Reducing I/O Complexity by Simulating Coarse Grained Parallel Algorithms

Frank Dehne, David Hutchinson, Anil Maheshwari
School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6, {*dehne, hutchins, maheshwa*}*@scs.carleton.ca*

Wolfgang Dittrich
Bosch Telecom GmbH, UC-ON/ERS
Gerberstraße 33, 71522 Backnang
Germany

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 parallel algorithms into (parallel) external memory algorithms. Specifically, we present a* deterministic *simulation technique which transforms* Coarse Grained Multicomputer *(CGM) algorithms into external memory algorithms for the* Parallel Disk Model. *Our technique 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. All of the (parallel) external memory algorithms obtained via simulation are analyzed with respect to the computation time, communication time and the number of I/O's. Our results answer to the challenge posed by the ACM working group on storage I/O for large-scale computing [8].

1 Introduction

Cormen and Goodrich [8] posed the *challenge* of combining BSP-like parallel algorithms with the requirements for parallel disk I/O. Solutions based on probabilistic methods were presented in [11] and [14]. 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 to that challenge. The analysis of the I/O complexity of our algorithms is done as in the PDM model. In addition, we analyze the running time and communication time.

We first review some definitions for the BSP and CGM model; consult [11, 23, 13] 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*. The notion of an h-relation is often used in the analysis of parallel algorithms based on BSP-like models. An h-relation is a communication superstep in which each of the v processors sends and receives at most h data items. An algorithm for a CGM with multiple disks attached to each processor 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 $v \geq p$, $M = \Theta(\mu)$, $N = \Omega(vDB)$ and $B = O(\frac{N}{v^2})$. G is the time (in processor cycles) for a parallel I/O operation.

   Let $g(N)$, $L(N)$, $v(N)$ be increasing functions of N. If $\mathcal{A}$ is c-optimal (see [16, 11] for definition) 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 (see [11] for definitions) if $\beta = \Omega(\lambda\mu)$, $g \leq$

$g(N)$, $G = BD \cdot O(\frac{\beta}{\mu\lambda})$, 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. Once the expression for the general lower bounds reported by [1, 26, 3] is specialized for the subset of the parameter space that we consider, it becomes fully compatible with our results. This answers questions of Cormen [7] and Vitter [27] on the apparent contradictions between the results of [11] 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) uni-directional 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.

The results for Items (2), (3) and (4) 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 multiple processors and multiple disks. It has relatively few parameters, generalizes the PDM model, and answers to the challenge of [8].

Note that our results apply to a wide spectrum of parallel algorithms. For example, several well known simulation techniques map parallel algorithms designed for the PRAM model to the BSP model. Furthermore, 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 [24, 25].

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 [22]. This reduces the constraint to a surface $N^{c-1} = v^c B^{c-1}$ in three dimensional space. Any point on or above the surface represents a valid set of parameters for the elimination of the logarithmic factor. For 100 processors or less, for instance, any problem size greater than about 10 mega-items is sufficient.

Due to page restrictions for the conference proceedings, some details are omitted in this paper. An extended version with additional figures and charts can be found at *http://www.scs.carleton.ca/~dehne*.

2 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 [4])
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 ℓ^{th} word of msg_{ij} to local bin $(i + j + \ell)$ mod v

2 For $j = 0$ to $(v-1)$
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}.*

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. The proof of the maximum message sizes is given in [4]. In the following, we give a proof for the minimum message sizes. In Superstep 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. 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 $(i+k)$ mod v by Step 1. Superbin j now contains the message which is to be sent from processor j to processor k in Step 4.

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

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

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

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

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

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

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

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

Proof. 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. □

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.

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 $G \cdot O(\frac{N}{DB} + \frac{v\mu}{DB})$ provided that $M \geq \mu$, $D = O(\frac{N}{vB})$, and $B = O(\frac{N}{v^2})$.*

The proof is given below, following the presentation and discussion of Algorithm 2. This 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.

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 $(d+q)$ mod 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 $(d+S+q)$ mod 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).

