Low-Bandwidth Matrix Multiplication: Faster Algorithms and More General Forms of Sparsity

Chetan Gupta ⊠[©] IIT Roorkee, India

Janne H. Korhonen ⊠[®] ^{Finland}

Jan Studený $\square \square$ Aalto University, Finland

Jukka Suomela ⊠ [©] Aalto University, Finland

Hossein Vahidi ⊠ [□] Aalto University, Finland

— Abstract -

In prior work, Gupta et al. (SPAA 2022) presented a distributed algorithm for multiplying sparse $n \times n$ matrices, using *n* computers. They assumed that the input matrices are uniformly sparse—there are at most *d* non-zeros in each row and column—and the task is to compute a uniformly sparse part of the product matrix. Initially each computer knows one row of each input matrix, and eventually each computer needs to know one row of the product matrix. In each communication round each computer can send and receive one $O(\log n)$ -bit message. Their algorithm solves this task in $O(d^{1.907})$ rounds, while the trivial bound is $O(d^2)$.

We improve on the prior work in two dimensions: First, we show that we can solve the same task faster, in only $O(d^{1.832})$ rounds. Second, we explore what happens when matrices are not uniformly sparse. We consider the following alternative notions of sparsity: row-sparse matrices (at most d non-zeros per row), column-sparse matrices, matrices with bounded degeneracy (we can recursively delete a row or column with at most d non-zeros), average-sparse matrices (at most dn non-zeros in total), and general matrices.

We show that we can still compute X = AB in $O(d^{1.832})$ rounds even if one of the three matrices (A, B, or X) is average-sparse instead of uniformly sparse. We present algorithms that handle a much broader range of sparsity in $O(d^2 + \log n)$ rounds, and present conditional hardness results that put limits on further improvements and generalizations.

2012 ACM Subject Classification Theory of computation \rightarrow Distributed algorithms; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases distributed algorithms, low-bandwidth model, matrix multiplication

1 Introduction

In this work we study sparse matrix multiplication in a distributed setting, for the lowbandwidth model (which is closely related to the node-capacitated clique model). Our work improves over the prior work by [12] in two ways:

1. we present a faster algorithm, and

2. we study more general forms of sparsity.

1.1 Setting and prior work

The task is to compute the matrix product X = AB for $n \times n$ matrices using a network of n computers. Initially each computer holds its own part of A and B. Computation proceeds in rounds, and in one round each computer can send one $O(\log n)$ -bit message to another

computer and receive one message from another computer; we will assume that the elements of A, B, and X fit in one message. Eventually each computer has to report its own part of the product matrix X. How many communication rounds are needed to solve the task?

The trivial solution for *dense* matrices takes $O(n^2)$ rounds: everyone sends all information to computer number 1, which solves the task locally and then distributes the solution to other computers. However, we can do better: for matrix multiplication over semirings there is an algorithm that runs in $O(n^{4/3})$ rounds, and for matrix multiplication over fields there is an algorithm that runs in $O(n^{2-2/\omega})$ rounds, where ω is the exponent of centralized matrix multiplication [3]. By plugging in the latest value of $\omega < 2.371552$ [20], we obtain $O(n^{1.157})$ rounds.

The key question is how much better we can do when we multiply *sparse* matrices. Let us first look at the case of *uniformly sparse* matrices, with at most d nonzero elements in each row and column. We also assume that we are only interested in a uniformly sparse part of the product matrix X. Now it is natural to assume that each computer initially holds one row of A and one row of B, and it needs to know one row of X. We assume the supported setting: the sparsity structures of A, B, and X are known in advance, while the values of the nonzero elements are revealed at run time. There are two algorithms from prior work that are applicable in this setting: for moderately large values of d we can use the algorithm by [2], which runs in $O(dn^{1/3})$ rounds, while for small values of d the fastest algorithm is due to [12], and the round complexity is $O(d^{1.927})$ for semirings and $O(d^{1.907})$ for fields; see Table 1 for an overview.

1.2 Contribution 1: faster algorithm

Our first contribution is improvements in the running time: we design a faster algorithm that solves the case of semirings in $O(d^{1.867})$ rounds and the case of fields in $O(d^{1.832})$ rounds.

While there are no non-trivial unconditional lower bounds, we point out that $O(d^{1.333})$ for semirings or $O(d^{1.156})$ for fields would imply major breakthroughs for dense matrix multiplication (by simply plugging in d = n). In the following figure, we illustrate the progress we make towards these milestones:

The algorithm from prior work [12] is based on the idea of processing triangles. Here a triangle is a triple $\{i, j, k\}$ such that A_{ij} and B_{jk} are nonzeros and X_{ik} is one of the elements of interest, and we say that we process the triangle if we add the product $A_{ij}B_{jk}$ to the sum

Semirings	Fields	Reference
$\overline{O(n^2)}$	$O(n^2)$	trivial
$O(n^{4/3})$	$O(n^{1.157})$	[3, 20]
$O(dn^{1/3})$	$O(dn^{1/3})$	[2]
$O(d^2)$	$O(d^2)$	trivial, [12]
$O(d^{1.927})$	$O(d^{1.907})$	[12]
$O(d^{1.867})$	$O(d^{1.832})$	this work, Theorem 4

Table 1 Complexity of distributed sparse matrix multiplication

 X_{ik} . Once all such triangles are processed, we have computed all elements of interest in the product matrix.

There are two strategies that [12] uses for processing triangles:

- 1. If there are many triangles in total, we can find a dense cluster of them, we can interpret such a cluster as a tiny instance of dense matrix multiplication, and we can apply dense matrix multiplication algorithms to batch-process such clusters.
- 2. If there are few triangles in total, we can afford to process the triangles one by one.

Our improvements are in the second phase. Trivially, if we have a uniformly sparse instance, there are at most d^2n triangles in total, and we can process them in a naive manner in $O(d^2)$ rounds. The second phase of [12] shows that if the number of triangles is $d^{2-\varepsilon}n$, one can process them in $O(d^{2-\varepsilon/2})$ rounds. Here the key obstacle is the expression $\varepsilon/2$ in the exponent: even if we eliminate many triangles in the first phase, the second phase is still relatively expensive. We design a new algorithm that is able to process $d^{2-\varepsilon}n$ triangles in $O(d^{2-\varepsilon})$ rounds. By combining this with the general strategy of [12] and optimizing the trade-off between using time in the first phase vs. using time in the second phase, we obtain the new round complexities $O(d^{1.867})$ and $O(d^{1.832})$ —the difference between semirings and fields is in the dense matrix multiplication routine that we use in the first phase.

