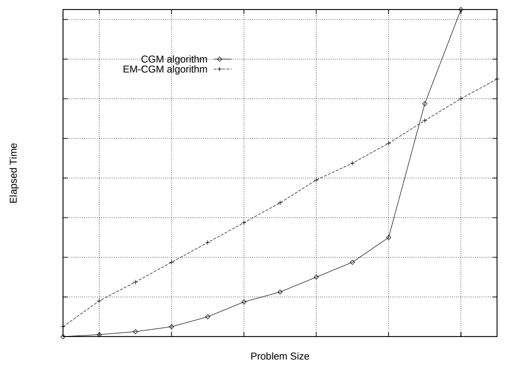

Figure 3:

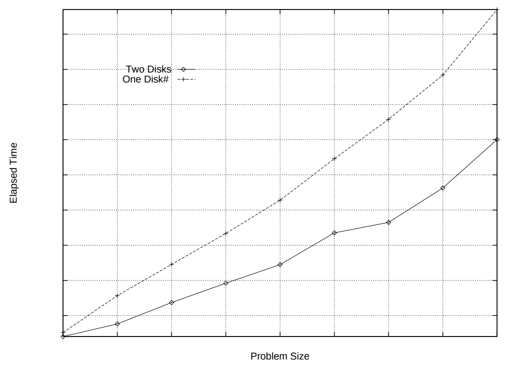

Figure 4:

on EM issues. References in this regard include the TPIE project [44], as well as [33, 14, 19, 18, 20, 22]. Our implementation results indicate that EM-CGM algorithms obtained via the techniques of this paper will likely be an important component of an EM workbench.

Preliminary implementation experiments with sorting support our predictions of linear running time. Figure 3 shows running times for a CGM sorting algorithm a) using virtual memory and LAM-MPI (see [1]), and b) converted to an EM-CGM algorithm by our deterministic simulation. As expected, multiple disks also reduce the running time. Figure 4 shows the running time of EM-CGM sort with one and two disks respectively.

5 Extensions

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

1. Conforming BSP algorithms can be converted to BSP* algorithms with $b = \frac{h_{min}}{v} - \frac{v-1}{2}$, where h_{min} is the minimum value of h used in any communication superstep.
2. Conforming BSP algorithms can be converted to EM-BSP algorithms and c-optimality is preserved.
3. Conforming BSP* algorithms can be converted to EM-BSP* algorithms and c-optimality is preserved.

The term "conforming" in items (1), (2), and (3) above refers to the need for the bounding concept of an h-relation to be a universal assumption in the analysis of the original BSP-like algorithm for each of its communication rounds. It is convenient, but not necessary, that the same value of h be used in every round.

Cache Memories: So far, we have considered only the interaction between the main memory and the external disk system. Many of the same issues also arise between the cache memory and main memory layers of the memory hierarchy. For simplicity, let us consider a computer with a two level memory hierarchy consisting of only a cache, and a main memory. The lower bounds for the number of block accesses required on fundamental problems such as sorting, permutation, matrix transpose [3] apply to this interface as well. Let $N = M$ be the size of a problem stored in the main memory, let M_I be the size of a cache memory I, and let B_I be the size of blocks transferred between I and the main memory. Our results indicate that if $(\frac{M_I}{B_I})^c = N$, the logarithmic term in the I/O complexity vanishes, to be replaced by a constant. There may therefore be concrete and significant savings to be realized in computation time on sequential computers if the cache design takes these results into account, and programs are formulated as parallel algorithms with virtual processor sizes tuned to the available cache memory. We refer the interested reader to the work of Vishkin [45, 46] for related ideas.

Acknowledgements
The authors thank Kumanan Yogaratnam for his programming work.

