Cache-Oblivious Simulation of Parallel Programs

Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri

Department of Information Engineering, University of Padova, Italy
{capri,geppo,silvest1}@dei.unipd.it

Abstract

This paper explores the relation between the structured parallelism exposed by the Decomposable BSP (D-BSP) model through submachine locality and locality of reference in multi-level cache hierarchies. Specifically, an efficient cache-oblivious algorithm is developed to simulate D-BSP programs on the Ideal Cache Model (ICM). The effectiveness of the simulation is proved by showing that optimal cache-oblivious algorithms for prominent problems can be obtained from D-BSP algorithms. Finally, a tight relation between optimality in the D-BSP and ICM models is established.

1. Introduction

The memory system of current microprocessors includes a hierarchical cascade of caches whose capacities and access times increase as they grow farther from the CPU and closer to main memory. In order to amortize the larger cost incurred when referencing data in distant levels of the hierarchy, the data is automatically replicated across the faster levels and transferred in blocks of contiguous locations. The rationale behind such a hierarchical organization is that the memory access costs of a computation can be reduced when the same data are frequently reused within a short time interval, and data stored at consecutive addresses are involved in consecutive operations, two properties known as *temporal* and *spatial locality of reference*, respectively.

In the last decade a number of computational models have been proposed to explicitly account for such a hierarchical nature of the memory system. The first attempts in this direction were the *Hierarchical Memory Model* (HMM) defined in [1], a random access machine where access to memory location x requires time $f(x)$, for a given nondecreasing function $f(x)$, and the *HMM with Block Transfer* model (BT) [2] which augments the HMM with the capability of moving memory blocks of arbitrary size at a reduced cost. Observe that both models reward temporal locality, while only BT rewards spatial locality. However, the HMM/BT algorithm designer is in complete charge of orchestrating data layout, which makes the models unsuitable to describe current caches, where data are automatically moved and replicated across the hierarchy.

A two-level memory organization is featured by the *External Memory* (EM) model [3, 18], which has been extensively used in the literature to develop I/O-efficient algorithms. This model is closer to actual caches, since it only features a fixed-size block transfer, however the programmer is still in charge of managing data placement and eviction in the fast memory level. Borrowing ideas from the EM model, only recently a faithful model of a cache-main memory system, the *Ideal Cache Model* (ICM), has been defined in [13, 14]. The ICM features automatic block transfer and eviction, and has been at the base of the vast literature on *cache-oblivious* algorithms, i.e., algorithms which run efficiently independently of the cache parameters (i.e., cache size and cache line size). Most importantly, cache-oblivious algorithms attaining minimal miss rate on the ICM can be shown to attain minimal miss rates at all levels of any multi-level cache hierarchy [13].

A number of works in the literature have explored the relation between (structured) parallelism and locality of reference. Earlier results [8, 10, 15] provided evidence that efficient EM algorithms for two-level hierarchies can be obtained by simulating parallel ones written for coarse-grained parallel models, such as BSP [17] or CGM [9], on the EM model. The main intuition behind these works is that the interleaving be-

This work was supported in part by the University of Padova under Grant CPDA033838, by MIUR of Italy under project "ALGONEXT", and by the EU/IST Project "AEOLUS".

1-4244-0054-6/06/$20.00 ©2006 IEEE

tween large local computation and bulk communication phases, which characterizes coarse-grained parallel algorithms, maps nicely on the two-level structure of the EM model. However, the flat parallelism offered by the above coarse-grained models seems unable to afford the finer exploitation of locality needed to achieve cache obliviousness.

A more general study on the relation between parallelism and locality of reference can be found in [12], where it is shown how a more structured form of parallelism, such as the one exhibited by the *Decomposable BSP (D-BSP)* (a clustered variant of BSP defined in [7]) can be simulated to yield efficient sequential algorithms on the HMM and BT models. The simulation strategy crucially relies on a nontrivial exploitation of the submachine locality exposed in D-BSP to guarantee that the resulting sequential computation exhibit a high degree of temporal and spatial locality.

The objective of this paper is to extend the above investigation by exploring the relation between structured parallelism and locality of reference in multi-level cache hierarchies, where the movement of data across the levels is outside the programmer's explicit control. The core technical result of the paper is an efficient algorithm that simulates any D-BSP program on the ICM model. The ICM simulation strategy makes sure that memory references trigger automatic data movements between cache and main memory analogous to those explicitly prescribed by the simulation algorithm developed in [12] for the HMM/BT models. The most important feature of the simulation is that the several degrees of submachine locality exposed by D-BSP algorithms are exploited to ensure that the resulting ICM computations adapt to the cache parameters without explicitly knowing them. Thus the simulation algorithm is cache oblivious, hence it can be efficiently ported onto any (multi-level) cache hierarchy.