Algorithm 2 simulates a compound superstep of a v-processor CGM on a single processor EM-CGM with D disks.

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, 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 consecutive 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.

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

(d) Write the packets which were sent by virtual processor i to the D disks in staggered format.

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

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[1].

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 $\left(i + j\frac{\mu}{B}\right) \ mod \ D$ using track $\left\lfloor \frac{i + j\frac{\mu}{B}}{D} \right\rfloor$. Since the context of each processor is now in 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 $(b_0 + b')$ mod 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' mod $D \neq 0$. Let $T_j = j \cdot \lceil \frac{vb'}{D} \rceil$ be the track offset for msg_{0j} (the first such message destined

[1] 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

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.*

We now begin with the **proof of Lemma 3**.

Algorithm 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 $O(G\frac{v\mu}{DB})$.

This concludes the proof of Lemma 3.

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 $v\beta + O(\lambda v\mu) + G\lambda \cdot O(\frac{v\mu}{DB})$ for $M = \Theta(\mu)$, $B = O(\frac{N}{v^2})$, 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 $v\beta + O(\lambda v\mu)$. 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 $G = DB \cdot O(\frac{\beta}{\lambda\mu})$. □

3 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 $B = O(\frac{N}{v^2})$.*

Proof. 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 $G \cdot \frac{v}{p}\frac{\mu}{DB}$. □

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 $B = O(\frac{N}{v^2})$.*
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 $L \leq L(N) \cdot \frac{p}{v}$. □

4 Conclusions and Future Work

The key result of this paper is a deterministic simulation theorem, that maps algorithms for coarse grained parallel computing models to external memory algorithms for the parallel disk model with multiple processors. As a corollary we obtain external memory algorithms for a spectrum of problems, and all of these algorithms are scalable in terms of the number of disks and processors.

Our recent work involves replacing Algorithm 1 by another algorithm for achieving fixed size messages in such a way that the slackness condition for the parallel algorithm improves to $N \geq v^{1+\epsilon}B$, where ϵ is a constant, $0 < \epsilon \leq 1$. Moreover this provides a better bound for N: $M \leq N \leq \frac{M^{1+\frac{1}{\epsilon}}}{B^{\frac{1}{\epsilon}}}$. However, the number of supersteps increases by a factor of at most $\frac{1}{\epsilon^2}$.

References

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

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

[3] 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.

[4] 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/.

[5] CÁCERES, E., DEHNE, F., FERREIRA, A., FLOCCHINI, P., REIPING, I., SANTORO, N., 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.

[6] 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.

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

[8] 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).

[9] 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.

[10] 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.

[11] 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.

[12] DEHNE, F., DITTRICH, W., HUTCHINSON, D., AND MAHESHWARI, A. Parallel Virtual Memory. 2 page abstract to appear in *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms* (1999), Baltimore, USA.

[13] 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.

[14] DITTRICH, W., HUTCHINSON, D., AND MAHESHWARI, A. Blocking in parallel multisearch problems. In *Proc. ACM Symp. on Parallel Algorithms and Architectures* (1998). (To appear).

[15] 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.

[16] GERBESSIOTIS, A. V., AND VALIANT, L. G. Direct Bulk-Synchronous Parallel Algorithms. *Journal of Parallel and Distributed Computing* (1994).

[17] 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.

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

[19] MEHLHORN, K. Personal communication, Dagstuhl seminar on data structures, 1998.

[20] 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.

[21] SIBEYN, J., AND KAUFMANN, M. Bsp-like external-memory computation. In *Proc. 3rd Italian Conference on Algorithms and Complexity* (1997).

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

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

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

[25] 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.

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

[27] VITTER, J. S. Personal communication, 1998.

[28] VITTER, J. S., AND SHRIVER, E. A. M. Algorithms for parallel memory, I: Two-level memories. *Algorithmica 12*, 2–3 (1994), 110–147.

[29] VITTER, J. S., AND SHRIVER, E. A. M. Algorithms for parallel memory, II: Hierarchical multilevel memories. *Algorithmica 12*, 2–3 (1994), 148–169.