More Optimizations to Sum-Check Proving

Quang Dao* Justin Thaler†

Abstract

Many fast SNARKs apply the sum-check protocol to an n-variate polynomial of the form $g(x) = \mathsf{eq}(w, x) \cdot p(x)$, where p is a product of multilinear polynomials, $w \in \mathbb{F}^n$ is a random vector, and eq is the multilinear extension of the equality function.

In this setting, we describe an optimization to the sum-check prover that substantially reduces the cost coming from the $\mathsf{eq}(w, x)$ factor. Our work further improves on a prior optimization by Gruen (ePrint 2023), and in the small-field case, can be combined with additional optimizations by Bagad, Domb, and Thaler (ePrint 2024), and Dao and Thaler (ePrint 2024).

Over large prime-order fields, our optimization eliminates roughly 2^{n+1} field multiplications compared to a standard linear-time implementation of the prover, and roughly 2^{n-1} field multiplications when considered on top of Gruen's optimization. These savings are about a 25% (respectively 10%) end-to-end prover speedup in common use cases, and potentially even larger when working over binary tower fields.

1 Introduction

The sum-check protocol [LFKN90] allows a verifier to offload the computation of the following quantity to an untrusted prover:

$$\sum_{x \in \{0,1\}^n} g(x). \tag{1}$$

Here, g is an n-variate polynomial defined over some finite field $\mathbb{F}$. From the verifier's perspective, the sum-check protocol acts as a reduction from the task of summing up g's evaluations over the $N = 2^n$ inputs in $\{0,1\}^n$ to the (hopefully easier) task of evaluating g at a *single* point in $\mathbb{F}^n$.

In this work, we focus on the case where g is of the form

$$g(x) = \mathsf{eq}(w, x) \cdot \prod_{i=1}^{d} p_i(x), \tag{2}$$

where $p(x) = \prod_{i=1}^{d} p_i(x)$ is a product of multilinear polynomials, $w \in \mathbb{F}^n$ is a vector of challenges chosen randomly by the verifier, and eq is the multilinear extension of the equality function, defined as

$$\mathsf{eq}(y, x) = \prod_{i=0}^{n-1} \left(x_i y_i + (1 - x_i)(1 - y_i)\right), \tag{3}$$

which satisfies

$$\mathsf{eq}(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise.} \end{cases}$$

This common case arises in the pervasive zero-check PIOP [BFL91, BTVW14, CFQ19, Set20], where we want to show that $p(x) = 0$ for all $x \in \{0,1\}^n$. It also captures other protocols such as Thaler's variation of the GKR protocol [GKR15] for grand-products [Tha13], where we want to show that $\prod_{v \in \{0,1\}^n} P(v) = t$ for some

*Carnegie Mellon University. Work done while at a16z crypto research

†a16z crypto research and Georgetown University

multilinear polynomial P and public value t. Both of these protocols are extensively used in Jolt [AST23], a leading candidate for zero-knowledge virtual machines (zkVMs) in terms of proving speed.

In this note, we provide further optimizations for the sum-check prover when it is applied to polynomials of the form of Expression (2), both in the setting where all arithmetic is performed over a large field $\mathbb{F}$, and in the setting where each factor $p_i(x)$ is defined over a smaller base field $\mathbb{B} \subseteq \mathbb{F}$, such as a binary tower field [Wie88].

Our optimization can be considered as a standalone speedup to the standard linear-time implementation [CTY11, Tha13] of the sum-check prover, or as a further speedup to optimizations of Gruen [Gru24], which decrease prover's computation (and proof size) related to the eq terms in exchange for increased verifier work.[1] In both settings, we do not change the protocol itself, only how the honest prover computes the sum-check messages in each round.

One of our main motivations for this note is to reduce prover time in the zkVM Jolt [AST23]. Specifically, Jolt uses Spartan [Set20] to prove satisfaction of an R1CS, and Jolt uses GKR-based grand product arguments within Lasso [STW23] and Spice [SAGL18] to prove lookup relations and perform memory-checking. Indeed, the *majority* of the prover's work in Jolt is applying the sum-check protocol to polynomials of the form of Equation (2) where $d = 2$.

1.1 Overview of our work

We first describe how our optimization improves upon the standard linear-time implementation of the sum-check prover. Our key idea is to leverage the fact that $\mathsf{eq}(w, x)$ is decomposable as a product of simpler multilinear polynomials: if $x = x_1 \| x_2$ and $w = w_1 \| w_2$ with $\|$ denoting concatenation, then

$$\mathsf{eq}(w, x) = \mathsf{eq}(w_1, x_1) \cdot \mathsf{eq}(w_2, x_2). \tag{4}$$

Because of this, we may rewrite the sum over the subset $x \in \{0,1\}^n$ as an iterated sum:

$$\sum_{x \in \{0,1\}^n} \mathsf{eq}(w, x) \cdot p(x) = \sum_{x_1 \in \{0,1\}^{n/2}} \mathsf{eq}(w_1, x_1) \cdot \sum_{x_2 \in \{0,1\}^{n/2}} \mathsf{eq}(w_2, x_2) \cdot p(x_1 \| x_2). \tag{5}$$

In the above, we have split w and x into two components, $w = w_1 \| w_2$, $x = x_1 \| x_2$, with $w_1, x_1 \in \{0,1\}^{n/2}$ and $w_2, x_2 \in \{0,1\}^{n/2}$. Our modified prover implementation now pre-computes two smaller tables:

$$\left\{\mathsf{eq}(w_1, x_1) : x_1 \in \{0,1\}^{n/2}\right\} \qquad \text{and} \qquad \left\{\mathsf{eq}(w_2, x_2) : x_2 \in \{0,1\}^{n/2}\right\}, \tag{6}$$

and uses them to compute the sum-check message in each of the first $n/2$ rounds of the protocol. The final $n/2$ rounds then follow the standard linear-time sum-check prover algorithm.