We provide evidence that our approach is successful by showing that optimal cache-oblivious algorithms for prominent problems can be obtained by simulating D-BSP ones. Furthermore, we also exhibit a tighter relation between the D-BSP and ICM models by arguing that achieving an optimal ICM algorithm by simulating a D-BSP algorithm implies, under certain reasonable conditions, the optimality of the latter.

The rest of the paper is organized as follows. Section 2 defines our reference models. The simulation algorithm is described and analyzed in Section 3. In Section 4 we apply the simulation to two relevant case studies, namely matrix multiplication and DFT. Finally, Section 5 discusses the relation between optimality in the D-BSP and ICM models.

2. Machine Models

D-BSP. The *Decomposable Bulk Synchronous Parallel* (D-BSP) model was introduced in [6] to capture submachine locality in a structured way through a hierarchical decomposition, and was further investigated in [4, 5, 11]. Let N be a power of two. A D-BSP$(N, \mathbf{g})$ is a collection of N RAM processors $\{P_j : 0 \leq j < N\}$ communicating through a router whose bandwidth characteristics are captured by vector $\mathbf{g} = \{g_0, \ldots, g_{\log N-1}\}$. (Throughout the paper all logarithms are taken to the base 2.) Specifically, for $0 \leq i \leq \log N$, the N processors are partitioned into 2^i fixed, disjoint i-*clusters* $C_0^i, C_1^i \ldots C_{2^i-1}^i$ of $N/2^i$ processors each, where the processors of a cluster are capable of communicating among themselves independently of the other clusters. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine, in the sense that $C_j^{\log N} = \{P_j\}$, for $0 \leq j < N$, and $C_j^i = C_{2j}^{i+1} \cup C_{2j+1}^{i+1}$, for $0 \leq i < \log N$ and $0 \leq j < 2^i$.

A D-BSP program consists of a sequence of *labeled supersteps*. In an i-*superstep*, $0 \leq i < \log N$, each processor executes internal computation on locally held data and sends messages exclusively to processors within its i-cluster (an output and an input queue for message exchange are part of each processor's local memory). The superstep ends with a barrier, which synchronizes processors independently within each i-cluster. Messages are of constant size and messages sent in one superstep are available at the destinations only at the beginning of the following superstep. It is also reasonable to assume that any D-BSP program ends with a global synchronization, that is, a 0-superstep. In an i-superstep, if each processor spends at most w units of time performing local computation during one superstep, and sends/receives at most h messages (i.e., the communication pattern is an h-relation), then the cost of the i-superstep is given by $w + hg_i$. The running time of a D-BSP program is obtained as the sum of the running times of its constituent supersteps. Observe that g_i is an inverse measure of an i-cluster's per-processor bandwidth, hence we can safely assume that $g_i \geq g_j$ for $0 \leq i < j < \log N$.

ICM. The *Ideal Cache Model* (ICM(Z, L)) was introduced in [13] and consists of a sequential processor equipped with a (data) cache and a main memory of arbitrary size. The cache contains Z words organized into lines of L words each, and it is ideal in the sense that it is fully associative and uses the optimal off-line strategy for cache-line replacement. The model rewards both spatial and temporal locality of reference. As in [13], when analyzing ICM(Z, L) algorithms, we

will make the *tall cache* hypothesis $Z = \Omega(L^2)$. A miss is said to be a *cold miss*, if it is caused by a reference to a previously unreferenced memory block, or a *capacity miss* if the referenced memory block was previously evicted from cache.

An ICM(Z, L) algorithm is characterized by its *work complexity* (number of operations) $W(N, Z, L)$ and *cache complexity* (number of misses) $Q(N, Z, L)$, where N denotes the input size. An algorithm is called *cache oblivious* if its specification is independent of the two parameters Z and L, and *cache aware* otherwise. Cache-oblivious algorithms for matrix multiplication, discrete Fourier Transform and sorting are presented in [13].

3. The Simulation Algorithm

In this section we describe the algorithm to simulate a D-BSP$(N, \mathbf{g})$ program $\mathcal{P}$ on an ICM(Z, L). The algorithm is similar to the one developed in [12] for the simulation of D-BSP on the BT model, with a number of nontrivial modifications due to the fact that, unlike the BT model, memory access costs on the ICM do not depend on the absolute addresses being referenced but, rather, on the temporal sequence of references. For the sake of completeness, we describe the entire algorithm rather than highlighting the differences with the algorithm in [12].