	Problem Description	PDM I/O Complexity	Complexity in CGM Model	New I/O Complexity[a]

Group A: Fundamental Algorithms[b]

1.	Sorting	$\Theta(\frac{N}{BD}\log_{\frac{M}{B}}\frac{N}{B})$ [3, 47]	$O(\frac{N\log N}{v})$ [31] $\lambda = O(1)$, $M = O(\frac{N}{B})$	$O(\frac{N}{pDB})$
2.	Permutation	$\Theta(min(\frac{N}{D}, \frac{N}{DB}\log_{\frac{M}{B}}\frac{N}{B})$ [3, 47]	$O(\frac{N\log N}{v})$ [this paper] $\lambda = O(1)$, $M = O(\frac{N}{B})$	$O(\frac{N}{pDB})$
3.	Matrix transpose[c]	$\Theta(\frac{N}{BD}\frac{\log min(M,k,\ell,N/B)}{\log(M/B)})$ [3, 47]	$O(\frac{N\log N}{v})$ [this paper] $\lambda = O(1)$, $M = O(\frac{N}{B})$	$O(\frac{N}{pDB})$

Group B: GIS and Computational Geometry Algorithms

1.	Polygon triangulation, Trapezoidal decomposition, Segment tree construction, Next Element Search on line segments[d]	$O(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B})$ [8]	$\tau = O(\frac{N\log N}{v})$, [13] $\lambda = O(1)$, $M = O(\frac{N\log N}{v})$	$O(\frac{N\log N}{pDB})$
2.	Batched planar point location	$O((\frac{N}{B}+k)\log_{\frac{M}{B}}\frac{N}{B})$ [8]	$\tau = O(\frac{N\log N}{v})$ [13] $\lambda = O(1)$, $M = O(\frac{N\log N}{v})$	$O(\frac{N\log N}{pDB})$
3.	3D convex hull, 2D Voronoi diagram, Delaunay triangulation [e]	$O(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B})$ [32]	$\tau = \tilde{O}(\frac{N\log N}{v})$ [24] $\lambda = \tilde{O}(1)$, $M = \tilde{O}(\frac{N}{v})$	$\tilde{O}(\frac{N}{pDB})$
4.	Lower envelope of non-intersecting line segments		$\tau = O(\frac{NlogN}{v})$ [27] $\lambda = O(1)$, $M = O(\frac{N}{B})$	$O(\frac{N}{pDB})$
5.	Generalized lower envelope of line segments		$\tau = O(\frac{NlogN}{v})$ [27] $\lambda = O(1)$, $M = O(\frac{N\alpha(N)}{B})$	$O(\frac{N\alpha(N)}{pDB})$
6.	Area of Union of Rectangles, 3D-maxima, 2D-nearest neighbors of a point set	$O(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B})$ [32]	$O(\frac{N\log N}{v})$ [27] $\lambda = O(1)$, $M = O(\frac{N}{B})$	$O(\frac{N}{pDB})$
7.	2D-weighted dominance counting, Uni-directional and multi-directional separability		$\tau = O(\frac{N\log N}{v})$, $\lambda = O(1), M = O(\frac{N}{B})$ [27]	$O(\frac{N}{pDB})$

Group C: Graph Algorithms

1.	List ranking, Euler tour of tree, Lowest common ancestor, Tree contraction, expression tree evaluation	$O(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B})$ [15]	$\lambda = O(\log v)$, $\tau = O(\frac{N}{v})$ $M = O(\frac{N}{v})$ [12]	$O(\frac{N\log v}{pDB})$
2.	Connected components, spanning forest, Ear and open ear decomposition, Biconnected components[f]	$O(\min\{\frac{V^2}{B}\log_{\frac{M}{B}}\frac{V}{B},$ $\log\frac{V}{M}\cdot\frac{E}{B}\log_{\frac{M}{B}}\frac{E}{B}\})$ [15]	$\lambda = O(\log v)$, $\tau = O(\frac{V+E}{v})$ $M = O(\frac{V+E}{v})$ [12]	$O(\frac{(V+E)\log v}{pDB})$

Figure 5: Overview of New EM Algorithms in Comparison To Previous Results.

[a]These results are subject to the conditions $N = \Omega(vDB)$, $N \geq v^2B + v^2(v-1)/2$, and $N > v^\kappa$, where $\kappa > 1$ is a constant that depends on the problem. For the problems examined in this paper, $\kappa \leq 3$.

[b]The PDM I/O complexities listed for this group apply also to multiple processors when the interconnection method is a shared RAM, hypercubic network, or cube-connected cycles. The PDM parameter D is the number of disks in total, while our parameter D is the number of disks on each processor. They are equivalent if $p = 1$.

[c]k, ℓ are the number of columns and rows, respectively, where $N = k\ell$.

[d]The PDM I/O complexities for this group are documented for the single processor, single disk case. We are not aware of multi-disk or multi-processor extensions.