1.3 Contribution 2: beyond uniform sparsity

So far we have followed the lead of [12] and discussed *uniformly sparse* matrices (i.e., each row and column has at most d nonzeros). This is a rather restrictive notion of sparsity—can we extend prior work to more general notions of sparsity?

Perhaps the most natural candidate to consider would be *average sparsity*: we would simply assume that the total number of nonzeros in A and B and the total number of elements of interest in X is at most dn. Unfortunately, it turns out that this notion of sparsity is way too general, and we cannot expect fast algorithms for this case: in Lemma 19 we show that if we could solve average-sparse matrix multiplication for $n \times n$ matrices using n computers in time that is independent of n or only mildly depends on n (say, polylogarithmic in n), it would imply major breakthroughs in our understanding of dense matrix multiplication.

In summary, in order to have any real hope of pushing the complexity of sparse matrix multiplication down to $O(d^{1.832})$ or even something more modest like $O(d^2 + \log n)$, we need to explore some *intermediate notions of sparsity*. In this work we make use of the following notions of sparsity:

- \bigcup US(d) = uniformly sparse: at most d nonzeros per row and column.
- \blacksquare RS(d) = row-sparse: at most d nonzeros per row.
- \blacksquare CS(d) = column-sparse: at most d nonzeros per column.
- **BD**(d) = bounded degeneracy: we can recursively eliminate the matrix so that at each step we delete a row or column with at most d nonzeros.
- \blacksquare AS(d) = average-sparse: at most dn nonzeros in total.
- \blacksquare GM = general matrices.

We will omit (d) when it is clear from the context, and we will use e.g. expressions such as $US \times BD = AS$ to refer to the task of computing AB = X such that $A \in US(d)$, $B \in BD(d)$, and $\hat{X} \in AS(d)$; here we write \hat{X} for the matrix that indicates which elements of the product X we are interested in.

Family BD may at first look rather unusual in the context of linear algebra. However, if we interpret a matrix A as a bipartite graph G (with an edge $\{i, j\}$ whenever A_{ij} is nonzero), then $A \in \mathsf{BD}(d)$ corresponds to the familiar graph-theoretic notion that G is d-degenerate.

This is a widely used notion of sparsity in the graph-theoretic setting, and also closely connected with other notions of sparsity such as bounded arboricity.

We also point out that any matrix $A \in \mathsf{BD}(d)$ can be written as a sum A = X + Y such that $X \in RS(d)$ and $Y \in CS(d)$; to see this, eliminate A by deleting sparse rows or sparse columns, and put each sparse row in X and put each sparse column in Y. In particular, matrix multiplication $\mathsf{BD} \times \mathsf{BD}$ decomposes into operations of the form $\mathsf{CS} \times \mathsf{CS}$, $\mathsf{RS} \times \mathsf{CS}$, $\mathsf{CS} \times \mathsf{RS}$, and $\mathsf{RS} \times \mathsf{RS}$. In summary, we have

$$\mathsf{US} \subseteq \left\{ \begin{matrix} \mathsf{RS} \\ \mathsf{CS} \end{matrix} \right\} \subseteq \mathsf{BD} \subseteq \mathsf{AS} \subseteq \mathsf{GM}.$$

As we will see, all of our algorithmic results are symmetric w.r.t. the three matrices. To help present such results, we will introduce the following shorthand notation. Let X, Y, and Z be families of matrices. We write [X : Y : Z] to refer to the following six operation: computing $X \times Y = Z$, $Y \times X = Z$, $X \times Z = Y$, $Z \times X = Y$, $Y \times Z = X$, and $Z \times Y = X$.

Now equipped with this notation, we can rephrase the main result from the prior work [12]: they show that [US : US : US] can be solved in $O(d^{1.927})$ rounds for semirings and in $O(d^{1.907})$ rounds for fields. On the other hand, in Lemma 19 we argue why it is unlikely to extend this all the way to [AS : AS : AS]. The key question is what happens between the two extremes, US and AS? In particular, could we solve [BD : BD] : BD] efficiently?

Our answer to this question consists of four parts:

- 1. We generalize prior work and show that we can tolerate *one* average-sparse matrix: [US:US:AS] can be computed in $O(d^{1.867})$ rounds for semirings and in $O(d^{1.832})$ rounds for fields. We emphasize that here the running time is still independent of n.
- 2. As soon as two of the matrices are BD or more general, the task becomes at least as hard as computing the sum of values distributed across n computers, or broadcasting a single value to n computers. Hence, it is natural to expect an additional $O(\log n)$ term in the running time.
- 3. We show that many combinations all the way to [BD : AS : AS] and [US : AS : GM] can be solved in $O(d^2 + \log n)$ rounds, for both semirings and fields.

4. Finally, we provide evidence that this is essentially as general as what we can hope for. The results are summarized in Table 2.

1.4 Related work and applications

Our model of computing is the low-bandwidth model; this is closely related to the model known as node-capacitated clique or node-congested clique in the literature [1]—in the lowbandwidth model each computer can send and receive one message, while the node-capacitated clique model is usually defined so that each computer can send and receive $O(\log n)$ messages per round. Both the low-bandwidth model and the node-capacitated clique model can be interpreted as variants of the congested clique model [16]. Indeed, any algorithm that runs in T(n) rounds in the congested clique model can be simulated in nT(n) rounds in the low-bandwidth model, and for many problems (such as the dense matrix multiplication) such a simulation also results in the fastest known algorithms for the low-bandwidth model. Both the low-bandwidth model and the congested clique model can be interpreted as special cases of the classic bulk synchronous parallel model [19].

Matrix multiplication in this context has been primarily studied in the congested clique model [2,3,5,15]. However, as observed by [12], the congested clique model is poorly-suited for the study of sparse matrix multiplication: one can only conclude that for sufficiently sparse

Sparsity	Upper bound		Conditional hardness
[US : US : US]	$O(d^{1.927})$	[12]	
[US : US : US]	$O(d^{1.867})$	Theorem 4	
 [US : US : AS]			
[US : US : GM]	$O(d^2 + \log n)$	Theorem 13	
[US : BD : BD]	$O(d^2 + \log n)$	Theorem 13	Lemma 18
 [US : AS : GM]			
[BD : BD : BD]	$O(d^2 + \log n)$	Theorem 17	Lemma 18
 [BD : AS : AS]			
[US : GM : GM]			Lemma 20
 [GM : GM : GM]			
[BD : BD : GM]			Lemma 21
 [GM : GM : GM]			
[AS : AS : AS]			Lemma 19
 [GM : GM : GM]			

Table 2 Summary of results (for semirings)

matrices, the problem is solvable in O(1) rounds, and one cannot explore more fine-grained differences between different algorithms.

