Space-Round Tradeoffs for MapReduce Computations

Andrea Pietracaprina
Dip. di Ingegneria dell'Informazione
Università di Padova
Padova, Italy
capri@dei.unipd.it

Geppino Pucci
Dip. di Ingegneria dell'Informazione
Università di Padova
Padova, Italy
geppo@dei.unipd.it

Matteo Riondato
Dept. of Computer Science
Brown University
Providence, RI USA
matteo@cs.brown.edu

Francesco Silvestri
Dip. di Ingegneria dell'Informazione
Università di Padova
Padova, Italy
silvest1@dei.unipd.it

Eli Upfal
Dept. of Computer Science
Brown University
Providence, RI USA
eli@cs.brown.edu

ABSTRACT

This work explores fundamental modeling and algorithmic issues arising in the well-established MapReduce framework. First, we formally specify a computational model for MapReduce which captures the functional flavor of the paradigm by allowing for a flexible use of parallelism. Indeed, the model diverges from a traditional processor-centric view by featuring parameters which embody only global and local memory constraints, thus favoring a more data-centric view. Second, we apply the model to the fundamental computation task of matrix multiplication presenting upper and lower bounds for both dense and sparse matrix multiplication, which highlight interesting tradeoffs between space and round complexity. Finally, building on the matrix multiplication results, we derive further space-round tradeoffs on matrix inversion and matching.

Categories and Subject Descriptors

C.1.4 [**Parallel Architectures**]: Distributed architectures; F.1.2 [**Modes of Computation**]: Parallelism and concurrency; F.1.3 [**Complexity Measures and Classes**]: Relations among complexity measures; F.2.1 [**Numerical Algorithms and Problems**]: Computations on matrices

General Terms

Algorithms, Design, Theory

Keywords

MapReduce, tradeoff, sparse and dense matrix multiplication, matrix inversion, matching

ICS'12, June 25–29, 2012, San Servolo Island, Venice, Italy.

1. INTRODUCTION

In recent years, MapReduce has emerged as a computational paradigm for processing large-scale data sets in a series of rounds executed on conglomerates of commodity servers [6], and has been widely adopted by a number of large Web companies (e.g., Google, Yahoo!, Amazon) and in several other applications (e.g., GPU and multicore processing). (See [20] and references therein.)

Informally, a MapReduce computation transforms an input set of key-value pairs into an output set of key-value pairs in a number of *rounds*, where in each round each pair is first individually transformed into a (possibly empty) set of new pairs (*map step*) and then all values associated with the same key are processed, separately for each key, by an instance of the same reduce function (simply called *reducer* in the rest of the paper) thus producing the next new set of key-value pairs (*reduce step*). In fact, as already noticed in [19], a reduce step can clearly embed the subsequent map step so that a MapReduce computation can be simply seen as a sequence of rounds of (augmented) reduce steps.

The MapReduce paradigm has a functional flavor, in that it merely requires that the algorithm designer decomposes the computation into rounds and, within each round, into independent tasks through the use of keys. This enables parallelism without forcing an algorithm to cater for the explicit allocation of processing resources. Nevertheless, the paradigm implicitly posits the existence of an underlying unstructured and possibly heterogeneous parallel infrastructure, where the computation is eventually run. While mostly ignoring the details of such an underlying infrastructure, existing formalizations of the MapReduce paradigm constrain the computations to abide with some local and aggregate memory limitations.

In this paper, we look at both modeling and algorithmic issues related to the MapReduce paradigm. We first provide a formal specification of the model, aimed at overcoming some limitations of the previous modeling efforts, and then derive interesting tradeoffs between memory constraints and round complexity for the fundamental problem of matrix multiplication and some of its applications.

1.1 Previous work

The MapReduce paradigm has been introduced in [6] without a fully-specified formal computational model for algorithm design and analysis. Triggered by the quickly gained popularity of the paradigm, a number of subsequent works have dealt more rigorously with modeling and algorithmic issues [16, 9, 7].

In [16], a MapReduce algorithm specifies a sequence of rounds as described in the previous section. Somewhat arbitrarily, the authors impose that in each round the memory needed by any reducer to store and transform its input pairs has size $O\left(n^{1-\epsilon}\right)$, and that the aggregate memory used by all reducers has size $O\left(n^{2-2\epsilon}\right)$, where n denotes the input size and ϵ is a fixed constant in $(0, 1)$. The cost of local computation, that is, the work performed by the individual reducers, is not explicitly accounted for, but it is required to be polynomial in n. The authors also postulate, again somewhat arbitrarily, that the underlying parallel infrastructure consists of $\Theta\left(n^{1-\epsilon}\right)$ processing elements with $\Theta\left(n^{1-\epsilon}\right)$ local memory each, and hint at a possible way of supporting the computational model on such infrastructure, where the reduce instances are scheduled among the available machines so to distribute the aggregate memory in a balanced fashion. It has to be remarked that such a distribution may hide non negligible costs for very fine-grained computations (due to the need of allocating multiple reducer with different memory requirements to a fixed number of machines) when, in fact, the algorithmic techniques of [16] do not fully explore the larger power of the MapReduce model with respect to a model with fixed parallelism. In [19] the same model of [16] is adopted but when evaluating an algorithm the authors also consider the total work and introduce the notion of work-efficiency typical of the literature on parallel algorithms.

An alternative computational model for MapReduce is proposed in [9], featuring two parameters which describe bandwidth and latency characteristics of the underlying communication infrastructure, and an additional parameter that limits the amount of I/O performed by each reducer. Also, a BSP-like cost function is provided which combines the internal work of the reducers with the communication costs incurred by the shuffling of the data needed at each round. Unlike the model of [16], no limits are posed to the aggregate memory size. This implies that in principle there is no limit to the allowable parallelism while, however, the bandwidth/latency parameters must somewhat reflect the topology and, ultimately, the number of processing elements. Thus, the model mixes the functional flavor of MapReduce with the more descriptive nature of bandwidth-latency models such as BSP [29, 3].

A model which tries to merge the spirit of MapReduce with the features of data-streaming is the MUD model of [7], where the reducers receive their input key-value pairs as a stream to be processed in one pass using small working memory, namely polylogarithmic in the input size. A similar model has been adopted in [5].

MapReduce algorithms for a variety of problems have been developed on the aforementioned MapReduce variants including, among others, primitives such as prefix sums, sorting, random indexing [9], and graph problems such as triangle counting [28] minimum spanning tree, s-t connectivity, [16], maximal and approximate maximum matching, edge cover, minimum cut [19], and max cover [5]. Moreover simulations of the PRAM and BSP in MapReduce have been presented in [16, 9]. In particular, it is shown that a T-step EREW PRAM algorithm can be simulated by an $O(T)$-round MapReduce algorithm, where each reducer uses constant-size memory and the aggregate memory is proportional to the amount of shared memory required by the PRAM algorithm [16]. The simulation of CREW or CRCW PRAM algorithms incurs a further $O(\log_m(M/m))$ slowdown, where m denotes the local memory size available for each reducer and M the aggregate memory size [9].

All of the aforementioned algorithmic efforts have been aimed at achieving the minimum number of rounds, possibly constant, provided that enough local memory for the reducer (typically sublinear in the input size) and enough aggregate memory is available. However, so far, to the best of our knowledge, there has been no attempt to fully explore the tradeoffs that can be exhibited for specific computational problems between the local and aggregate memory sizes, on one side, and the number of rounds, on the other, under reasonable constraints of the amount of total work performed by the algorithm. Our results contribute to filling this gap.

Matrix multiplication is a building block for many problems, including matching [25], matrix inversion [15], all-pairs shortest path [15], graph contraction [8], cycle detection [30], and parsing context free languages [27]. Parallel algorithms for matrix multiplication of dense matrices have been widely studied: among others, we note [14, 22] which provide upper and lower bounds exposing a tradeoff between communication complexity and processor memory. For sparse matrices, interesting results are given in [23, 21] for some network topologies like hypercubes, in [18] for PRAM, and in [4] for a BSP-like model. In particular, techniques in [22, 18] are used in the following sections for deriving efficient MapReduce algorithms. In the sequential settings, some interesting works providing upper and lower bounds are [13, 17] for dense matrix multiplication, and [12, 31, 11] for sparse matrix multiplication.

