# Time-Optimal Interactive Proofs for Circuit Evaluation

Justin Thaler[⋆]

Harvard University, School of Engineering and Applied Sciences.

**Abstract.** Several research teams have recently been working toward the development of practical general-purpose protocols for verifiable computation. These protocols enable a computationally weak *verifier* to offload computations to a powerful but untrusted *prover*, while providing the verifier with a guarantee that the prover performed the requested computations correctly. Despite substantial progress, existing implementations require further improvements before they become practical for most settings. The main bottleneck is typically the extra effort required by the prover to return an answer with a guarantee of correctness, compared to returning an answer with no guarantee.

We describe a refinement of a powerful interactive proof protocol due to Goldwasser, Kalai, and Rothblum [20]. Cormode, Mitzenmacher, and Thaler [14] show how to implement the prover in this protocol in time $O(S \log S)$, where $S$ is the size of an arithmetic circuit computing the function of interest. Our refinements apply to circuits with sufficiently "regular" wiring patterns; for these circuits, we bring the runtime of the prover down to $O(S)$. That is, our prover can evaluate the circuit with a guarantee of correctness, with only a constant-factor blowup in work compared to evaluating the circuit with no guarantee.

We argue that our refinements capture a large class of circuits, and we complement our theoretical results with experiments on problems such as matrix multiplication and determining the number of distinct elements in a data stream. Experimentally, our refinements yield a 200x speedup for the prover over the implementation of Cormode et al., and our prover is less than 10x slower than a C++ program that simply evaluates the circuit. Along the way, we describe a special-purpose protocol for matrix multiplication that is of interest in its own right.

Our final contribution is the design of an interactive proof protocol targeted at general data parallel computation. Compared to prior work, this protocol can more efficiently verify complicated computations as long as that computation is applied independently to many different pieces of data.

## 1 Introduction

Protocols for verifiable computation enable a computationally weak *verifier* $\mathcal{V}$ to offload computations to a powerful but untrusted *prover* $\mathcal{P}$. These protocols aim to provide the verifier with a guarantee that the prover performed the requested computations correctly, without requiring $\mathcal{V}$ to perform the computations herself.

Surprisingly powerful protocols for verifiable computation were discovered within the computer science theory community several decades ago, in the form of interactive proofs (IPs) and their brethren, interactive arguments (IAs) and probabilistically checkable proofs (PCPs). In these protocols, the prover $\mathcal{P}$ solves a problem using her

[⋆] Supported by an NSF Graduate Research Fellowship and NSF grants CNS-1011840 and CCF-0915922.

(possibly vast) computational resources, and tells $\mathcal{V}$ the answer. $\mathcal{P}$ and $\mathcal{V}$ then engage in a randomized protocol involving the exchange of one or more messages. During this exchange, $\mathcal{P}$'s goal is to convince $\mathcal{V}$ that the answer is correct.

Results quantifying the power of IPs, IAs, and PCPs are some of the most celebrated in all of computational complexity theory, but until recently they were mainly of theoretical interest, far too inefficient for actual deployment. In fact, the main applications of these results have traditionally been in negative applications – showing that many problems are just as hard to approximate as they are to solve exactly.

However, the surging popularity of cloud computing has brought renewed interest in positive applications of protocols for verifiable computation. A typical motivating scenario is as follows. A business processes billions or trillions of transactions a day. The volume is sufficiently high that the business cannot or will not store and process the transactions on its own. Instead, it offloads the processing to a commercial cloud computing service. The offloading of any computation raises issues of trust: the business may be concerned about relatively benign events like dropped transactions, buggy algorithms, or uncorrected hardware faults, or the business may be more paranoid and fear that the cloud operator is deliberately deceptive or has been externally compromised. Either way, each time the business poses a query to the cloud, the business may demand that the cloud also provide a guarantee that the returned answer is correct.

This is precisely what protocols for verifiable computation accomplish, with the cloud acting as the prover in the protocol, and the business acting as the verifier. In this paper, we describe a refinement of an existing general-purpose protocol originally due to Goldwasser, Kalai, and Rothblum [14, 20]. When they are applicable, our techniques achieve asymptotically optimal runtime for the prover, and we demonstrate that they yield protocols that are significantly closer to practicality than prior work.

We also make progress toward addressing another issue of existing interactive proof implementations: their applicability. The protocol of Goldwasser et al. (henceforth the GKR protocol) applies in principle to any problem computed by a small-depth arithmetic circuit, but this is not the case when more fine-grained considerations of prover and verifier efficiency are taken into account. In brief, existing implementations of interactive proof protocols for circuit evaluation require that the circuit have a highly regular wiring pattern [14, 37]. If this is not the case, then these implementations require the verifier to perform an expensive (though data-independent) preprocessing phase to pull out information about the wiring of the circuit, and they require a substantial factor blowup (logarithmic in the circuit size) in runtime for the prover relative to evaluating the circuit without a guarantee of correctness. Developing a protocol that avoids these pitfalls and applies to more general computations remains an important open question.

Our approach is the following. We do not have a magic bullet for dealing with irregular wiring patterns: if we want to avoid an expensive pre-processing phase for the verifier and minimize the blowup in runtime for the prover, we do need to make an assumption about the structure of the circuit we are verifying. Acknowledging this, we ask whether there is some general structure in real-world computations that we can leverage for efficiency gains.

To this end, we design a protocol that is highly efficient for data parallel computation. By data parallel computation, we mean any setting in which one applies the same computation independently to many pieces of data. Many outsourced computations are data parallel, with Amazon Elastic MapReduce[1] being one prominent example of a cloud computing service targeted specifically at data parallel computations. Crucially, we do not want to make significant assumptions on the sub-computation that is being applied, and in particular we want to handle sub-computations computed by circuits with highly irregular wiring patterns.

The verifier in our protocol still has to perform an offline phase to pull out information about the wiring of the circuit, but the cost of this phase is proportional to the size of a *single* instance of the sub-computation, avoiding any dependence on the number of pieces of data to which the sub-computation is applied. Similarly, the blowup in runtime suffered by the prover is the same as it would be if the prover had run the basic GKR protocol on a single instance of the sub-computation.

Our final contribution is to describe a new protocol specific to matrix multiplication that is of interest in its own right. It avoids circuit evaluation entirely, and reduces the overhead of the prover (relative to running *any* unverifiable algorithm) to an additive low-order term. A major message of our results is that the more structure that exists in a computation, the more efficiently it can be verified, and that this structure exists in many real-world computations.

## 1.1 Prior Work

**Work on Interactive Proofs.** Goldwasser, Kalai, and Rothblum described a powerful general-purpose interactive proof protocol in [20]. This protocol is framed in the context of *circuit evaluation*. Given a layered arithmetic circuit $C$ of depth $d$, size $S(n)$, and fan-in 2, the GKR protocol allows a prover to evaluate $C$ with a guarantee of correctness in time $\text{poly}(S(n))$, while the verifier runs in time $\tilde{O}(n + d \log S(n))$, where $n$ is the length of the input and the $\tilde{O}$ notation hides polylogarithmic factors in $n$.

Cormode, Mitzenmacher, and Thaler showed how to bring the runtime of the prover in the GKR protocol down from $\text{poly}(S(n))$ to $O(S(n) \log S(n))$ [14]. They also built a full implementation of the protocol and ran it on benchmark problems. These results demonstrated that the protocol does indeed save the verifier significant time in practice (relative to evaluating the circuit locally); they also demonstrated surprising scalability for the prover, although the prover's runtime remained a major bottleneck. With the implementation of [14] as a baseline, Thaler et al. [35] described a parallel implementation of the GKR protocol that achieved 40x-100x speedups for the prover and 100x speedups for the (already fast) implementation of the verifier.