Similar to [12], we work in the *supported* version of the low-bandwidth model. In general, the supported model [10, 11, 18] refers to a setting in which the structure of the input is known in advance (we can do arbitrary *preprocessing* based on the structure of the input), while the specific instance is revealed at run time. In the case of graph problems, we can see an unweighted graph G in advance, while the adversary reveals a weighted version G' of G(a special case being a 0/1-weighted graph, which can be interpreted as a subgraph of G). For matrix multiplication, we know the sparsity structure in advance (i.e., which elements are potentially nonzero), while the adversary reveals the concrete values at run time.

One of the main applications that we have in mind here is triangle detection, which has been extensively studied in the distributed setting [4, 6, 7, 9, 12-14, 17]. If we can multiply matrices, we can also easily detect triangles in a graph. Moreover, [US : US : US] corresponds to triangle detection in a bounded-degree graph, while e.g. [AS : AS : AS] corresponds to triangle detection in a sparse graph.

1.5 Open questions for future work

In this work we initiated the study of different notions of sparsity in the context of lowbandwidth matrix multiplication. This work also gives rise to a number of questions for

future work.

First, our running times— $O(d^{1.867})$ rounds for semirings and $O(d^{1.832})$ for fields—are still far from the conditional lower bounds $\Omega(d^{1.333})$ and $\Omega(d^{1.156})$, respectively. Pushing the running time further down is a major challenge for future work. If we follow the two-phase approach of [12], our present work essentially makes the second phase optimal; the main hope for future improvements now lies in the first phase of the algorithm.

Our algorithms are designed for the supported model. Eliminating the knowledge of the support (i.e., knowledge of the sparsity structure) is a major challenge for future work.

Our hardness results are conditional. Turning these into unconditional lower bounds is yet another question for future work.

Finally, there are some gaps in Table 2. For example, we do not have conditional hardness result for [US:US:GM], and also Lemmas 20 and 21 do not cover all permutations of the matrix families.

2 Preliminaries

We work in the low-bandwidth model. There are n computers. Initially each computer holds its own part of A and B, and eventually each computer has to report its own part of X. When we study sparse matrices with at most dn nonzeros, we assume that each computer holds at most d elements, while when we study dense matrices, we assume that each computer holds at most n elements.

For our algorithms, it does not matter how the input and output is distributed among the computers—with an additional O(d) time we can permute the input and output as appropriate. Our conditional lower bounds hold for any fixed distribution of input and output.

2.1 Supported model and indicator matrices

When we study matrix multiplication in the supported model, we assume that we know in advance indicator matrices \hat{A} , \hat{B} , and \hat{X} that encode the structure of our instance: $\hat{A}_{ij} = 0$ implies $A_{ij} = 0$, $\hat{B}_{jk} = 0$ implies $B_{jk} = 0$, and $\hat{X}_{ik} = 0$ indicates that we do not need to compute the value of X_{ik} . When we make assumptions on sparsity, our assumptions refer to the sparsity of \hat{A} , \hat{B} , and \hat{X} . For example, when we study $\mathsf{US} \times \mathsf{BD} = \mathsf{AS}$, we assume that $\hat{A} \in \mathsf{US}(d)$, $\hat{B} \in \mathsf{BD}(d)$, and $\hat{X} \in \mathsf{AS}(d)$. It then also follows that $A \in \mathsf{US}(d)$ and $B \in \mathsf{BD}(d)$.

2.2 Tripartite graph and triangles

It will be convenient to assume that our matrices are indexed with indices from three disjoint sets I, J, and K, each of size n. For example, elements of matrix A are indexed with A_{ij} where $i \in I$ and $j \in J$. Following [12], we write $\hat{\mathcal{T}}$ for the set of all *triangles*, which are triples $\{i, j, k\}$ with $i \in I$, $j \in J$, and $k \in K$ such that $\hat{A}_{ij} \neq 0$, $\hat{B}_{jk} \neq 0$, and $\hat{X}_{ik} \neq 0$.

For a collection of triangles \mathcal{T} , we write $G(\mathcal{T})$ for the tripartite graph $G(\mathcal{T}) = (V, E)$, where the set of nodes is $V = I \cup J \cup K$ and there is an edge $\{u, v\} \in E$ if there is a triangle $T \in \hat{\mathcal{T}}$ with $\{u, v\} \subseteq T$. For a set of nodes $U \subseteq V$, we write $\mathcal{T}[U] = \{T \in \mathcal{T} : T \subseteq U\}$ for the set of triangles induced by U.

We say that we process a triangle $\{i, j, k\}$ if we have added the product $A_{ij}B_{jk}$ to the sum X_{ik} . The key observation is that processing all triangles is exactly equivalent to computing all values of interest in the product matrix X = AB.

2.3 Clusters and clusterings

We say that $U \subseteq V$ is a *cluster* if $U = I' \cup J' \cup K'$ for $I' \subseteq I$, $J' \subseteq J$, $K' \subseteq K$, |I'| = d, |J'| = d, and |K'| = d. We say that a collection of triangles \mathcal{P} is *clustered* if there are disjoint clusters U_1, \ldots, U_k such that $\mathcal{P} = \mathcal{P}[U_1] \cup \cdots \cup \mathcal{P}[U_k]$.

A clustered collection of triangles can be processed efficiently by applying an algorithm for dense matrix multiplication in parallel to each cluster [12]. By applying the strategy of [3] and the latest value of ω from [20], we obtain:

▶ Lemma 1. A clustered instance of matrix multiplication can be solved in $O(d^{4/3})$ rounds over semirings, and in $O(d^{1.156671})$ rounds over fields.

3 Handling few triangles fast

We will start by a general result that will form the foundation for all of our upper-bound results. [12, Lemma 5.2] observed that if we have a *balanced* instance in which each node touches at most κ triangles, we can solve matrix multiplication in $O(\kappa)$ rounds. However, the key challenge is what to do in *unbalanced* instances: we have at most κn triangles, but some nodes can touch many more than κ triangles. The following lemma shows how to handle also such instances efficiently:

▶ Lemma 2. Let \mathcal{T} be a set of triangles with $|\mathcal{T}| \leq \kappa n$, and assume that for each pair of nodes $u, v \in V$ there are at most m triangles $T \in \mathcal{T}$ with $\{u, v\} \in T$. Assume that each computer initially holds at most d elements of A and at most d elements of B, and it will need to hold at most d elements of X. Then all triangles in \mathcal{T} can be processed (over fields or semirings) in $O(\kappa + d + \log m)$ rounds.

We emphasize that this strictly improves over [12, Lemma 5.1] in two different ways: our result is applicable in a more general setting, and we avoid the factor-2 loss in the exponent of the running time.