1.2 New results

The contribution of this paper is twofold, since it targets both modeling and algorithmic issues.

We first provide a more general and polished version of the MapReduce model of [16]. In particular, we generalize the model by letting the local and aggregate memory sizes be two independent parameters, m and M, respectively. Moreover, we enforce a clear separation between the model and underlying execution infrastructure: for instance, we do not impose a bound on the number of available machines, thus fully decoupling the degree of parallelism exposed by a computation from the one of the machine where the computation will be eventually executed. This decoupling greatly simplifies algorithm design, which has been one of the original objectives of the MapReduce paradigm. (In Section 2, we quantify the cost of implementing a round of our model on a system with fixed parallelism.)

Our algorithmic contributions concern the study of attainable tradeoffs in MapReduce for several variants of the fundamental primitive of matrix multiplication. In particular, we develop deterministic upper and lower bounds for dense-dense, sparse-dense and sparse-sparse matrix multiplication, and a more efficient randomized upper bound for the sparse-sparse case. As a by-product of this latter result, we also obtain a randomized procedure to approximate

the number of nonzero entries in the output matrix. Our algorithms are parametric in the m and M, and achieve optimal or quasi-optimal round complexity in the entire range of variability of these parameters. Finally, building on the matrix multiplication results, we derive space-round tradeoffs for matrix inversion and matching, which are important by-products of matrix multiplication.

To the best of our knowledge, no previous work in the MapReduce literature has explicitly addressed the above fundamental problems, and the round complexity of our MapReduce algorithms for these problems exhibit a logarithmic improvement upon what could be obtained by simulating the best known PRAM algorithms using the techniques by [16]. Moreover, we show that, for suitable values of the memory parameters, constant number of rounds (the holy grail of algorithmic research in MapReduce) are achievable for both dense and sparse matrix multiplication.

1.3 Organization of the paper

The rest of the paper is structured as follows. In Section 2 we introduce our computational model for MapReduce and describe important algorithmic primitives (sorting and prefix sums) that we use in our algorithms. Section 3 deals with matrix multiplication in our model, presenting theoretical bounds to the complexity of algorithms to solve this problem. We apply these results in Section 4 to derive algorithms for matrix inversion and for matching in graphs.

2. MODEL DEFINITION AND BASIC PRIMITIVES

Our model is defined in terms of two integral parameters m and M, whose meaning will be explained below, and is named MR(m, M). Algorithms specified in this model will be referred to as *MR-algorithms*. An MR-algorithm specifies a sequence of *rounds*: the r-th round, with $r \geq 1$ transforms a multiset W_r of key-value pairs into two multisets W_{r+1} and O_r of key-value pairs, where W_{r+1} is the input of the next round (empty, if r is the last round), and O_r is a (possibly empty) subset of the final output. The input of the algorithm is represented by W_1 while the output is represented by $\cup_{r\geq 1} O_r$, with $\cup$ denoting the union of multisets. The universes of keys and values may vary at each round, and we let U_r denote the universe of keys of W_r. The computation performed by Round r is defined by a *reducer* function ρ_r which is applied independently to each multiset $W_{r,k} \subseteq W_r$ consisting of all entries in W_r with key $k \in U_r$.

Let n be the input size. The two parameters m and M specify the memory requirements that each round of an MR-algorithm must satisfy. In particular, let $m_{r,k}$ denote the space needed to compute $\rho_r(W_{r,k})$ on a RAM, including the space taken by the input (i.e., $m_{r,k} \geq |W_{r,k}|$) and the work space, but excluding the space taken by the output, which contributes either to O_r (i.e., the final output) or to W_{r+1}. The model imposes that $m_{r,k} \in O(m)$, for every $r \geq 1$ and $k \in U_r$, that $\sum_{k\in U_r} m_{r,k} \in O(M)$, for every $r \geq 1$, and that $\sum_{r\geq 1} O_r = O(M)$. Note that the size of the output generated by a reducer is not limited by the local memory m. The complexity of an MR-algorithm is the number of rounds that it executes in the worst case, and it is expressed as a function of the input size n and of parameters m and M. The dependency on the parameters m and M allows for a finer analysis of the cost of an MR-algorithm.

As in [16], we require that each reducer function runs in time polynomial in n. In fact, it can be easily seen that the model defined in [16] is equivalent to the MR(m, M) model with $m \in O\left(n^{1-\epsilon}\right)$ and $M \in O\left(n^{2-2\epsilon}\right)$, for some fixed constant $\epsilon \in (0, 1)$, except that we eliminate the additional restrictions that the number of rounds of an algorithm be polylogarithmic in n and that the number of physical machines on which algorithms are executed be $\Theta\left(n^{1-\epsilon}\right)$, which in our opinion should not be imposed at the model level.

Compared to the model in [9], our MR(m, M) model introduces the parameter M to limit the size of the aggregate memory required at each round, whereas in [9] this size is virtually unbounded, and it imposes a less stringent bound on the output size of each reducer[1]. Moreover, the complexity analysis in MR(m, M) focuses on the tradeoffs between m and M, on one side, and the number of rounds on the other side, while in [9] a more complex cost function is defined which accounts for the overall message complexity of each round, the time complexity of each reducer computation, and the latency and bandwidth characteristics of the executing platform.

2.1 Sorting and prefix sum computations

Sorting and prefix sum primitives are used in the algorithms presented in this paper. The input to both primitives consists of a set of n key-value pairs (i, a_i) with $0 \leq i < n$ and $a_i \in S$, where S denotes a suitable set. For sorting, a total order is defined over S and the output is a set of n key-value pairs (i, b_i), where the b_i's form a permutation of the a_i's and $b_{i-1} \leq b_i$ for each $0 < i < n$. For prefix sums, a binary associative operation $\oplus$ is defined over S and the output consists of a collection of n pairs (i, b_i) where $b_i = a_0 \oplus \ldots \oplus a_i$, for $0 \leq i < n$.

By straightforwardly adapting the results in [9] to our model we have:

THEOREM 1. *The sorting and prefix sum primitives for inputs of size n can be performed in $O(\log_m n)$ rounds in MR(m, M) for any $M = \Omega(n)$.*

We remark that each reducer in the implementation of the sorting and prefix primitives makes use of $\Theta(m)$ memory words. Hence, the same round complexity can be achieved in a more restrictive scenario with fixed parallelism. In fact, our MR(m, M) model can be simulated on a platform with $\Theta(M/m)$ processing elements, each with internal memory of size $\Theta(m)$, at the additional cost of one prefix computation per round. Therefore, $O(\log_m n)$ can be regarded as an upper bound on the relative power of our model with respect to one with fixed parallelism.

In [9], the authors claim that the round complexities stated in Theorem 1 are optimal in their model, as a consequence of the lower bound for computing the OR of n bits on the BSP model [10]. It can be shown that the optimality carries through to our model for any m and any $M = \Omega(n)$.

3. MATRIX MULTIPLICATION

Let A and B be two $\sqrt{n} \times \sqrt{n}$ matrices and let $C = A \cdot B$. We use $a_{i,j}, b_{i,j}$ and $c_{i,j}$, with $0 \leq i, j < \sqrt{n}$, to denote the entries of A, B and C, respectively. In this section we present upper and lower bounds for computing the

[1]For clarity, we remark that in [9] M denotes the maximum amount of space used by a reducer, that is, the quantity denoted by m in our model.

product C in MR(m, M). All our algorithms envision the matrices as conceptually divided into submatrices of size $\sqrt{m} \times \sqrt{m}$, and we denote these submatrices with $A_{i,j}$, $B_{i,j}$ and $C_{i,j}$, respectively, for $0 \leq i, j < \sqrt{n/m}$. Clearly, $C_{i,j} = \sum_{h=0}^{\sqrt{n/m}-1} A_{i,h} \cdot B_{h,j}$.

