Power circuits: a new arithmetization for GKR-styled sumcheck

Lev Soukhanov *

October 2023

Abstract

Goldwasser-Kalai-Rothblum protocol (GKR) for layered circuits is a sumcheck-based argument of knowledge for layered circuits, running in $\sim 2\mu\ell$ amount of rounds, where ℓ is the amount of layers and μ is the average layer logsize.

For a layer i of size 2^{μ_i} the main work consists of running a sumcheck protocol of the form

$$\sum_{x,y} \mathrm{Add}_i(x,y,z)(f(x)+f(y)) + \mathrm{Mul}_i(x,y,z)f(x)f(y)$$

over a $2^{2\mu_i}$-dimensional cube, where $\mathrm{Add}_i(x,y,z)$ and $\mathrm{Mul}_i(x,y,z)$ are (typically relatively sparse) polynomials called "wiring predicates".

We present a different approach, based on the (trivial) observation that multiplication can be expressed through linear operations and squaring.

This leads to the different wiring, which is marginally more efficient even in a worst-case scenario, and decreases the amount of communication $\sim 2\times$ in the case where wiring predicates are sparse.

1 Introduction

In the last year, the protocol of Goldwasser-Kalai-Rothblum [1] has got a resurgence due to the applications to the field of succinct non-interactive snarks.

Specifically, it was realized that structured circuits used in lookup arguments and permutation arguments can be efficiently delegated to a run of

*Privacy Scaling Exporations, Ethereum Foundation. email:0xdeadfae@gmail.com

GKR protocol. Two already existing constructions of this kind are Lasso [4] and logup-GKR [3].

The GKR protocol has excellent proving time, and, importantly, only requires the user to commit to the witness and not the full execution trace, which has an important implication to the lookup arguments in question - as values committed by the prover are typically of small magnitude, and values appearing in the execution trace itself are much larger.

Despite that, usage of GKR protocol in other succinct contexts is blocked by a relatively large amount of rounds it requires. We suggest a different wiring strategy, leading to the same worst-case performance and concrete gains in sparse wiring scenario (and, specifically, for Thaler's product circuit [5] and for fractional summation protocol from logup-GKR paper [3]) .

This work was done at research seminar in Seoul organized by Privacy Scaling Exploration. Author expresses gratitude to the participants of the seminar: without them the work would likely never happen.

2 Reminder on GKR protocol

Let's consider a layered arithmetic circuit over a large field $\mathbb{F}$, with n layers, layer i being of the size 2^{μ_i}. The elements of the layer are interpreted as the vertices of a boolean hypercube, i.e. vectors $(x_0, ..., x_{\mu_i - 1}) \in \{0,1\}^{\mu_i}$.

Each layer is connected to the next one by a pair of *wiring predicates*: for variables x, y in layer i and variable z in layer $i+1$, we set $\mathrm{Add}_i(x,y,z) = 1$ iff the values in cells x and y should be added and result written in a cell z, and 0 otherwise. Similarly, we declare a wiring predicate $\mathrm{Mul}_i(x,y,z)$ which is equal to 1 iff there is an instruction multiplying values in x and y and writing the product into z, and 0 otherwise.

We denote the unique **multilinear extensions** of Add and Mul with the same symbols, and hope that it will not lead to any confusion.

Then, for a run of this circuit, we denote by the W_i the (multilinear extension of) values of the execution trace on the layer i. Then, it is clear that

$$W_{i+1}(z) = \sum_{x,y \in \mathbb{B}^{\mu_i}} (\mathrm{Add}_i(x,y,z)(W_i(x) + W_i(y)) + \mathrm{Mul}_i(x,y,z) W_i(x) W_i(y)$$

Sumcheck protocol is a fundamental protocol [2] allowing the prover to reduce the claim about the sum of the multivariate polynomial $P(x)$ of degree d in each variable over a hypercube of dimension μ to a claim about its value in a challenge point. This short note is not a good and place to go into

the details, but we refer to any of the cited papers, or excellent Thaler's lectures on the topic https://people.cs.georgetown.edu/jthaler/sumcheck.pdf. Sumcheck protocol runs in μ rounds with prover messages of size d.