3.1 High-level plan

Our high-level plan for proving Lemma 2 goes as follows:

- 1. We start with an arbitrary collection of κn triangles \mathcal{T} over the original set of 3n nodes $V = I \cup J \cup K$.
- 2. We construct a new virtual collection of κn triangles \mathcal{T}' over a new set of O(n) virtual nodes V'.
- 3. The new virtual collection of triangles is *balanced* in the sense that each virtual node touches only κ triangles.
- 4. The virtual nodes are assigned to real computers so that each real computer is responsible for only O(1) virtual nodes.
- 5. We show how we can route original data to the virtual nodes, so that each virtual node only needs to do work for each triangle that it touches, and we can also aggregate the results back to the original nodes that need it.

The most challenging part is the final step, routing.

3.2 Virtual instance

For each node $v \in V$, let $\mathcal{T}(v)$ be the set of triangles in $T \in \mathcal{T}$ with $v \in T$. Let $t(v) = |\mathcal{T}(v)|$ and $\ell(v) = \lfloor t(v)/\kappa \rfloor$. Our set of virtual nodes is

 $V' = I' \cup J' \cup K',$ $I' = \{(i, x) \mid i \in I, x \in \{1, 2, \dots, \ell(i)\}\},$ $J' = \{(k, x) \mid j \in J, x \in \{1, 2, \dots, \ell(j)\}\},$ $K' = \{(j, x) \mid k \in K, x \in \{1, 2, \dots, \ell(k)\}\}.$

The key observation is that the set of virtual nodes is not too large:

$$|V'| = \sum_{v} \ell(v) \le \sum_{v} (t(v)/\kappa + 1) \le 3|\mathcal{T}|/\kappa + n = 4n.$$

Now we can construct a new balanced collection of triangles \mathcal{T}' over V' such that each virtual node $v' \in V'$ touches at most κ triangles: for the first κ triangles in $\mathcal{T}(v)$ we replace v with the virtual node (v, 1), for the next κ triangles we replace v with (v, 2), etc.

We assign the virtual nodes to real computers so that each computer is responsible for at most 4 virtual nodes. Therefore, in what follows, we can assume that the virtual nodes are computational entities, and we can simulate their work in the real computer network with constant overhead.

3.3 Routing

Now let us see how we will process the triangles. Consider a triangle $\{i, j, k\} \in \mathcal{T}$, and let its unique virtual copy be $\{i', j', k'\} \in \mathcal{T}'$. Conceptually, we proceed as follows:

- 1. The real computer p(i, j) that initially holds A_{ij} transmits it to virtual computer i'.
- 2. The real computer p(j,k) that initially holds B_{jk} transmits it to virtual computer i'.
- 3. Virtual computer i' computes the product $A_{ij}B_{jk}$ and transmits it to the real computer p(i,k) that is responsible for storing X_{ik} .

If we did this in a naive manner, it would be prohibitively expensive; a more careful routing scheme is needed; see Figure 1 for an illustration.

First consider step (1): the transmission of A_{ij} from p = p(i, j) to i'. By assumption, i, j is part of at most m triangles, and a single real computer initially holds at most d distinct values of matrix A. On the other hand, a single virtual computer needs to receive up to κ messages, so the total number of messages to transmit is bounded by κn .

The key idea is that we will do intermediate-point routing. Form an array of all triples (i, j, i') such that A_{ij} is needed by i', and order the array lexicographically (i, j, i'). For each (i, j) the first triple (i, j, i') is called the *anchor* element for (i, j). The array has at most κn triples, and we can assign the triples to our real computers so that each computer has at most κ triples. For each (i, j) we write q(i, j) for the computer that holds the anchor element for (i, j).

Now we first route $A_{i,j}$ from computer p(i,j) to computer q(i,j). This can be completed in $O(d + \kappa)$ rounds, as each computer has d outgoing messages and κ incoming messages. We can implement this, for example, by considering the bipartite graph in which on one side we have sender, on one side we have recipients: the sender-side has maximum degree d and the recipient-side has maximum degree κ , and hence we can find a proper edge coloring with $O(d + \kappa)$ colors and use the color classes to schedule the messages. This way in each round each computer sends and receives at most one message.



Figure 1 Routing scheme from the proof of Lemma 2

Next, we will spread the messages so that each computer that holds a triple (i, j, i') knows the value of A_{ij} . Thanks to the array being sorted, this is now easy. Let us focus on some pair (i, j). Recall that q(i, j) is the first computer that holds any triple of the form (i, j, \cdot) , and it already knows the value of A_{ij} . Let r(i, j) be the last computer that holds any triple of the form (i, j, \cdot) . Now our task is to simply spread A_{ij} from q(i, j) to all computers in the range $q(i, j) + 1, \ldots, r(i, j)$. If r(i, j) = q(i, j), there is nothing to be done. On the other hand, if $r(i, j) \neq q(i, j)$, we have a convenient situation: computer q(i, j) only needs to spread the value of A_{ij} , and computers $q(i, j) + 1, \ldots, r(i, j)$ only need to receive the value of A_{ij} . Hence, we can first let q(i, j) inform q(i, j) + 1, and then $q(i, j) + 1, \ldots, r(i, j)$ form a broadcast tree that distributes A_{ij} to each of the computers. Note that all these broadcast trees involve disjoint sets of computers, and we can implement broadcast in parallel. We have at most m triples of form (i, j, \cdot) , and hence (using a very sloppy estimate) at most mcomputers involved in each broadcast operation. A broadcast tree of depth $O(\log m)$ suffices.

At this point each real computer holds up to κ triples of (i, j, i') and knows the corresponding value A_{ij} . Then we route A_{ij} to virtual computer i'; as everyone needs to send and receive up to κ messages, this can be implemented in $O(\kappa)$ rounds.

Step (2) is essentially identical to step (1); instead of values A_{ij} and triples (i, j, i'), we transmit values B_{jk} and use triples (j, k, i').

For step (3) we do the converse of step (1). We now form triples (i, k, i') that indicate that virtual computer i' holds a product $A_{ij}B_{jk}$ that needs to be accumulated to X_{ik} . We sort the array as above, and then first use $O(\kappa)$ rounds to route the products from virtual computers to real computers. Now each real computer that holds multiple triples of the form (i, k, \cdot) can locally aggregate all these products into a single sum. Then we define q(i, j) and r(i, j) as above, and construct a converge ast tree with which computers $q(i, j) + 1, \ldots, r(i, j)$ compute the grand total of the products $A_{ij}B_{jk}$ they have received, and finally q(i, j) + 1relays this information to the anchor q(i, j), which can compute the sum X_{ij} . Finally, we route X_{ij} to the computer that needs to report it, using additional $O(\kappa + d)$ rounds.

This completes the proof of Lemma 2. In what follows, we will present applications of this result.

