Distributing Frequency-Dependent Data Stream Computations

Sumit Ganguly
sganguly@cse.iitk.ac.in
Indian Institute of Technology, Kanpur, India

Abstract

For time-efficiency, data stream computations are often performed in a highly distributed fashion (e.g., internet applications and sensor networks). A distributed computation is modeled as a binary tree, whose leaf nodes contain the input stream fragments $\sigma_i$, each of which is reduced to a message $\phi(\sigma_i)$ that is sent to its parent. Internal nodes compose the messages received from its children using a message composition function $\odot$ and relay the composite message upwards in the tree. Finally, the root node obtains a summary composite message of the input stream fragments and processes it to return an answer. A maximally flexible distributed algorithm is one where all possible computation trees over the same set of leaf nodes compute a correct answer, and each tree is identified with a distinct distributed computation. A basic question here is: what kind of data stream computations can be distributed in a flexible manner?

(Feldman et al. 2008) show that any $s(m)$-space, $c(m)$-communication complexity1 data stream algorithm that computes a total and symmetric function of its input stream can be used to simulate a maximally flexible distributed algorithm for computing the same function that uses $c(m)$-communication, $O((s(m))^2)$-space and $2^{O(s(m))}$-time for the message composition function.

We consider distributed computations for approximating total functions over the frequency vector of a data stream (e.g., estimating frequencies, frequent items, quantiles, etc.). We show that if there is a $c(m)$-communication data streaming algorithm for computing such a function, then there is a maximally flexible algorithm for computing the same approximation function over the sum of distributed $n$-dimensional vectors requiring at most $c(m)$-communication and $O(c(n))$-space and time for the message composition function. We also show the somewhat surprising result that given an apparently inflexible distributed algorithm (i.e., only one tree is known) for computing an arbitrary approximation to a function of the sum of distributed vectors, there exists a maximally flexible distributed algorithms for the same computation with the above mentioned relation between resource consumed between the two algorithms. This improves previous work in terms of the efficiency and generalization of simulation parameters.

1 Introduction

Modern distributed data-centric applications typically process massive amounts of data over a collection of nodes that are organized in the form of a tree. The applications include internet data processing and sensor networks, among others. An example is the class of applications that use the Google MAPREDUCE framework of scalable and flexible distributed processing as presented by (Dean & Ghemawat 2004). In this model, data resides at the hundreds or even thousands of nodes that form the leaf nodes of a depth one tree. Data at each of the nodes is locally reduced, and the reduced data is sent to the root site, which combines the locally reduced copies into a single copy and then computes an answer from it. In sensor networks, the sensors are organized in the form of a directed spanning tree and local data at nodes is reduced and combined up the tree. Many such practical algorithms are maximally flexible in the sense that all trees over the same set of leaf nodes may be used to compute the answer.

There is another practical model of data processing, namely, data stream processing, that also processes massive amounts of data, but, in a sequential, online fashion. Typically, a low space summary of the data stream is constructed and maintained with respect to newly arriving data or updates. Although the data streaming model is sequential, it is an oft-spoken virtue of data stream processing that typical stream summary structures are efficiently composable. By composability, we mean that there is a binary composition operation $\odot$ that takes instances of the summary structure that are maintained independently over distributed streams and combines them into a single instance whose state is the same as, or is equivalent to, the state of the summary structure after processing the concatenation of the distributed streams at a single site. The property of composable summaries makes it viable to have data-parallel distributed streaming computation with substantial flexibility, as illustrated by the following example.

Example 1. Consider a distributed computation consisting of $k$ nodes such that the input at each node is an $n$-dimensional vector $f_i$, $i = 1, 2, \ldots, k$. The problem is to design an efficient distributed algorithm to compute an approximation to the $l_2$ norm of the sum of the vectors $f_1 + \ldots + f_k$. This is possible using the randomized linear sketch technique by (Alon et al. 1998) for approximating the $l_2$ norm of a vector in the data stream model. Each site computes a random sketch of its input vector using the same random bits and sends it to a central site. The central site composes the sketches received from the sites, by simply adding them like vectors in the sketch space and then applies the $l_2$-approximation function on the composed sketches. The composition operation is time-efficient, as well as associative and com-
mutative. So any binary tree over the set of leaf nodes may be used to compute the answer, making the distributed computation highly flexible. □

The composability property of stream structures presents the possibility that sequential data stream execution can be molded into data-parallel execution over distributed streams. This was explored in the MUD model (acronym for Massive Unordered Distributed algorithms) presented by (Feldman et al. 2008). We first outline the data streaming model and then discuss the MUD model and the results by (Feldman et al. 2008).