All our algorithms exploit the following partition of the $(n/m)^{3/2}$ products between submatrices (e.g., $A_{i,h} \cdot B_{h,j}$) into $\sqrt{n/m}$ *groups*: group G_ℓ, with $0 \leq \ell < \sqrt{n/m}$, consists of products $A_{i,h} \cdot B_{h,j}$, for every $0 \leq i, j < \sqrt{n/m}$ and for $h = (i + j + \ell) \mod \sqrt{n/m}$. Observe that each submatrix of A and B occurs exactly once in each group G_ℓ, and that each product in G_ℓ contributes to a distinct submatrix of C.

We focus our attention on matrices whose entries belong to a semiring $(S, \oplus, \odot)$ such that for any $a \in S$ we have $a \odot 0 = 0$, where 0 is the identity for $\oplus$. In this setting, efficient matrix multiplication techniques such as Strassen's cannot be employed. Moreover, for the sake of the analysis, we make the reasonable assumption that the inner products of any row of A and of any column of B with overlapping nonzero entries never cancel to zero.

In our algorithms, any input matrix X ($X = A, B$) is provided as a set of key-value pairs $(k_{i,j}, (i, j, x_{i,j}))$ for all elements $x_{i,j} \neq 0$. Key $k_{i,j}$ represents a progressive index, e.g., the number of nonzero entries preceding $x_{i,j}$ in the row-major scan of X. We call a $\sqrt{n} \times \sqrt{n}$ matrix *dense* if the number of its nonzero entries is $\Theta(n)$, and we call it *sparse* otherwise. In what follows, we present different algorithms tailored for the multiplication of dense-dense (Section 3.1), sparse-sparse (Section 3.2), and sparse-dense matrices (Section 3.3). For suitable values of m and M the algorithms complete in a constant number of rounds. We also derive lower bounds which demonstrate that our deterministic algorithms are either optimal or close to optimal (Section 3.4), and an algorithm for estimating the number of nonzero entries in the product of two sparse matrices (Section 3.2.4).

3.1 Dense-Dense Matrix Multiplication

In this section we provide a simple, deterministic algorithm for multiplying two dense matrices, which will be proved optimal in Subsection 3.4. The algorithm is a straightforward adaptation of the well-established three-dimensional algorithmic strategy for matrix multiplication of [22, 14], however we describe a few details of its implementation in MR(m, M) since the strategy is also at the base of algorithms for sparse matrices. W.l.o.g. we may assume that $m \leq 2n$, since otherwise matrix multiplication can be executed by a trivial sequential algorithm. We consider matrices A and B as decomposed into $\sqrt{m} \times \sqrt{m}$ submatrices and subdivide the products between submatrices into groups as described above.

In each round, the algorithm computes all products within $K = \min\{M/n, \sqrt{n/m}\}$ consecutive groups: namely, at round $r \geq 1$, all multiplications in G_ℓ are computed, with $(r-1)K \leq \ell < rK$. The idea is that in a round all submatrices of A and B can be replicated K times and paired in such a way that each reducer performs a distinct multiplication in $\cup_{(r-1)K \leq \ell < rK} G_\ell$. Then, each reducer sums the newly computed product to a partial sum which accumulates all of the products contributing to the same submatrix of C belonging to groups with the same index modulo K dealt with in previous rounds. At the end of the $\sqrt{n}/(K\sqrt{m})$-th round, all submatrix products have been computed. The final matrix C is then obtained by adding together the K partial sums contributing to each entry of C through a prefix computation[2]. We have the following result.

THEOREM 2. *The above MR(m, M)-algorithm multiplies two $\sqrt{n} \times \sqrt{n}$ dense matrices in*

$$O\left(\frac{n^{3/2}}{M\sqrt{m}} + \log_m n\right)$$

rounds.

PROOF. The algorithm clearly complies with the memory constraints of MR(m, M) since each reducer multiplies two $\sqrt{m} \times \sqrt{m}$ submatrices and the degree of replication is such that the algorithm never exceeds the aggregate memory bound of M. Also, the $(n/m)^{3/2}$ products are computed in $n^{3/2}/(M\sqrt{m})$ rounds, while the final prefix computation requires $O(\log_m K + 1) = O(\log_m n)$ rounds. □

We remark that the multiplication of two $\sqrt{n} \times \sqrt{n}$ dense matrices can be performed in a constant number of rounds whenever $m = \Omega(n^\epsilon)$, for constant $\epsilon > 0$, and $M\sqrt{m} = \Omega\left(n^{3/2}\right)$.

3.2 Sparse-Sparse Matrix Multiplication

Consider two $\sqrt{n} \times \sqrt{n}$ sparse matrices A and B and denote with $\tilde{n} < n$ the maximum number of nonzero entries in any of the two matrices, and with $\tilde{o}$ the number of nonzero entries in the product $C = A \cdot B$. Below, we present two deterministic MR-algorithms (D1 and D2) and a randomized one (R1), each of which turns out to be more efficient than the others for suitable ranges of parameters. We consider only the case $m < 2\tilde{n}$, since otherwise matrix multiplication can be executed by a trivial one-round MR-algorithm using only one reducer. We also assume that the value $\tilde{n}$ is provided in input. (If this were not the case, such a value could be computed with a simple prefix computation in $O(\log_m n)$ rounds, which does not affect the asymptotic complexity of our algorithms.) However, we do not assume that $\tilde{o}$ is known in advance since, unlike $\tilde{n}$, this value cannot be easily computed. In fact, the only source of randomization in algorithm R1 stems from the need to estimate $\tilde{o}$.

3.2.1 Deterministic algorithm D1

This algorithm is based on the following strategy adapted from [18]. For $0 \leq i < \sqrt{n}$, let a_i (resp., b_i) be the number of nonzero entries in the ith column of A (resp., ith row of B), and let Γ_i be the set containing all nonzero entries in the ith column of A and in the ith row of B. It is easily seen that all of the $a_i b_i$ products between entries in Γ_i (one from A and one from B) must be computed. The algorithm performs a sequence of *phases* as follows. Suppose that at the beginning of Phase t, with $t \geq 0$, all products between entries in Γ_i, for each $i \leq r-1$ and for a suitable value r (initially, $r = 0$), have been computed and added to the appropriate entries of C. Through a prefix computation, Phase t computes the largest K_t such that $\sum_{j=r}^{r+K_t} a_j b_j \leq M$. Then, all products between entries in Γ_j, for every $r \leq j \leq r + K_t$, are computed using one reducer (with constant memory) for each such product.

[2]The details of the key assignments needed to perform the necessary data redistributions among reducers are tedious but straightforward, and will be provided in the full version of this abstract.

The products are then added to the appropriate entries of C using again a prefix computation.

THEOREM 3. *Algorithm D1 multiplies two sparse $\sqrt{n} \times \sqrt{n}$ matrices with at most $\tilde{n}$ nonzero entries each in*

$$O\left(\left\lceil \frac{\tilde{n}\min\{\tilde{n},\sqrt{n}\}}{M} \right\rceil \log_m M\right)$$

rounds, on MR(m, M).