Vu, Setty, Blumberg, and Walfish [37] further refine and extend the implementation of Cormode et al. [14]. In particular, they combine the GKR protocol with a compiler from a high-level programming language so that programmers do not have to explicitly express computation in the form of arithmetic circuits as was the case in the implementation of [14]. This substantially extends the reach of the implementation, but it should be noted that their approach generates circuits with irregular wiring patterns, and hence only works in a *batching* model, where the cost of a fairly expensive offline setup phase is amortized by verifying many instances of a single computation in batch. They also

[1] `http://aws.amazon.com/elasticmapreduce/`

build a hybrid system that statically evaluates whether it is better to use the GKR protocol or a different, cryptography-based argument system called Zaatar (see Section 1.1), and runs the more efficient of the two protocols in an automated fashion.

A growing line of work studies protocols for verifiable computation in the context of *data streaming*. In this context, the goal is not just to save the verifier time (compared to doing the computation without a prover), but also to save the verifier space. The protocols developed in this line of work allow the client to make a single streaming pass over the input (which can occur, for example, while the client is uploading data to the cloud), keeping only a very small summary of the data set. The interactive version of this model was introduced by Cormode, Thaler, and Yi [15], who observed that many protocols from the interactive proofs literature, including the GKR protocol, can be made to work in this restrictive setting. The observations of [15] imply that all of our protocols also work with streaming verifiers. Non-interactive variants of the streaming interactive proofs model have also been studied in detail [12, 13, 22, 25].

**Work on Argument Systems.** There has been a lot of work on the development of efficient interactive arguments, which are essentially interactive proofs that are secure only against dishonest provers that run in polynomial time. A substantial body of work in this area has focused on the development of protocols targeted at specific problems (e.g. [2, 5, 16]). Other works have focused on the development of general-purpose argument systems. Several papers in this direction (e.g. [8, 10, 11, 18]) have used fully homomorphic encryption, which unfortunately remains impractical despite substantial recent progress. Work in this category by Chung et al. [10] focuses on streaming settings, and is therefore particularly relevant.

Several research teams have been pursuing the development of general-purpose argument systems that might be suitable for practical use. Theoretical work by Ben-Sasson et al. [4] focuses on the development of short PCPs that might be suitable for use in practice – such PCPs can be compiled into efficient interactive arguments. As short PCPs are often a bottleneck in the development of efficient argument systems, other works have focused on avoiding their use [3, 6, 7, 19]. In particular, Gennaro et al. [19] and Bitansky et al. [9] develop argument systems with a clear focus on implementation potential. Very recent work by Parno et al. [28] describes a near-practical general-purpose implementation, called Pinocchio, of an argument system based on [19]. Pinocchio is additionally non-interactive and achieves public verifiability.

Another line of implementation work focusing on general-purpose interactive argument systems is due to Setty et al. [31–33]. This line of work begins with a base argument system due to Ishai et al. [23], and substantially refines the theory to achieve an implementation that approaches practicality. The most recent system in this line of work is called Zaatar [33], and is also based on the work of Gennaro et al. [19]. An empirical comparison of the GKR-based approach and Zaatar performed by Vu et al. [37] finds the GKR approach to be significantly more efficient for quasi-straight-line computations (e.g. programs with relatively simple control flow), while Zaatar is appropriate for programs with more complicated control flow.

## 1.2 Our Contributions

Our primary contributions are three-fold. Our first contribution addresses one of the biggest remaining obstacles to achieving a truly practical implementation of the GKR protocol: the logarithmic factor overhead for the prover. That is, Cormode et al. show how to implement the prover in time $O(S(n) \log S(n))$, where $S(n)$ is the size of the arithmetic circuit to which the GKR protocol is applied, down from the $\Omega(S(n)^3)$ time required for a naive implementation. The hidden constant in the Big-Oh notation is at least 3, and the $\log S(n)$ factor translates to well over an order of magnitude, even for circuits with a few million gates. We remove this logarithmic factor, bringing $\mathcal{P}$'s runtime down to $O(S(n))$ for a large class of circuits. Informally, our results apply to any circuit whose wiring pattern is sufficiently "regular". We formalize the class of circuits to which our results apply in Theorem 1.

We experimentally demonstrate the generality and effectiveness of Theorem 1 via two case studies. Specifically, we apply an implementation of the protocol of Theorem 1 to a circuit computing matrix multiplication (MATMULT), as well as to a circuit computing the number of distinct items in a data stream (DISTINCT). Experimentally, our refinements yield a 200x speedup for the prover over the state of the art implementation of Cormode et al. [14]. A serial implementation of our prover is less than 10x slower than a C++ program that simply evaluates the circuit sequentially, a slowdown that is likely tolerable in realistic outsourcing scenarios where cycles are plentiful for the prover.

Our second contribution is to specify a highly efficient protocol for verifiably outsourcing arbitrary data parallel computation. Compared to prior work, this protocol can more efficiently verify complicated computations, as long as that computation is applied independently to many different pieces of data. We formalize this protocol and its efficiency guarantees in Theorem 2.

Our third contribution is to describe a new protocol specific to matrix multiplication that we believe to be of interest in its own right. This protocol is formalized in Theorem 3. Given any *unverifiable* algorithm for $n \times n$ matrix multiplication that requires time $T(n)$ using space $s(n)$, Theorem 3 allows the prover to run in time $T(n) + O(n^2)$ using space $s(n) + o(n^2)$. Note that Theorem 3, which is specific to matrix multiplication, is much less general than Theorem 1, which applies to any circuit with a sufficiently regular wiring pattern. However, Theorem 3 achieves optimal runtime and space usage for the prover up to leading constants, assuming there is no $O(n^2)$ time algorithm for matrix multiplication. While these properties are also satisfied by a classic protocol due to Freivalds [17], the protocol of Theorem 3 is significantly more amenable for use as a primitive when verifying computations that repeatedly invoke matrix multiplication. We complement Theorem 3 with experimental results demonstrating its extreme efficiency.

Do to space constraints, full proofs of are omitted from this extended abstract, and can be found in the full version of the paper.

## 2 Preliminaries

We begin by defining a valid interactive proof protocol for a function $f$.

**Definition 1.** *Consider a prover $\mathcal{P}$ and verifier $\mathcal{V}$ who wish to compute a function $f : \{0,1\}^n \to \mathcal{R}$ for some set $\mathcal{R}$. After the input is observed, $\mathcal{P}$ and $\mathcal{V}$ exchange a sequence of messages. Denote the output of $\mathcal{V}$ on input x, given prover $\mathcal{P}$ and $\mathcal{V}$'s random bits R, by out($\mathcal{V}, x, R, \mathcal{P}$). $\mathcal{V}$ can output $\perp$ if $\mathcal{V}$ is not convinced that $\mathcal{P}$'s claim is valid. We say $\mathcal{P}$ is a* valid prover *with respect to $\mathcal{V}$ if for all inputs x, $Pr_R[out(\mathcal{V}, x, R, \mathcal{P}) = f(x)] =$*

1. *The property that there is at least one valid prover $\mathcal{P}$ with respect to $\mathcal{V}$ is called* completeness. *We say $\mathcal{V}$ is a* valid verifier *for $f$ with* soundness probability $\delta$ *if there is at least one valid prover $\mathcal{P}$ with respect to $\mathcal{V}$, and for all provers $\mathcal{P}'$ and inputs $x$, $Pr[out(\mathcal{V}, A, R, \mathcal{P}') \notin \{f(x), \perp\}] \leq \delta$. A prover-verifier pair $(\mathcal{P}, \mathcal{V})$ is a* valid interactive proof protocol *for $f$ if $\mathcal{V}$ is a valid verifier for $f$ with soundness probability $1/3$, and $\mathcal{P}$ is a valid prover with respect to $\mathcal{V}$. If $\mathcal{P}$ and $\mathcal{V}$ exchange $r$ messages, we say the protocol has $\lceil r/2 \rceil$ rounds.*

Informally, the completeness property guarantees that an honest prover will convince the verifier that the claimed answer is correct, while the soundness property ensures that a dishonest prover will be caught with high probability. An *interactive argument* is an interactive proof where the soundness property holds only against polynomial-time provers $\mathcal{P}'$. We remark that the constant $1/3$ used for the soundness probability in Definition 1 is chosen for consistency with the interactive proofs literature, where $1/3$ is used by convention. In our actual implementation, the soundness probability will always be less than $2^{-45}$.