1.1 Data Stream Model

A general data stream over the domain \([n]\) is modeled as a sequence of individual records of the form \((index, i, \delta)\), where, \(index\) represents the position of this record in the sequence, \(i \in [n]\) and \(\delta\) is either 1, or \(-1\), signifying, respectively, insertion or deletion of an instance of \(i\). The data streaming algorithm may be viewed as a pair of functions \(\oplus, output\). Here, \(\oplus\) is the configuration transition function of the streaming algorithm, that is, if the current configuration of the algorithm is \(s\) and the input is the sequence \(\sigma\), then, the configuration of the algorithm starting from \(s\) and after processing \(\sigma\) is \(s \circ \sigma\). The output function takes the current configuration and returns an output; the function output may take multiple arguments. The space and time requirement of a data stream algorithm are, respectively, the space and time requirement of the \(\oplus\) operation. The communication complexity of a data stream algorithm is defined as \([\log Q(m)]\) where, \(Q(m)\) is the set of reachable configurations of the stream algorithm for inputs of size at most \(m\). The communication complexity gives a lower bound on the number of bits that needs to be communicated in order to precisely convey the current configuration of the algorithm.

The frequency of \(i \in [n]\) in the stream \(\sigma\), denoted by \(f_i(\sigma)\), is defined as the number of current instances of \(i\), counting insertions and substracting deletions:

\[f_i(\sigma) = \sum_{(index, i, \delta)} \delta, \quad i \in [n].\]

The frequency vector \(f(\sigma)\) is the \(n\)-dimensional vector whose \(i\)th coordinate is the value \(f_i(\sigma)\). A popular class of data processing applications over data streams can be modeled as exact or approximate computations over the frequency vector of the stream. Examples of such functions include, finding frequent items in a data stream, finding the median/quantiles, approximating the frequency vector by a histogram, computing the \(\ell_p\) norms of the frequency vector, etc.

This class of functions are called frequency-dependent functions and denoted as \(\text{str}[\text{freq}]\).

1.2 The MUD Model

A MUD algorithm models distributed computation that is organized in the form of a binary operator tree \(T\), where, data resides only at the leaf nodes. We will view the data at the leaf nodes as a stream over \(\Sigma^+\) (i.e., a sequence from \(\Sigma^+\)). A MUD algorithm is represented as a triple \((\phi, \odot, \psi)\). Each leaf node \(v_l \in T\) applies the reduction function \(\phi: \Sigma^+ \rightarrow M\) to its local stream \(\sigma_l\) to obtain a message \(\phi(\sigma_l)\) that is sent to its parent. Each internal node applies the message composition function \(\odot : M \times M \rightarrow M\) to combine the messages received from its left and right children to produce a new summary message that is then conveyed to its parent. Let \(\psi(\sigma) \in M\) denote the data summary obtained at the root of the tree \(T\) at the end of this process, where, \(\sigma\) is the concatenation of the data streams in a left to right traversal of the leaf nodes of \(T\). The output of the computation \(\psi(\sigma)\) is then conveyed to its parent. Let \(\psi(T, \sigma)\) denote the output of the computation \(\psi(\sigma)\) at the root of the tree \(T\). The MUD algorithm is said to compute a function \(g(\sigma)\) provided

\[\psi(\sigma(T)) = g(\sigma), \quad \forall \, \text{trees } T \text{ over leaf sequence } \sigma.\]

In other words, the MUD model by (Feldman et al. 2008) admits only those algorithms whose output is independent of the tree \(T\). The space and time requirements of a MUD algorithm \(P = (\phi, \odot, \psi)\) are measured as the maximum of the respective requirements for the functions \(\phi\) and \(\odot\) respectively (the output function \(\psi\) is excluded). The communication is measured as the maximum number of bits sent from a child node to its parent, as a function of the input stream size.

Extending MUD. The MUD proposal does not allow general approximations, for example, MUD disallows an algorithm that computes a function over the frequency vector of its input stream such that the output of the algorithm is different for two different input streams, both having the same frequency vector and both satisfying approximation criterion. We extend the MUD proposal as follows. The approximate computation of a function \(g : \Sigma^+ \rightarrow O\) is specified by a binary approximation predicate \(\text{APPROX}(\hat{g}, g)\) that returns true if \(\hat{g}(\sigma)\) is an acceptable approximation to the exact value \(g(\sigma)\) and is false otherwise. A MUD algorithm \(P = (\phi, \odot, \psi)\) is said to approximately compute \(g : \Sigma^+ \rightarrow O\) with respect to \(\text{APPROX}\), if the output of the tree satisfies \(\text{APPROX}(P(T, \sigma), g(\sigma)) = \text{TRUE}\).

We will also relax MUD’s requirement that all binary trees over the ordered left to right sequence of leaf nodes must yield the same answer. This was required so that MUD algorithms would be highly flexible and implied that its corresponding streaming algorithm be symmetric, that is, it gives the same answer for all permutations of the input stream. We classify a MUD algorithm \(P = (\phi, \odot, \psi)\) as maximally flexible (or, indifferent), if \(\phi, \odot, \psi\) is a family of trees \(\{T_k\}_{k \geq 1}\), containing one tree \(T_k\) for each number \(k\) of leaf nodes and the output satisfies \(\text{APPROX}(\sigma(T_k), g(\sigma)) = \text{TRUE}\), for all \(k \geq 1\). and for all input streams \(\sigma\) partitioned left to right into \(k\) contiguous sub-streams, where, each sub-stream is given as input to one leaf node. A MUD algorithm \(P = (\phi, \odot, \psi)\) is maximally flexible if for any binary tree \(T\) of \(k\) leaf nodes, \(\text{APPROX}(\sigma(T), g(\sigma)) = \text{TRUE}\). The original MUD proposal allowed only maximally flexible algorithms.