In contrast to our work, which pre-computes two tables of size $2^{n/2} = \sqrt{N}$, prior sum-check prover implementations [CTY11, Tha13] pre-computed a table of size N, namely:

$$\{\mathsf{eq}(w, x) \colon x \in \{0,1\}^n\}. \tag{7}$$

This computation can be done with N field operations [VSBW13], but this is nonetheless a constant fraction of the sum-check prover's total runtime.

At a high level, our use of two small tables instead of one big table saves the prover $2N - O(\sqrt{N}) \approx 2N$ field multiplications, with half of the savings coming from the reduced time to construct the tables, and half coming from the reduced time to update the tables after each round of the sum-check protocol. Our storage cost also goes from N down to $2\sqrt{N}$, which is significant for large invocations of sum-check.[2]

[1] With Gruen's optimization [Gru24, Section 3], the verifier either needs to invert one randomly chosen field element per round, or else offset the improvement in proof size, say by having the prover provide the claimed inverse y^{-1} and having the verifier do an extra multiplication to check that $y \cdot y^{-1} = 1$.

[2] In general, we could use c tables of size $N^{1/c}$ for any integer $c > 1$, which would ultimately lead to even less space devoted to processing $\mathsf{eq}(w, x)$ evaluations, but a slower overall prover. This approach is essentially identical to the evaluation algorithm for sparse multilinear polynomials described in [STW23, Section 3].

One may initially worry that we later "give back the work savings" because we do not precompute the entire N-sized table $\{\mathsf{eq}(w,x)\colon x \in \{0,1\}^n\}$, and hence every time the prover has to multiply a value $p(x) = \prod_{i=1}^{d} p_i(x)$ by $\mathsf{eq}(w,x)$ vis-a-vis Equation (2), the prover incurs two multiplications rather than one (i.e., first multiplying $p(x)$ by $\mathsf{eq}(w_2,x_2)$, and then multiplying be $\mathsf{eq}(w_1,x_1)$).

We avoid this by "aggregating" values across different $x = x_1 \| x_2$ based on x_1. This aggregation works because values only get aggregated together if they need to be multiplied by the *same* value $\mathsf{eq}(w_1,x_1)$. In this way, we need only at most $2^{n/2} = \sqrt{N}$ multiplications to process all of the $\mathsf{eq}(w_1,x_1)$ values in each round,[3] rather than 2^n such operations. This simple aggregation technique is the main technical novelty in this note.

Combining with Gruen's optimization. We also consider how our optimization can be directly built on top of Gruen's changes to the sum-check protocol [Gru24, Section 3], which trade off improved prover computation and proof size in exchange for increased verifier work. In this setting, our speedup is more modest, since Gruen's optimization already removed substantial cost for the prover's computation related to the eq terms. However, we show that our work still leads to a meaningful efficiency improvement on top of Gruen's in common applications (as well as significant space reductions).

Overview of Gruen's changes. Recall that in each round i of the sum-check protocol, the prover sends a univariate polynomial $s_i(X)$ whose degree is equal to the degree $d+1$ of the polynomial g being summed. Gruen [Gru24, Section 3] showed that if g is of the form in Equation (2), then the prover can actually send a polynomial of degree d, by removing the contribution of the first i factors in the definition of $\mathsf{eq}(w,x)$ (Equation (3)) to the computation of g. In other words, the prover will instead compute and send the following polynomial:

$$h_i(X) = \sum_{x' \in \{0,1\}^{n-i}} \mathsf{eq}(w',x') \cdot p(r,X,x'), \tag{8}$$

where $w' = (w_{i+1},\ldots,w_n)$, and $r = (r_1,\ldots,r_{i-1})$ is the vector of challenges so far. The verifier will then compute the missing eq terms, namely $\mathsf{eq}(w_1 \ldots w_{i-1}, r_1 \ldots r_{i-1}) \cdot \mathsf{eq}(w_i, X)$, to reconstruct $s_i(X)$ from $h_i(X)$.

Gruen's changes leads to the following decreases in prover's cost:

- The first is in the computation related to the eq terms. Instead of having to pre-compute the full (N-sized) table of eq evaluations, and update it after each round, the prover in Gruen's setting only needs to pre-compute a $(N/2)$-sized table of eq evaluations (missing the first factor $\mathsf{eq}(w_1,X)$) for the first round.[4] This saves roughly $2N - N/2 = 3N/2$ field multiplications compared to the standard linear-time sum-check prover. Here, $N/2$ multiplications are saved due to the built table having size $N/2$ rather than N, and N more multiplications are saved by avoiding the need to update the table after each sum-check round.
- The second is in having to compute one fewer evaluation per round, because the univariate polynomial sent by the prover in each round has degree that is one less than its degree in the standard sum-check protocol. This leads to a saving of roughly $d \cdot N/2^i$ field multiplications per round $i = 1,\ldots,n$, which is about $d \cdot N$ in total.