Given a D-BSP program $\mathcal{P}$, we refer to the local memory used by a processor during the execution of $\mathcal{P}$ as the processor's *context*. Let μ be an upper bound to the size of any context. The ICM main memory is divided into N *blocks*, each of size $\Theta(\mu)$, where the i-th block stores the context of processor P_i, followed by an auxiliary free space of size $c\mu$, for a suitable constant $c > 1$, which is used for bookkeeping purposes. The simulation is divided into *rounds*, where a round simulates an i-superstep for a certain i-cluster and identifies the cluster involved in the following round. Let the supersteps of $\mathcal{P}$ be numbered consecutively and let i_s denote the label of the s-th superstep, for $s \geq 0$.

Consider a generic round that simulates the s-th superstep of $\mathcal{P}$ for an i_s-cluster C. The simulation is divided into two parts: the execution of local computations for each processor of C, and the data movements corresponding to the message distribution. The simulation of local computations is done recursively in the two $(i_s + 1)$-clusters contained in C, until $\log N$-clusters are reached. As in [12], message distribution is simulated through sorting as follows. The blocks containing the contexts of the processors in C, which are consecutive in main memory, are rearranged so to pack the actual contexts in the first $\mu N/2^{i_s}$ words, followed by the $c\mu N/2^{i_s}$ words of free space. Then, the contexts are partitioned into $\Theta(\mu N/2^{i_s})$ constant-sized elements which are tagged with keys in such a way that, after sorting, contexts are still ordered by processor number and all messages are brought to the end of the respective destination processors' contexts. Both tagging and sorting need auxiliary space that is extracted from the segment of free space created via packing. After sorting, the keys are removed and the initial layout, with each context followed by free space, is restored.

Finally, the cluster that will be simulated in the next round is identified by determining its first processor P. Specifically, if $i_{s+1} \geq i_s$ then P remains the first processor of C, and the next round will simulate the $(s + 1)$-st superstep for the first i_{s+1}-cluster contained in C. If instead $i_{s+1} < i_s$ we have two cases. Let $\hat{C}$ be the i_{s+1}-cluster containing C. If the s-th superstep has been simulated for all i_s-clusters contained in $\hat{C}$, then P will be the first processor of $\hat{C}$, otherwise P will be the first processor of the next i_s-cluster contained in $\hat{C}$ (hence sibling of C) for which the s-th superstep has not been simulated yet.

The pseudocode of the simulation algorithm is provided in the appendix at the end of the paper. We have:

Theorem 1. *The simulation algorithm is correct.*

Proof (sketch). The correctness of the algorithm can be proved by modifying the argument presented in [12], taking into account that no explicit context movements are now specified in the algorithm but are automatically induced by the caching mechanism. Full details are found in [16] and will be provided in the full version of this extended abstract. □

The simulation strategy outlined above exploits locality of reference by proceeding unevenly on the different D-BSP clusters. In particular, the same cluster could be simulated for several consecutive supersteps so to avoid repeated, expensive reloads of its processors' contexts in cache. We remark that by using a cache-oblivious sorting algorithm for implementing Line 5 of `Communicate(`C`)` (e.g., the algorithm by [13]), the simulation algorithm becomes cache-oblivious since it makes no use of parameters Z and L of the ICM model.

3.1 Complexity analysis

We analyze the work and cache complexities of the simulation algorithm described in the previous section. In the analysis, we assume that $Z = \Omega(\mu)$ so that, for every superstep, the simulation of the local computations of the D-BSP processors never incurs capacity misses. In this way we are able to directly relate the

cache complexity of the ICM simulation to the context switching overhead induced by the parallelism of the D-BSP program. In fact, this assumption on Z is often satisfied by fine-grained D-BSP algorithms exploiting maximum parallelism with respect to the input size.