A sub-class of MUD algorithms, called MUD[VECSUM]\(^1\) is defined as follows. Here, each leaf node \(v_l\) has an \(n\)-dimensional integer vector \(f_l\). The algorithm computes an approximation to some function \(g: \Sigma^+ \rightarrow O\) defined as the application of \(g\) to the sum of the distributed vectors, that is, \(g(f_1 + \ldots + f_k)\). The class MUD[VECSUM] contains interesting practical algorithms. For example, in a large network, there may be a collection of \(k\) distinct traffic matrix monitoring centers, corresponding to nodes. It is needed to aggregate the data by adding the matrices (or, vectors, tensors, etc.) coordinate-wise, and then, to perform an approximate computation over the resulting “global” matrix.

Results of (Feldman et al. 2008). Feldman et al. ask the following question. Suppose that there
is a data streaming algorithm that computes (or approximates) a function \( g \). When does this imply the existence of a maximally flexible MUD algorithm for computing \( g \) over distributed data streams, and, what are the resource bounds of such a MUD algorithm in terms of the resource bounds of the streaming algorithm? The main result of (Feldman et al. 2008) is given below and uses the following terminology. A function \( g : \Sigma^* \rightarrow O \) is said to be total if it is defined for all sequences in \( \Sigma^* \). It is said to be partial (also called a promise function) if it is only defined for some specific subset of \( \Sigma^* \). A function \( g : \Sigma^* \rightarrow O \) is said to be symmetric if \( g(\sigma) = g(\pi(\sigma)) \) for any permutation \( \pi \) over \( \sigma \), and any \( \sigma \in \Sigma^* \).

**Theorem 1** (Feldman et al. 2008). Every total data streaming algorithm that computes a symmetric function \( g \) over its input stream using space \( s(m) \), communication \( c(m) \) and time \( t(m) \) can be transformed into a maximally flexible MUD algorithm, that uses communication \( c(m) \), space \( O(s(m)^2) \) and time \( 2^{O(s(m))} \). Furthermore, there exist promise functions and streaming algorithms for such a function that require \( \log n \) space and communication and for which any maximally flexible MUD algorithm requires \( \Omega(m) \) communication.

In Theorem 1, the MUD algorithm corresponding to the streaming algorithm requires space \( (s(n))^2 \) and time \( 2^{O(s(m))} \). These are the space and time requirements respectively of the message composition function \( \circ \) of the MUD algorithm; the space requirement of the \( \phi \) function remains \( s(m) \), since the given data stream algorithm is used unchanged by each leaf node and the message formed is the configuration of the streaming algorithm. The quadratic space and exponential time complexity expression arise essentially due to the use of a Savitch-like simulation for the message composition operation. An important fact used in obtaining the simulation results of Theorem 1 is that the function that is being computed is symmetric, that is, invariant of the order of the input.

### 1.3 Contributions

We prove theoretical properties regarding simulation of MUD algorithms from data streaming algorithms\(^2\). Our first result presents an isomorphism between the resource usage of total algorithms for approximating frequency-dependent functions over data streams and computing the same approximate function over the sum of distributed vectors.

**Theorem 2** Given a communication \( c(m) \) total data streaming algorithm for approximating a function \( g : \mathbb{Z}^n \rightarrow O \) over the frequency vector of its input stream with respect to approximation predicate \( \text{APPROX} \), there is a maximally flexible MUD[VECSUM] algorithm \( P = (\phi, \circ, \psi) \) for approximating \( g \) over the sum of the vectors distributed over the leaves under the approximation predicate \( \text{APPROX} \) that requires \( c(m) \)-communication and whose message composition function \( \circ \) requires time and space \( O(c(m)) \).

Theorem 2 differs from Theorem 1 in several ways. First, it does not assume that the streaming algorithm that computes a general approximation (also called indeterminate approximation by (Feldman et al. 2008)) of a function \( g(f(\sigma)) \) is a symmetric

\[^2\]The converse, namely, the simulation of a data streaming algorithm from a MUD algorithm may be done as follows. Given \( P = (\phi, \circ, \psi) \), define \( s \oplus (index, v) = s \circ \phi(1, v) \) and \( output(s) = \psi(s) \).

\[ f(\sigma) = \sum_{(index,v)\in\sigma} v \cdot 
\]

The concatenation of two streams \( \sigma \) and \( \tau \) is denoted by \( \sigma \circ \tau \). The size of a data stream \( \sigma \) is defined as follows.