Our optimization in Gruen's setting. Similar to the standalone case, and focusing on the first round (where the prover cost is the greatest), we may rewrite the sum in Equation (8) over the subset $x' \in \{0,1\}^{n-1}$ as an iterated sum:

$$h_1(X) = \sum_{x'_1 \in \{0,1\}^{(n-1)/2}} \mathsf{eq}(w'_1,x'_1) \cdot \sum_{x'_2 \in \{0,1\}^{(n-1)/2}} \mathsf{eq}(w'_2,x'_2) \cdot g(X,x'_1,x'_2), \tag{9}$$

where $w'_1, w'_2 \in \mathbb{F}^{(n-1)/2}$ and $x'_1, x'_2 \in \{0,1\}^{(n-1)/2}$. Our modified prover implementation now pre-computes two smaller tables:

$$\left\{\mathsf{eq}(w'_1,x'_1) : x'_1 \in \{0,1\}^{(n-1)/2}\right\} \qquad \text{and} \qquad \left\{\mathsf{eq}(w'_2,x'_2) : x'_2 \in \{0,1\}^{(n-1)/2}\right\}, \tag{10}$$

[3] We actually only incur $2^{n/2-i} = \sqrt{N}/2^i$ field multiplications in round i to process these values, as the number of values of x_1 that are summed over halves in each round of the sum-check protocol.

[4] Technically speaking, the prover also needs to pre-compute a $(N/2^i)$-sized table of eq evaluations for each round $i = 1,\ldots,n$. All these tables can be computed at the same time using only $N/2$ field multiplications via known algorithms (see Section 2.1).

		# field multiplications		
		w/o our opt.	w/ our opt.	Percentage saving
Standard	Prime field	$\approx 8.5N$	$\approx 6.5N$	$\approx 23.5\%$
	Binary tower field	$\approx 7N$	$\approx 5N$	$\approx 28.6\%$
Gruen's	Prime field	$\approx 5.5N$	$\approx 5N$	$\approx 9.1\%$
	Binary tower field	$\approx 4N$	$\approx 3.5N$	$\approx 12.5\%$

Table 1: Concrete savings of our optimization in the first sum-check invocation of the Spartan protocol, as it is used in Jolt. In the above, $N = 2^n$ is the size of the hypercube $\{0,1\}^n$ being summed over. We consider various settings: with or without Gruen's optimization, and whether we are working over prime fields or binary tower fields. In the binary tower field case, we ignore the cost of multiplying a base-field element by a big-field element, as such multiplications are cheap in the tower basis. We also ignore potential further savings from prior works [Gru24, BDT24]. Thus, our estimated percentage speedup in this setting is highly conservative; see Section 4 for details.

and uses them to compute the sum-check message in each round of the protocol. Since we further decompose the eq table needed for the first round of Gruen's optimization, we obtain an extra saving of roughly $N/2$ field multiplications *on top of* Gruen's savings.

Combining with prior works in the tower field setting. In our prior note [DT24], we showed that when $\mathbb{F}$ is a degree-2^k tower extension of a base field $\mathbb{B}$ (such as $\mathsf{GF}(2)$), and each $p_i(x)$ in Equation (2) is defined over $\mathbb{B}$, the first k entries of w can often be set deterministically to "multiplication-friendly" field elements $z_0, \ldots, z_{k-1}$. In another prior work [BDT24], the authors show that in the same case of tower fields, one can further decrease the number of $\mathbb{F}$-multiplications in the first few rounds of sum-check, and instead replace them with $\mathbb{B}$-by-$\mathbb{F}$ multiplications (i.e., one base-field element multiplied by one extension-field element), which are much cheaper when working over tower fields than $\mathbb{F}$-by-$\mathbb{F}$ multiplications. These two optimizations can be trivially combined with the one in this work, and lead to further efficiency in the tower field case.

Concrete savings from our optimization. In Table 1, we summarize our concrete savings in our situation of interest, which is first invocation of sum-check in Spartan as it is used in Jolt. We show that for the standard linear-time prover implementation, our optimization gives around a 25% reduction in prover computation, and for the protocol with Gruen's changes, our optimization gives around a 10% reduction.[5] We refer to Section 4 for details.

1.2 Further Discussion

Relationship to recent work of Ron Rothblum [Rot24]. Rothblum [Rot24] shows that all $\mathsf{eq}(w, x)$ evaluations for $x \in \{0,1\}^n$ can be iteratively generated (specifically, in Gray code order) with roughly $N = 2^n$ field multiplications and just $\log(n)$ space (i.e., the space required to store $\log(n)$ elements of $\mathbb{F}$).[6] Thus, Rothblum's result almost eliminates any space cost for processing the $\mathsf{eq}(w, x)$ evaluations, but does not save the prover any field multiplications compared to prior algorithms. Our work uses $O(\sqrt{N})$ space rather than $O(\log N)$ space to store data relevant to the $\mathsf{eq}(w, x)$ evaluations, but saves on the prover time.