Theorem 2. *Consider a D-BSP$(N, \boldsymbol{g})$ program $\mathcal{P}$ using contexts of size μ, and let τ be the aggregate time for local computations, summed over all processors and all supersteps. Let k_i be the number of i-supersteps in $\mathcal{P}$, $0 \leq i < \log N$. When $Z = \Omega(\mu)$, $\mathcal{P}$ can be simulated on an ICM(Z, L) with work and cache complexities:*

$$W(N, Z, L) = \Theta\left(\tau + \mu N \sum_{i=0}^{\log N - 1} k_i \log \frac{\mu N}{2^i}\right) \quad (1)$$

$$Q(N, Z, L) = \Theta\left(1 + \frac{\mu N}{L}\left(1 + \sum_{i=0}^{\lambda-1} k_i \frac{\log \frac{\mu N}{2^i}}{\log Z}\right)\right), \quad (2)$$

where $\lambda = \max\left\{0, \left\lceil \log \frac{\tilde{\mu} N}{Z} \right\rceil\right\}$, with $\tilde{\mu} = (c+1)\mu$ for a suitable constant $c \geq 1$.

Proof. Consider a round simulating the s-th superstep for an i_s-cluster C, and let B_C denote the segment of $N/2^{i_s}$ contiguous memory blocks associated with the processors of C. The work complexity of the round is obtained by adding the contributions of the call to `Compute(`C`)` and the call to `Communicate(`C`)`, since the other operations account only for $O(1)$ work. It is easy to see that the work complexity of `Compute(`C`)` is proportional to the sum of the local computations of all processors in C in the superstep being simulated. As for `Communicate(`C`)`, the required packing and tagging operations and the respective unpacking and key deletion can be easily performed in $O\left(\mu N/2^{i_s}\right)$ work, through a constant number of scans of B_C. By employing the algorithm proposed in [13], the sorting step requires additional $O\left((\mu N/2^{i_s}) \log(\mu N/2^{i_s})\right)$ work. Equation 1 follows by combining the above contributions and summing over all rounds.

For the analysis of the cache complexity, observe that the value λ defined in the statement of the theorem represents the index of the largest cluster $C^\star$ such that $B_{C^\star}$ fits in cache. Note that if $\lambda = 0$, then $C^\star$ is the entire D-BSP machine, hence the simulation incurs only $O\left(1 + \mu N/L\right)$ cold misses. Instead, if $\lambda > 0$, the cache complexity of a round depends on the size of the cluster being simulated in the round.

Consider again a round simulating the s-th superstep for an i_s-cluster C. If $i_s < \lambda$, B_C cannot be contained entirely in cache. In this case, the cache complexity of `Compute(`C`)` is given by $Q_{Comp}(i_s, N, Z, L)$, where Q_{Comp} obeys the following recurrence:

$$Q_{Comp}(j, N, Z, L) = \begin{cases} O(\frac{\mu N}{2^j L}) & \text{if } j \geq \lambda \\ 2Q_{Comp}(j+1, N, Z, L) + O(1) & \text{if } j < \lambda. \end{cases}$$

It is easily seen that $Q_{Comp}(i_s, N, Z, L) = O(\mu N/(2^{i_s} L))$. Also, procedure `Communicate(`$C$`)` requires $O\left(\mu N/(2^{i_s} L)\right)$ misses for packing, unpacking and tagging, and $O\left(\mu N \log(\mu N/2^{i_s})/(2^{i_s} L \log Z)\right)$ misses for sorting [13]. Therefore, in this case the cache complexity of the round is $O\left(\mu N \log(\mu N/2^{i_s})/(2^{i_s} L \log Z)\right)$.

Suppose now $i_s \geq \lambda$, and note that B_C can be entirely contained in cache, hence the only misses that may occur are those triggered by the initial accesses to B_C. If $s = 0$ then the (cold) misses incurred in the simulation of the 0-th superstep for all of the 2^{i_0} i_0-clusters is bounded from above by $O(1 + \mu N/L)$. Instead, if $s > 0$ we show that the misses, if any, can be neglected without affecting the asymptotic cache complexity of the entire simulation. We distinguish among the following three cases depending on the label i_{s-1} of the previous superstep.

Case I $(i_{s-1} < \lambda \leq i_s)$ In this case the misses incurred when accessing B_C, as well as those incurred in all rounds simulating the s-th superstep for the siblings of C contained in the same i_{s-1}-cluster $\hat{C}$, are amortized by the $\Omega\left(\mu N/(2^{i_{s-1}} L)\right)$ misses incurred in a previous round which simulated the $(s-1)$-th superstep for $\hat{C}$.

Case II $(\lambda \leq i_{s-1} \leq i_s)$ In this case no misses occur since B_C is contained in $B_{\hat{C}}$, with $\hat{C}$ the i_{s-1}-cluster that contains C, and $B_{\hat{C}}$ is still resident in cache where it was loaded in a previous round which simulated the $(s-1)$-th superstep for $\hat{C}$.