PROOF. The correctness is trivial and the memory constraints imposed by the model are satisfied since in each phase at most M elementary products are performed. The theorem follows by observing that the maximum number of elementary products is $\tilde{n}\min\{\tilde{n},\sqrt{n}\}$ and that two consecutive phases compute at least M elementary products in $O(\log_m M)$ rounds. □

3.2.2 Deterministic algorithm D2

The algorithm exploits the same three-dimensional algorithmic strategy used in the dense-dense case and consists of a sequence of phases. In Phase t, $t \geq 0$, all $\sqrt{m} \times \sqrt{m}$-size products within K_t consecutive groups are performed in parallel, where K_t is a phase-specific value. Observe that the computation of all products within a group G_ℓ requires space $M_\ell \in [\tilde{n}, \tilde{n} + \tilde{o}]$, since each submatrix of A and B occurs only once in G_ℓ and each submatrix product contributes to a distinct submatrix of C. However, the value M_ℓ can be determined in $\Theta(\tilde{n})$ space and $O(\log_m n)$ rounds by "simulating" the execution of the products in G_ℓ (without producing the output values) and adding up the numbers of nonzero entries contributed by each product to the output matrix. The value K_t is determined as follows. Suppose that, at the beginning of Phase t, groups G_ℓ have been processed, for each $\ell \leq r-1$ and for a suitable value r (initially, $r = 0$). The algorithm replicates the input matrices $K'_t = \min\{M/\tilde{n}, \sqrt{n/m}\}$ times. Subsequently, through sorting and prefix computations the algorithm computes M_ℓ for each $r \leq \ell < r + K'_t$ and determines the largest $K_t \leq K'_t$ such that $\sum_{\ell=r}^{r+K_t} M_\ell \leq M$. Then, the actual products in G_ℓ, for each $r \leq \ell \leq r + K_t$ are executed and accumulated (again using a prefix computation) in the output matrix C. We have the following theorem.

THEOREM 4. *Algorithm D2 multiplies two sparse $\sqrt{n} \times \sqrt{n}$ matrices with at most $\tilde{n}$ nonzero entries each in*

$$O\left(\left\lceil \frac{(\tilde{n}+\tilde{o})\sqrt{n}}{M\sqrt{m}} \right\rceil \log_m M\right)$$

rounds on MR(m, M), where $\tilde{o}$ denotes the maximum number of nonzero entries in the output matrix.

PROOF. The correctness of the algorithm is trivial. Phase t requires a constant number of sorting and prefix computations to determine K_t and to add the partial contributions to the output matrix C. Each value M_ℓ is $O(\tilde{n} + \tilde{o})$ and the groups are $\sqrt{n/m}$, then $K_t = \Omega\left(\min\{M/(\tilde{n}+\tilde{o}), \sqrt{n/m}\}\right)$ and the theorem follows. □

We remark that the value $\tilde{o}$ appearing in the stated round complexity needs not be explicitly provided in input to the algorithm. We also observe that with respect to Algorithm D1, Algorithm D2 features a better exploitation of the local memories available to the individual reducers, which compute $\sqrt{m} \times \sqrt{m}$-size products rather than working at the granularity of the single entries.

By suitably combining Algorithms D1 and D2, we can get the following result.

COROLLARY 1. *There is a deterministic algorithm which multiplies two sparse $\sqrt{n} \times \sqrt{n}$ matrices with at most $\tilde{n}$ nonzero entries each in*

$$O\left(\left\lceil \frac{\min\{\tilde{n}^2, \tilde{n}\sqrt{n}, (\tilde{n}+\tilde{o})\sqrt{n/m}\}}{M} \right\rceil \log_m M\right)$$

rounds on MR(m, M), where $\tilde{o}$ denotes the maximum number of nonzero entries in the output matrix.

3.2.3 Randomized algorithm R1

Algorithm D2 requires $O(\log_m M)$ rounds in each Phase t for computing the number K_t of groups to be processed. However, if $\tilde{o}$ were known, we could avoid the computation of K_t and resort to the fixed-K strategy adopted in the dense-dense case, by processing $K = M/(\tilde{n}+\tilde{o})$ consecutive groups per round. This would yield an overall $O((\tilde{n}+\tilde{o})\sqrt{n}/(M\sqrt{m}) + \log_m M)$ round complexity, where the $\log_m M$ additive term accounts for the complexity of summing up, at the end, the K contributions to each entry of C. However, $\tilde{o}$ may not be known a priori. In this case, using the strategy described in Section 3.2.4 we can compute a value $\widehat{o}$ which is a 1/2-approximation to $\tilde{o}$ with probability at least $1 - 1/n$. (We say that $\widehat{o}$ ϵ-approximates $\tilde{o}$ if $|\tilde{o} - \widehat{o}| < \epsilon\tilde{o}$.) Hence, in the algorithm we can plug in $2\widehat{o}$ as an upper bound to $\tilde{o}$. By using the result of Theorem 6 with $\epsilon = 1/2$ and $\delta = 1/(2n)$, we have:

THEOREM 5. *Let $m = \Omega(\log^2 n)$. Algorithm R1 multiplies two sparse $\sqrt{n} \times \sqrt{n}$ matrices with at most $\tilde{n}$ nonzero entries in*

$$O\left(\frac{(\tilde{n}+\tilde{o})\sqrt{n}}{M\sqrt{m}} + \log_m M\right)$$

rounds on MR(m, M), with probability at least $1 - 1/n$.

By comparing the rounds complexities stated in Corollary 1 and Theorem 5, it is easily seen that the randomized algorithm R1 outperforms the deterministic strategies when $m \in (\Omega(\log^2 n), o(M^\epsilon))$, for any constant ϵ, $\tilde{n} \geq \sqrt{n/m}/\log_m M$, and $\tilde{o} \leq \tilde{n}\min\{\tilde{n}, \sqrt{m}\}\log_m M$. For a concrete example, R1 exhibits better performance when $\tilde{n} > \sqrt{n}$, $\tilde{o} = \Theta(\tilde{n})$, and m is polylogarithmic in M. Moreover, both the deterministic and randomized strategies can achieve a constant round complexity for suitable values of the memory parameters.

3.2.4 Evaluation of the number of nonzero entries

Observe that a $\sqrt{n}$-approximation to $\tilde{o}$ derives from the following simple argument. Let a_i and b_i be the number of nonzero entries in the ith column of A and in the ith row of B respectively, for each $0 \leq i < \sqrt{n}$. Then, $\tilde{o} \leq \sum_{i=0}^{\sqrt{n}-1} a_i b_i \leq \tilde{o}\sqrt{n}$. Evaluating the sum requires $O(1)$ sorting and prefix computations, hence a $\sqrt{n}$-approximation of $\tilde{o}$ can be computed in $O(\log_m \tilde{n})$ rounds. However, such an approximation is too weak for our purposes and we show below how to achieve a tighter approximation by adapting a strategy born in the realm of streaming algorithms.

Let $\epsilon > 0$ and $0 < \delta < 1$ be two arbitrary values. An ϵ-approximation to $\tilde{o}$ can be derived by adapting the algorithm of [2] for counting distinct elements in a stream $x_0 x_1 \ldots$, whose entries are in the domain $[n] = \{0, \ldots, n-1\}$. The algorithm of [2] makes use of a very compact data structure, customarily called *sketch* in the literature, which consists of $\Delta = \Theta(\log(1/\delta))$ lists, $L_1, L_2, \ldots, L_\Delta$. For $0 \leq w < \Delta$, L_w contains the $t = \Theta\left(\lceil 1/\epsilon^2 \rceil\right)$ distinct smallest values of the set $\{\phi_w(x_i) : i \geq 0\}$, where $\phi_w : [n] \to [n^3]$ is a hash function picked from a pairwise independent family. It is shown in [2] that the median of the values $tn^3/v_0, \ldots tn^3/v_{\Delta-1}$, where v_w denotes the tth smallest value in L_w, is an ϵ-approximation to the number of distinct elements in the stream, with probability at least $1-\delta$. In order to compute an ϵ-approximation of $\tilde{o}$ for a product $C = A \cdot B$ of $\sqrt{n} \times \sqrt{n}$ matrices, we can modify the algorithm as follows. Consider the stream of values in $[n]$ where each element of the stream corresponds to a distinct product $a_{i,h} b_{h,j} \neq 0$ and consists of the value $j + i\sqrt{n}$. Clearly, the number of distinct elements in this stream is exactly $\tilde{o}$. (A similar approach has been used in [1] in the realm of sparse boolean matrix products.) We now show how to implement this idea on $\mathrm{MR}(m, M)$.