GKR protocol applies the following reduction n times: it goes back, starting from the final layer, and uses sumcheck over variables x and y to reduce a statement about the value of a polynomial in some point

$$W_{i+1}(s_{i+1}) = S_{i+1}$$

into a pair of statements

$$W_i(l_i) = L_i$$

$$W_i(r_i) = R_i$$

. Then, with additional round of communication of size μ_i, it reduces this statement to a single statement of a form

$$W_i(s_i) = S_i$$

by the standard argument: prover sends the coefficients of $w_i(t)$ of the restriction of W_i on a line $l_i + t(r_i - l_i)$, and verifier responds with a random challenge τ and specializes evaluation into a challenge point: $s_i = l_i + \tau(r_i - l_i)$, $S_i = w_i(\tau)$

Therefore, total amount of communication for a single layer i is $7\mu_i$, coming from $2\mu_i$ messages of size 3, and an additional reduction protocol with a single message of size μ_i.

3 Power circuits protocol

We start off with a rather simple observation - usage of addition predicate feels wasteful, because the same effect could be achieved by a single sumcheck emulating a linear operator:

$$\sum_{x,y} \mathrm{Add}_i(x, y, z)(f(x) + f(y)) = \sum_x M_i(z, x) f(x)$$

by defining

$$\Delta(p, q, x) = \mathrm{Eq}(p, x) + \mathrm{Eq}(q, x)$$

and then noticing

$$f(p) + f(q) = \Delta(p, q, x) f(x)$$

and setting

$$M_i(z,x) = \sum_{p,q} \mathrm{Add}_i(p,q,z)\Delta(p,q,x)$$

This does not improve verification cost by itself, because there is still a multiplication sumcheck which runs over a double set of variables. However, it is well known that it is possible to emulate multiplication using only linear operations and squaring:

$$ab = \frac{1}{2}((a+b)^2 - a^2 - b^2)$$

Motivated by this, we give the following definition:

Definition 1. *A power circuit of depth n is given by a sequence of variable layers of size $\mu_0, ...\mu_n$, the sequence of n affine mappings, encoded as tuples (M, C), where $M_i(y,x)$ is a matrix and an $C_i(y)$ is an affine shift, with x running over layer i and y over the layer $i+1$, and a sequence of powers $d_1, ..., d_{n-1}$ (and we set $d_0 = 1$ by convention, to start from a linear layer).*

The execution of the circuit is given by the following rule:

$$W_{i+1}(y) = \sum_x M_i(y,x)W_i^{d_i}(x) + C_i(y)$$

We will also frequently consider the special case, when $d_1 = ... = d_{n-1} = 2$, which we call a "squaring circuit".

Theorem 1. *It is possible to emulate a GKR circuit by a squaring circuit, with the size $2c_i + q_i$, where c_i is the size of the original layer, and q_i is the amount of multiplication gates in layer i - i.e. amount of pairs (x,y) such that $\exists z | \mathrm{Mul}_i(x,y,z) \neq 0$.*

Note: it is also always possible to skip the factor 2 before c_i if our circuit is homogeneous.

Convention: in what follows, we will sometimes write some vectors like $W_i(x)$ (meaning multilinear function with x running over vertices of the boolean hypercube), and sometimes as the list of their values, written as $(W_i)_0, (W_i)_1, ...,$. We do not explicitly spell out how these indices are related to the vertices of a hypercube, but pretend that this indexing is given.

Proof. We define a pair of affine mappings (recall that affine mapping is a multiplication by matrix and addition of a vector), the extension mapping and the reduction mapping. The extension mapping maps the original layer

(in which the variable x lives) into an extended layer $\tilde{x}$ and works by appending to the vector $(W_i)_0, ..., (W_i)_{c_i-1}$ the vector $(, ..., (W_i)_x + (W_i)_y, ...,)$ running over all pairs x, y that appear in multiplication gates, and the vector $((W_i)_0 + 1, ..., (W_i)_{c_i-1} + 1)$.

This mapping is clearly affine. We denote it $\mathcal{E}_i$.