[e]The PDM I/O complexities previously reported for these problems apply to a PRAM-like interconnection only.

[f]for a graph of V vertices and E edges

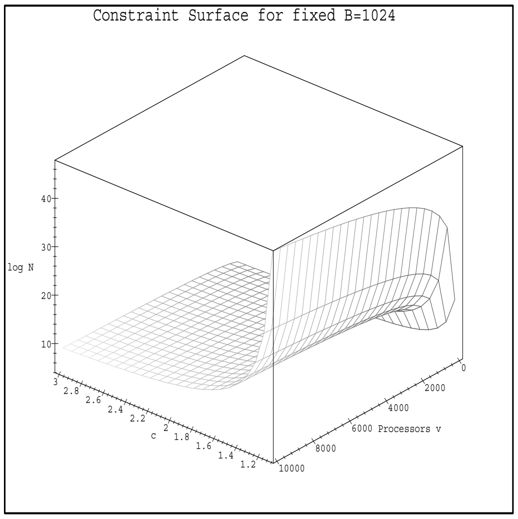

Figure 6: The surface $N^{c-1} = v^c B^{c-1}$.

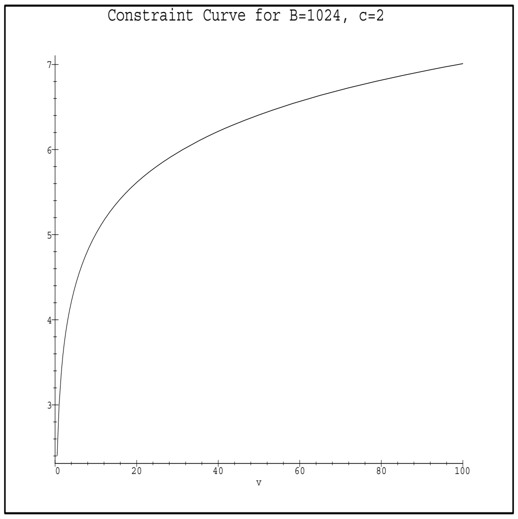

Figure 7: 2D Projection of Figure 6 for fixed $c = 2$, clipped to $v \leq 100$. The vertical axis is $log_{10}N$.

References

[1] http://www.osc.edu/lam.html.

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

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

[4] Arge, L. The buffer tree: A new technique for optimal I/O-algorithms. In *Proc. Workshop on Algorithms and Data Structures, LNCS 955* (1995), pp. 334–345. A complete version appears as BRICS technical report RS-96-28, University of Aarhus.

[5] Arge, L. The I/O-complexity of ordered binary-decision diagram manipulation. In *Proc. Int. Symp. on Algorithms and Computation, LNCS 1004* (1995), pp. 82–91. A complete version appears as BRICS technical report RS-96-29, University of Aarhus.

[6] Arge, L. *Efficient External-Memory Data Structures and Applications.* PhD thesis, University of Aarhus, February/August 1996.

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

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

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

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

[11] Bäumker, A., Dittrich, W., and auf der Heide, F. M. Efficient parallel algorithms: c-optimal multisearch for an extension of the bsp model. In *Proc. Annual European Symposium on Algorithms* (1995), pp. 17–30.

[12] Cáceres, E., Dehne, F., Ferreira, A., Flocchini, P., Reiping, I., Santoro, N.,

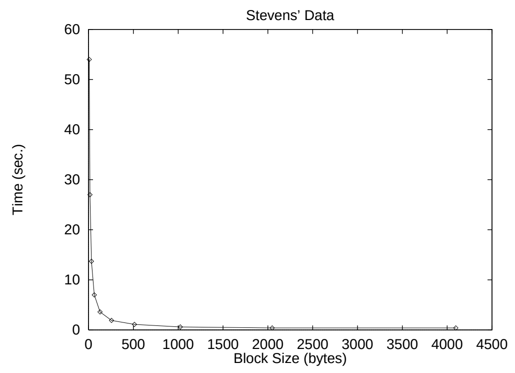

Figure 8: Stevens' measurements on the effects of varying the block-size.

AND SONG, S. Efficient parallel graph algorithms for coarse grained multicomputers and bsp. In *Proc. Int. Colloquium Algorithms, Languages and Programming, LNCS 1256* (1997), pp. 390–400.