The MR-algorithm is based on the crucial observation that if the stream of values defined above is partitioned into segments, the sketch for the entire stream can be obtained by combining the sketches computed for the individual segments. Specifically, two sketches are combined by merging each pair of lists with the same index and selecting the t smallest values in the merged list. The $\mathrm{MR}(m, M)$-algorithm consists of a number of phases, where each phase, except for the last one, produces set of M/m sketches, while the last phase combines the last batch of M/m sketches into the final sketch, and outputs the approximation to $\tilde{o}$.

We refer to the partition of the matrices into $\sqrt{m} \times \sqrt{m}$ submatrices and group the products of submatrices as done before. In Phase t, with $t \geq 1$, the algorithm processes the products in $K = \min\{M/\tilde{n}, \sqrt{n/m}\}$ consecutive groups, assigning each pair of submatrices in one of the K groups to a distinct reducer. A reducer receiving $A_{i,h}$ and $B_{h,j}$, each with at least a nonzero entry, either computes a sketch for the stream segment of the nonzero products between entries of $A_{i,h}$ and $B_{h,j}$, if the total number of nonzero entries of $A_{i,h}$ and $B_{h,j}$ exceeds the size of the sketch, namely $H = \Theta\left((1/\epsilon^2)\log(1/\delta)\right)$ words, or otherwise leaves the two submatrices untouched (observe that in neither case the actual product of the two submatrices is computed). In this latter case, we refer to the pair of (very sparse) submatrices as a *pseudosketch*. At this point, the sketches produced by the previous phase (if $t > 1$), together with the sketches and pseudosketches produced in the current phase are randomly assigned to M/m reducers. Each of these reducers can now produce a single sketch from its assigned pseudosketches (if any) and merge it with all other sketches that were assigned to it. In the last phase ($t = \sqrt{n/m}/K$) the M/m sketches are combined into the final one through a prefix computation, and the approximation to $\tilde{o}$ is computed.

THEOREM 6. *Let $m = \Omega\left((1/\epsilon^2)\log(1/\delta)\log(n/\delta)\right)$ and let $\epsilon > 0$ and $0 < \delta < 1$ be arbitrary values. Then, with probability at least $1 - 2\delta$, the above algorithm computes an ϵ-approximation to $\tilde{o}$ in*

$$O\left(\frac{\tilde{n}\sqrt{n}}{M\sqrt{m}} + \log_m M\right)$$

rounds, on $MR(m, M)$

PROOF. The correctness of the algorithm follows from the results of [2] and the above discussion. Recall that the value computed by the algorithm is an ϵ-approximation to $\tilde{o}$ with probability $1 - \delta$. As for the rounds complexity we observe that each phase, except for the last one, requires a constant number of rounds, while the last one involves a prefix computation thus requiring $O(\log_m M)$ rounds. We only have to make sure that in each phase the memory constraints are satisfied (with high probability). Note also that a sketch of size $H \leq m$ is generated either in the presence of a pair of submatrices $A_{i,h}$, $B_{h,j}$ containing at least H entries, or within one of the M/m reducers. By the choice of K, it is easy to see that in any case, the overall memory occupied by the sketches is $O(M)$. As for the constraint on local memories, a simple modification of the standard balls-into-bins argument [24] and the union bound suffices to show that with probability $1 - \delta$, in every phase when sketches and pseudosketches are assigned to M/m reducers, each reducer receives in $O\left(m + (1/\epsilon^2)\log(1/\delta)\log(n/\delta)\right) = O(m)$ words. The theorem follows. (More details will be provided in the full version of the paper.) □

3.3 Sparse-Dense matrix multiplication

Let A be a sparse $\sqrt{n} \times \sqrt{n}$ matrix with at most $\bar{n}$ nonzero entries and let B be a dense $\sqrt{n} \times \sqrt{n}$ matrix (the symmetric case, where A is dense and B sparse, is equivalent). The algorithm for dense-dense matrix multiplication does not exploit the sparsity of A and takes $O(n\sqrt{n}/(M\sqrt{m}) + \log_m n)$ rounds. Also, if we simply plug $\tilde{n} = n$ in the complexities of the three algorithms for the sparse-sparse case (where $\tilde{n}$ represented the maximum number of nonzero entries of A or B) we do not achieve a better round complexity. However, a careful analysis of algorithm D1 in the sparse-dense case reveals that its round complexity is $O(\lceil \bar{n}\sqrt{n}/M \rceil \log_m M)$. By using the fastest algorithm, depending on the relative values of the parameters, we obtain:

COROLLARY 2. *The multiplication on $MR(m, M)$ between a sparse $\sqrt{n} \times \sqrt{n}$ matrix with at most $\bar{n}$ nonzero entries and a dense $\sqrt{n} \times \sqrt{n}$ matrix requires a number of rounds which is the minimum between $O(\lceil \bar{n}\sqrt{n}/M \rceil \log_m M)$ and $O(n\sqrt{n}/(M\sqrt{m}) + \log_m n)$.*

The above sparse-dense strategy outperforms all previous algorithms for instance when $\bar{n} = o(n/(\sqrt{m}\log_m M))$.

3.4 Lower bounds

In this section we provide lower bounds for deterministic dense-dense and sparse-sparse matrix multiplication. We restrict our attention to algorithms performing all nonzero *elementary products* $a_{i,k} \cdot b_{k,j}$, which we refer to as *conventional sparse* algorithms, extending a similar terminology introduced in [14] for the dense case. Although this assumption limits the class of algorithms, ruling out Strassen-like techniques, the following lemma generalizes a result in [17] to prove that computing all nonzero elementary products is indeed necessary when entries of the input matrices are from the semiring $(\mathbb{N}, +, \cdot)$. (A similar lemma holds for the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$, where ∞ is the identity of the min operation, which is usually adopted for shortest path computations.)

LEMMA 1. *Consider an algorithm $\mathcal{A}$ which multiples two $\sqrt{n} \times \sqrt{n}$ matrices A and B with $\tilde{n}_A$ and $\tilde{n}_B$ nonzero entries,*

respectively, from the semiring $(\mathbb{N}, +, \cdot)$. *Then, for each* $\tilde{n}_A$ *and* $\tilde{n}_B$, *algorithm* $\mathcal{A}$ *must perform all the nonzero elementary products.*

PROOF. The result in [17] focuses on algorithms for multiplying $\sqrt{n} \times \sqrt{n}$ matrices which do not exploit sparsity. The proof in [17] identifies some *specific* pairs of $\sqrt{n} \times \sqrt{n}$ matrices so that any algorithm that does not compute all elementary products is not correct on at least one such pair. Our generalization identifies similar matrix pairs containing $\tilde{n}_A$ and $\tilde{n}_B$ nonzero entries so that any algorithm that does not compute all *nonzero* elementary products is not correct on at least one such pair. More details will be provided in the full version of the paper. □

The following theorem exhibits a tradeoff in the lower bound between the amount of local and aggregate memory and the round complexity of a conventional sparse matrix multiplication algorithm. The proof is similar to the one proposed in [14] for lower bounding the communication complexity of dense-dense matrix multiplication in a BSP-like model: however, differences arise since we focus on round complexity and our model does not assume the outdegree of a reducer to be bounded. In the proof of the theorem we use the following lemma which was proved using the red-blue pebbling game in [13] and then restated in [14] as follows.