\[ |\sigma| = \max_{\sigma \text{′} \text{ sub-sequence of } \sigma} \|f(\sigma')\|_{\infty} .\]
A deterministic stream automaton is an abstraction for deterministic algorithms for processing data streams. It is defined as a two-tape Turing machine, where the first tape is a one-way (unidirectional) input tape that contains the sequence $\sigma$ of updates $\sigma$ that constitute the stream. Each update is a member of $\Sigma$, that is, it is an elementary vector or its inverse, $e_i$ or $-e_i$. The second tape is a (bidirectional) two way work-tape. A configuration of a stream automaton is modeled as a triple $(q, h, w)$, where, $q$ is a state of the finite control, $h$ is the current head position of the work-tape and $w$ is the content of the work-tape. The set of configurations of a stream automaton $A$ that are reachable from the initial configuration $o$ on some input stream is denoted as $C(A)$. The set of configurations of an automaton $A$ that is reachable from the origin $o$ for some input stream $\sigma$ with $|\sigma| \leq m$ is denoted by $C_m(A)$. A stream automaton may be viewed as a tuple $(n, C, o, \oplus, \psi)$, where, $n$ is a state of the finite control, $o$ is the state of the work-tape and $\psi$ is the content of the work-tape.

We present the basic theorem of stream automaton. For processing records of an input stream whose size is at most $n$, the space usage of approximating a total function $g : \mathbb{Z}^n \rightarrow O$ isupper bounded by $O((n - \dim M) \log m)$, where, $\dim M$ is the dimension of $M$.

(6.) Conversely, given any sub-module $M \subset \mathbb{Z}^n$, a stream automaton $A = (n, C_A, o_A, \oplus_A, \psi_A)$ can be constructed such that there is an isomorphic map $\varphi : C_A \rightarrow \mathbb{Z}^n/M$ such that for any stream $\sigma$

$$\varphi(a \oplus \tau) = \varphi(a) \bigoplus [f(\sigma)]$$

where, $x \mapsto [x]$ is the canonical homomorphism from $\mathbb{Z}^n$ to $\mathbb{Z}^n/M$ (that is, $[x]$ is the unique coset of $M$ to which $x$ belongs).

We generally write $\approx$ for a stream automaton $A = (n, C_A, o_A, \oplus_A, \psi_A)$ such that an estimate $\hat{f}(\sigma)$ is viewed as a module with binary addition operation $\bigoplus$, such that for any stream $\sigma$

$$\varphi(a \oplus \tau) = \varphi(a) \bigoplus [f(\sigma)]$$

where, $\bigoplus$ is the addition operation of $\mathbb{Z}^n/M$, and

$$\approx = O((n - \dim M) \log m)$$

Randomized stream automata can be defined as a deterministic stream automaton with one additional tape for the random bits. The random bit string $R$ is initialized on the random bit tape before any input record is read; thereafter the random bit string is used in a two way read-only manner. The rest of the execution of the finite control as before. We will say that a randomized stream automaton correctly computes an approximation given by the predicate $\approx$ to a function $g$ of the frequency vector of the input stream, provided, for all feasible streams $\sigma$

$$\approx(g(f(\sigma))) = \text{true}$$

for at least $3/4$ fraction of the possible values taken by the random bit string $R$.

### 3 MUD[VECSUM] Algorithm from STR[FREQ]

In this section, we show an isomorphism between the resource usage of approximating a total function $g : \mathbb{Z}^n \rightarrow O$ with respect to an approximation predicate $\approx$ as an MUD[VECSUM] algorithm and a data streaming algorithm from STR[FREQ]. We first define a sub-class of MUD[VECSUM] algorithms that we call reducible algorithms, and show its equivalence with data streaming computations, in the above sense. We then show that corresponding to any MUD algorithm for computing approximations to total functions of vector sums, one can construct a reducible algorithm for the same problem with the same or less communication.

**Notation:** Given a vector sequence $\tau = x_1, x_2, \ldots, x_k$ where the $x_j$’s belong to $\mathbb{Z}^n$, we use the following...
abbreviations. For any function $\phi : \mathbb{Z}^n \to D$, we extend the notation $\phi : (\mathbb{Z}^n)^* \to D^*$ as follows.

$$\phi(\tau) \overset{\text{denote}}{=} (\phi(x_1), \ldots, \phi(x_k)) .$$

For $\tau \in (\mathbb{Z}^n)^*$, let

$$\sum_{\tau} \overset{\text{denote}}{=} x_1 + x_2 + \ldots + x_k .$$

In the following definition, let $M$ be the set of messages and $O$ be the domain of output.