Case III $(\lambda \leq i_s \leq i_{s-1})$ In this case, no misses occur since prior to the round being considered, the $(s-1)$-th superstep has been simulated for all i_{s-1}-clusters contained in C, hence the blocks associated with these clusters (i.e., all blocks in B_C) are in cache when the simulation of the s-th superstep for C begins.

Equation 2 is obtained by summing the initial $O(1+\mu N/L)$ cold misses and the misses incurred by all rounds simulating supersteps with label smaller than λ. □

A number of remarks are in order. First, we observe that the complexity gain of our simulation strategy over the trivial approach, where supersteps are simulated one after the other for all processors, becomes

apparent in Equation 2, where the summation is truncated at index λ, which is a function of the cache size, rather than going up to $\log N - 1$. This is made possible by the fact that the simulation proceeds unevenly on different clusters, thus fully exploiting temporal locality when dealing with small clusters. In fact, the performance improvement could be substantial for D-BSP programs with high submachine locality, that is, confining most of the computation within i-clusters with $i \geq \lambda$.

Second, we note that the logarithmic terms occurring in the summations in both the work and cache complexities are introduced by the sorting employed for simulating communications. In general the cache complexity of the simulation cannot be improved since arbitrary communication patterns in D-BSP translate into permutations on the ICM model for which a matching superlinear lower bound is known [3]. In the next section we show that for D-BSP algorithms that use certain structured communication patterns, a faster simulation can be obtained, yielding more efficient (in fact, optimal) ICM algorithms.

Finally, recall that the ICM model assumes an ideal cache with optimal (yet practically unfeasible) replacement policy. In many cases, however, the cache complexity on the more realistic cache model with the LRU replacement policy remains unaffected asymptotically as long as the algorithm satisfies a certain condition. More precisely, an ICM algorithm is called *regular* if its (ideal) cache complexity satisfies $Q(N, Z, L) = O(Q(N, 2Z, L))$. In [13] it is proved that the cache complexity of a regular algorithm remains asymptotically unaffected under the LRU replacement policy.

We define a corresponding regularity condition in the parallel setting. We say that a D-BSP program $\mathcal{P}$ is *p-regular* if $k_0 = O(1)$ and $k_i = O\left(\sum_{j=0}^{i-1} k_j\right)$, $0 < i < \log N$, where k_i is the number of i-supersteps in $\mathcal{P}$. We have:

Theorem 3. *Let $\mathcal{P}$ be a p-regular program for a D-BSP$(N, \boldsymbol{g})$ with contexts of size μ. Then, its simulation satisfies the regularity condition on an ICM(Z, L) with $Z = \Omega(\mu)$, hence its asymptotic cache complexity remains unchanged under the LRU replacement policy.*

Proof. If the cache size doubles, the value of λ defined in Theorem 2 decreases by at most one unit. By the p-regularity of $\mathcal{P}$, we have:

$$
\begin{aligned}
&N \sum_{i=0}^{\lambda-1} k_i \frac{\mu \log \frac{\mu N}{2^i}}{L \log Z} = \\
&\quad O\left(N \sum_{i=0}^{\lambda-2} k_i \frac{\mu \log \frac{\mu N}{2^i}}{L \log Z} + N k_{\lambda-1} \frac{\mu \log \frac{\mu N}{2^{\lambda-1}}}{L \log Z}\right) = \\
&\quad O\left(N \sum_{i=0}^{\lambda-2} k_i \frac{\mu \log \frac{\mu N}{2^i}}{L \log Z} + N \sum_{i=0}^{\lambda-2} k_i \frac{\mu \log \frac{\mu N}{2^{\lambda-1}}}{L \log Z}\right) = \\
&O\left(N \sum_{i=0}^{\lambda-2} k_i \frac{\mu \log \frac{\mu N}{2^i}}{L \log Z}\right), \qquad (3)
\end{aligned}
$$

and the theorem follows. □

4. Application to case-study problems

We apply the simulation presented in the previous section to two D-BSP algorithms for the prominent problems of Matrix Multiplication (MM) (limited to semiring operations) and Discrete Fourier Transform (DFT). The D-BSP algorithms, which are briefly outlined below, are those used in [12] to obtain efficient HMM and BT counterparts. (For simplicity, in the algorithms we assume that all relevant quantities are integral. Easy modifications are sufficient to handle the general case without affecting the asymptotic complexities.)