**Cost Model.** Whenever we work over a finite field $\mathbb{F}$, we assume that a single field operation can be computed in a single machine operation. For example, when we say that the prover $\mathcal{P}$ in our interactive protocols requires time $O(S(n))$, we mean that $\mathcal{P}$ must perform $O(S(n))$ additions and multiplications within the finite field over which the protocol is defined.

**Input Representation.** Following prior work [12, 14, 15], all of the protocols we consider can handle inputs specified in a general data stream form. Each element of the stream is a tuple $(i, \delta)$, where $i \in [n]$ and $\delta$ is an integer. The $\delta$ values may be negative, thereby modeling deletions. The data stream implicitly defines a frequency vector $a$, where $a_i$ is the sum of all $\delta$ values associated with $i$ in the stream. When checking the evaluation of a circuit $C$, we consider the inputs to $C$ to be the entries of the frequency vector $a$. We emphasize that in all of our protocols, $\mathcal{V}$ only needs to see the raw stream and not the aggregated frequency vector $a$. Notice that we may interpret the frequency vector $a$ as an object other than a vector, such as a matrix or a string. For example, in MATMULT, the data stream defines two matrices to be multiplied.

When we refer to a *streaming verifier* with space usage $s(n)$, we mean that the verifier can make a single pass over the stream of tuples defining the input, regardless of their ordering, while storing at most $s(n)$ elements in the finite field over which the protocol is defined.

**Problem Definitions.** To focus our discussion, we give special attention to two problems also considered in prior work [14,28,31–33,35,37]. In the MATMULT problem, the input consists of two $n \times n$ matrices $A, B \in \mathbb{Z}^{n \times n}$, and the goal is to compute the matrix product $A \cdot B$. In the DISTINCT problem, the input is a data steam consisting of $m$ tuples $(i, \delta)$ from a universe of size $n$. The stream defines a frequency vector $a$, and the goal is to compute $|\{i : a_i \neq 0\}|$, the number of items with non-zero frequency.

**Additional Notation.** Let $\mathbb{F}$ be a field. For any $d$-variate polynomial $p(x_1, \ldots, x_d) : \mathbb{F}^d \to \mathbb{F}$, we use $\deg_i(p)$ to denote the degree of $p$ in variable $i$. A $d$-variate polynomial $p$ is said to be *multilinear* if $\deg_i(p) = 1$ for all $i \in [d]$. Given a function $V : \{0,1\}^d \to$

$\{0,1\}$ whose domain is the $d$-dimensional Boolean hypercube, the *multilinear extension* (MLE) of $V$ over $\mathbb{F}$, denoted $\tilde{V}$, is the unique multilinear polynomial $\mathbb{F}^d \to \mathbb{F}$ that agrees with $V$ on all Boolean-valued inputs, i.e., $\tilde{V}(x) = V(x)$ for all $x \in \{0,1\}^d$.

## 3 Time-Optimal Protocols for Circuit Evaluation

### 3.1 Technical Background

**Sum-Check Protocol.** Our main technical tool is the well-known sum-check protocol of Lund et al. [27], and we briefly describe this protocol and summarize the properties that are most important in our analysis. Suppose we are given a $v$-variate polynomial $g$ defined over a finite field $\mathbb{F}$, such that $\deg_i(g) = O(1)$ for all $i \in \{1,\dots,v\}$. The purpose of the sum-check protocol is to compute the sum:

$$H := \sum_{b_1\in\{0,1\}} \sum_{b_2\in\{0,1\}} \cdots \sum_{b_v\in\{0,1\}} g(b_1,\dots,b_v).$$

The protocol proceeds in $v$ rounds as follows. In the first round, the prover sends a polynomial $g_1(X_1)$, and claims that $g_1(X_1) = \sum_{x_2,\dots,x_v\in\{0,1\}^{v-1}} g(X_1,x_2,\dots,x_v)$. Observe that if $g_1$ is as claimed, then $H = g_1(0)+g_1(1)$. Also observe that the polynomial $g_1(X_1)$ has degree $\deg_1(g) = O(1)$. Hence $g_1$ can be specified by sending the evaluation of $g$ at each point in the $O(1)$-sized set $\{0,1,\dots,\deg_1(g)\}$.

Then, in round $j > 1$, $\mathcal{V}$ chooses a value $r_{j-1}$ uniformly at random from $\mathbb{F}$ and sends $r_{j-1}$ to $\mathcal{P}$. We refer to this step by saying that variable $j-1$ gets *bound* to value $r_{j-1}$. In return, the prover sends a polynomial $g_j(X_j)$, and claims that

$$g_j(X_j) = \sum_{(x_{j+1},\dots,x_v)\in\{0,1\}^{v-j}} g(r_1,\dots,r_{j-1},X_j,x_{j+1},\dots,x_v). \tag{1}$$

The verifier then checks that $g_{j-1}(r_{j-1}) = g_j(0)+g_j(1)$, rejecting otherwise.

In the final round, the prover has sent $g_v(X_v)$ which is claimed to be $g(r_1,\dots,r_{v-1},X_v)$. $\mathcal{V}$ now checks that $g_v(r_v) = g(r_1,\dots,r_v)$. Notice that in order to perform this check, the verifier needs to be able to evaluate $g(r_1,\dots,r_v)$ without assistance from the prover. If this test succeeds, and so do all previous tests, then the verifier accepts, and is convinced that $H = g_1(0)+g_1(1)$.

**Discussion of costs.** For our purposes, the key cost of the sum-check protocol is the prover's runtime. Notice that the number of terms defining the value $g_j(i)$ in Equation (1) falls geometrically with $j$: in the $j$th message, there are only $2^{v-j}$ terms. The total number of terms that must be evaluated over the course of the protocol is therefore $O\left(\sum_{j=1}^{v} 2^{v-j}\right) = O(2^v)$. Consequently, if $\mathcal{P}$ is given oracle access to (evaluations of) the polynomial $g$, then $\mathcal{P}$ will require $O(2^v)$ time. Unfortunately, in our applications $\mathcal{P}$ will not have oracle access to $g$. The key to our results is to show that in our applications $\mathcal{P}$ can nonetheless evaluate $g$ at the necessary points in $O(2^v)$ total time.

**The GKR Protocol at a Glance.** In the GKR protocol, $\mathcal{P}$ and $\mathcal{V}$ first agree on an arithmetic circuit $C$ of fan-in 2 over a finite field $\mathbb{F}$ computing the function of interest ($C$ may have multiple outputs). Each gate of $C$ performs an addition or multiplication over $\mathbb{F}$. $C$ is assumed to be in layered form, meaning that the circuit can be decomposed into layers, and wires only connect gates in adjacent layers. Suppose the circuit has depth

$d$; we will number the layers from 1 to $d$ with layer $d$ referring to the input layer, and layer 1 referring to the output layer.

In the first message, $\mathcal{P}$ tells $\mathcal{V}$ the (claimed) output of the circuit. The protocol then works its way in iterations towards the input layer, with one iteration devoted to each layer. The purpose of iteration $i$ is to reduce a claim about the values of the gates at layer $i$ to a claim about the values of the gates at layer $i+1$, in the sense that it is safe for $\mathcal{V}$ to assume that the first claim is true as long as the second claim is true. This reduction is accomplished by applying the sum-check protocol to a certain polynomial $f^{(i)}$.

More concretely, the GKR protocol starts with a claim about the values of the output gates of the circuit, but $\mathcal{V}$ cannot check this claim without evaluating the circuit herself, which is precisely what we want to avoid. So the first iteration uses a sum-check protocol to reduce this claim about the outputs to a claim about the gate values at layer 2 (more specifically, to a claim about an evaluation of the multilinear extension (MLE) of the gate values at layer 2). Once again, $\mathcal{V}$ cannot check this claim herself, so the second iteration uses another sum-check protocol to reduce the latter claim to a claim about the gate values at layer 3, and so on. Eventually, $\mathcal{V}$ is left with a claim about the inputs to the circuit, and $\mathcal{V}$ can check this claim on her own.