**Definition 1** A **MUD[VECSUM]** algorithm $Q = (\phi, \psi)$ is said to be reducible if there exists a symmetric function $\Upsilon : (\mathbb{Z}^n)^* \to M$ such that $\psi(\phi(\tau)) = \psi(\Upsilon(\tau))$ and $\circ(\phi(\tau)) = \Upsilon(\tau)$, for all $\tau \in (\mathbb{Z}^n)^*$, and,

$$\Upsilon(\tau) = \Upsilon(\sum_{\tau}) , \text{ and}$$

$$\forall \sigma, \tau, \tau' \in (\mathbb{Z}^n)^* \text{ s.t. } \Upsilon(\tau) = \Upsilon(\tau') . \square$$

Reducible algorithms are denoted by the pair $(\Upsilon, \psi)$. The $\Upsilon$ function essentially represents an aggregating function such that the vector of messages $\phi(\tau)$ is equivalent to the single message $\Upsilon(\tau)$. In addition, reducible MUD[VECSUM] algorithms are maximally flexible. This follows from the fact that $\Upsilon$ function is symmetric. For any tree $T$ with left to right sequence of input streams at the leaves $\sigma_1, \ldots, \sigma_t$, let $\sigma = \sigma_1 \circ \ldots \circ \sigma_t$ and let $x_i = f(\sigma_i)$. Then,

$$\circ_\tau(\sigma) = \circ(\phi(\sigma_1), \ldots, \phi(\sigma_t)) = \Upsilon(\sigma_1, \ldots, \sigma_t) = \Upsilon(\tau_1 + \ldots + \tau_t) .$$

Thus, all trees on the same set of leaf nodes give the same output, which implies that reducible MUD[VECSUM] algorithms are fully flexible.

The proof outline is as follows. We first show that any total data stream algorithm that approximates a function of the frequency vector of its input stream, that is, an algorithm in the STR[FREQ] class can always be used to design a reducible distributed streaming algorithm, that is, a maximally flexible distributed algorithm in the MUD[VECSUM] class, for computing the same approximation, using essentially no more resources than that used by the streaming algorithm. This proves the statement of Theorem 2.

The proof of Theorem 3 requires an additional step. We first prove the converse of the above statement, that is, a reducible distributed streaming algorithm for approximating a function of the sum of distributed vectors can be used to design a total data streaming algorithm for computing the same approximation to the function over the frequency vector of its input stream using, essentially, no more resources than that used by the given reducible algorithm. This essentially shows that the class of reducible MUD[VECSUM] algorithms and total data streaming algorithms STR[FREQ] are equivalent.

The problem now reduces to showing that corresponding to any MUD[VECSUM] algorithm for approximating a function of sum of distributed vectors, and one that is not necessarily reducible, it is possible to design a reducible MUD[VECSUM] algorithm for approximating the same function. This step is proved as follows. We define a MUDFLAT[VECSUM] distributed algorithm as a tree consisting of two levels, namely, the root and the leaves. Each leaf node reduces its input stream say $\sigma_1$ to the message $\phi(\sigma_1)$ and relays it to the root. The root node applies the message composition function $\circ(\phi(\sigma_1), \ldots, \phi(\sigma_t))$ and applies $\psi$ to this quantity. The MUDFLAT[VECSUM] distributed algorithm may in general be inflexible. Any MUD[VECSUM] algorithm may be viewed as a MUDFLAT algorithm, by simply aggregating the internal nodes of the tree into a single root. We then show that corresponding to any MUDFLAT algorithm, one can design a reducible algorithm for approximating the same function. The proof essentially proceeds by showing that by a module $K \subset \mathbb{Z}^n$ can be constructed from the MUDFLAT algorithm such that a total streaming algorithm from STR[FREQ] can be designed for computing the same approximation such that $K$ is the kernel of the automaton. By Theorem 2, a total streaming algorithm from STR[FREQ] implies the existence of a fully flexible MUD[VECSUM] algorithm. This proves Theorem 3.

### 3.1 STR[FREQ] gives reducible algorithms

**Lemma 1** Suppose there is a total data stream algorithm over the domain $[n]$ for approximating $g(f(\sigma))$ using communication complexity $c(m)$ bits. Then, there exists a reducible MUD[VECSUM] algorithm $P$ for approximating $g(f_1 + \ldots + f_k)$ with communication $c(nm)$.

**Proof:** Let $A = (n, C_A, o_A, \psi_A)$ be a total stream algorithm that approximates $g(f(\sigma))$. Then, by Theorem 4, there exists a stream automaton $B = (n, C_B, o_B, \psi_B)$ such that $\text{COMM}(A, m) \geq \text{COMM}(B, m)$ and there exists a sub-module $M$ of $\mathbb{Z}^n$ and an isomorphism $\varphi$ from the set of configurations of $B, C_B$ to the factor module $\mathbb{Z}^n/M, \oplus$ that preserves the transition function, that is,

$$\varphi(a \circ \sigma) = \varphi(a) \oplus [x] .$$

The communication algorithm $P = (\varphi_D, o_D, \psi_D)$ is as follows.

$$\phi_D(f) = \{f_1, \ldots, f_k\}, \circ_D([f_1], \ldots, [f_k]) = [f_1 + \ldots + f_k]$$

$$\psi_D([f]) = \psi_B(\psi^{-1}(B)) .$$

The algorithm is reducible, since, the $\Upsilon$ function is defined as $\{x_1, \ldots, x_k\} \mapsto [x_1 + \ldots + x_k]$. The correctness of $P$ follows from the commutative, associative addition operation $\oplus$ in a module. The communication from each party is equal to the communication required by $B$ which is $c(m)$ bits, by Theorem 4.