MM We consider the natural recursive algorithm for multiplying two $\sqrt{N} \times \sqrt{N}$ matrices on a D-BSP$(N, \mathbf{g})$. Initially, the N elements of each matrix are evenly distributed among the N processors. Then, by subdividing each input matrix into four quadrants, the input instance is decomposed into eight MM subproblems of size $\sqrt{N}/2 \times \sqrt{N}/2$ solved in two phases, where in each phase four subproblems are solved recursively within the four distinct 2-clusters. To keep space requirements at a minimum, the subproblems are partitioned among the two phases in such a way that each submatrix is required exactly once in each phase. Before each phase, elements are suitably redistributed between the clusters through a 0-superstep. It is easy to see that the number of i-supersteps is $k_i = 0$, if i is odd, and $k_i = \Theta(2^{i/2})$, if i is even, and that the aggregate time for local computations is $\tau = \Theta(N^{3/2})$. Note that this algorithm is p-regular and needs contexts of size $O(1)$. By Theorems 2 and 3, its simulation yields a regular cache-oblivious algorithm on the ICM(Z, L), exhibiting $W(N, Z, L) = O\left(N^{3/2}\right)$ and $Q(N, Z, L) = O\left(1 + N/L + N^{3/2}/(L\sqrt{Z})\right)$ work and

cache complexities, which match the respective optimal complexities of the cache-oblivious matrix multiplication algorithm given in [13].

DFT The DFT of an N-element vector can be computed on a D-BSP$(N, \mathbf{g})$ by recursively decomposing the N-input FFT dag into 2 layers of $\sqrt{N}$ independent $\sqrt{N}$-input FFT subdags, which are computed recursively in two phases by the $\sqrt{N}$ $((\log N)/2)$-clusters, one layer per phase. The outputs of the first layer become the inputs of the second layer where the correspondence is realized through a matrix-transpose permutation executed as a 0-superstep. It is easy to see that the number of i-supersteps executed by the algorithm is $k_i = \Theta(2^j)$, when $i = ((1 - 1/2^j) \log N)$ (for every $0 \leq j \leq \log \log N$), and $k_i = 0$ otherwise. Moreover, the aggregate time for local computations is $\tau = \Theta(N \log N)$ and, as before, the algorithm is p-regular and needs contexts of size $O(1)$. By Theorems 2 and 3, its simulation yields a regular cache-oblivious algorithm on the ICM(Z, L), exhibiting $W(N, Z, L) = O(N \log N \log \log N)$ and $Q(N, Z, L) = O(1 + N/L + N \log N \log(\log N / \log Z)/(L \log Z))$ work and cache complexities, respectively. These complexities are only a doubly logarithmic factor away from the optimal complexities achieved by the cache-oblivious algorithm presented in [13].

It can be noted that the nonoptimality of the ICM algorithm for DFT obtained through the simulation of the D-BSP algorithm is mainly caused by the slowdown introduced by the recourse to sorting for simulating communications. As already observed in the previous section, although the complexity of sorting cannot be avoided in the case of arbitrary communication patterns, more efficient approaches can be employed when the patterns exhibit some regularity. This is indeed the case of the D-BSP algorithm for DFT outlined above. Each i-superstep executed by the algorithm executes a matrix-transpose permutation within each i-cluster. By using the regular cache-oblivious matrix-transposition algorithm proposed in [13] in place of sorting, the work and cache complexities of a round simulating an i-superstep for an i-cluster become $O(\mu N/2^i)$ and $O(1 + \mu N/2^i L)$, respectively, hence the entire simulation yields a regular cache-oblivious algorithm for DFT on the ICM(Z, L), exhibiting $W(N, Z, L) = O(N \log N)$ and $Q(N, Z, L) = O(1 + N/L + N \log N/(L \log Z))$ work and cache complexities, respectively, which are optimal.

In general, consider a D-BSP program $\mathcal{P}$ and suppose that for every i-superstep executed by the program there exists an ICM procedure that can simulate the communication pattern required by the superstep within each i-cluster with $O(\mu N/2^i)$ work and $O(1 + \mu N/2^i L)$ misses, using $O(\mu N/2^i)$ auxiliary space. Then, by replacing sorting with these procedures, the simulation of $\mathcal{P}$ on the ICM(Z, L) attains the following work and cache complexities

$$
\begin{aligned}
Q(N, Z, L) &= O\left(1 + \frac{\mu N}{L} + N \frac{\mu}{L} \sum_{i=0}^{\lambda-1} k_i\right) \\
W(N, Z, L) &= O\left(\tau + \mu N \sum_{i=0}^{\log N - 1} k_i\right),
\end{aligned}
$$

where τ is the aggregate time for local computations and $\lambda = \max\left\{0, \left\lceil \log \frac{\tilde{\mu} N}{Z} \right\rceil\right\}$, with $\tilde{\mu} = \Theta(\mu)$. We refer to this type of simulations as *ad-hoc simulations*.