[13] CHAN, A., DEHNE, F., AND RAU-CHAPLIN, A. Coarse grained parallel next element search. In *Proc. International Parallel Processing Symposium* (1997), pp. 320–325.

[14] CHIANG, Y.-J. *Dynamic and I/O-Efficient Algorithms for Computational Geometry and Graph Problems: Theoretical and Experimental Results.* PhD thesis, Brown University, August 1995.

[15] CHIANG, Y.-J., GOODRICH, M. T., GROVE, E. F., TAMASSIA, R., VENGROFF, D. E., AND VITTER, J. S. External-memory graph algorithms. In *Proc. ACM-SIAM Symp. on Discrete Algorithms* (1995), pp. 139–149.

[16] CORMEN, T. H. Personal communication, 1997.

[17] CORMEN, T. H., AND GOODRICH, M. T. Position Statement, ACM Workshop on Strategic Directions in Computing Research: Working Group on Storage I/O for Large-Scale Computing. *ACM Computing Surveys 28A(4)* (Dec. 1996).

[18] CORMEN, T. H., AND HIRSCHL, M. Early Experiences in Evaluating the Parallel Disk Model with the ViC* Implementation. Tech. Rep. PCS-TR96-293, Dartmouth College, Computer Science, Hanover, NH, September 1996.

[19] CORMEN, T. H., AND KOTZ, D. Integrating Theory and Practice in Parallel File Systems. Tech. Rep. PCS-TR93-188, Dartmouth College, Computer Science, Hanover, NH, 1993.

[20] CORMEN, T. H., AND NICOL, D. M. Performing Out-of-Core FFTs on Parallel Disk Systems. Tech. Rep. PCS-TR96-294, Dartmouth College, Computer Science, Hanover, NH, September 1996.

[21] CORMEN, T. H., SUNDQUIST, T., AND WISNIEWSKI, L. F. Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. Tech. Rep. PCS-TR94-223, Dartmouth College Dept. of Computer Science, July 1994. To appear in SIAM J. on computing.

[22] CORMEN, T. H., WEGMANN, J., AND NICOL, D. M. Multiprocessor Out-of-Core FFTs with Distributed Memory and Parallel Disks. Tech. Rep. PCS-TR97-303, Dartmouth College, Computer Science, Hanover, NH, January 1997.

[23] CRAUSER, A., FERRAGINA, P., MEHLHORN, K., MEYER, U., AND RAMOS, E. Randomized external memory algorithms for geometric problems. In *Proc. ACM Annual Conference on Computational Geometry* (1998). (to appear).

[24] DEHNE, F., DENG, X., DYMOND, P., FABRI, A., AND KHOKHAR, A. A randomized parallel 3d convex hull algorithm for coarse grained multicomputers. In *Proc. ACM Symp. on Parallel Algorithms and Architectures* (1995), pp. 27–33.

[25] DEHNE, F., DITTRICH, W., AND HUTCHINSON, D. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *Proc. ACM Symp. on Parallel Algorithms and Architectures* (1997), pp. 106–115.

[26] DEHNE, F., FABRI, A., AND KENYON, C. Scalable and architecture independent parallel geometric algorithms with high probability optimal time. In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing* (1994), pp. 586–593.

[27] DEHNE, F., FABRI, A., AND RAU-CHAPLIN, A. Scalable parallel geometric algorithms for

coarse grained multicomputers. In *Proc. ACM Annual Conference on Computational Geometry* (1993), pp. 298–307.

[28] Dittrich, W., Hutchinson, D., and Maheshwari, A. Blocking in parallel multisearch problems. In *Proc. ACM Symp. on Parallel Algorithms and Architectures* (1998). (To appear).

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

[30] Gibson, G. A., Vitter, J. S., and Wilkes, J. Strategic directions in storage i/o issues in large-scale computing. *ACM Computing Surveys 28(4)* (Dec. 1996), 779–793.

[31] Goodrich, M. Communication efficient parallel sorting. In *Proc. ACM Symp. on Theory of Computation* (1996).

[32] Goodrich, M. T., Tsay, J.-J., Vengroff, D. E., and Vitter, J. S. External-memory computational geometry. In *Proc. IEEE Symp. on Foundations of Comp. Sci.* (1993), pp. 714–723.