Since reducible algorithms are maximally flexible, Lemma 1 implies Theorem 2.

### 3.2 Reducibility yields STR[FREQ] algorithm

We now prove the converse of the Lemma 1. That is, we will show that any reducible MUD[VECSUM] algorithm for computing an approximation to some function $g$ of the sum of $n$-dimensional vectors that are distributed in the leaf nodes of a tree implies the existence of a total data streaming algorithm over the domain $[n]$ for computing the same approximation to $g$ over the frequency vector of its input stream. The symbols $\sigma, \tau, \rho$ etc. will be used to refer only to sequences of vectors in $\mathbb{Z}^n$, whereas, $x, y, z$ etc., will refer to vectors from $\mathbb{Z}^n$.

**Lemma 2** Suppose $g : \mathbb{Z}^n \to O$ is approximated by a reducible algorithm $P = (\Upsilon, \psi)$ with respect to the approximation predicate APPROX with communication $\text{COMM}(P, m)$. Then, there is a total stream automaton $B$ that approximates $g(f(\sigma))$ over any input stream $\sigma$ over $[n]$ with respect to APPROX with communication complexity at most $\text{COMM}(P, m)$ bits.
Proof: For \(x, y \in \mathbb{Z}^n\), define
\[
x R y \text{ if } \exists \sigma, \tau \in (\mathbb{Z}^n)^* \text{ s.t. } \sum \sigma = x, \sum \tau = y
\]
and \(\Upsilon(\tau) = \Upsilon(\sigma)\).

By construction, the relation \(R\) is reflexive and symmetric.

(1). Suppose \(x R y\). Then, there exist vector sequences \(\sigma, \tau\) such that \(\sum \sigma = x, \sum \tau = y\) and \(\Upsilon(\tau) = \Upsilon(\sigma)\). Since the algorithm is reducible, for any \(z \in \mathbb{Z}^n\), \(\Upsilon(x, \sigma, \tau, z) = \Upsilon(x, \sigma, \tau, z)\), or that \(x + z R y + z\) for any \(z \in \mathbb{Z}^n\). Setting \(z = -y\) we have, \(x \sim R 0\).

Let \(x R y\) and \(y R z\). Then, there exists vector sequences \(\sigma, \tau, \sigma'\) and \(\rho\) such that \(\sum \tau = x, \sum \sigma = \sum \sigma' = y\) and \(\sum \rho = z\) such that \(\Upsilon(\tau) = \Upsilon(\sigma)\) and \(\Upsilon(\sigma') = \Upsilon(\rho)\). By the previous paragraph, \(x R y\) implies \(x - y R 0\) and \(y R z\) implies \(y - z R 0\). By property of reducibility,
\[
\Upsilon(0) = \Upsilon(\tau, -\sigma, \sigma', -\rho) = \Upsilon(\tau, -\sigma, \sigma, \rho) = \Upsilon(\tau, -\rho)
\]
or that \(x - z R 0\). So, there exists \(\sigma'\) such that \(\sum \sigma' = x - z\) and \(\Upsilon(\sigma') = \Upsilon(0)\). Therefore,
\[
\Upsilon(x) = \Upsilon(\sigma', z) = \Upsilon(0, z) = \Upsilon(z)
\]
or that \(x R z\). Thus, \(R\) is an equivalence relation.

(2). Suppose \(x R z\). Then, \(\Upsilon(0, x - x) = \Upsilon(0, \tau) = \Upsilon(-x)\), or that \(-x R 0\). Suppose \(x R 0\) and \(y R 0\). Then, \(-x R 0\) and \(-y R 0\). Thus,
\[
\Upsilon(0) = \Upsilon(x + y, -x, -y) = \Upsilon(x + y, 0, 0) = \Upsilon(x + y)
\]
or, \(x + y R 0\).

Define the algorithm \(Q = (\psi, \psi', \sigma)\) as follows. Let \(K = [0]_{\mathbb{R}}\), that is, \(K\) is the equivalence class to which \(0\) belongs. By properties (1) and (2), \(K\) is a module. Define \(\psi' : \mathbb{Z}^n \to \mathbb{Z}^n/K\), where, \((\mathbb{Z}^n/K, +)\) is the quotient module whose elements are \([x] \mid x \in \mathbb{Z}^n\) and \([x]\) is the coset \(x + K\) to which \(x\) belongs. Let
\[
\circ([x_1], \ldots, [x_r]) = [x_1 + \ldots + x_r]
\]
and
\[
\psi'(x_1, \ldots, x_r) = \psi(\Upsilon(y)), \text{ for any } y \in \{x_1 + \ldots + x_r\}.
\]