Generality. Other protocols such as those for proving AIR or Plonkish constraint systems may have $p(x)$ of the form $p(x) = h(p_1(x), \ldots, p_\ell(x))$, where $p_1(x), \ldots, p_\ell(x)$ are multilinear and $h(y)$ is a degree-d

[5] We stress that our estimates in this table only count the primary cost of field multiplications, and ignore lower-order terms such as the cost of field additions or (in the extension field setting) base-field-by-big-field multiplications; these may affect the final savings once our optimization is implemented.

[6] Rothblum's focus is on evaluating an arbitrary multilinear extension polynomial quickly and in small space, but can also benefit the sum-check prover when the polynomial being summed has the form of Equation (2).

multivariate polynomial. The result in this note applies similarly to these protocols, where we simply apply the optimization to each monomial of the multivariate polynomial $h(y)$.

Organization. In Section 2, we present our optimization over the standard linear-time implementation of the sum-check prover. In Section 3, we present our optimization when considered on top of Gruen's optimization. Finally, in Section 4, we present analysis of concrete savings for our optimization in the case of the Spartan protocol as used in Jolt.

2 Speedup over Standard Linear-Time Prover

2.1 Recap: Linear-Time Prover for Sum-check

We recall the standard implementation of a linear-time prover for sum-check [Tha13, XZZ+19, Tha22].

1. **Pre-computation:** The prover initializes $d+1$ arrays $E^{(1)}, P_1^{(1)}, \ldots, P_d^{(1)}$, each of size 2^n, by setting $E^{(1)}[x] := \mathsf{eq}(w, x)$, $P_1^{(1)}[x] := p_1(x)$, ..., and $P_d^{(1)}[x] := p_d(x)$ for all $x \in \{0,1\}^n$.
2. In round $i = 1, \ldots, n$:
   (a) The prover needs to compute the polynomial $s_i(X)$, defined as:

$$s_i(X) = \sum_{x' \in \{0,1\}^{n-i}} \mathsf{eq}(w, r_1 \ldots r_{i-1} \| X \| x') \cdot \prod_{k=1}^{d} p_k(r_1 \ldots r_{i-1} \| X \| x').$$

   To do so, it suffices for the prover to compute and send the evaluations $s_i(0), s_i(2), \ldots, s_i(d+1)$:[7]

$$\begin{aligned}
s_i(0) &= \sum_{x' \in \{0,1\}^{n-i}} E^{(i)}[0, x'] \cdot \prod_{k=1}^{d} P_k^{(i)}[0, x'], \\
s_i(1) &= \sum_{x' \in \{0,1\}^{n-i}} E^{(i)}[1, x'] \cdot \prod_{k=1}^{d} P_k^{(i)}[1, x'], \\
s_i(j) &= \sum_{x' \in \{0,1\}^{n-i}} \Big((1-j) \cdot E^{(i)}[0, x'] + j \cdot E^{(i)}[1, x'] \Big) \\
&\quad \cdot \prod_{k=1}^{d} \Big((1-j) \cdot P_k^{(i)}[0, x'] + j \cdot P_k^{(i)}[1, x'] \Big), \qquad \text{for all } j = 2, \ldots, d+1.
\end{aligned}$$

   Note that the prover does not need to compute and send $s_i(1)$, since it could be derived by the verifier from the other evaluations and the claimed sum-check value, i.e., $s_i(1) = s_i(0) - s_{i-1}(r_{i-1})$.

   (b) Upon receiving the i-th challenge $r_i \in \mathbb{F}$, the prover then updates the arrays

$$\begin{aligned}
E^{(i+1)}[x'] &:= \Big(E^{(i)}[1, x'] - E^{(i)}[0, x'] \Big) \cdot r_i + E^{(i)}[0, x'], \\
P_k^{(i+1)}[x'] &:= \Big(P_k^{(i)}[1, x'] - P_k^{(i)}[0, x'] \Big) \cdot r_i + P_k^{(i)}[0, x'],
\end{aligned}$$

   for all $k = 1, \ldots, d$ and $x' \in \{0,1\}^{n-i}$. Up to re-arranging, this computation keeps the invariant that $P_k^{(i+1)}[x'] = p_k(r_{[1:i]} \| x')$ for all k, and similarly for the $E^{(i+1)}$ table.

[7] We pick the evaluations points $0, 2, \ldots, d+1$ assuming that the characteristic of the field is larger than $d+1$. For fields of small characteristic, such as binary tower fields, we instead pick other distinct evaluation points. An efficient choice would be to pick according to the multilinear ordering $0, 1, z_0, z_0+1, z_1, \ldots$, where $z_0, z_1, \ldots$ are the special field elements of the tower extension $\mathbb{B} \subset \mathbb{F}$ (see [DT24] for details about these special field elements).