[33] Hutchinson, D., Maheshwari, A., Sack, J.-R., and Velicescu, R. Early experiences in implementing the buffer tree. In *Proceedings of the Workshop on Algorithm Engineering* (1997). http://www.dsi.unive.it/ wae97/proceedings.

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

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

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

[37] Nodine, M. H., and Vitter, J. S. Paradigms for optimal sorting with multiple disks. In *Proc. of the 26th Hawaii Int. Conf. on Systems Sciences* (1993).

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

[39] Sibeyn, J., and Kaufmann, M. Bsp-like external-memory computation. In *Proc. 3rd Italian Conference on Algorithms and Complexity* (1997).

[40] Stevens, R. W. *Advanced Programming in the Unix Environment.* Addison-Wesley, Don Mills, Ont., Canada, 1995.

[41] Ullman, J. D., and Yannakakis, M. The input/output complexity of transitive closure. *Annals of Mathematics and Artificial Intellegence 3* (1991), 331–360.

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

[43] van Kreveld, M., Nievergelt, J., Roos, T., and Widmayer, P., Eds. *Algorithmic Foundations of Geographic Information Systems, LNCS 1340.* Springer, 1997.

[44] Vengroff, D. E., and Vitter, J. S. Supporting I/O-efficient scientific computation in TPIE. In *Proc. IEEE Symp. on Parallel and Distributed Computing* (1995). Appears also as Duke University Dept. of Computer Science technical report CS-1995-18.

[45] Vishkin, U. Can parallel algorithms enhance serial implementations? *Communications of the ACM 39(9)* (1996), 88–91.

[46] Vishkin, U. From algorithm parallelism to instruction-level parallelism: An encode-decode chain using prefix-sum. In *Proc. ACM Symp. on Parallel Algorithms and Architectures* (June 1997), pp. 260–270.

[47] Vitter, J. S. External memory algorithms. *Proc. ACM Symp. Principles of Database Systems* (1998), 119–128.

[48] Vitter, J. S. Personal communication, 1998.

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

[50] Vitter, J. S., and Shriver, E. A. M. Algorithms for parallel memory, II: Hierarchical multilevel memories. *Algorithmica 12*, 2–3 (1994), 148–169.

6 Appendix

6.1 BSP-like Models

We adopt a family of models in this paper which were first introduced in [25], and which incorporate the parameters of the PDM, except for two small differences. We will assume that *each* of the p processors has D local disk drives, and M local memory. Our models are described in more detail in Section 6.2.

The **BSP** (Bulk Synchronous Parallel) Model was introduced in 1990 [42]. 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 comsists 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

$\hat{g}$ is the minimum time required by the router to deliver a unit of data

All times are in basic computation units.

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

The **BSP*** model was introduced in 1995 [11] 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 *packet size*, 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. A BSP* computer can be characterized by the following parameters:

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

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 $w^i_{comm,j} = max\{L, g(\sum_{u=1}^{j'} \lceil \frac{r_i}{b} \rceil + \sum_{u=1}^{j''} \lceil \frac{s_i}{b} \rceil)\}$.

The **CGM** (Coarse Grained Multicomputer) model was introduced in 1993 [27, 26, 24]. It uses only two parameters, n and p, and assumes a collection of p processors with n/p local memory, each 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 $h \leq n/p$. The cost w^i_{comm} 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 $T_{comm} = \lambda H_{n,p}$. Algorithms do usually require a lower bound on $n/p >> 1$, e.g. $n/p \geq p$ or $n/p \geq p^{\epsilon}$ [27, 26, 24] .

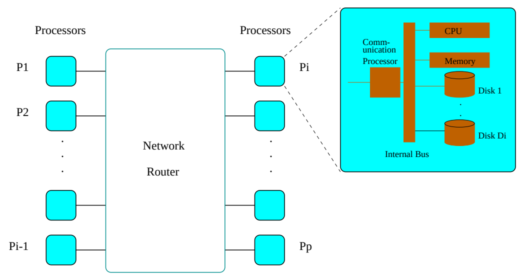

Figure 9: Illustration of a Parallel Machine with External Memory

The CGM model works particularly well in the case where the overall computation speed is considerably larger than the overall communication speed.