4 Algorithm for [US : US : AS]

In our terminology, [12] proved the following statement:

▶ **Theorem 3** ([12]). In the supported low-bandwidth model, sparse matrix multiplication of the form [US : US : US] can be computed in $O(d^{1.927})$ rounds over semirings and in $O(d^{1.907})$ rounds over fields.

In this section, we both improve the running time and widen the scope of applicability:

▶ **Theorem 4.** In the supported low-bandwidth model, sparse matrix multiplication of the form [US : US : AS] can be computed in $O(d^{1.867})$ rounds over semirings and in $O(d^{1.832})$ rounds over fields.

4.1 **Preliminaries**

We start with the following technical lemma:

▶ Lemma 5. Let $\hat{\mathcal{T}}$ be defined by an instance of [US : US : AS]. For each node $x \in V$ there are at most d^2 triangles $T \in \hat{\mathcal{T}}$ with $x \in T$.

Proof. Consider first the case of $US \times AS = US$. If $x \in I$, then x is incident to at most d nodes $j \in J$ and at most d nodes $k \in K$, and hence in total there can be at most d^2 triangles of the form $\{x, j, k\}$. If $x \in J$, then x is incident to at most d nodes $i \in I$, and each of them is incident to at most d nodes $k \in K$, and hence in total there can be at most d^2 triangles of the form $\{i, x, k\}$. The case of $x \in K$ is similar to $x \in J$. For the cases of $AS \times US = US$ and $US \times US = AS$, permute the roles of I, J, and K as appropriate.

We have two simple corollaries that will be useful shortly:

▶ Corollary 6. Let $\hat{\mathcal{T}}$ be defined by an instance of [US : US : AS]. For each pair of nodes $u, v \in V$ there are at most d^2 triangles $T \in \hat{\mathcal{T}}$ with $\{u, v\} \in T$.

▶ Corollary 7. The total number of triangles in any [US : US : AS] instance is bounded by d^2n .

4.2 Part 1: handling dense parts

In the first part we follow the basic structure of [12], with two changes:

- 1. we generalize it to [US : US : AS],
- **2.** we choose different parameter values so that we can fully benefit from the new algorithm for part 2.

We will then diverge from [12] in part 2, where we will apply Lemma 2.

Similar to [12], we first show that if there are many triangles, we can find a *dense cluster*:

▶ Lemma 8. Let $\hat{\mathcal{T}}$ be defined by an instance of [US : US : AS], and let $\mathcal{T} \subseteq \hat{\mathcal{T}}$. If $|\mathcal{T}| \ge d^{2-\varepsilon}n$ for some $\varepsilon > 0$, then there exists a cluster $U \subseteq V$ with $|\mathcal{T}[U]| \ge d^{3-4\varepsilon}/24$.

Proof. This is a generalization of [12, Lemma 3.1], which holds for [US : US : US]. The original proof makes use of the following facts about $G(\mathcal{T})$:

- \blacksquare there are at most dn edges between J and K,
- = each $i \in I$ is incident to at most d nodes of J,
- each $i \in I$ is incident to at most d nodes of K,
- = each $j \in J$ is incident to at most d nodes of I,
- each $k \in K$ is incident to at most d nodes of I.

All of these also hold for $US \times AS = US$. For the cases of $AS \times US = US$ and $US \times US = AS$, permute the roles of *I*, *J*, and *K* as appropriate.

Then by plugging in Lemma 8 instead of [12, Lemma 3.1] in the proof of [12, Lemma 4.1], we obtain the following corollary:

▶ Lemma 9. Let $\delta > 0$ and $\varepsilon > 0$, and assume that d is sufficiently large. Let $\hat{\mathcal{T}}$ be defined by an instance of $[\mathsf{US} : \mathsf{US} : \mathsf{AS}]$, and let $\mathcal{T} \subseteq \hat{\mathcal{T}}$. Assume $|\mathcal{T}| \ge d^{2-\varepsilon}n$. Then we can partition \mathcal{T} into disjoint sets \mathcal{P} and \mathcal{T}' such that \mathcal{P} is clustered and $|\mathcal{P}| \ge d^{2-5\varepsilon-4\delta}n$.

Proof. We follow the strategy in the proof of [12, Lemma 4.1]: Apply Lemma 8 to find a cluster U. Put the triangles $T \in \mathcal{T}[U]$ that are fully contained in U to \mathcal{P} , and put the triangles $T \in \mathcal{T}$ that only partially touch U to \mathcal{T}' . Repeat until there are sufficiently few triangles left; put all remaining triangles to \mathcal{T}' .

The original proof makes use of the fact that each node $x \in V$ is contained in at most d^2 triangles. This is trivial for [US : US : US]. For [US : US : AS] we apply Lemma 5, and then the original analysis holds verbatim.

Step	δ	γ	ε	α	β
1	0.00001	0.00000	0.10672	1.86698	1.89328
2	0.00001	0.10672	0.12806	1.86696	1.87194
3	0.00001	0.12806	0.13233	1.86697	1.86767
4	0.00001	0.13233	0.13319	1.86700	1.86681

Table 3 Parameters for the proof of Lemma 11 (semirings)

Finally, by plugging in Lemma 9 instead of [12, Lemma 4.1] in the proof of [12, Lemma 4.2], we obtain the following corollary:

▶ Lemma 10. Let $\delta > 0$ and $0 \le \gamma < \varepsilon$, and assume that d is sufficiently large. Let $\hat{\mathcal{T}}$ be defined by an instance of [US : US : AS], and let $\mathcal{T} \subseteq \hat{\mathcal{T}}$. Assume $|\mathcal{T}| \le d^{2-\gamma}n$. Then we can partition \mathcal{T} into disjoint sets $\mathcal{T} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_L \cup \mathcal{T}'$ such that each \mathcal{P}_i is clustered, $L \le 144d^{5\varepsilon-\gamma+4\delta}$ and $|\mathcal{T}'| \le d^{2-\varepsilon}n$.

Proof. We follow the proof of [12, Lemma 4.2]: Apply Lemma 9 repeatedly to construct clusterings $\mathcal{P}_1, \mathcal{P}_2, \ldots$ Stop once the residual set of triangles \mathcal{T}' is sufficiently small. The original analysis holds verbatim.

Now we are ready to prove the following result. Note that unlike the analogous result from [12], we have got here a convenient formulation in which the exponent in the time complexity matches the exponent in the size of the residual part:

▶ Lemma 11. Let $\hat{\mathcal{T}}$ be defined by an instance of $[\mathsf{US} : \mathsf{US} : \mathsf{AS}]$. We can partition $\hat{\mathcal{T}}$ into $\mathcal{T}_1 \cup \mathcal{T}_2$ such that:

- 1. For matrix multiplication over semirings, \mathcal{T}_1 can be solved in $O(d^{1.867})$ rounds and $|\mathcal{T}_2| \leq d^{1.867}n$.
- 2. For matrix multiplication over fields, \mathcal{T}_1 can be solved in $O(d^{1.832})$ rounds and $|\mathcal{T}_2| \leq d^{1.832}n$.