We now prove the correctness of the algorithm \(P\). For any set of input vectors \(x_1, \ldots, x_k\), it is easy to see by induction on the height of the computation tree that the output is
\[
g_Q(T) = \psi'(x_1 + \ldots + x_k) = \psi(y)
\]
for some \(y \in \{x_1 + \ldots + x_k\}\). Thus, \(y R x_1 + \ldots + x_k\) and there exists \(\sigma, \tau \in (\mathbb{Z}^n)^*\) such that \(\sum \sigma = y, \sum \tau = x_1 + \ldots + x_k\) and \(\Upsilon(\tau) = \Upsilon(\sigma)\). Therefore,
\[
\Upsilon(x_1 + \ldots + x_k) = \Upsilon(\tau) = \Upsilon(\sigma) = \Upsilon(\sum \sigma) = \Upsilon(y).
\]

Therefore, the output of the tree \(T\) with input vectors \(x_1, \ldots, x_k\) under the algorithm \(Q\) is
\[
g_Q(T) = \psi'(x_1 + \ldots + x_k) = \psi(\Upsilon(y)) = \psi(\Upsilon(\tau)).
\]

Hence,
\[
\text{APPROX}(g_Q(T), g(x_1 + \ldots + x_k)) = \text{APPROX}(\psi(\Upsilon(\tau)), g(x_1 + \ldots + x_k))
\]
\[
= \text{APPROX}(\psi(\Upsilon(x_1 + \ldots + x_k)), g(x_1 + \ldots + x_k)) = \text{TRUE}
\]
since it is given that the reducible algorithm \(P = (\phi, T)\) satisfies the approximation predicate APPROX.

Since, \(\phi'(x) = \phi(y)\) if \([x] = [y]\), therefore,
\[
\text{comm}(Q, m) \geq \log \left| \phi'(\mathbb{Z}^n) \right|
\]
\[
\geq \log \left| \{x + K \mid x \in \mathbb{Z}^n \} \right|.
\]

The algorithm \(P = (\psi', \sigma)\) is converted into a stream automaton \(A = (n, \alpha, \mathbb{Z}^n/K, \odot, \psi_A)\) defined as
\[
[x] \odot \sigma = [x + \sum \sigma], \ [x] \in \mathbb{Z}^n/K
\]
and
\[
\psi_A([x]) = \psi(\Upsilon(y)), \text{ for some } y \in [x].
\]

Thus, by Theorem 4,
\[
\text{comm}(A, m) = \log \left| \{x + K \mid x \in \mathbb{Z}^n \} \right| \leq \text{comm}(Q, m).
\]

### 3.3 MUD[VECSUM] is essentially reducible

We now show that given any \(c(m)\)-communication, \(s(m)\)-space and \(t(m)\)-time MUD[VECSUM] algorithm for computing the approximation to a function \(q\) of the sum of distributed vectors, there exists a reducible MUD[VECSUM] algorithm requiring at most \(c(m)\)-communication, \(c(m) + O(\log m)\) space and \(O(c(m) + \log m)\) time. Essentially, this shows that without increasing the communication or space resource requirements, one may always assume MUD[VECSUM] algorithms to be maximally flexible.

In order to show this property, we introduce the MUDFLAT model. In the MUDFLAT model, the computation tree \(T\) is a two level tree consisting of the root and the leaves. The root is allowed to be have arbitrary arity. The leaf nodes contain the data, that yields the input sequence when traversed from left to right. A MUDFLAT algorithm is denoted by a pair \(Q = (\phi, \Upsilon, \psi)\). Each leaf node \(v_j\) applies the function \(\phi : \mathbb{Z}^n \to M\) to its input \(\sigma_i\) (say) to obtain \(\phi(\sigma_i)\) that is sent to the root. The root node applies the function \(\odot : M^s \to M\) to merge the messages into a single message as
\[
\odot(\phi(\sigma_i), \ldots, \phi(\sigma_k)).
\]

The output is obtained as
\[
P(\sigma_1, \ldots, \sigma_k) = \psi(\odot(\phi(\sigma_1), \ldots, \phi(\sigma_k))).
\]

The notion of approximation is defined with respect to an approximation predicate as before. A MUDFLAT[VECSUM] algorithm for the function \(q : \mathbb{Z}^n \to O\) assumes that the leaf nodes contain vectors from \(\mathbb{Z}^n\) and the algorithm approximates the function \(g\) applied to the sum of the vectors at the leaves.

**Lemma 3** Suppose \(P = (\phi, \Upsilon, \psi)\) is a MUDFLAT[VECSUM] algorithm that approximates \(g(f_1 + \ldots + f_k)\) with respect to a given approximation predicate APPROX. Then, there exists a reducible algorithm \(Q = (\psi', \Upsilon', \phi') = (\phi, \Upsilon', \phi')\) such that \(Q\) computes \(g\) approximately with respect to APPROX. Further, \(\text{comm}(Q, m) \leq \text{comm}(P, m)\) and the space and time requirement of \(\phi'\) function is \(O(\text{comm}(Q, m))\).