We shall refer to the BSP, BSP*, and CGM models as *BSP-like models*. The term *h-relation* is commonly used to refer to a communication step on such a computer in which messages of length at most h bytes are sent and received by every processor.

6.2 The EM-CGM Model

The EM-CGM Model was first introduced in [25]. The basic idea is illustrated in Figure 9. In addition to its local memory, each processor has an external memory in the form of a set of hard disks. The CGM model is extended to its external memory version **EM-CGM** by the addition of the following 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 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 CGM model, the computation on the EM-CGM model proceeds in a succession of supersteps. We adapt communication and computation supersteps from the CGM model and allow multiple I/O-operations during a single computation superstep. For the EM-CGM model, the computation cost, t_{comp}, and communication cost, t_{comm}, are the same as for the CGM model. The total cost of each superstep is defined as $t_{comp} + t_{comm} + t_{I/O} + L$. Each I/O operation costs G time units. The I/O cost $t_{I/O}$ of each CGM round is identical, and represents the I/O cost for simulating an h-relation with $h = n/p$.

Note that the model gives incentives to access all disk drives using block transfers. For instance, a single processor EM-CGM 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.

6.3 The EM-BSP and EM-BSP* Models

The EM-BSP and EM-BSP* models are defined analogously to the EM-CGM model. To the BSP and BSP* models the following additonal parameters are added:

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 or BSP* superstep is defined as $t_{comp} + t_{comm} + t_{I/O} + L$ where t_{comp} and t_{comm} refer to the computation time and communication time as defined for the BSP* model. The term $t_{I/O}$ is the additional I/O cost charged for the superstep, where $t_{I/O} = \max_{j=1}^{p}\{w_{I/O}^{j}\}$, and $w_{I/O}^{j}$ is the I/O-cost incurred by processor j. Each I/O operation costs G time units.

We shall refer to the BSP, BSP* and CGM models collectively as *BSP-like* models.

6.4 EM Cost Model

We adopt the cost model proposed in [25, 28].

Definition 1 *[25] Let $\mathcal{A}$ be the optimal sequential algorithm on the RAM for the problem under consideration, and let $T(\mathcal{A})$ be its worst case running time. 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 $N \to \infty$.

We say that a EM algorithm is *one-optimal* if it is c-optimal for $c = 1$. The constraint on ϕ is another way of saying that $\mathcal{A}^*$ must be *work optimal*. The constraints on ξ and η ensure that the communication and I/O time do not affect the asymptotic running time, even by a constant factor.

We will use the terms *communication-efficient* and *I/O-efficient* to describe an algorithm for which ξ and η, respectively, are $O(1)$. An algorithm which is work-optimal, communication-efficient, and I/O-efficient, therefore, is one whose running time complexity is no worse than the complexity $T(\mathcal{A})/p$. Constant factors are ignored. We will call an algorithm *I/O-optimal* if the number of I/O operations matches the lower bound for the number of I/Os required to solve the problem.

This definition applies equally well to EM-BSP, EM-BSP* and EM-CGM algorithms. In the interests of brevity and ease of exposition, we focus primarily on the EM-CGM case. Conditions on the input size n and the 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 are functions $p(N)$, $b(N)$, $g(N)$, $B(N)$, $G(N)$, $L(N)$, and $M(N)$ 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.

6.5 Proof of Theorem 1

The proof of the maximum message sizes is given in [10]. In the following, we give a proof for the minimum message sizes. In round A each processor initially has $\frac{\bar{n}}{v}$ data. At the end of Superstep B, each processor will have at most $\bar{h}$ data.

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

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

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

6.6 Proof of Corollary 1

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

We can assume a minimum message size of $\frac{h}{v} - \frac{v-1}{2}$ in the second round because the cost of communication is bounded by the assumption of an h-relation. When every processor is the destination of h data, it does not affect the worst case complexity of the superstep. We can therefore assume that every processor receives h data (by adding dummy items for the sake of the argument). Hence the minimum message size for any processor in Superstep B becomes $\frac{h}{v} - \frac{v-1}{2}$ without affecting the asymptotic communication cost of the superstep. $\square$

6.7 Proof of Lemma 1

From Corollary 1, we can achieve a minimum message size b_{min} provided that $b_{min} \leq \frac{N}{v^2} - \frac{v-1}{2}$. $\square$

6.8 Proof of Lemma 2

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

6.9 Proof of Lemma 3

We use the results of Lemma 2. Since messages are at most $2 \cdot \frac{h}{v} \leq 2\frac{N}{v^2}$ in size we can allocate fixed sized slots for them on the disks while preserving an $O(v\mu)$ disk space requirement. In many practical EM situations $2\frac{N}{v^2}$ will be a significant overestimate of the maximum message size, as $v << \frac{N}{v^2}$. The assurance of minimum message size $\Omega(B)$ implies that I/O operations will be blocked[3].