In summary, the GKR protocol uses a sum-check protocol at each level of the circuit to enable $\mathcal{V}$ to go from verifying a randomly chosen evaluation of the MLE of the gate values at layer $i$ to verifying a (different) evaluation of the MLE of the gate values at layer $i+1$. Importantly, apart from the input layer and output layer, $\mathcal{V}$ does not ever see all of the gate values at a layer. Instead, $\mathcal{V}$ relies on $\mathcal{P}$ to do the hard work of actually evaluating the circuit, and uses the power of the sum-check protocol to force $\mathcal{P}$ to be consistent and truthful over the course of the protocol.

**Further Details.** Suppose we are given a layered arithmetic circuit $C$ of depth $d$ and fan-in two. Let $S_i$ denote the number of gates at layer $i$ of the circuit $C$. Assume $S_i$ is a power of 2 and let $S_i = 2^{s_i}$. To explain how each iteration of the GKR protocol proceeds, we must introduce several functions, each of which encodes certain information about the circuit. Number the gates at layer $i$ from 0 to $S_i - 1$, and let $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ denote the function that takes as input a binary gate label, and outputs the corresponding gate's value at layer $i$. The GKR protocol makes use of the multilinear extension $\tilde{V}_i$ of the function $V_i$.

The GKR protocol also makes use of the notion of a "wiring predicate" that encodes which pairs of wires from layer $i+1$ are connected to a given gate at layer $i$ in $C$. We define two functions, $\text{add}_i$ and $\text{mult}_i$ mapping $\{0,1\}^{s_i+2s_{i+1}}$ to $\{0,1\}$, which together constitute the wiring predicate of layer $i$ of $C$. Specifically, these functions take as input three gate labels $(j_1, j_2, j_3)$, and return 1 if gate $j_1$ at layer $i$ is the addition (respectively, multiplication) of gates $j_2$ and $j_3$ at layer $i+1$, and return 0 otherwise. Let $\tilde{\text{add}}_i$ and $\tilde{\text{mult}}_i$ denote the multilinear extensions of $\text{add}_i$ and $\text{mult}_i$ respectively. Finally, let $\beta_{s_i}(z,p)$ denote the function $\beta_{s_i}(z,p) = \prod_{j=1}^{s_i}\left((1-z_j)(1-p_j) + z_j p_j\right)$. It can be shown that for any $z \in \mathbb{F}^{s_i}$,

$$\tilde{V}_i(z) = \sum_{(p,\omega_1,\omega_2)\in\{0,1\}^{s_i+2s_{i+1}}} f^{(i)}(p,\omega_1,\omega_2), \text{where}$$

$$f^{(i)}(p,\omega_1,\omega_2) = \beta_{s_i}(z,p)\cdot\left(\tilde{\text{add}}_i(p,\omega_1,\omega_2)(\tilde{V}_{i+1}(\omega_1)+\tilde{V}_{i+1}(\omega_2)) + \tilde{\text{mult}}_i(p,\omega_1,\omega_2)\tilde{V}_{i+1}(\omega_1)\cdot\tilde{V}_{i+1}(\omega_2)\right).$$

Iteration $i$ begins with a claim by $\mathcal{P}$ about the value of $\tilde{V}_i(z)$ for some $z \in \mathbb{F}^{s_i}$. In order to verify this claim, the sum-check protocol is applied to the polynomial $f^{(i)}$. However, $\mathcal{V}$ can only execute her part of the sum-check protocol if she can evaluate the polynomial $f^{(i)}$ at a random point $f^{(i)}(r_1, \ldots, r_{s_i+2s_{i+1}})$. In particular, this requires evaluating $\tilde{V}_{i+1}(\omega_2^*)$, and $\tilde{V}_{i+1}(\omega_1^*)$, but $\mathcal{V}$ cannot perform these evaluations on her own without evaluating the circuit. At a high level, $\mathcal{V}$ instead asks $\mathcal{P}$ to simply tell her these two values, and uses iteration $i+1$ to verify that these values are as claimed. The full version of the paper spells out the remaining details.

### 3.2 Achieving Optimal Prover Runtime for Regular Circuits

In Theorem 1 below, we describe a protocol for circuit evaluation that brings $\mathcal{P}$'s runtime down to $O(S(n))$ for a large class of circuits, while maintaining the same verifier runtime as in prior implementations of the GKR protocol. Informally, Theorem 1 applies to any circuit whose wiring pattern is sufficiently "regular".

Our protocol follows the same outline as the GKR protocol, in that we proceed in iterations from the output layer of the circuit to the input layer, using a sum-check protocol at iteration $i$ to reduce a claim about the gate values at layer $i$ to a claim about the gate values at layer $i+1$. However, at each iteration $i$ we apply the sum-check protocol to a carefully chosen polynomial that differs from the ones used in prior work [14, 20]. In each round $j$ of the sum-check protocol, our choice of polynomial allows $\mathcal{P}$ to reuse work from prior rounds in order to compute the prescribed message for round $j$, allowing us to shave a $\log S(n)$ factor from the runtime of $\mathcal{P}$ relative to the $O(S(n) \log S(n))$-time implementation due to Cormode et al. [14].

Specifically, at iteration $i$, the polynomial $f^{(i)}$ that is used in the GKR protocol is defined over $\log S_i + 2 \log S_{i+1}$ variables, where $S_i$ is the number of gates at layer $i$. The "truth table" of $f^{(i)}$ is sparse on the Boolean hypercube, in the sense that $f^{(i)}(x)$ is non-zero for at most $S_i$ of the $S_i \cdot S_{i+1}^2$ inputs $x \in \{0,1\}^{\log S_i + 2 \log S_{i+1}}$. Cormode et al. leverage this sparsity to bring the runtime of $\mathcal{P}$ in iteration $i$ down to $O(S_i \log S_i)$ from a naive bound of $\Omega(S_i \cdot S_{i+1}^2)$. However, this same sparsity prevents $\mathcal{P}$ from reusing work from prior iterations as we seek to do.

In contrast, we use a polynomial $g^{(i)}$ defined over only $\log S_i$ variables rather than $\log S_i + 2 \log S_{i+1}$ variables. Moreover, the truth table of $g^{(i)}$ is dense on the Boolean hypercube, in the sense that $g^{(i)}(x)$ may be non-zero for all of the $S_i$ Boolean inputs $x \in \{0,1\}^{\log S_i}$. This density allows $\mathcal{P}$ to reuse work from prior iterations in order to speed up her computation in round $i$ of the sum-check protocol.

In more detail, in each round $j$ of the sum-check protocol, the prover's prescribed message is defined via a sum over a large number of terms, where the number of terms falls geometrically fast with the round number $j$. Moreover, it can be shown that in each round $j$, each gate at layer $i+1$ contributes to exactly one term of this sum [14]. Essentially, what we do is group the gates at layer $i+1$ by the term of the sum to which they contribute. Each such group can be treated as a single unit, ensuring that in any round of the sum-check protocol, the amount of work $\mathcal{P}$ needs to do is proportional to the number of terms in the sum rather than the number of gates $S_i$ at layer $i$.

**Formal Statement.** Our protocol makes use of the following functions that capture the wiring structure of an arithmetic circuit $C$.

**Definition 2.** *Let $C$ be a layered arithmetic circuit of depth $d(n)$ and size $S(n)$ over finite field $\mathbb{F}$. For every $i \in \{1,\ldots,d-1\}$, let $in_1^{(i)} : \{0,1\}^{s_i} \to \{0,1\}^{s_{i+1}}$ and $in_2^{(i)} : \{0,1\}^{s_i} \to \{0,1\}^{s_{i+1}}$ denote the functions that take as input the binary label $p$ of a gate at layer $i$ of $C$, and output the binary label of the first and second in-neighbor of gate $p$ respectively. Similarly, let $type^{(i)} : \{0,1\}^{s_i} \to \{0,1\}$ denote the function that takes as input the binary label $p$ of a gate at layer $i$ of $C$, and outputs 0 if $p$ is an addition gate, and 1 if $p$ is a multiplication gate.*

Intuitively, the following two definitions capture functions whose outputs are simple bit-wise transformations of their inputs.