Proof. For the case of semirings, we apply Lemma 10 with the parameters δ , γ , and ε shown in Table 3. In the first step, we start with $\gamma = 0$; note that by Corollary 7 we satisfy the assumption that our initial set of triangles has size at most $d^{2-\gamma}n = d^2n$. We pick a very small δ and choose a suitable ε . Then we use Lemma 1 to process each of \mathcal{P}_i . The total processing time will be $O(d^{\alpha})$, where $\alpha = 5\varepsilon - \gamma + 4\delta + 4/3$. Our choice of ε is optimized so that $\alpha \leq 1.867$ and hence we do not exceed our time budget. After processing all of \mathcal{P}_i , we are left with the residual set of triangles of size $|\mathcal{T}'| \leq d^{\beta}n$, where $\beta = 2 - \varepsilon$. We then repeat the same process, starting with $\gamma = 2 - \beta$, and again optimizing ε . We continue until $\beta \leq 1.867$.

For the case of fields, the idea is the same, but we make use of the parameter values in Table 4.

4.3 Part 2: handling few triangles

Let us recap: We started with an arbitrary instance, with possibly up to d^2n triangles. Then we have used $O(d^{\alpha})$ rounds to process "dense" parts, and we are left with only $d^{\alpha}n$ triangles; here $\alpha = 1.867$ for semirings and $\alpha = 1.832$ for fields.

Step	δ	γ	ε	α	β
1	0.00001	0.00000	0.13505	1.83197	1.86495
2	0.00001	0.13505	0.16206	1.83197	1.83794
3	0.00001	0.16206	0.16746	1.83196	1.83254
4	0.00001	0.16746	0.16854	1.83196	1.83146

Table 4 Parameters for the proof of Lemma 11 (fields)

Now we can apply Lemma 2 to handle all these remaining $d^{\alpha}n$ triangles in $O(d^{\alpha})$ rounds. When we apply the lemma, we set $\kappa = d^{\alpha}$, and by Corollary 6 we have $m = d^2$. The overall running time is $O(d^{\alpha} + d + \log d^2) = O(d^{\alpha})$. This completes the proof of Theorem 4.

5 More general algorithms

We will now step beyond [US : US : AS] and consider more general settings. We will show that we can handle much more general notions of sparsity in $O(d^2 + \log n)$ rounds. For each case we prove a bound on the total number of triangles, and then apply Lemma 2.

5.1 Algorithm for [US : AS : GM]

▶ Lemma 12. Let $\hat{\mathcal{T}}$ be defined by an instance of [US : AS : GM]. Then $|\hat{\mathcal{T}}| \leq d^2 n$.

Proof. We give a proof for $US \times AS = GM$; the remaining cases can be proved similarly. We can bound the number of triangles of the form $\{i, j, k\}$ with $i \in I$, $j \in J$, and $k \in K$ as follows: there are at most dn edges of the form $\{j, k\}$, since $B \in AS$. For each such edge $\{j, k\}$ there are at most d edges of the form $\{i, j\}$, since $A \in US$. It follows that the number of triangles is at most d^2n .

▶ **Theorem 13.** In the supported low-bandwidth model, sparse matrix multiplication of the form [US : AS : GM] can be computed in $O(d^2 + \log n)$ rounds over semirings and fields.

Proof. Follows from Lemma 12 and Lemma 2 by setting $\kappa = d^2$ and m = n.

5.2 Algorithm for [BD : AS : AS]

When we consider [BD : AS : AS], it will be convenient to decompose BD into RS and CS.

▶ Lemma 14. Let $\hat{\mathcal{T}}$ be defined by an instance of [RS : AS : AS]. Then $|\hat{\mathcal{T}}| \leq d^2 n$.

Proof. Let us consider the case $RS \times AS = AS$; the other permutations are similar. We can bound the number of triangles of the form $\{i, j, k\}$ with $i \in I$, $j \in J$, and $k \in K$ as follows: there are at most dn edges of the form $\{i, k\}$, since $X \in AS$. For each such edge $\{i, k\}$ there are at most d edges of the form $\{i, j\}$, since $A \in RS$. It follows that the number of triangles is at most d^2n .

▶ Lemma 15. Let $\hat{\mathcal{T}}$ be defined by an instance of [CS : AS : AS]. Then $|\hat{\mathcal{T}}| \leq d^2 n$.

Proof. Transpose the matrices in Lemma 14.

▶ Lemma 16. Let $\hat{\mathcal{T}}$ be defined by an instance of [BD : AS : AS]. Then $|\hat{\mathcal{T}}| \leq 2d^2n$.

Proof. Let us consider the case $BD \times AS = AS$; the other permutations are similar. As we discussed in Section 1.3, we can decompose X = AB into $X = A_1B + A_2B$, where $A_1 \in RS$ and $A_2 \in CS$. The claim follows by applying Lemma 14 to $X_1 = A_1B$ and Lemma 15 to $X_2 = A_2B$.

▶ **Theorem 17.** In the supported low-bandwidth model, sparse matrix multiplication of the form [BD : AS : AS] can be computed in $O(d^2 + \log n)$ rounds over semirings and fields.

Proof. Follows from Lemma 16 and Lemma 2 by setting $\kappa = 2d^2$ and m = n.

6 Conditional hardness

In this section we connect the hardness of sparse matrix multiplication with simpler primitives, such as broadcasting.

6.1 Broadcasting and aggregation

For settings between [US : BD : BD] and [US : AS : GM] our upper bound from Theorem 13 has an additive $O(\log n)$ term, which comes from broadcast and convergecast operations. In the following lemma, we show that some broadcasting and/or aggregation is needed in all of these cases. We note that a simple low-bandwidth adaptation of the proof in [8] shows an $\Omega(\log n)$ -round lower bound for computing a sum over values held by all computers, and hence this result also implies an unconditional lower bound.

▶ Lemma 18. In the supported low-bandwidth model, sparse matrix multiplication of the form [US : BD : BD] is at least as hard as computing a sum of n values distributed among n computers, or broadcasting a single value to n computers, even for d = 1.