5. Discussion

The case studies analyzed in the previous section provide evidence that efficient parallel algorithmic strategies can be transformed automatically into efficient cache-oblivious ones. In this section, we show a tighter coupling between optimality in the parallel and memory hierarchy scenarios. To this purpose, we need to exploit the following important feature of cache-oblivious algorithms. Consider an extension of the ICM model consisting of a processor and k memory levels. In [13] it is proved that a regular cache-oblivious algorithm exhibiting optimal work W and cache complexity Q on the two-level ICM, also exhibits optimal work W and optimal cache complexity Q_i at each cache level i, with $0 \leq i < k - 1$, under certain reasonable assumptions on the relative values of the parameters of adjacent cache levels and under the LRU replacement policy.

We say that a D-BSP$(N, \mathbf{g})$ program which uses contexts of size μ is a *full program* if the communication required in every superstep is a $\Theta(\mu)$-relation. We have:

Theorem 4. *Let $\mathcal{P}$ be a full p-regular D-BSP$(N, \boldsymbol{g})$ program using contexts of size μ, with $g_i = \Omega(\log(\mu N/2^i))$, $0 \leq i < \log N$. If the simulation of $\mathcal{P}$ is optimal for the two-level ICM, then $\mathcal{P}$ exhibits optimal time complexity among all full p-regular D-BSP$(N, \boldsymbol{g})$ programs for the same problem which use contexts of size $\Theta(\mu)$.*

Proof (sketch). Let $T(\mathcal{P}, N, \mathbf{g})$ be the running time of program $\mathcal{P}$ on a D-BSP$(N, \mathbf{g})$. For the sake of contradiction, assume that there exists a full p-regular program for the same problem $\mathcal{P}'$ using contexts of size $\Theta(\mu)$, such that $T(\mathcal{P}', N, \mathbf{g}) = o(T(\mathcal{P}, N, \mathbf{g}))$. Consider an extended ICM with $\log N$ cache levels, where the

level-i cache has size $Z_i = \Theta(2^i\mu)$ and cache line size $L_i = O(1)$, for $0 \leq i < \log N$. Let $W(\mathcal{P})$, $Q_i(\mathcal{P})$ and $W(\mathcal{P}')$, $Q_i(\mathcal{P}')$ be the work and cache complexities attained by the simulations of $\mathcal{P}$ and $\mathcal{P}'$ in such a model, respectively. From the results in [13] it follows that the Q_i's can be obtained from Theorem 2 by plugging in the values Z_i and L_i in Equation 2. It is easy to determine nonnegative coefficients $t_0, t_1, \ldots, t_{\log N-1}$ such that

$$\begin{aligned} W(\mathcal{P}) + \sum_{i=0}^{\log N-1} t_i Q_i(\mathcal{P}) &= NT(\mathcal{P}, N, \mathbf{g}) \\ W(\mathcal{P}') + \sum_{i=0}^{\log N-1} t_i Q_i(\mathcal{P}') &= NT(\mathcal{P}', N, \mathbf{g}). \end{aligned}$$

Since $T(\mathcal{P}', N, \mathbf{g}) = o(T(\mathcal{P}, N, \mathbf{g}))$, we must have that either $W(\mathcal{P}') = o(W(\mathcal{P}))$ or $Q_i(\mathcal{P}') = o(Q_i(\mathcal{P}))$ for some index i, which is impossible since the simulation of $\mathcal{P}$ is cache oblivious and optimal in the two-level ICM, hence also optimal on any multi-level ICM [13]. □

An almost identical property can be proved for ad-hoc cache-oblivious simulations, with the only difference that in this case D-BSP optimality holds with no restrictions on the vector $\mathbf{g}$. As a consequence, the D-BSP MM and DFT algorithms described in Section 4 turn out to be optimal among all D-BSP$(N, \mathbf{g})$ algorithms for the corresponding problem using constant-size contexts (the reader can convince himself/herself that an ad-hoc cache-oblivious simulation for the MM algorithm exists).

References

[1] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proc. of the 19th ACM Symp. on Theory of Computing*, pages 305–314, 1987.