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

[3]Although a CGM algorithm may occasionally use smaller messages than B, it is charged for an $\frac{N}{v}$-relation in each superstep as if every processor sent and received h data

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

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

Details of Algorithm 2

Details of Steps (a) and (e): Since we know the size of the contexts of the processors, we can distribute the 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 V_j of virtual processor j into blocks of size B and store the i-th block of V_j on disk $(i + j\frac{\mu}{B}) \mathit{mod} D$ using track $\left\lfloor \frac{i+j\frac{\mu}{B}}{D} \right\rfloor$. Since the context of each processor is now in striped format on the disks, we can easily read and write the contexts using D disks in parallel for every I/O operation.

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

Details of Step (d): After the Computation

Phase, all messages sent by the current virtual processor have been generated and stored in internal memory. The coarse-grained nature of the underlying BSP-like algorithm results in large messages, (see Lemma 2) which are as long or longer than the block size B. We cut the messages into blocks of size B. Each block inherits the destination address from its original message. In $\frac{\gamma}{BD}$ rounds, we write the blocks out to the disks, as described in detail below.

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

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

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

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

We can avoid this extra space requirement, however, as follows.

Observation 2 *By alternating between* $\{$*consecutive reads, staggered writes*$\}$ *and* $\{$*staggered reads, consecutive writes*$\}$ *in successive compound supersteps, the simulation can achieve fully parallel I/O on all message blocks with a single copy of the messaging matrix.*

SeqCompoundSuperstep loads each virtual processor into the real memory, requiring that $M \geq \mu$. Since the messages sent or received in a superstep by a virtual processor are $h = \Theta(\frac{N}{v})$ in total size, we require that $\frac{N}{vB} = \Omega(D)$ to ensure that our I/O scheme for messages is efficient. This means that $D = O(\frac{N}{vB})$.
Disk Space: The disk space needed by the simulation is the total context size $v\mu$, which includes space for messages. At most one track is wasted for each virtual processor. The space used on each disk is $O(\frac{v\mu}{DB})$, since $\frac{N}{v} = \Omega(DB)$.
Computation Time: Steps (a) and (e) of algorithm SeqCompoundSuperstep require computation time $O(v\mu)$. In Steps (b) and (d), $O(\frac{N}{v})$ message data is routed for each virtual processor. Over all v processors, this adds $O(N)$ computation time overall, which can be ignored. Step (c) consumes $v\tau$ computation time.
I/O Time: Steps (a) and (e) consume $O(G\frac{v\mu}{DB})$ time, and steps (b) and (d) consume $O(G\frac{N}{DB})$ time. Since $O(N/p)$ message data is sent in each superstep, and $N/p \leq \mu$ we have time $O(G\frac{v\mu}{DB})$ due to I/O overall.

Thus, overall, the computation time is $v\tau + O(v\mu)$ and the I/O-time is $O(G\frac{v\mu}{DB})$. □

6.10 Proof of Lemma 4

We have $p \leq v$ real processors, so the time to simulate Step (c) is $\frac{v}{p}\tau$. Computational overhead is contributed by $\frac{v}{p}(O(\mu)+O(\frac{N}{p}))$, due to swapping of contexts (Steps (a),(d)) and messaging I/O (Steps (b),(d)). As before, the computational overhead is dominated by the cost of swapping contexts. The original compound superstep is replaced by $\frac{v}{p}$ compound supersteps on the target machine.

The I/O time is determined by the cost of swapping contexts plus the cost of simulating the original messaging via I/O. These costs are $G \cdot \frac{v}{p}\frac{\mu}{DB}$ and $G \cdot \frac{v}{p}O(\frac{N}{vD})$ respectively, and the total is dominated by $G \cdot \frac{v}{p}\frac{\mu}{DB}$. □