Proof. Consider first $BD \times BD = US$. A special case of this is computing AB = X such that all nonzeros in A are in row 1, all nonzeros in B are in column 1, and we are only interested in computing element (1, 1) of X. Furthermore, let all nonzeros of B be 1; we have the following task (where "?" indicates elements of the result matrix that we are not interested in):

$\begin{bmatrix} a_1 \\ 0 \end{bmatrix}$	$\begin{array}{c} a_2 \\ 0 \end{array}$	 	$a_n \\ 0$	×	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	0 0	· · · ·	0 0	=	$\begin{bmatrix} x \\ ? \end{bmatrix}$? ?	· · · ·	? ?	
$\begin{bmatrix} \dots \\ 0 \end{bmatrix}$	0	· · · · · ·	0		 1	 0		0		 ?	· · · · ?	· · · · ·	 ?	

Now this is equivalent to computing a sum $x = \sum_j a_j$, in a setting in which initially each computer holds one a_j .

Then consider $BD \times US = BD$. A special case of this is computing AB = X such that all nonzeros in A are in column 1, the only nonzero of B is at (1, 1), and we are only interested in computing elements in the first column of X. Furthermore, let all nonzeros of A be 1; we have the following task:

$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{array}{c} 0 \\ 0 \end{array}$	· · · ·	$\begin{bmatrix} 0\\ 0 \end{bmatrix}$	×	$\begin{bmatrix} b \\ 0 \end{bmatrix}$	0 0	· · · ·	$\begin{bmatrix} 0\\ 0 \end{bmatrix}$	_	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$? ?	· · · ·	? ?	
 1	 0	· · · · ·	 0		 0	 0	· · · · ·	 0		$\begin{bmatrix} \dots \\ x_4 \end{bmatrix}$	· · · · ?	· · · · ·	· · · ?	

Now we need to output $x_1 = \ldots = x_n = b$, and each computer has to hold one of these values. Hence, this is equivalent to broadcasting the value b to all computers.

-

The case of $US \times BD = BD$ is also a broadcast task, by a similar argument.

6.2 Dense matrix multiplication

Let us now argue why [AS : AS] is unlikely to be solvable in $O(d^2 + \log n)$ rounds, or even in $f(d) + O(n^{\varepsilon})$ rounds for sufficiently small values of ε .

▶ Lemma 19. Computing [AS : AS : AS] for d = 1 in T(n) rounds implies an algorithm that solves dense matrix multiplication in $T'(n) = nT(n^2)$ rounds.

Proof. Assume \mathcal{A} solves $[\mathsf{AS} : \mathsf{AS}]$ in T(n) rounds for d = 1. Let $m = \sqrt{n}$. We use \mathcal{A} as a black box to design an algorithm \mathcal{A}' that multiplies dense $m \times m$ matrices using m computers in T'(m) rounds. Algorithm \mathcal{A}' works as follows: given an input $X' = \mathcal{A}'B'$ with dimensions $m \times m$, construct an instance $X = \mathcal{A}B$ of dimensions $n \times n$ by padding with zeros:

 $\begin{bmatrix} A' & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} B' & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} X' & ? \\ ? & ? \end{bmatrix}.$

This is now an average-sparse instance with $m^2 = n$ nonzeros. Then apply algorithm \mathcal{A} , so that each of the *m* computers simulates $m = \sqrt{n}$ computers; this way we have *n* virtual computers available for running \mathcal{A} . The running time of \mathcal{A} is $T(n) = T(m^2)$, and hence the simulation completes in time $T'(m) = mT(m^2)$.

In particular, if we could achieve $T(n) = o(n^{1/6})$ for sparse matrix multiplication over semirings, then we could do dense matrix multiplication over semirings in $T'(n) = o(n^{4/3})$ rounds, and it would imply major breakthroughs in our understanding of dense matrix multiplication.

Similarly, if we could achieve $T(n) = O(n^{0.01})$ over fields, then we could do dense matrix multiplication over fields in $T'(n) = O(n^{1.02})$ rounds, which is far from the current upper bound of $O(n^{1.157})$.

This is the main justification why our upper bound in Theorem 17 is applicable up to [BD: AS: AS], but not all the way to [AS: AS].

6.3 Routing

Next we consider problem settings that are at least as hard as routing many elements to a single computer. We will assume here that the assignment of the input and output values to computers only depends on the structure of the input, and not on the numerical values. We will write A^v , B^v , and X^v to denote the set of elements of matrices A, B, and X that are held by computer v.

We first consider [US : GM : GM]. We prove a hardness result for $US \times GM = GM$; the case of $GM \times US = GM$ is symmetric, but $GM \times GM = US$ is left for future work.

▶ Lemma 20. In the supported low-bandwidth model, to solve $US \times GM = GM$, at least one node needs to output $\Omega(\sqrt{n})$ values originally held by other nodes.

Proof. For the sake of simplicity, we give a proof when matrix A contains at most 2n nonzero elements, i.e. d = 2. Let all the elements of A except $a_{i,i}, a_{i,(i \mod n)+1}$ for all $i \in [n]$ are equals to 0; that is, our task is to compute

 $\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & \dots & 0 & 0 \\ 0 & a_{2,2} & a_{2,3} & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,1} & 0 & 0 & 0 & \dots & 0 & a_{n,n} \end{bmatrix} \times B = X,$

where B is a general (dense) matrix, and we are interested in all values of X.

Each computer in the network contains two (nonzero) elements of A, n elements of B and

n elements of X. We divide the analysis into the following two cases; consider a computer v:
1. If X^v contains at least √n elements from one of the columns of X: let us assume that X^v contains √n elements from column j of X i.e.,

 $x_{i_1,j}, x_{i_2,j}, \ldots, x_{i_{\sqrt{n}},j} \in X^v.$

We know that

 $x_{i,j} = a_{i,i}b_{i,j} + a_{i,(i \mod n)+1}b_{(i \mod n)+1,j}.$

Let us set $b_{i,j} = 1$, and $a_{i,(i \mod n)+1} = 0$ for all $i, j \in [n]$. This implies $x_{i_l,j} = a_{i_l,i_l}$ for all $i_l \in [\sqrt{n}]$. Since computer v initially holds only two of these a_{i_l,i_l} , we need to route at least $\Omega(\sqrt{n})$ values from other computers to computer v.