LEMMA 2 ([14]). *Consider a parallel algorithm computing the product* $C = A \cdot B$, *where* A *and* B *are two arbitrary matrices. A processor that uses* N_A *entries of* A *and* N_B *entries of* B, *and computes elementary products for* N_C *entries of* C, *can compute at most* $(N_A N_B N_C)^{1/2}$ *elementary products.*

THEOREM 7. *Consider a conventional sparse* $MR(m, M)$*-algorithm* $\mathcal{A}$ *for multiplying two* $\sqrt{n} \times \sqrt{n}$ *matrices. Let* P *and* $\tilde{o}$ *denote the number of nonzero elementary products and the number of nonzero entries in the output matrix, respectively. Then, the round complexity of* $\mathcal{A}$ *is*

$$\Omega\left(\left\lceil \frac{P}{M\sqrt{m}} \right\rceil + \log_m \left(\frac{P}{\tilde{o}}\right)\right).$$

PROOF. Let $\mathcal{A}$ be an R-round $\mathrm{MR}(m, M)$-algorithm computing $C = A \cdot B$. We prove that $R = \Omega(P/(M\sqrt{m}))$. Consider the r-th round, with $1 \leq r \leq R$, and let k be an arbitrary key in U_r and $K_r = |U_r|$. We denote with $o_{r,k}$ the space taken by the output of $\rho_r(W_{r,k})$ which contributes either to O_r or to W_{r+1}, and with $m_{r,k}$ the space needed to compute $\rho_r(W_{r,k})$ including the input and working space but excluding the output. Clearly, $m_{r,k} \leq m$, $\sum_{k \in U_r} m_{r,k} \leq M$, and $\sum_{k \in U_r} o_{r,k} \leq \tilde{o} \leq M$.

Suppose $M/K_r \geq m$. By Lemma 2, the reducer ρ_r with input $W_{r,k}$ can compute at most $m\sqrt{o_{r,k}}$ elementary products, since $N_A, N_B \leq m$ and $N_C \leq o_{r,k}$, where N_A and N_B denote the entries of A and B used in $\rho_r(W_{r,k})$ and N_C the entries of C for which contributions are computed by $\rho_r(W_{r,k})$. Then, the number of terms computed in the r-th round is at most $\sum_{k \in U_r} m\sqrt{o_{r,k}} \leq m\sqrt{MK_r} \leq M\sqrt{m}$, since $K_r \leq M/m$ and the summation is maximized when $o_{r,k} = M/K_r$ for each $k \in U_r$.

Suppose now that $M/K_r < m$. Partition the keys in U_r into K'_r sets $S_0, \ldots S_{K'_r - 1}$ such that $m \leq \sum_{k \in S_j} m_{r,k} \leq 2m$ for each $0 \leq j < K'_r$ (the lower bound may be not satisfied for $j = K'_r - 1$). Clearly, $\lfloor M/2m \rfloor \leq K'_r \leq \lceil M/m \rceil$. By Lemma 2, the number of elementary products computed by all the reducers $\rho_r(W_{r,k})$ with keys in a set S_j is at most $\sum_{k \in S_j} (m_{r,k} m_{r,k} o_{r,k})^{1/2}$. Since $(xyz)^{1/2} + (x'y'z')^{1/2} \leq ((x + x')(y + y')(z + z'))^{1/2}$ for each non negative assignment of the x, y, z, x', y', z' variables and since $\sum_{k \in S_j} m_{r,k} \leq 2m$, it follows that at most $2m\sqrt{O_{r,j}}$ elementary products can be computed using keys in S_j, where $O_{r,j} = \sum_{k \in S_j} o_{r,k}$. Therefore, the number of elementary products computed in the r-th round is at most $\sum_{j=0}^{K'_r - 1} 2m\sqrt{O_{r,j}} \leq 2m\sqrt{MK'_r} \leq 2M\sqrt{2m}$, since $K'_r \leq \lceil M/m \rceil$ and the sum is maximized when $O_{r,j} = M/K'_r$ for each $0 \leq j < K'_r$.

Therefore, in each round $O(M\sqrt{m})$ nonzero elementary products can be computed, and then $R = \Omega(\lceil P/M\sqrt{m} \rceil)$. The second term of the lower bound follows since there is at least one entry of C given by the sum of $P/\tilde{o}$ nonzero elementary products. □

We now specialize the above lower bound for algorithms for generic dense-dense and sparse-sparse matrix multiplication.

COROLLARY 3. *Consider a conventional sparse* $MR(m, M)$*-algorithm* $\mathcal{A}$ *for multiplying two* $\sqrt{n} \times \sqrt{n}$ *matrices. For input matrices with at most* $\tilde{n}$ *nonzero entries each, the round complexity of* $\mathcal{A}$ *is*

$$\Omega\left(\left\lceil \frac{\tilde{n} \min\{\tilde{n}, \sqrt{n}\}}{M\sqrt{m}} \right\rceil + \log_m \tilde{n}\right).$$

PROOF. Consider two matrices with $\tilde{n}$ nonzero entries each. Depending on which of the two terms in the complexity dominates, we can distribute the nonzero entries in the input matrices so that $P = \tilde{n} \min\{\tilde{n}, \sqrt{n}\}$ or $P/\tilde{o} = \Omega(\tilde{n})$. □

All deterministic algorithms provided in this section comply with the hypotheses of the lower bound, hence the above corollary applies (even in the dense-dense case, by setting $\tilde{n} = n$). Thus, the algorithm for dense-dense matrix multiplication described in Section 3.1 is optimal for any value of the parameters. On the other hand, the deterministic algorithm D2 for sparse-sparse matrix multiplication given in Section 3.2.2 is optimal whenever $\tilde{n} \geq \sqrt{n}$, $\tilde{o} = O(\tilde{n})$ and m is polynomial in M.

4. APPLICATIONS

We now apply the algorithms presented in Section 3 to derive efficient algorithms for inverting a square matrix and for solving several variants of the matching problem in a graph. For specific values of m and M, these MR-algorithms complete in $O(\log n)$ rounds, whereas the corresponding PRAM algorithms [15, 25], simulated using the technique from [16, Theorem 7.1], take $O(\log^2 n)$ rounds.

As done in other parallel models (see e.g. [15]), we assume that each memory word is able to store any value that occurs in the computation. Detailed descriptions of the algorithms and proofs of the theorems will appear in the full version of the paper.

4.1 Inverting a lower triangular matrix

In this section we study the problem of inverting a lower triangular matrix A of size $\sqrt{n} \times \sqrt{n}$. We adapt the simple recursive algorithm which leverages on the easy formula for

inverting a 2×2 lower triangular matrix [15, Sect. 8.2]. We have

$$\begin{bmatrix} a & 0 \\ b & c \end{bmatrix}^{-1} = \begin{bmatrix} a^{-1} & 0 \\ -c^{-1}ba^{-1} & c^{-1} \end{bmatrix}. \quad (1)$$

Since Equation (1) holds even when a, b, c are matrices, it is possible to derive a simple recursive algorithm for inverting a lower triangular matrix. Such an algorithm is easily implemented on MR(m, M) by partitioning A into square matrices of size $\sqrt{m} \times \sqrt{m}$ and proceed from the bottom up. The following theorem formalizes the complexity of such an algorithm.

THEOREM 8. *The above recursive algorithm computes the inverse of a nonsingular lower triangular $\sqrt{n} \times \sqrt{n}$ matrix A in*

$$O\left(\frac{n^{3/2}}{M\sqrt{m}} + \frac{\log^2 n}{\log m}\right)$$

rounds on MR(m, M).

When $M\sqrt{m}$ is $\Omega\left(n^{3/2}\right)$ and $m = \Omega\left(n^{\epsilon}\right)$ for some constant ϵ, the complexity reduces to $O(\log n)$ rounds.

It is also possible to compute A^{-1} using the closed formula derived by unrolling a blocked forward substitution. In general, the closed formula contains an exponential number of terms. There are nonetheless special cases of matrices for which a large number of terms in the sum are zero and only a polynomial number of terms is left. This is, for instance, the case for triangular band matrices. A $\sqrt{n} \times \sqrt{n}$ lower triangular band matrix A with bandwidth $b < \sqrt{n}$ is a matrix such that for all entries $a_{i,j}$ with $|i-j| \geq b$ or $i < j$ we have $a_{i,j} = 0$. Note that the inverse of a triangular band matrix is triangular but not necessarily a triangular band matrix. It is possible to compute the inverse of these matrices in a constant number of rounds, as formalized by the following theorem.