[2] A. Aggarwal, A. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. of the 28th IEEE Symp. on Foundations of Computer Science*, pages 204–216, 1987.

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

[4] G. Bilardi, C. Fantozzi, A. Pietracaprina, and G. Pucci. On the effectiveness of D-BSP as a bridging model of parallel computation. In *Proc. of the Int. Conference on Computational Science*, LNCS 2074, pages 579–588, 2001.

[5] G. Bilardi, A. Pietracaprina, and G. Pucci. A quantitative measure of portability with application to bandwidth-latency models for parallel computing. In *Proc. of EUROPAR 99*, LNCS 1685, pages 543–551, Sept. 1999.

[6] P. De la Torre and C. Kruskal. A structural theory of recursively decomposable parallel processor networks. In *Proc. of the 7th IEEE Symposium on Parallel and Distributed Processing*, pages 570–578, Oct. 1995.

[7] P. De la Torre and C. Kruskal. Submachine locality in the bulk synchronous setting. In *Proc. of EUROPAR 96*, LNCS 1124, pages 352–358, Aug. 1996.

[8] F. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *Proc. of 9th ACM Symp. on Parallel Algorithms and Architectures*, pages 106–115, 1997.

[9] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.

[10] F. Dehne, D. Hutchinson, D. Maheshwari, and W. Dittrich. Reducing I/O complexity by simulating coarse-grained parallel algorithms. In *Proc. of 13th International Parallel Processing Symposium*, pages 14–20, 1999.

[11] C. Fantozzi, A. Pietracaprina, and G. Pucci. A general PRAM simulation scheme for clustered machines. *Intl. Journal of Foundations of Computer Science*, 14(6):1147–1164, 2003.

[12] C. Fantozzi, A. Pietracaprina, and G. Pucci. Translating submachine locality into locality of reference. *Journal of Parallel and Distributed Computing*, 2006. In Print. Special issue on 18th IPDPS.

[13] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. of 40th IEEE Symp. on Foundations of Computer Science*, pages 285–298, 1999.

[14] S. Sen, S. Chatterjee, and N. Dumir. Towards a theory of cache-efficient algorithms. *Journal of the ACM*, 49(6):828–858, 2002.

[15] J. Sibeyn and M. Kaufmann. BSP-like external-memory computation. In *Proc. of 3rd CIAC*, LNCS 1203, pages 229–240, 1999.

[16] F. Silvestri. Simulazione di algoritmi paralleli per il modello D-BSP su una gerarchia di cache ideali, Oct. 2005. Laurea Thesis (in italian). University of Padova, Italy.

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

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

Appendix: Simulation Pseudocode

The pseudocode of the simulation algorithm is given below.

Algorithm Simulation($\mathcal{P}$)

$P \leftarrow 0$;
while *true* **do**
  $s \leftarrow$ index of next superstep to be simulated for P;
  $C \leftarrow i_s$-cluster of P;
  `Compute`(C);
  `Communicate`(C);
  if P *ended* **then** `Exit()`;
  if $i_{s+1} < i_s$ **then**
    Let $\hat{C}$ be the i_{s+1}-cluster containing C and let $\hat{C}_0 \ldots \hat{C}_{2^{i_s - i_{s+1}} - 1}$ be its component i_s-clusters, with $C = \hat{C}_j$ for some index j;
    if $j = i_s - i_{s+1} + 1$ **then**
      $P \leftarrow P + \frac{N}{2^{i_s}} - \frac{N}{2^{i_{s+1}}}$ {smallest index of a processor of $\hat{C}$};
    else
      $P \leftarrow P + \frac{N}{2^{i_s}}$ {smallest index of a processor of $\hat{C}_{j+1}$};
    end
  end
end

Procedure `Compute`(C)

$i \leftarrow$ label of C;
if $i = \log N$ **then**
  Simulate the local computation of C's unique component processor;
else
  Let $C = \hat{C}_0 \hat{C}_1$, where the $\hat{C}_i$'s are $(i+1)$-clusters;
  `Compute`($\hat{C}_0$);
  `Compute`($\hat{C}_1$);
end

Procedure `Communicate`(C)

$i \leftarrow$ label of C;
$B \leftarrow$ the segment of $N/2^i$ blocks associated with the processors of C;
Pack the $N/2^i$ contexts of C in the first $\mu N/2^i$ words of B;
Break the contexts into $O(1)$-word elements;
Sort the elements tagged with suitable keys so to bring messages to the end of their destination processors' contexts;
Remove the keys and unpack the contexts restoring B's initial layout;