**Definition 3.** *Let $f$ be a function mapping $\{0,1\}^v$ to $\{0,1\}^{v'}$. Number the $v$ input bits from $1$ to $v$, and the $v'$ output bits from $1$ to $v'$. We say that $f$ is regular if $f$ can be evaluated on any input in constant time, and there is a subset of input bits $\mathcal{S} \subseteq [v]$ with $|\mathcal{S}| = O(1)$ such that:*

1. *Each input bit in $[v] \setminus \mathcal{S}$ affects $O(1)$ of the output bits of $f$. Moreover, for any $j \in [v] \setminus \mathcal{S}$, the set $\mathcal{S}_j$ of output bits affected by the $j$th input bit can be enumerated in constant time.*
2. *Each output bit of $f$ depends on at most one input bit.*

**Definition 4.** *We say that $in_1^{(i)}$ and $in_2^{(i)}$ are* similar *if there is a set of output bits $\mathcal{T} \subseteq [s_{i+1}]$ with $|\mathcal{T}| = O(1)$ such that for all inputs $x$, the $j$th output bit of $in_1^{(i)}$ equals the $j$th output bit of $in_2^{(i)}$ for all $j \in [s_{i+1}] \setminus \mathcal{T}$.*

**Theorem 1.** *Let $C$ be an arithmetic circuit, and suppose that for all layers $i$ of $C$, $in_1^{(i)}$, $in_2^{(i)}$, and $type^{(i)}$ are regular. Suppose moreover that $in_1^{(i)}$ is similar to $in_2^{(i)}$ for all but $O(1)$ layers $i$ of $C$. Then there is a valid interactive proof protocol $(\mathcal{P},\mathcal{V})$ for the function computed by $C$, with the following costs. The total communication cost is $|\mathcal{O}| + O(d(n)\log S(n))$ field elements, where $|\mathcal{O}|$ is the number of outputs of $C$. The time cost to $\mathcal{V}$ is $O(n\log n + d(n)\log S(n))$, and $\mathcal{V}$ can make a single streaming pass over the input, storing $O(\log(S(n)))$ field elements. The time cost to $\mathcal{P}$ is $O(S(n))$.*

The asymptotic costs of the protocol whose existence is guaranteed by Theorem 1 are identical to those of the implementation of the GKR protocol due to Cormode et al. in [14], except that in Theorem 1 $\mathcal{P}$ runs in time $O(S(n))$ rather than $O(S(n)\log S(n))$. While the conditions of Theorem 1 may appear unnatural, our techniques in fact capture a large class of circuits. Theorem 1 applies for example to circuits computing naive $n \times n$ matrix multiplication (MATMULT), computing the number of distinct items in a data stream (DISTINCT), pattern matching (which is useful, e.g., for searching email data stored in the cloud), and FFTs. To the best of our our knowledge Theorem 1 yields the fastest known prover among all interactive proof protocols for DISTINCT and for pattern matching with sublinear space and communication costs. More importantly, we will leverage the techniques underlying Theorem 1 to achieve our improved protocol for data parallel computation described in Theorem 2.

**Experimental Results.** We implemented the protocols implied by Theorem 1 as applied to circuits computing MATMULT and DISTINCT. The circuits are over the field $\mathbb{F}_q$ with

$q = 2^{61} - 1$. The soundness probability in all cases is less than $2^{-45}$ (this probability is proportional to $\frac{d(n)\log S(n)}{q}$). These experiments serve as case studies to demonstrate the feasibility of Theorem 1 in practice, and to quantify the improvements over prior implementations. While Section 5 describes a specialized protocol for MATMULT that is more efficient than the protocol implied by Theorem 1, MATMULT serves as an important case study for the costs of the more general protocol described in Theorem 1, and allows for direct comparison with prior implementation work that also evaluated general-purpose protocols via their performance on the MATMULT problem [14, 28, 32, 33, 35, 37].

The main takeaways of our experiments are as follows. When Theorem 1 is applicable, the prover in the resulting protocol is 200x-250x faster than the previous state of the art implementation of the GKR protocol, and is just 5x-10x times slower than a C++ program that simply evaluates the circuit with no correctness guarantee. The communication costs and the number of rounds required by our protocols are also 2x-3x smaller than the previous state of the art. The verifier in our implementation takes essentially the same amount of time as in prior implementations of the GKR protocol; this time is much smaller than the time to perform the computation locally without a prover. See Table 1 for detailed results – in this table, our comparison point is the implementation of Cormode et al. [14], with some of the refinements of Vu et al. [37] included.

Most of the 200x speedup can be attributed directly to our improvements in protocol design over prior work: the circuit for 512x512 matrix multiplication is of size $2^{28}$, and hence our $\log S(n)$ factor improvement the runtime of $\mathcal{P}$ likely accounts for at least a 28x speedup. The 3x reduction in the number of rounds accounts for another 3x speedup. The remaining speedup factor of roughly 2x may be due to a more streamlined implementation relative to prior work, rather than improved protocol design per se.

| Problem | Implementation | Problem Size | $\mathcal{P}$ Time | $\mathcal{V}$ Time | Rounds | Total Communication | Circuit Eval Time |
|---|---|---|---|---|---|---|---|
| MATMULT | Previous state of the art | 512 x 512 | 9759 s | 0.10 s | 767 | 17.97 KBs | 6.07 s |
| MATMULT | Theorem 1 | 512 x 512 | 37.85 s | 0.10 s | 236 | 5.48 KBs | 6.07 s |
| DISTINCT | Previous state of the art | $n = 2^{20}$ | 3400 s | 0.20 s | 3916 | 91.3 KBs | 1.88 s |
| DISTINCT | Theorem 1 | $n = 2^{20}$ | 17.28 s | 0.20 s | 1361 | 40.76 KBs | 1.88 s |

**Table 1.** Experimental results for Theorem 1. For the MATMULT problem, the Total Communication column does not count the communication required to specify the answer.

## 4 Verifying General Data Parallel Computations

Theorem 1 only applies to circuits with regular wiring patterns, as do other existing implementations of interactive proof protocols for circuit evaluation [14, 37]. For circuits with irregular wiring patterns, these implementations require the verifier to perform an expensive preprocessing phase (requiring time proportional to the size of the circuit) to pull out information about the wiring of the circuit, and they require a substantial factor blowup (logarithmic in the circuit size) in runtime for the prover relative to evaluating the circuit without a guarantee of correctness.

To address these bottlenecks, we do need to make an assumption about the structure of the circuit we are verifying. Ideally our assumption will be satisfied by many real-world computations. To this end, Theorem 2 below describes a protocol that is highly efficient for any data parallel computation, by which we mean any setting in which the same sub-computation is applied independently to many pieces of data, before possibly aggregating the results. We do not want to make significant assumptions on the sub-computation that is being applied (in particular, we want to handle sub-computations computed by circuits with irregular wiring patterns), but we are willing to assume that the sub-computation is applied to many pieces of data.

For example, Theorem 2 applies to arbitrary *counting queries* on a database. In a counting query, one applies some function independently to each row of the database and sums the results. For instance, one may ask "How many people in the database satisfy Property $P$?" Our protocol allows one to verifiably outsource such a counting query with overhead that depends minimally on the size of the database, but that necessarily depends on the complexity of the property $P$.

**Overview of the Protocol.** Let $C$ be a circuit of size $S$ with an arbitrary wiring pattern, and let $C^*$ be a "super-circuit" that applies $C$ independently to $B$ different inputs before possibly aggregating the results in some fashion. For example, in the case of a counting query, the aggregation phase simply sums the results of the data parallel phase. We assume that the aggregation step is sufficiently simple that the aggregation itself can be verified using existing techniques such as the basic GKR protocol or Theorem 1, and we focus on verifying the data parallel part of the computation. For instance, in the case of a counting query, the aggregation phase simply sums the outputs, and this is easily handled via Theorem 1. We stress that our protocol applies even if there is no aggregation phase; in this case $\mathcal{P}$ will begin the protocol by sending $\mathcal{V}$ all outputs of $C^*$, and the protocol can then be used to prove the validity of those outputs.