THEOREM 9. *Let A be a $\sqrt{n} \times \sqrt{n}$ triangular band matrix with bandwidth $b = n^{\epsilon}$, for a constant $\epsilon \in (0, 1/2)$. Then, if $m = \Omega\left(n^{2\epsilon+\alpha}\right)$ for any constant $\alpha \in (0, 1-2\epsilon)$ and $M = \Omega\left(n^{3/2-\epsilon}\right)$, computing A^{-1} takes $O(1)$ rounds in MR(m, M).*

4.2 Inverting a general matrix

Building on the inversion algorithm for triangular matrices presented in the previous subsection, and on the dense-dense matrix multiplication algorithm, in this section we develop an MR(m, M)-algorithm to invert a general $\sqrt{n} \times \sqrt{n}$ matrix A. Let the trace $tr(A)$ of A be defined as $\sum_{i=0}^{n-1} a_{i,i}$, where $a_{i,i}$ denotes the entry of A on the i-th row and i-th column. The algorithm is based on the following known strategy (see e.g., [15, Sect. 8.8]).

1. Compute the powers $A^2, \ldots, A^{\sqrt{n}-1}$.
2. Compute $s_k = \sum_{i=1}^{\sqrt{n}} tr(A^k)$, for $1 \leq k \leq \sqrt{n} - 1$.
3. Compute the coefficients c_i of the characteristic polynomial of A by solving a lower triangular system of $\sqrt{n}$ linear equations involving the values s_k (the system is specified below).
4. Compute $A^{-1} = -(1/c_0) \sum_{i=1}^{\sqrt{n}} c_i A^{i-1}$.

We now provide more details on the MR implementation of above strategy. The algorithm requires $M = \Omega\left(n^{3/2}\right)$, which ensures that enough aggregate memory is available to store all the $\sqrt{n}$ powers of A. In Step 1, the algorithm computes naively the powers in the form A^{2^i}, $1 \leq i \leq \log\sqrt{n}$, by performing a sequence of $\log\sqrt{n}$ matrix multiplications using the algorithm in Section 3.1. Then, each one of the remaining powers is computed using $M/\sqrt{n} \geq n$ aggregate memory and by performing a sequence of at most $\log\sqrt{n}$ multiplications of the matrices A^{2^i} obtained earlier. In Step 2, the $\sqrt{n}$ values s_k are computed in parallel using a prefix-like computation, while the coefficients c_i of the characteristic polynomial are computed in Step 3 by solving the linear system $L \cdot C = -S$ (i.e., computing $C = -L^{-1}S$). L is a lower triangular matrix whose elements are $\ell_{i,j} = s_{i-j}$, for $0 \leq j < i < \sqrt{n}$, and $\ell_{i,i} = i+1$ for $0 \leq i < \sqrt{n}$, C is the vector of the unknown coefficient c_i, and S is the vector of the values s_k. In order to compute the coefficients in C the algorithm inverts the $\sqrt{n} \times \sqrt{n}$ lower triangular matrix L as described in Section 4.1, and computes the product between L^{-1} and S, to obtain C. Finally, Step 4 requires a prefix-like computation. We have the following theorem.

THEOREM 10. *The above algorithm computes the inverse of any nonsingular $\sqrt{n} \times \sqrt{n}$ matrix A in*

$$O\left(\frac{n^2 \log n}{M\sqrt{m}} + \frac{\log^2 n}{\log m}\right)$$

rounds on MR(m, M), with $M = \Omega\left(n^{3/2}\right)$.

If $M\sqrt{m}$ is $\Omega\left(n^2 \log n\right)$ and $m = \Omega\left(n^{\epsilon}\right)$ for some constant ϵ, the complexity reduces to $O(\log n)$ rounds.

4.3 Approximating the inverse of a matrix

The above algorithm for computing the inverse of any nonsingular matrix requires $M = \Omega\left(n^{3/2}\right)$. In this section we provide an MR(m, M)-algorithm providing a strong approximation of A^{-1} only assuming the natural constraint $M = \Omega(n)$. A matrix B is a *strong approximation* of the inverse of an $\sqrt{n} \times \sqrt{n}$ matrix A if $\|B - A^{-1}\|/\|A^{-1}\| \leq 2^{-n^c}$, for some constant $c > 0$. The norm $\|A\|$ of a matrix A is defined as

$$\|A\| = \max_{\mathbf{x} \neq 0} \|A\mathbf{x}\|_2 / \|\mathbf{x}\|_2$$

where $\|\cdot\|_2$ denotes the Euclidean norm of a vector. The condition number $\kappa(A)$ of a matrix A is defined as $\kappa(A) = \|A\|\|A^{-1}\|$.

An iterative method to compute a strong approximation of the inverse of a $\sqrt{n} \times \sqrt{n}$ matrix A is proposed in [15, Sect. 8.8.2]. The method works as follows. Let B_0 be a $\sqrt{n} \times \sqrt{n}$ matrix satisfying the condition $\|I_{\sqrt{n}} - B_0 A\| = q$ for some $0 < q < 1$ and where $I_{\sqrt{n}}$ is the $\sqrt{n} \times \sqrt{n}$ identity matrix. For a $\sqrt{n} \times \sqrt{n}$ matrix C let $r(C) = I_{\sqrt{n}} - CA$. We define $B_k = (I_{\sqrt{n}} + r(B_{k-1}))B_{k-1}$, for $k > 0$. We have

$$\frac{\|B_k - A^{-1}\|}{\|A^{-1}\|} \leq q^{2^k}.$$

By setting $B_0 = \alpha A^T$ where

$$\alpha = \max_i \{ \sum_{j=0}^{\sqrt{n}-1} |a_{i,j}| \} \max_j \{ \sum_{i=0}^{\sqrt{n}-1} |a_{i,j}| \},$$

we have $q = 1 - 1/(\kappa(A)^2 n)$ [26]. Then, if $\kappa(A) = O(n^c)$ for some constant $c \geq 0$, B_k provides a strong approximation when $k = \Theta(\log n)$. From the above discussion, it is easy to derive an efficient MR(m, M)-algorithm to compute a strong approximation of the inverse of a matrix using the algorithm for dense matrix multiplication in Section 3.1.

THEOREM 11. *The above algorithm provides a strong approximation of the inverse of any nonnegative $\sqrt{n} \times \sqrt{n}$ matrix A in*

$$O\left(\frac{n^{3/2}\log n}{M\sqrt{m}} + \frac{\log^2 n}{\log m}\right)$$

rounds on MR(m, M) when $\kappa(A) = O(n^c)$ for some constant $c \geq 0$.

If $M\sqrt{m}$ is $\Omega\left(n^{3/2}\log n\right)$ and $m = \Omega(n^\epsilon)$ for some constant ϵ, the complexity reduces to $O(\log n)$ rounds.

4.4 Matching of general graphs

Given a graph $G = (V, E)$, with $|V| = \sqrt{n}$ and $|E| = k$, a matching is a set of edges without common vertices. Matching problems are natural and ubiquitous in computer science. In this section we present an algorithm to compute, with high probability, a perfect matching of a general graph. A *perfect matching* is defined as a matching containing an edge for each vertex in V. The following strategy to compute a perfect matching with probability at least $1/2$ is presented in [25]:

1. Let the input of the algorithm be the adjacency matrix A of a graph $G = (V, E)$ with $\sqrt{n}$ vertices and k edges.
2. Let B be the matrix obtained from A by substituting the entries $a_{i,j} = a_{j,i} = 1$ corresponding to edges in the graph with the integers $2^{w_{i,j}}$ and $-2^{w_{i,j}}$ respectively, for $0 \leq i < j < \sqrt{n}$, where $w_{i,j}$ is an integer chosen independently and uniformly at random from $[1, 2k]$. We denote the entry on the ith row and jth column of B as $b_{i,j}$.
3. Compute the determinant $det(B)$ of B and the greatest integer w such that 2^w divides $det(B)$.
4. Compute $adj(B)$, the adjugate matrix of B, and denote the entry on the ith row and jth column as $adj(B)_{i,j}$.
5. For each edge $(v_i, v_j) \in E$, compute

$$z_{i,j} = \frac{b_{i,j} \cdot adj(B)_{i,j}}{2^w}.$$

   If $z_{i,j}$ is odd, then add the edge (v_i, v_j) to the matching.