Then, the reduction mapping is another affine mapping that expresses required products from the squares of the extended vector. Precisely, it is clear that we can linearly express every product of $(W_i)_x$ and $(W_i)_y$ as $\frac{1}{2}(((W_i)_x + (W_i)_y)^2 - (W_i)_x^2 - (W_i)_y^2)$, and it is possible to express $(W_i)_x$ (and hence any of its linear combinations) back affinely as $\frac{1}{2}(((W_i)_x + 1)^2 - (W_i)_x^2 - 1)$. Therefore, we can construct such reduction mapping $\mathcal{R}_i$ that

$$\mathcal{R}_i(\mathcal{E}_i(W_i)^2) = W_{i+1}$$

Now, we can use the fact that composition of affine mappings is affine and define

$$\widetilde{M_i} = \mathcal{E}_{i+1} \circ \mathcal{R}_i$$

The components of this mapping define our matrix and a constant term:

$$(\widetilde{M_i} W_i)(y) = \sum_x M_i(y, x) W_i(x) + C_i(y)$$

□

Performance. The prover work seems to be comparable (though it largely depends on optimizations involved in evaluation of wiring predicates). Amount of communication drops significantly compared to GKR - in each layer, instead of doing $2\mu_i$ rounds with messages of size 3, parties now only exchange $\tilde{\mu}_i = \lceil \log_2(2c_i + q) \rceil$ messages of size 3. In the dense scenario total communication is the same as before, and with multiplication predicate being relatively sparse it decreases the amount of work $\sim 2\times$. This also completely skips an additional subprotocol which is required to combine two claims about the value of W_i into a single one, saving us μ amount of communication and another round.

Higher degree circuits. It is not inconceivable that power circuits of larger degree can also be used. For example, computing 4-th power in a single layer yields the same amount of communication, but lesser amounts of rounds (and in a non-interactive Fiat-Shamir setting this is also a welcome tradeoff, because hashing larger strings at the same time is faster). However, amount of data which needs to be computed on a linear layer grows significantly in a worst-case scenario, which suggests that such optimizations are

better used for a very structured circuits consisting of the copies of the same polynomial gate.

4 Examples

Lets consider two simple but important examples.

Example 1. *Binary product tree. The initial layer has size* 2^n*, and every next layer is twice smaller, keeping products of elements* $w_0 = v_0v_1, w_1 = v_2v_3, ...$

Then, our extension operation writes elements

$$v_{01} = v_0 + v_1, v_{23} = v_2 + v_3, ...$$

. Reduction operation, on the other hand, computes

$$w_0 = \frac{1}{2}(v_{01}^2 - v_0^2 - v_1^2), w_1 = \frac{1}{2}(v_{23}^2 - v_2^2 - v_3^2, ...)$$

Combining this with the next extension layer, we get the linear expression which represents the new extended layer as the linear combination of squares of the previous one:

$$w_0 = \frac{1}{2}(v_{01}^2 - v_0^2 - v_1^2), w_1 = \frac{1}{2}(v_{23}^2 - v_2^2 - v_3^2, ...),$$

$$w_{01} = \frac{1}{2}(v_{01}^2 - v_0^2 - v_1^2 + v_{23}^2 - v_2^2 - v_3^2)$$

As one can see, the size of the layer only increased twofold (so, $\tilde{\mu}_i = \mu_i + 1$).

Example 2. *Logup-gkr fraction addition tree. It works as follows: along binary tree, we are adding pairs of the form* (a, b) *using the addition formula* $(a, b) \oplus (c, d) = (ad + bc, bd)$*.*

The details of the example are almost the same, we only need to see how extension works here. To compute $ad + bc$ and bd it is enough to consider squares of following elements: $a, b, c, d, b + d, a + d, b + c$. Indeed, $2bd = (b + d)^2 - b^2 - d^2$, and ad and bc are similarly expressed through squares of $a + d$ and $b + c$.

References

[1] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[2] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, oct 1992.

[3] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using gkr. Cryptology ePrint Archive, Paper 2023/1284, 2023. https://eprint.iacr.org/2023/1284.

[4] 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.

[5] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. Cryptology ePrint Archive, Paper 2013/351, 2013. https://eprint.iacr.org/2013/351.