Cost estimate. We estimate the cost of the above approach by counting the number of field multiplications, since this represents the dominant cost.[8]

1. In the pre-computation phase, assuming that the values $p_1(x), \ldots, p_d(x)$ are immediately available to the prover,[9] the prover then only needs to compute the evaluations of eq, which takes roughly 2^n field multiplications (we refer to [DT24, Section 8] for such an algorithm).
2. In each subsequent round $i = 1, \ldots, n$, the prover computes roughly $d \cdot 2^{n-i}$ field multiplications for the computation of each evaluation $s_i(j)$ for $j = 0, 1, \ldots, d+1$, and $(d+1) \cdot 2^{n-i}$ field multiplications for updating the arrays $E^{(i+1)}, P_1^{(i+1)}, \ldots, P_d^{(i+1)}$.

In total, this gives a cost of roughly

$$2^n + ((d+1) \cdot d + (d+1)) \cdot \sum_{i=1}^{n} 2^{n-i} \approx (d^2 + 2d + 2) \cdot N$$

field multiplications, and a similar amount of field additions (which are less expensive).

2.2 Our Approach

We now present our optimization. Given a round split parameter $m \approx n/2$ (which will be determined later), we split the computation of the sum in each round $i \leq m$ into two layers, with each layer having about half of the eq factors via the decomposition

$$\mathsf{eq}(w, r_{[1:i-1]} \| X \| x') := \mathsf{eq}(w_1, r_{[1:i-1]} \| X \| x'_1) \cdot \mathsf{eq}(w_2, x'_2), \qquad \text{where} \quad w_1 \in \mathbb{F}^m, \quad w_2 \in \mathbb{F}^{n-m}.$$

In other words, for each round $i \leq m$, we will compute the evaluations $s_i(j)$ for $j = 0, 2, \ldots, d+1$ as follows:

$$\begin{aligned} s_i(j) &= \sum_{x' \in \{0,1\}^{n-i}} \mathsf{eq}(w, r_{[1:i-1]} \| j \| x') \cdot p(r_{[1:i-1]} \| j \| x') \\ &= \sum_{x'_1 \in \{0,1\}^{m-i}} \mathsf{eq}(w_1, r_{[1:i-1]} \| j \| x'_1) \cdot \sum_{x'_2 \in \{0,1\}^{n-m}} \mathsf{eq}(w_2, x'_2) \cdot p(r_{[1:i-1]} \| j \| x'_1 \| x'_2). \end{aligned}$$

The benefit of this decomposition is that we no longer have to materialize the 2^n-sized table of evaluations $\{\mathsf{eq}(w, x) : x \in \{0,1\}^n\}$ and updating it after each round, which cost $2 \cdot 2^n$ field multiplications in total. Instead, we only need to compute and update two smaller tables $\{\mathsf{eq}(w_1, x_1) : x_1 \in \{0,1\}^m\}$ and $\{\mathsf{eq}(w_2, x_2) : x_2 \in \{0,1\}^{n-m}\}$, of size 2^{m-i} and 2^{n-m} respectively. This cost $2 \cdot (2^m + 2^{n-m})$ field multiplications in total, plus an additional $(d+1) \cdot 2^{m-i}$ field multiplications in each round $i \leq m$, since we incur an extra multiplication per outer eq term. Putting these costs together, we want to minimize the quantity:

$$2 \cdot (2^m + 2^{n-m}) + (d+1) \cdot \sum_{i=1}^{m} 2^{m-i} = (d+3) \cdot 2^m + 2 \cdot 2^{n-m}.$$

Since the product of two terms is a constant (in terms of n and d), their sum is minimized when they are equal, i.e., when $(d+3) \cdot 2^m = 2 \cdot 2^{n-m}$, or $m = n/2 - \log\left(\frac{d+3}{2}\right)/2 \approx n/2$. For the rest of this work, we will assume for simplicity that $m = n/2$ exactly, since the term $\log\left(\frac{d+3}{2}\right)/2$ is a small constant, and the extra savings are in the low-order terms $O(2^{n/2})$ anyway.[10]

[8] Over a 256-bit field $\mathbb{F}$, a multiplication can be an order of magnitude more expensive than a field addition. The difference may be less pronounced for smaller fields such as the binary extension field $\mathsf{GF}(2^{128})$.

[9] This is the case for applications such as Jolt, where the $p_i(x)$ evaluations roughly correspond to entries of the execution trace of the CPU. This execution trace is materialized once, then used in different subprotocols. Generating the execution trace is currently less than 10% of the Jolt prover's runtime.

[10] A further optimization, that only affects the square-root term, is to further split the outer eq term into two pieces, thus rewriting the computation as a triply-iterated sum. This drives down some of the $2^{n/2}$ cost to a $2^{n/4}$ cost, but since the improvement is so marginal, we do not discuss this further.

Summary. For completeness, we now present the full implementation of the sum-check prover following our optimization.

1. **Pre-computation:** As before, the prover initializes d arrays $P_1^{(1)}, \ldots, P_d^{(1)}$, each of size 2^n, by setting $P_1^{(1)}[x] := p_1(x)$, $\ldots$, and $P_d^{(1)}[x] := p_d(x)$ for all $x \in \{0,1\}^n$.

   The prover then initializes two smaller arrays $E_1^{(1)}$ and E_2, each of size $2^{n/2}$, by setting $E_1^{(1)}[x_1] := \mathsf{eq}(w_1, x_1)$ and $E_2[x_2] := \mathsf{eq}(w_2, x_2)$ for all $x_1, x_2 \in \{0,1\}^{n/2}$.