An MR(m, M)-algorithm for perfect matching easily follows from the above strategy. To obtain B (Step 2), A is partitioned into square $\sqrt{m} \times \sqrt{m}$ submatrices $A_{\ell,h}$, $0 \leq \ell, h < \sqrt{n/m}$, and then each pair of submatrices $(A_{\ell,h}, A_{h,\ell})$ is assigned to a different reducer. This assignment ensures that each pair of entries $(a_{i,j}, a_{j,i})$ of A is sent to the same reducer. The reducer receiving the pair $(A_{\ell,h}, A_{h,\ell})$ (i.e., receiving the pairs $(a_{i,j}, a_{j,i})$, where $\ell\sqrt{m} \leq i < (\ell+1)\sqrt{m}$, $h\sqrt{m} \leq j < (h+1)\sqrt{m}$), chooses a $w_{i,j}$ independently and uniformly at random from $[1, 2k]$ for each pair $(a_{i,j}, a_{j,i})$ such that $a_{i,j} = a_{j,i} = 1$, and sets $b_{i,j}$ to $2^{w_{i,j}}$ and $b_{j,i}$ to $-2^{w_{i,j}}$. For all the other entries $a_{i,j} = a_{j,i} = 0$, the reducer sets $b_{i,j} = b_{j,i} = 0$. Let c_k, $0 \leq k \leq \sqrt{n}$ be the coefficients of the characteristic polynomial of B, which can be computed as described in Section 4.2. Using the fact that the determinant of B is c_0 and $adj(B) = -(c_1 I + c_2 B + c_3 B^2 + \cdots + c_{\sqrt{n}} B^{\sqrt{n}-1})$, it is easy to implement Steps 3 and 4. Finally (Step 5), matrices B and $adj(B)$ are partitioned in square submatrices of size $\sqrt{m} \times \sqrt{m}$, and corresponding submatrices assigned to the same reducer, which computes the values $z_{i,j}$ for the entries in its submatrices and outputs the edges belonging to the matching.

Executing this procedure $\Theta(\log n)$ times in parallel allows us to obtain a perfect matching with high probability, as formalized by the following theorem.

THEOREM 12. *The above algorithm computes, with high probability, a perfect matching of the vertices of a graph G, in*

$$O\left(\frac{n^2\log n}{M\sqrt{m}} + \frac{\log^2 n}{\log m}\right)$$

rounds on MR(m, M), where $M = \Omega\left(n^{3/2}\log n\right)$.

Our algorithm can be extended (as in [25]) to other variants of the matching problem like minimum weight perfect matching and maximum matching. The maximum matching problem in particular was already studied in the MapReduce framework in [19]. The authors of this work present an algorithm to obtain, with high probability, an 8-approximation to the maximum matching in a graph with $k = n^{(1-c)}$ for some constant c. Their algorithm runs in a constant (4) number of rounds if $m = n^{1/2+1/(3-6c)}$ and $M = k$. In comparison our algorithm runs in $O(\log n)$ rounds if $M\sqrt{m}$ is $\Omega\left(n^2 \log n\right)$ and $m = \Omega(n^\epsilon)$ for some constant ϵ, but computes an exact solution with high probability.

5. CONCLUSIONS

In this paper, we provided a formal specification of a computational model for the MapReduce paradigm which is parametric in the local and aggregate memory sizes and retains the functional flavor originally intended for the paradigm. Performance in the model is represented by the round complexity, which is consistent with the idea that when processing large data sets the dominant cost is the reshuffling of the data. The two memory parameters featured by the model allow the algorithm designer to explore a wide spectrum of tradeoffs between round complexity and memory availability. In the paper, we covered interesting such tradeoffs for the fundamental problem of matrix multiplication and some of its applications. The study of similar tradeoffs for other important applications (e.g., graph problems) constitutes an interesting open problem.

6. ACKNOWLEDGMENTS

The work of Pietracaprina, Pucci and Silvestri was supported, in part, by MIUR of Italy under project AlgoDEEP, by the University of Padova under the Strategic Project STPD08JA32 and Project CPDA099949/09, and by Amazon Web Services in Education grant. The work of Riondato and Upfal was supported, in part, by NSF award IIS-0905553 and by the University of Padova through the Visiting Scientist 2010/2011 grant.

7. REFERENCES

[1] R. Amossen, A. Campagna, and R. Pagh. Better size estimation for sparse matrix products. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 6302 of *Lecture Notes in Computer Science*, pages 406–419, 2010.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. of the 6th Intl. Workshop on Randomization and Approximation Techniques*, pages 1–10, 2002.

[3] G. Bilardi and A. Pietracaprina. Theoretical models of computation. In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 1150–1158. Springer, 2011.

[4] A. Buluç and J. R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *Proc. of 37th Intl. Conference on Parallel Processing*, pages 503–510, 2008. See also CoRR abs/1006.2183.

[5] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in Map-Reduce. In *Proc. of the 19th World Wide Web Conference*, pages 231–240, 2010.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4), 2010.

[8] J. Gilbert, V. Shah, and S. Reinhardt. A unified framework for numerical and combinatorial computing. *Computing in Science Engineering*, 10(2):20–25, 2008.

[9] M. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the MapReduce framework. In *Proc. of the 22nd International Symp. on Algorithms and Computation*, 2011. See also CoRR abs/1004.470.

[10] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[11] G. Greiner and R. Jacob. The I/O complexity of sparse matrix dense matrix multiplication. In *Proc. of 9th Latin American Theoretical Informatics*, volume 6034, pages 143–156, 2010.

[12] F. G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978.

[13] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th ACM Symp. on Theory of Computing*, pages 326–333, 1981.

[14] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[15] J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1992.

[16] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. of the 21st ACM-SIAM Symp. On Discrete Algorithms*, pages 938–948, 2010.

[17] L. R. Kerr. *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplication*. PhD thesis, Cornell University, 1970.

[18] C. P. Kruskal, L. Rudolph, and M. Snir. Techniques for parallel manipulation of sparse matrices. *Theoretical Computer Science*, 64(2):135–157, 1989.

[19] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proc. of the 23rd ACM Symp. on Parallel Algorithms and Architectures*, pages 85–94, 2011.

[20] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool, 2010.

[21] G. Manzini. Sparse matrix computations on the hypercube and related networks. *Journal of Parallel and Distributed Computing*, 21(2):169–183, 1994.

[22] W. F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24(3):287–297, 1999.

[23] M. Middendorf, H. Schmeck, H. Schröder, and G. Turner. Multiplication of matrices with different sparseness properties on dynamically reconfigurable meshes. *VLSI Design*, 9(1):69–81, 1999.

[24] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge MA, 2005.

[25] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[26] V. Pan and J. Reif. Efficient parallel solution of linear systems. In *Proc. of the 17th ACM Symp. on Theory of Computing*, pages 143–152, 1985.

[27] G. Penn. Efficient transitive closure of sparse matrices over closed semirings. *Theoretical Computer Science*, 354(1):72–81, 2006.

[28] C. Tsourakakis, U. Kang, G. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *Proc. of the 15th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, pages 837–849, 2009.

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

[30] R. Yuster and U. Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proc. of 15th ACM-SIAM Symp. On Discrete Algorithms*, pages 254–260, 2004.

[31] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005.