If we naively apply the GKR protocol to the super-circuit $C^*$, $\mathcal{V}$ might have to perform an expensive pre-processing phase to evaluate the wiring predicate of $C^*$ at the necessary locations – this would require time $\Omega(B \cdot S)$. Moreover, when applying the basic GKR protocol to $C^*$, $\mathcal{P}$ would require time $\Theta(B \cdot S \cdot \log(B \cdot S))$. A different approach was taken by Vu et al [37], who applied the GKR protocol $B$ independent times, once for each copy of $C$. This causes both the communication cost and $\mathcal{V}$'s online check time to grow linearly with $B$, the number of sub-computations.

In contrast, our protocol achieves the best of both prior approaches. We observe that although each sub-computation $C$ can have a very complicated wiring pattern, the super-circuit $C^*$ is maximally regular between sub-computations, as the sub-computations do not interact at all. Therefore, each time the basic GKR protocol would apply the sum-check protocol to a polynomial derived from the wiring predicate of $C^*$, we instead use a simpler polynomial derived only from the wiring predicate of $C$. By itself, this is enough to ensure that $\mathcal{V}$'s pre-processing phase requires time only $O(S)$, rather than $O(B \cdot S)$ as in a naive application of the GKR protocol to $C^*$. That is, the cost of $\mathcal{V}$'s pre-processing phase in our protocol is proportional to the cost of applying the basic GKR protocol only to $C$, not to $C^*$.

Furthermore, by combining this observation with the ideas underlying Theorem 1, we can bring the runtime of $\mathcal{P}$ down to $O(B \cdot S \cdot \log S)$. That is, the blowup in runtime suffered by the prover, relative to performing the computation without a guarantee of correctness, is just a factor of $\log S$ – the same as it would be if the prover had run the basic GKR protocol on a *single* instance of the sub-computation.

**Notation.** Let $C$ be an arithmetic circuit over $\mathbb{F}$ of depth $d$ and size $S$ with an arbitrary wiring pattern, and let $C^*$ be the circuit of depth $d$ and size $B \cdot S$ obtained by laying $B$ copies of $C$ side-by-side, where $B = 2^b$. We will use the same notation as in Section 3.1, using $^*$'s to denote quantities referring to $C^*$. For example, layer $i$ of $C$ has size $S_i = 2^{s_i}$ and gate values specified by the function $V_i$, while layer $i$ of $C^*$ has size $S_i^* = 2^{s_i^*}$ and gate values specified by the function $V_i^*$. We denote the length of the input to $C^*$ by $n^*$.

We assume at the start of our protocol that $\mathcal{P}$ has made a claim about $\tilde{V}_1^*(z)$ for some $z \in \mathbb{F}^{s_1^*}$, in the sense that it is safe for $\mathcal{V}$ to believe $\mathcal{P}$ has followed the prescribed protocol as long as $\tilde{V}_1^*(z)$ is as claimed. Such a claim about $\tilde{V}_1^*(z)$ would be obtained by first applying existing verification techniques such as Theorem 1 to the aggregation phase of the data parallel computation.

**Theorem 2.** *For any point $z \in \mathbb{F}^{s_1^*}$, there is a valid interactive proof protocol for computing $\tilde{V}_1^*(z)$ with the following costs. $\mathcal{V}$ spends $O(S)$ time in a pre-processing phase, and $O(n^* \log n^* + d \cdot \log(B \cdot S))$ time in an online verification phase. $\mathcal{P}$ runs in total time $O(S \cdot B \cdot \log S)$. The total communication is $O(d \cdot \log(B \cdot S))$ field elements.*

*Proof sketch*: Consider layer $i$ of $C^*$. Let $p = (p_1, p_2) \in \{0,1\}^{s_i} \times \{0,1\}^b$ be the binary label of a gate at layer $i$ of $C^*$, where $p_2$ specifies which "copy" of $C$ the gate is in, while $p_1$ designates the label of the gate within the copy. Similarly, let $\omega = (\omega_1, \omega_2) \in \{0,1\}^{s_{i+1}} \times \{0,1\}^b$ and $\gamma = (\gamma_1, \gamma_2) \in \{0,1\}^{s_{i+1}} \times \{0,1\}^b$ be the labels of two gates at layer $i+1$. It is straightforward to check that for all $(p_1, p_2) \in \{0,1\}^{s_i} \times \{0,1\}^b$,

$$V_i^*(p_1, p_2) = \sum_{\omega_1 \in \{0,1\}^{s_{i+1}}} \sum_{\gamma_1 \in \{0,1\}^{s_{i+1}}} g^{(i)}(p_1, p_2, \omega_1, \gamma_1), \text{where } g^{(i)}(p_1, p_2, \omega_1, \gamma_1) \text{ is defined as:}$$

$$\beta_{s_i^*}(z, (p_1, p_2)) \cdot \left( \tilde{\text{add}}_i(p_1, \omega_1, \gamma_1) \left( \tilde{V}_{i+1}^*(\omega_1, p_2) + \tilde{V}_{i+1}^*(\gamma_1, p_2) \right) + \tilde{\text{mult}}_i(p_1, \omega_1, \gamma_1) \left( \tilde{V}_{i+1}^*(\omega_1, p_2) \cdot \tilde{V}_{i+1}^*(\gamma_1, p_2) \right) \right)$$

Essentially, the above says that a gate $p = (p_1, p_2) \in \{0,1\}^{s_i + b}$ is connected to gates $\omega = (\omega_1, \omega_2) \in \{0,1\}^{s_{i+1}+b}$ and $\gamma = (\gamma_1, \gamma_2) \in \{0,1\}^{s_{i+1}+b}$ if and only if $p$, $\omega$, and $\gamma$ are all in the same copy of $C$, and $p$ is connected to $\omega$ and $\gamma$ within the copy. The above derivation can be shown to imply that for any $z \in \mathbb{F}^{s_i^*}$,

$$\tilde{V}_i^*(z) = \sum_{(p_1, p_2, \omega_1, \gamma_1) \in \{0,1\}^{s_i} \times \{0,1\}^b \times \{0,1\}^{s_{i+1}} \times \{0,1\}^{s_{i+1}}} g^{(i)}(p_1, p_2, \omega_1, \gamma_1).$$

Thus, in iteration $i$ of our protocol, we apply the sum-check protocol to $g^{(i)}$. This reduces $\mathcal{P}$'s claim about $\tilde{V}_i^*(z)$ to a claim about $\tilde{V}_{i+1}^*(z')$ for some $z' \in \mathbb{F}^{s_{i+1}^*}$, exactly as in the $i$th iteration of the GKR protocol.

**Costs for $\mathcal{V}$.** The bottleneck in $\mathcal{V}$'s runtime is that, in the last round of the sum-check protocol, $\mathcal{V}$ must evaluate $g^{(i)}$ at a single point. This requires evaluating $\beta_{s_i^*}$, $\tilde{\text{add}}_i$, $\tilde{\text{mult}}_i$, and $\tilde{V}_{i+1}^*$ at a constant number of points. The $\tilde{V}_{i+1}^*$ evaluations are provided by $\mathcal{P}$ in all iterations $i$ of the protocol except the last. The bottleneck in the evaluation is the $\tilde{\text{add}}_i$ and $\tilde{\text{mult}}_i$ computations. These can be done in pre-processing in time $O(S_i)$ by enumerating the in-neighbors of each of the $S_i$ gates at layer $i$ [14, 37]. Adding up the pre-processing time across all iterations $i$ of our protocol, $\mathcal{V}$'s pre-processing time is $O(\sum_i S_i) = O(S)$ as claimed.

**Costs for $\mathcal{P}$.** Notice $g^{(i)}$ is a polynomial in $v := s_i + 2s_{i+1} + b$ variables. We order the sum in this sum-check protocol so that the $s_i + 2s_{i+1}$ variables in $p_1$, $\omega_1$, and $\gamma_1$ are bound first in arbitrary order, followed by the variables of $p_2$. $\mathcal{P}$ can compute the prescribed messages in the first $s_i + 2s_{i+1} = O(\log S)$ rounds exactly as in the implementation of Cormode et al. [14], who show that each gate at layers $i$ and $i+1$ of $C^*$ contributes to exactly one term in the sum defining $\mathcal{P}$'s message in any given round of the sum-check protocol. Hence the total time required by $\mathcal{P}$ to handle these rounds is $O(B \cdot (S_i + S_{i+1}) \cdot \log S)$.