**Proof:** Let
\[
K = \{x - y \mid \phi(x) = \phi(y)\}.
\]
Suppose $\phi(x) = \phi(y)$. Given the input vector sequence $(x, -y)$, the central site $C$ receives $(\phi(x), \phi(-y))$, which is the same as $(\phi(y), \phi(-y))$, the latter being the message sequence received at $C$ for input vector sequence $(-y, y)$. Since,

$$\sum (x, -y) = x - y \text{ and } \sum (y, -y) = y - y = 0$$

it follows that

$$\psi \circ \phi(x, y) = \psi \circ \phi(y, y).$$

Here, $\psi \circ \phi$ is the composite function $(\psi \circ \phi)(t_1, ..., t_k) = \psi(\phi(t_1, ..., t_k))$. Therefore, the output $\psi \circ \phi(x, y)$ may be mapped under a new computation function $\psi' \circ \phi'(x, y) = \psi(\phi(0))$ while satisfying APPROX($\psi(\phi(0)), g(x - y)$).

Let $M = (K)$ be the ideal generated by $K$. Suppose $y \in M$. Then there exists $r$ such that $y = x_1 + x_2 + ... + x_r, x_i \in K$, for $1 \leq i \leq r$. Therefore,

$$\psi(\phi(x_1), ..., \phi(x_k)) = \psi(\phi(0), ..., \phi(0)).$$

Thus, the output may be considered to be the same as $\psi(\phi(0))$, since the sum is 0. Therefore, APPROX($\psi(\phi(0)), g(y)$) = $\psi(\phi(0))$ and $y$ could be mapped under $\phi(0)$ under $\phi'$. In a similar manner, it is argued that all elements of $M$ could be mapped to $\phi(0)$ under $\phi'$, that is, $\phi'(M) = \phi(0)$.

Consider a coset $x + M$ and let $y \in x + M$. By a similar argument as above, it may be inferred that

$$\text{APPROX}(\psi(\phi(0)), g(x - y)) = \text{true}.$$ 

Since $x - y$ is indistinguishable from 0, therefore, $x$ and $y$ could be mapped to the same image by $\phi'$ without error.

We can now construct the communication function $\psi'(x)$ of the new algorithm $Q' = (\phi', \circ', \psi')$. Define $\phi'(x)$ to be an encoding of the coset $x + M$. By definition, if $\phi(x) = \phi(y)$, then, $x - y \in K \subset M$ and therefore, $x + M = y + M$ and so, $\phi'(x) = \phi'(y)$. This implies that for any $m$,

$$\left| \{x : \|x\|_\infty \leq m\} \right| \leq \left| \{x + M : \|x\|_\infty \leq m\} \right| = \left| \{\phi'(x) : \|x\|_\infty \leq m\} \right|.$$ 

Therefore,

$$\text{COMM}(Q, m) \geq \log \left| \{x : \|x\|_\infty \leq m\} \right| \geq \log \left| \{\phi'(x) : \|x\|_\infty \leq m\} \right| = \text{COMM}(Q', m).$$

The function $\circ'$ is defined as: $\circ'(x_1 + M, x_2 + M) = x_1 + x_2 + M$ and is implemented as the sum of elements of the factor module. The space requirement is therefore $O(\text{COMM}(Q', m))$. The output function $\psi'(x + M)$ is set to $\psi(y)$, where $y$ is any member of $x + M$. This defines the MUD[VECSUM] algorithm $Q = (\psi', \circ, \psi')$. The algorithm $Q'$ is reducible with the reducing function $\tau(\tau)$ is $O(\tau) + M$. \hfill \Box

We can now prove Theorem 3.

**Proof [Of Theorem 3]:** Any MUD algorithm with computation along a tree $T$ with ordered leaf inputs $\sigma_1, ..., \sigma_k$ may be viewed as a MUDFLAT algorithm by aggregating the tree operator into a single operator $\psi'$. That is, for input sequence $\sigma_1, ..., \sigma_k$, define

$$\psi'(\phi(\sigma_1), ..., \phi(\sigma_k)) = \psi(\circ_T(\sigma_1, ..., \sigma_k)).$$

By Lemma 3, we know that corresponding to any MUDFLAT[VECSUM] algorithm for approximating a function of the sum of distributed vectors that uses communication $c(m)$, there is a reducible MUD[VECSUM] algorithm that computes the same approximation using communication $c(m)$. Further, the message composition function $\circ$ of the reducible MUD[VECSUM] algorithm requires space and time $O(\epsilon(m))$. Since, reducible algorithms are maximally flexible, this proves Theorem 3.

Remark. It would be interesting to know properties of the class of compressible $d \times n$ linear maps (matrices) $A$, such that all entries of $A$ can be obtained from a small seed. This would be help to characterize the space complexity of the $\phi$ function of the MUD[VECSUM] algorithms obtained in this work.

References