2. If X^{v} contains less than \sqrt{n} elements from every column of X: let

$$I = \{ (i, j) \mid x_{i, j} \in X^v \}.$$

If we set $a_{i,i} = 1$ and $a_{i,((i-1) \mod n)+1} = 0$ for all $i \in [n]$, we have X = B. Thus, v must output all the elements of $b_{i,j}$ such that $(i, j) \in I$. Therefore, v must receive all these elements. Now it might happen that these n elements of B are already stored at v. However, we will prove that computer v also requires at least \sqrt{n} more elements stored at other computers in order to produce correct output for different values of nonzero elements of A and B. Note that if X^v contains less than \sqrt{n} nonzero elements from each column of X, then the number of columns of X from which X^v contains an element is more than \sqrt{n} . Let C^v be the set of columns of X such that X^v contains some element from these columns. From above, we can say that in each $C_i \in C^v$ there exist two elements $x_{j,i}, x_{j+1,i}$ such that $x_{j,i} \in X^v$ and $x_{j+1,i} \notin X^v$ (because X^v contains less than \sqrt{n} elements from each column). Let

$$J = \{j \mid x_{j,i} \in X^v \text{ and } x_{j+1,i} \notin X^v\}$$

Now we take a different assignment where we keep matrix B the same, and in matrix A we set $a_{j,j} = 0$ and $a_{j,(j \mod n+1)} = 1$ for all $j \in J$. This will result in $x_{j,i} = b_{j+1,i}$ for all i and j such that $x_{j,i} \in X^v$ and $x_{j+1,i} \notin X^v$. We know that there exist more than \sqrt{n} such $x_{j,i}$. From the above paragraph, we know that v must contain all $b_{j,i}$ such that $x_{j,i} \in X^v$. Therefore, we can say that v does not contain $b_{j+1,i}$ such that $x_{j,i} \in X^v$ and $x_{j+1,i} \notin X^v$. Thus, v needs to receive $\Omega(\sqrt{n})$ values from other computers.

Next we consider [BD : BD : GM]. We prove a hardness result for $BD \times BD = GM$; the case of $BD \times GM = BD$ and $GM \times BD = BD$ is left for future work. We consider the special case of $RS \times CS = GM$; the case of $BD \times BD = GM$ is at least as hard:

▶ Lemma 21. In the supported low-bandwidth model, to solve $RS \times CS = GM$, at least one node needs to output $\Omega(\sqrt{n})$ values originally held by other nodes.

Proof. We prove the lower bound for the case when d = 1, i.e. each node in the network contains one element of A, one element of B and n elements of X. Let A and B be matrices such that all the elements of these matrices are zero except $a_{i,1}$ and $b_{1,i}$ for all $i \in [n]$:

$$\begin{bmatrix} a_{1,1} & 0 & \dots & 0 \\ a_{2,1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & 0 & \dots & 0 \end{bmatrix} \times \begin{bmatrix} b_{i,1} & b_{1,2} & \dots & b_{1,n} \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix} = X.$$

Similar to the above, we divide the analysis into two cases.

1. If X^v contains at least \sqrt{n} elements from one of the columns of X: let us assume that X^v contains \sqrt{n} elements from column j of X i.e.,

$$x_{i_1,j}, x_{i_2,j}, \ldots, x_{i_{\sqrt{n}},j} \in X^v.$$

Let us set $b_{1,i} = 1$ for all $i \in n$. This implies that $x_{i_k,j} = a_{i_k,1}$ for all $k \in \sqrt{n}$. We know that v contains only one element of A, and therefore we need to receive $\Omega(\sqrt{n})$ elements from other computers.

2. If X^v contains less than \sqrt{n} elements from each column of X: in such cases we know that there exist at least \sqrt{n} elements $x_{i_1,j_1}, x_{i_2,j_2}, \ldots x_{i_{\sqrt{n}},j_{\sqrt{n}}}$ such that $j_i \neq j_k$ for $i \neq k$ and $i, k \in [\sqrt{n}]$. Now consider an assignment where we set all $a_{i,1} = 1$. This implies $x_{i_k,j_k} = b_{1,j_k}$ for all $k \in \sqrt{n}$. Again we can conclude that we need to receive $\Omega(\sqrt{n})$ elements from other computers.

— References

- 1 John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 69–79. ACM, 2019. doi:10.1145/3323165.3323195.
- 2 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 34(6):463-487, 2021. doi: 10.1007/S00446-020-00380-5.
- 3 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019. doi:10.1007/S00446-016-0270-2.
- 4 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In Yuval Emek and Christian Cachin, editors, PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020, pages 474–482. ACM, 2020. doi:10.1145/3382734.3405742.
- 5 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. In Jiannong Cao, Faith Ellen, Luís Rodrigues, and Bernardo Ferreira, editors, 22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China, volume 125 of LIPIcs, pages 4:1-4:17. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS. OPODIS.2018.4.
- 6 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *Journal of the ACM*, 68(3):21:1–21:36, 2021. doi:10.1145/3446330.
- 7 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In Peter Robinson and Faith Ellen, editors, Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019, pages 66–73. ACM, 2019. doi:10.1145/3293611.3331618.
- 8 Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. SIAM Journal on Computing, 15(1):87–97, 1986. doi:10.1137/0215006.
- Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In Marcos K. Aguilera, editor, Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings, volume 7611 of Lecture Notes in Computer Science, pages 195–209. Springer, 2012. doi:10.1007/978-3-642-33651-5_14.

- 10 Klaus-Tycho Foerster, Juho Hirvonen, Stefan Schmid, and Jukka Suomela. On the power of preprocessing in decentralized network optimization. In 2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 May 2, 2019, pages 1450–1458. IEEE, 2019. doi:10.1109/INFOCOM.2019.8737382.
- 11 Klaus-Tycho Foerster, Janne H. Korhonen, Joel Rybicki, and Stefan Schmid. Does preprocessing help under congestion? In Peter Robinson and Faith Ellen, editors, Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019, pages 259–261. ACM, 2019. doi:10.1145/3293611.3331581.
- 12 Chetan Gupta, Juho Hirvonen, Janne H. Korhonen, Jan Studený, and Jukka Suomela. Sparse matrix multiplication in the low-bandwidth model. In Kunal Agrawal and I-Ting Angelina Lee, editors, SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 14, 2022, pages 435–444. ACM, 2022. doi:10.1145/3490148. 3538575.
- 13 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389. ACM, 2017. doi:10.1145/3087801.3087811.
- 14 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CON-GEST. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, 21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017, volume 95 of LIPIcs, pages 4:1-4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.0PODIS.2017.4.
- 15 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 2016. doi:10.1007/978-3-662-53426-7_5.
- 16 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in o(log log n) communication rounds. In Arnold L. Rosenberg and Friedhelm Meyer auf der Heide, editors, SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003), pages 94–100. ACM, 2003. doi:10.1145/777412.777428.
- 17 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. *ACM Transactions on Parallel Computing*, 8(2):7:1–7:28, 2021. doi:10.1145/3460900.
- 18 Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In Nate Foster and Rob Sherwood, editors, Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013, pages 121–126. ACM, 2013. doi: 10.1145/2491185.2491198.
- Leslie G. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103-111, 1990. doi:10.1145/79173.79181.
- 20 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. Society for Industrial and Applied Mathematics, 2024. doi:10.1137/1.9781611977912.134.