It remains to show how $\mathcal{P}$ can compute the prescribed messages in the final $b$ rounds of the sum-check protocol while investing $O((S_i + S_{i+1}) \cdot B)$ across all rounds of the protocol. The idea is that once the variables of $p_1$, $\omega_1$, and $\gamma_1$ are bound, the truth table of $g^{(i)}$, viewed as a function of the unbound variables, is dense on the Boolean hypercube, in the sense of Section 3.2. We therefore exploit the reuse-of-work techniques underlying Theorem 1 to achieve the desired runtime for the prover.

## 5 Optimal Space and Time Costs for MATMULT

In Theorem 3 below, we describe a special-purpose protocol for $n \times n$ MATMULT in Theorem 3. The idea behind this protocol is as follows. The GKR protocol, as well the protocols of Theorems 1 and 2, only make use of the multilinear extension $\tilde{V}_i$ of the function $V_i$ mapping gate labels at layer $i$ of the circuit to their values. In some cases, there is something to be gained by using a higher-degree extension of $V_i$. This is precisely what we exploit here. In more detail, our special-purpose protocol can be viewed as an extension of our circuit-checking techniques applied to a circuit $C$ performing naive matrix multiplication, but using a quadratic extension of the gate values in this circuit. This allows us to verify the computation using a single invocation of the sum-check protocol. More importantly, $\mathcal{P}$ can evaluate this higher-degree extension at the necessary points without explicitly materializing all of the gate values of $C$, which would not be possible if we had used the multilinear extension of the gate values of $C$.

In the protocol of Theorem 3, $\mathcal{P}$ just needs to compute the correct output (possibly using an algorithm that is much more sophisticated than naive matrix multiplication), and then perform $O(n^2)$ additional work to prove the output is correct. We obtain the $O(n^2)$ bound on the extra work required by $\mathcal{P}$ by exploiting the reuse-of-work technique underlying Theorems 1 and 2.

Since $\mathcal{P}$ does not have to evaluate $C$ in full, this protocol is perhaps best viewed outside the lens of circuit evaluation. Still, the idea underlying Theorem 3 extends those underlying our circuit evaluation protocols, and we believe similar ideas may yield further improvements to general-purpose protocols in the future.

**Theorem 3.** *There is a valid interactive proof protocol for $n \times n$ matrix multiplication over the field $\mathbb{F}_q$ with the following costs. The communication cost is $n^2 + O(\log n)$ field elements. The runtime of the prover is $T(n) + O(n^2)$ and the space usage is $s(n) + o(n^2)$, where $T(n)$ and $s(n)$ are the time and space requirements of any (unverifiable) algorithm for $n \times n$ matrix multiplication. The verifier can make a single streaming pass over the input as well as over the claimed output in time $O(n^2 \log n)$, storing $O(\log n)$ field elements.*

## 5.1 Comparison to Prior Work

It is worth comparing Theorem 3 to a well-known protocol due to Freivalds [17]. Let $D^*$ denote the claimed output matrix. In Freivalds' algorithm, the verifier stores a random vector $x \in \mathbb{F}^n$, and computes $D^*x$ and $ABx$, accepting if and only if $ABx = D^*x$. Freivalds showed that this is a valid protocol. In both Freivalds' protocol and that of Theorem 3, the prover runs in time $T(n) + O(n^2)$ (in the case of Freivalds' algorithm, the $O(n^2)$ term is 0), and the verifier runs in linear or quasilinear time. We now highlight several properties of our protocol that are not achieved by prior work.

**Utility as a Primitive.** A major advantage of Theorem 3 relative to prior work is its utility as a primitive that can be used to verify more complicated computations. This is important as many algorithms repeatedly invoke matrix multiplication as a subroutine. For concreteness, consider the problem of computing $A^{2^k}$ via repeated squaring. By iterating the protocol of Theorem 3 $k$ times, we obtain a valid interactive proof protocol for computing $A^{2^k}$ with communication cost $n^2 + O(k \log(n))$. The $n^2$ term is due simply to specifying the output $A^{2^k}$, and can often be avoided in applications – see for example the diameter protocol described two paragraphs hence. The $i$th iteration of the protocol for computing $A^{2^k}$ reduces a claim about an evaluation of the multilinear extension of $A^{2^{k-i+1}}$ to an analogous claim about $A^{2^{k-i}}$. Crucially, the prover in this protocol never needs to send the verifier the intermediate matrices $A^{2^{k'}}$ for $k' < k$. In contrast, applying Freivalds' algorithm to this problem would require $O(kn^2)$ communication, as $\mathcal{P}$ must specify each of the intermediate matrices $A^{2^i}$.

The ability to avoid having $\mathcal{P}$ explicitly send intermediate matrices is especially important in settings in which an algorithm repeatedly invokes matrix multiplication, but the desired output of the algorithm is smaller than the size of the matrix. In these cases, it is not necessary for $\mathcal{P}$ to send *any* matrices; $\mathcal{P}$ can instead send just the desired output, and $V$ can use Theorem 3 to check the validity of the output with only a polylogarithmic amount of additional communication. This is analogous to how the verifier in the GKR protocol can check the values of the output gates of a circuit without ever seeing the values of the interior gates of the circuit.

As a concrete example illustrating the power of our matrix multiplication protocol, consider the fundamental problem of computing the diameter of an unweighted (possibly directed) graph $G$ on $n$ vertices. Let $A$ denote the adjacency matrix of $G$, and let $I$ denote the $n \times n$ identity matrix. Then it is easily verified that the diameter of $G$ is the least positive number $d$ such that $(A+I)^d_{ij} \neq 0$ for all $(i, j)$. We therefore obtain the following natural protocol for diameter. $\mathcal{P}$ sends the claimed output $d$ to $V$, as well as an $(i, j)$ such that $(A+I)^{d-1}_{ij} = 0$. To confirm that $d$ is the diameter of $G$, it suffices for $\mathcal{V}$ to check two things: first, that all entries of $(A+I)^d$ are non-zero, and second that $(A+I)^{d-1}_{ij}$ is indeed non-zero.

The first task is accomplished by combining our matrix multiplication protocol of Theorem 3 with our DISTINCT protocol from Theorem 1. Indeed, let $d_j$ denote the $j$th bit in the binary representation of $d$. Then $(A+I)^d = \prod_j^{\lceil \log d \rceil} (A+I)^{2^j}$, so computing the number of non-zero entries of $(A+I)^d$ can be treated as a sequence of $O(\log d)$ matrix multiplications, followed by a DISTINCT computation. The second task, of verifying that $(A+I)^{d-1}_{ij}=0$, is similarly accomplished using $O(\log d)$ invocations of the matrix multiplication protocol of Theorem 3 – since $\mathcal{V}$ is only interested in one entry of $(A+I)^{d-1}$, $\mathcal{P}$ need not send the matrix $(A+I)^{d-1}$ in full, and the total communication here is just polylog$(n)$.

$\mathcal{V}$'s runtime in this diameter protocol is $O(m\log n)$, where $m$ is the number of edges in $G$. $\mathcal{P}$'s runtime in the above diameter protocol matches the best known unverifiable diameter algorithm up to a low-order additive term [30, 38], and the communication is just polylog$(n)$. We know of no other protocol achieving this.

In many settings, practitioners will not tolerate even a 2x slowdown to achieve verifiability, so the fact that $\mathcal{P}$'s slowdown is a low-order additive term is critical. Moreover, for a graph with $n=1$ million nodes, the total communication cost of the above protocol would be on the order of KBs – in contrast, if $\mathcal{P}$ had to send the matrices $(I+A)^d$ or $(I+A)^{d-1}$ explicitly (as required in prior work, e.g., Cormode et al. [13]), the communication cost would be at least $n^2=10^{12}$ words of communication, which translates to terabytes of data.