2. In round $i = 1, \ldots, n/2$:

   (a) The prover computes the evaluations $s_i(0), s_i(2), \ldots, s_i(d+1)$ as follows. First, the prover computes the inner sums:

   $$\begin{aligned} S_i(x_1, j) &= \sum_{x_2 \in \{0,1\}^{n/2}} \mathsf{eq}(w_2, x_2) \cdot \prod_{k=1}^{d} \left((1-j) \cdot p_k(r_{[1:i-1]} \| j \| x_1 \| x_2) + j \cdot p_k(r_{[1:i-1]} \| j \| x_1 \| x_2)\right) \\ &= \sum_{x_2 \in \{0,1\}^{n/2}} E_2[x_2] \cdot \prod_{k=1}^{d} \left((1-j) \cdot P_k^{(i)}[x_1 \| x_2] + j \cdot P_k^{(i)}[x_1 \| x_2]\right). \end{aligned}$$

   for all $x_1 \in \{0,1\}^{n/2-i}$ and $j = 0, 2, \ldots, d+1$. The prover then uses these inner sum values to compute the evaluations:

   $$\begin{aligned} s_i(j) &= \sum_{x_1 \in \{0,1\}^{n/2-i}} \mathsf{eq}(w_1, r_{[1:i-1]} \| j \| x_1) \cdot S_i(x_1, j) \\ &= \sum_{x_1 \in \{0,1\}^{n/2-i}} \left((1-j) \cdot E_1^{(i)}[0, x_1] + j \cdot E_1^{(i)}[1, x_1]\right) \cdot S_i(x_1, j). \end{aligned}$$

   (b) The prover then updates the arrays:

   $$\begin{aligned} E_1^{(i+1)}(x_1) &:= \left(E_1^{(i)}[0 \| x_1] - E_1^{(i)}[1 \| x_1]\right) \cdot r_i + E_1^{(i)}[1 \| x_1], \\ P_k^{(i+1)}(x_1) &:= \left(P_k^{(i)}[0 \| x_1] - P_k^{(i)}[1 \| x_1]\right) \cdot r_i + P_k^{(i)}[1 \| x_1], \end{aligned}$$

   for all $k = 1, \ldots, d$ and $x_1 \in \{0,1\}^{n/2-i}$.

3. In round $i = n/2 + 1, \ldots, n$: we proceed the same as the linear-time prover implementation. Note that it is possible for us to switch to this algorithm since our algorithm above still retains all the necessary information to run the linear-time prover. This includes the arrays $P_k^{(n/2+1)}$ for all $k = 1, \ldots, d$, along with the array $E^{(n/2+1)} := \mathsf{eq}(w, r_{[1:n/2]} \| x_2)$, which can be computed as $E^{(n/2+1)} = E_1^{(n/2)} \cdot E_2$.

From the discussion above, the total cost saving of our optimization over the baseline is roughly $2 \cdot 2^n - (d+5) \cdot 2^{n/2} \approx 2N$ field multiplications.

3 Speedup over Gruen's Optimization

In this section, we present how our optimization can be applied directly on top of Gruen's changes [Gru24], leading to a further reduction of roughly 2^{n-1} field multiplications.

Recap: Gruen's optimization. The core idea of Gruen's changes to the sum-check protocol is to remove the contribution of the eq factor to the prover's message in each round, since the verifier can compute that contribution on its own. This lowers both the prover computation and the proof size (though it increases verifier work). Below, we give the prover implementation for Gruen's variation of the sum-check protocol when it is applied to a polynomial of the form of Equation (2).

1. **Pre-computation:** The prover initializes d arrays $P_1^{(1)}, \ldots, P_d^{(1)}$, each of size $N = 2^n$, by setting $P_1^{(1)}[x] := p_1(x)$, ..., and $P_d^{(1)}[x] := p_d(x)$ for all $x \in \{0,1\}^n$. The prover also computes the tables $E^{(1)}, \ldots, E^{(n-1)}$ satisfying:

$$E^{(i)}[x'] := \mathsf{eq}(w_{[i+1:n]}, x') \quad \text{for all} \quad x' \in \{0,1\}^{n-i} \quad \text{and} \quad i = 1, \ldots, n-1.$$

   Existing algorithms [DT24, Section 8] may compute all $E^{(i)}$ tables with roughly 2^{n-1} field multiplications.

2. In round $i = 1, \ldots, n$:

   (a) The prover needs to compute the polynomial $s_i(X)$, defined as:

$$s_i(X) = \sum_{x' \in \{0,1\}^{n-i}} \mathsf{eq}(w_{[i+1:n]}, x') \cdot \prod_{k=1}^{d} p_k(r_{[1:i-1]} \| X \| x').$$

   To do so, it suffices for the prover to compute and send the evaluations $s_i(0), s_i(2), \ldots, s_i(d)$:

$$s_i(0) = \sum_{x' \in \{0,1\}^{n-i}} E^{(i)}(x') \cdot \prod_{k=1}^{d} P_k^{(i)}[0 \| x'],$$

$$s_i(j) = \sum_{x' \in \{0,1\}^{n-i}} E^{(i)}(x') \cdot \prod_{k=1}^{d} \left((1-j) \cdot P_k^{(i)}[0 \| x'] + j \cdot P_k^{(i)}[1 \| x'] \right), \quad \text{for all } j = 2, \ldots, d.$$

   (b) Upon receiving the i-th challenge $r_i \in \mathbb{F}$, the prover then updates the arrays:

$$P_k^{(i+1)}(x') := \left(P_k^{(i)}[1 \| x'] - P_k^{(i)}[0 \| x'] \right) \cdot r_i + P_k^{(i)}[0 \| x'],$$

   for all $k = 1, \ldots, d$ and $x' \in \{0,1\}^{n-i}$.