**Small-Space Streaming Verifiers.** In Freivalds' algorithm, $\mathcal{V}$ has the store the random vector $x$, which requires $\Omega(n)$ space. There are methods to reduce $\mathcal{V}$'s space usage by generating $x$ with limited randomness: Kimbrel and Sinha [24] show how to reduce $\mathcal{V}$'s space to $O(\log n)$, but their solution does not work if $\mathcal{V}$ must make a streaming pass over arbitrarily ordered input. Chakrabarti et al. [12] extend the method of Kimbrel and Sinha to work with a streaming verifier, but this requires $\mathcal{P}$ to play back the input matrices $A,B$ in a special order, increasing proof length to $3n^2$. Our protocol works with a streaming verifier using $O(\log n)$ space, and our proof length is $n^2+O(\log n)$, where the $n^2$ term is due to specifying $AB$ and can be avoided in applications such as the diameter example considered above.

## 5.2 Protocol Details

When multiplying matrices $A$ and $B$ such that $AB=D$, let $A(i,j)$, $B(i,j)$ and $D(i,j)$ denote functions from $\{0,1\}^{\log n}\times\{0,1\}^{\log n}\to\mathbb{F}_q$ that map input $(i,j)$ to $A_{ij}$, $B_{ij}$, and $D_{ij}$ respectively. Let $\tilde{A}$, $\tilde{B}$, and $\tilde{D}$ denote their multilinear extensions.

**Lemma 1.** *For all* $(p_1,p_2)\in\mathbb{F}^{\log n}\times\mathbb{F}^{\log n}$*,*

$$\tilde{D}(p_1,p_2)=\sum_{p_3\in\{0,1\}^{\log n}}\tilde{A}(p_1,p_3)\cdot\tilde{B}(p_3,p_2)$$

*Proof.* For all $(p_1,p_2)\in\{0,1\}^{\log n}\times\{0,1\}^{\log n}$, the right hand side is easily seen to equal $D(p_1,p_2)$, using the fact that $D_{ij}=\sum_k A_{ik}B_{kj}$ and the fact that $\tilde{A}$ and $\tilde{B}$ agree with the functions $A(i,j)$ and $B(i,j)$ at all Boolean inputs. Moreover, the right hand side is a multilinear polynomial in the variables of $(p_1,p_2)$. Putting these facts together implies that the right hand side is the unique multilinear extension of the function $D(i,j)$.

Lemma 1 implies the following valid interactive proof protocol for matrix multiplication: $\mathcal{P}$ sends a matrix $D^*$ claimed to equal the product $D=AB$. $\mathcal{V}$ evaluates $\tilde{D}^*(r_1,r_2)$ at a random point $(r_1,r_2)\in\mathbb{F}^{\log n}\times\mathbb{F}^{\log n}$. It can be shown that it is safe for $\mathcal{V}$ to believe $D^*$ is as claimed, as long as $\tilde{D}^*(r_1,r_2)=\tilde{D}(r_1,r_2)$. In order to check that $\tilde{D}^*(r_1,r_2)=\tilde{D}(r_1,r_2)$, we invoke a sum-check protocol on the polynomial $g(p_3)=\tilde{A}(r_1,p_3)\cdot\tilde{B}(p_3,r_2)$.

$\mathcal{V}$'s final check in this protocol requires her to compute $g(r_3)$ for a random point $r_3 \in \mathbb{F}^{\log n}$. $\mathcal{V}$ can do this by evaluating both of $\tilde{A}(r_1,r_3)$ and $\tilde{B}(r_3,r_2)$ with a single streaming pass over the input, and then multiplying the results. The prover can be made to run in time $T(n)+O(n^2)$ across all rounds of the sum-check protocol using the reuse-of-work technique underlying Theorem 1. Moreover, the space requirements of $\mathcal{P}$ are just $s(n)+o(n^2)$.

**Implementation.** We implemented the protocol of Theorem 3 over the field with $q = 2^{61}-1$ elements. The results are shown in Table 2, where the column labelled "Additional Time for $\mathcal{P}$" denotes the time required to compute $\mathcal{P}$'s prescribed messages after $\mathcal{P}$ has already computed the correct answer. We report the naive matrix multiplication time both when the computation is done using standard multiplication of 64-bit integers, as well as when the computation is done using finite field arithmetic over $\mathbb{F}_q$. The main takeaways from Table 2 are that the verifier does indeed save substantial time relative to performing matrix multiplication locally, and that the runtime of the prover is hugely dominated by the time required simply to compute the answer.

| Problem Size | Naive Matrix Multiplication Time | Additional Time for $\mathcal{P}$ | $\mathcal{V}$ Time | Rounds |
|---|---|---|---|---|
| $2^{10} \times 2^{10}$ | 2.17 s over $\mathbb{Z}$; 9.11 s over $\mathbb{F}_q$ | 0.03s | 0.67 s | 11 |
| $2^{11} \times 2^{11}$ | 18.23 s over $\mathbb{Z}$; 73.65 s over $\mathbb{F}_q$ | 0.13s | 2.89 s | 12 |

**Table 2.** Experimental results for the $n \times n$ MATMULT protocol of Theorem 3.

## References

1. S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
2. S. Benabbas, R. Gennaro, Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111-131, 2011.
3. E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *ITCS*, pages 401-414, 2013.
4. E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete-efficiency threshold of probabilistically-checkable proofs. In *STOC*, 2013.
5. D. Boneh and D. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149-168, 2011.
6. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326-349, 2012.
7. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *STOC*, 2013.
8. N. Bitansky, and A. Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, pages 255-272, 2012.
9. N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315-333, 2013.
10. K-M. Chung, Y. Tauman Kalai, F-H. Liu, R. Raz. Memory delegation. In *CRYPTO*, pages 151-168, 2011.
11. K-M. Chung, Y. Tauman Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483-501, 2010.
12. A. Chakrabarti, G. Cormode, and A. McGregor. Annotations in data streams. In *ICALP (1)*, pages 222-234, 2009.

13. G. Cormode, M. Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409-442, 2013.
14. G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, pages 90-112, 2012.
15. G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011.
16. D. Fiore, R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS*, pages 501-512, 2012.
17. R. Freivalds. Fast probabilistic algorithms. In *MFCS*, pages 57–69, 1979.
18. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, pages 465-482, 2010.
19. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succint NIZKs without PCPs. In *EUROCRYPT*, pages 626-645, 2013.
20. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.
21. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321-340, 2010.
22. T. Gur and R. Raz Arthur-Merlin Streaming Complexity. In *ICALP (1)*, 2013.
23. Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *CCC*, pages 278–291, 2007.
24. T. Kimbrel and R. K. Sinha. A probabilistic algorithm for verifying matrix products Using O(n$^2$) time and $\log_2 n + O(1)$ random bits. *Inf. Process. Lett.* 45(2):107-110, 1993.
25. H. Klauck, and V. Prakash. Streaming computations with a loquacious prover. In *ITCS*, pages 305-320, 2013.
26. H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero- knowledge arguments. In *TCC*, pages 169-189, 2012.
27. C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39:859–868, 1992.
28. B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy (Oakland)*, 2013.
29. G. Rothblum. Delegating computation reliably: paradigms and constructions. Ph.D. Thesis. Available online at `http://hdl.handle.net/1721.1/54637`, 2009.
30. R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *JCSS*, 51(3):400-403, 1995.
31. S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, 2012.
32. S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security*, 2012.
33. S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walsh. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys*, pages 71-84, 2013.
34. A. Shamir. IP = PSPACE. *J. ACM*, 39:869–877, 1992.
35. J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *HotCloud*, 2012.
36. J. Thaler. Source code. Available online at `http://http://people.seas.harvard.edu/~jthaler/Tcode.htm`
37. V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. Pre-print, November 2012. In *IEEE Symposium on Security and Privacy (Oakland)*, May 2013.
38. R. Yuster, Computing the diameter polynomially faster than APSP. *CoRR*, Vol. abs/1011.6181, 2010.