Cost savings from Gruen's optimization. Gruen's optimization over the standard protocol in Section 2.1 decreases the prover computation in two ways:

1. First, the prover only has to materialize the smaller tables $E^{(1)}, \ldots, E^{(n-1)}$, instead of computing and updating the N-sized table $\{\mathsf{eq}(w, x) : x \in \{0,1\}^n\}$. As discussed in the introduction, this saves $3N/2$ field multiplications.
2. Second, the prover only has to compute and send d evaluations $s_i(j)$ for $j = 0, 2, \ldots, d$, instead of $d+1$ evaluations $s_i(j)$ for $j = 0, 2, \ldots, d+1$. This saves another $d \cdot 2^{n-i}$ field multiplications in each round i, leading to a total of $d \cdot N$ further multiplications saved.

Our optimization on top of Gruen's. Similar to our optimization in Section 2.2, we decompose the eq factors in each round, and compute the evaluations via a two-layer sum:

$$s_i(j) = \sum_{x'_1 \in \{0,1\}^{n/2-i}} \mathsf{eq}(w_{[i+1:n/2]}, x'_1) \cdot \sum_{x'_2 \in \{0,1\}^{n-n/2}} \mathsf{eq}(w_{[n/2+1:n]}, x'_2) \cdot p(r_{[1:i-1]} \| j \| x'_1 \| x'_2).$$

This saves the cost of materializing the $(N/2)$-sized table $\left\{\mathsf{eq}(w_{[2:n]}, x') : x' \in \{0,1\}^{n-1}\right\}$ in the first round, instead replacing it with low-order terms with asymptotic $O(\sqrt{N})$.[11] Therefore, our optimization saves roughly $N/2 - O(\sqrt{N}) \approx N/2$ field multiplications in the setting of Gruen's changes.

[11] We also omit discussing further savings in the constant-factor of the $\sqrt{N}$ term, by e.g. dividing the eq evaluations into two equal-sized factors in each round, which is possible in this setting.

4 Case Study of Concrete Savings

In this section, we discuss the concrete savings of our optimization in a realistic setting, namely that of the Spartan protocol [Set20] as it is currently used in Jolt [AST23]. We first recall the Spartan protocol for uniform instances of R1CS, and then present the concrete savings of our optimization when applied to Spartan in various settings: with or without Gruen's optimization, and whether in a large prime field or in binary tower fields. The reader may refer to Table 1 for a summary.

Recap: The Spartan protocol. Spartan is a proof system for the R1CS relation, which takes the form

$$(A \cdot Z) \circ (B \cdot Z) = C \cdot Z,$$

where $A, B, C \in \mathbb{F}^{n \times m}$ are public matrices, $x \in \mathbb{F}^k$ is the public input, $v \in \mathbb{F}^{m-k-1}$ is the private witness, $Z = (1, x, v)$, and $\circ$ denotes Hadamard (element-wise) product. To prove this relation, Spartan invokes the sum-check protocol twice, with the first invocation applied to the polynomial:

$$F(x) := \mathsf{eq}(w, x) \cdot \left(\overline{A}(x) \cdot \overline{B}(x) - \overline{C}(x)\right).$$

Here $w \in \mathbb{F}^n$ is a random challenge vector, and $\overline{A}(x), \overline{B}(x), \overline{C}(x)$ are the multilinear extensions of the vectors $A \cdot Z$, $B \cdot Z$, and $C \cdot Z$, respectively. For the specific use-case of Spartan in Jolt, these vectors can be assumed to be pre-computed as the prover builds the execution trace (see Footnote 9), and they also have small entries that are at most the word size of the RISC-V virtual machine (which is currently 32 bits in Jolt). By linearity, the honest sum-check prover's message in each round, when applied to F, is the sum of the messages sent when applied to

$$\mathsf{eq}(w, x) \cdot \overline{A}(x) \cdot \overline{B}(x) \qquad \text{and} \qquad \mathsf{eq}(w, x) \cdot \overline{C}(x). \tag{11}$$

We will focus on applying the sum-check protocol specifically to

$$g(x) := \mathsf{eq}(w, x) \cdot \overline{A}(x) \cdot \overline{B}(x),$$

as it is the bottleneck compared to the sum-check computation for $\mathsf{eq}(w, x) \cdot \overline{C}(x)$.[12]

Spartan in general also involves a second invocation of the sum-check protocol, but in Jolt this second invocation can be effectively eliminated owing to the highly uniform nature of the constraint system arising in Jolt.[13]

Summary of costs, with and without our optimization. We first give a summary of the number of field multiplications (in the large prime field case) used in applying sum-check for the polynomial $g(x)$ in Equation (11), which is the dominant cost of Spartan as applied in Jolt. Following our analysis in Section 2.1, the number of field multiplications for the standard linear-time sum-check prover implementation is as follows:

- $N + N = 2N$ to build and then collapse $\mathsf{eq}(w, \cdot)$;
- $N + N = 2N$ to collapse both $\overline{A}$ and $\overline{B}$;
- $3N/2$ to compute the 3 evaluations $s_1(0), s_1(2), s_1(3)$ in the first round of sum-check. Note here that we save on $3N/2$ field multiplications due to the fact that the products $\overline{A}(j, x') \cdot \overline{B}(j, x')$ for all $j = 0, 2, 3$ and $x' \in \{0, 1\}^{n-1}$ are between small field elements (roughly 32-bit values), and hence can be considered "free";
- $3 \cdot 2 \cdot (N/4 + N/8 + \dots) \approx 3N$ to compute the evaluations $s_i(0), s_i(2), s_i(3)$ for rounds $i = 2, 3, \dots, n$.

In total, the standard linear-time prover implementation computes about $8.5N$ field multiplications. With our optimization, we reduce this cost to roughly $6.5N$ field multiplications, due to cutting the first cost related to the eq terms, which is a 23.5% reduction.

[12] Since sum-check is less expensive overall when applied to $\mathsf{eq}(w, x) \cdot \overline{C}(x)$, our optimization will yield an even bigger percentage saving if we were to take this cost into account.

[13] See `https://github.com/a16z/jolt/issues/347` for details.

With Gruen's optimization, and without our optimization, the prover cost will be about $5.5N$ field multiplications. This comes from a $3N/2$ reduction in `eq` computation, and another $N/2 + 2 \cdot (N/4 + N/8 + \dots) = 3N/2$ reduction coming from not having to compute $s_i(3)$ over all rounds $i = 1, \dots, n$. With our optimization on top, we save roughly another $N/2$ field multiplications, which makes the cost about $5N$ field multiplications. This is roughly a 9% reduction.

Finally, over binary tower fields of the form $\mathbb{B} \subset \mathbb{F}$, where $\overline{A}, \overline{B}$ have evaluations belonging in $\mathbb{B}$, the percentage prover speedup of our optimizations is significantly amplified. One reason for this is $\mathbb{B}$-by-$\mathbb{F}$ multiplications are much cheaper than $\mathbb{F}$-by-$\mathbb{F}$ multiplications when working over the tower basis. In particular, the first round of sum-check features $3N/2$ number of $\mathbb{B}$-by-$\mathbb{F}$ multiplications of the form $\mathtt{eq}(w', x') \cdot \big(\overline{A}(j, x') \cdot \overline{B}(j, x')\big)$. Since these multiplications are much cheaper than $\mathbb{F}$-by-$\mathbb{F}$ multiplications when working over the tower basis, we do not factor them into our final counts of $\mathbb{F}$-by-$\mathbb{F}$ multiplications for the Spartan prover (both with and without our optimizations). This makes the $2N$ or $N/2$ $\mathbb{F}$-by-$\mathbb{F}$ multiplications saved by our optimizations a much bigger percentage of the total prover work.

We summarize the estimated savings in Table 1, which clearly show that our optimization leads to a bigger percentage savings in the setting of binary tower fields, compared to the large prime field case. Even here, the percentage savings we report for the case of binary tower fields are highly conservative, as we do not account for further prover cost reductions in this setting due to prior works (i.e., the modified sum-check prover algorithm in [BDT24, Algorithm 3], or the univariate skip technique in [Gru24, Section 4]).

Disclosures. Justin Thaler is a Research Partner at a16z crypto and is an investor in various blockchain-based platforms, as well as in the crypto ecosystem more broadly (for general a16z disclosures, see `https://www.a16z.com/disclosures/`.)

References

[AST23] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. *Cryptology ePrint Archive*, 2023.

[BDT24] Suyash Bagad, Yuval Domb, and Justin Thaler. The sum-check protocol over fields of small characteristic. Cryptology ePrint Archive, Paper 2024/1046, 2024. `https://eprint.iacr.org/2024/1046`.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1:3–40, 1991.

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. ePrint Report 2014/846, 2014.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2075–2092, 2019.

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, 2011.

[DT24] Quang Dao and Justin Thaler. Constraint-packing and the sum-check protocol over binary tower fields. Cryptology ePrint Archive, Paper 2024/1038, 2024. `https://eprint.iacr.org/2024/1038`.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[Gru24] Angus Gruen. Some improvements for the piop for zerocheck. *Cryptology ePrint Archive*, 2024.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, October 1990.

[Rot24] Ron D. Rothblum. A note on efficient computation of the multilinear extension. Cryptology ePrint Archive, Paper 2024/1103, 2024. https://eprint.iacr.org/2024/1103.

[SAGL18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2018.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with Lasso. Cryptology ePrint Archive, Paper 2023/1216, 2023. https://eprint.iacr.org/2023/1216.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2013.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends in Privacy and Security*, 4(2–4):117–660, 2022.

[VSBW13] Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2013.

[Wie88] Doug Wiedemann. An iterated quadratic extension of gf (2). *Fibonacci Quart*, 26(4):290–295, 1988.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2019.