A Note on Efficient Computation of the Multilinear Extension

Ron D. Rothblum[*]

July 11, 2024

Abstract

The multilinear extension of an m-variate function $f : \{0,1\}^m \to \mathbb{F}$, relative to a finite field $\mathbb{F}$, is the unique multilinear polynomial $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on inputs in $\{0,1\}^m$.

In this note we show how, given oracle access to $f : \{0,1\}^m \to \mathbb{F}$ and a point $z \in \mathbb{F}^m$, to compute $\hat{f}(z)$ using exactly 2^{m+1} multiplications, 2^m additions and $O(m)$ additional operations. The amount of space used corresponds to $O(m)$ field elements.

1 Introduction

The multilinear extension is a method for encoding the truth table of a function in a redundant form that is extremely useful in many applications. In particular, multilinear extensions have proven to be extremely useful in the development of efficient proof-systems. We refer the reader to the recent book by Thaler [Tha22], for further details on the key role that multilinear extensions play in these applications.

The Multilinear Extension. Let $\mathbb{F}$ be a finite field and $m \in \mathbb{N}$ be an integer. For every function $f : \{0,1\}^m \to \mathbb{F}$ there exists a unique multilinear polynomial $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on $\{0,1\}^m$. We refer to $\hat{f}$ as the *multilinear extension* of f.

The multilinear extension $\hat{f}$ can be expressed explicitly as:

$$\hat{f}(z) = \sum_{b \in \{0,1\}^m} eq(z,b) \cdot f(b), \tag{1}$$

where $eq(z,b) = \prod_{i \in [m]} eq_1(z_i, b_i)$ and $eq_1(z_i, b_i) = z_i \cdot b_i + (1 - z_i) \cdot (1 - b_i)$.

Efficient Evaluation of the Multilinear Extension. Consider the following basic computational task: given as input $z \in \mathbb{F}^m$ and the truth table of $f : \{0,1\}^m \to \mathbb{F}$, output $\hat{f}(z)$. A few methods for performing this computation have been described in the literature:

1. A direct method, described in [CTY11] (see also [Tha22, Lemma 3.7]) iterates over all indices $b \in \{0,1\}^m$ and for each index computes the contribution to the sum in Eq. (1). This involves computing the sequence of values $\big(eq(z,b)\big)_{b \in \{0,1\}^m}$. Each value in this sequence can be generated using $O(m)$ arithmetic operations, which leads to an overall cost of $O(m \cdot 2^m)$ field operations.

---

[*] Technion. Email: `rothblum@cs.technion.ac.il`.

2. An alternate method, due to [VSBW13] (see also [Tha22, Lemma 3.8]), observes that the values $eq(z, b)$'s in the foregoing sequence are in fact related, and uses a memoization approach to generate them. Using their approach to compute $\hat{f}(z)$ requires 2^{m+1} multiplications and 2^{m+1} additions.[1] This memoization technique however also requires $O(2^m)$ space, in contrast to the "direct" approach that uses $O(m)$ space. Their approach is also incompatible with some applications, such as in the context of streaming algorithms [CTY11, CMT12].

3. Chiesa *et al.* [CFFZ24, Section 4.1] implicitly gave a method for computing the multilinear extension using $O(2^m)$ field operations and $O(m)$ space. We remark that a similar result was obtained as an unpublished observation due to Victor Vu in 2013 [Tha24]. We describe Vu's approach in Appendix A.

Our Result. We describe a new approach for computing the multilinear extension, which improves on the aforementioned results – it requires only 2^{m+1} multiplications, 2^m additions (and an additional $O(m)$ field operations, including inversions) and uses only $O(m)$ space. In particular, the amount of space used is exponentially smaller than the [VSBW13] approach and the number of additions is about half. The main beneift over [CFFZ24, Section 4.1] is that the number of multiplications is halved.

In a nutshell, our approach follows the direct approach but utilizes the relations between the sequence of values $eq(z, b)$, to generate it more efficiently on-the-fly and while leveraging a particularly convenient order of the b's. We remark that our algorithm can be used in a streaming setting (i.e., when the values of f are given as a stream of data), as long as the stream is given in a specific order, which we describe below.

1.1 Applications and Related Works

We note that, typically, proof-systems have a high space usage regardless of the computation of the multilinear extensions. In such cases, the *asymptotic* improvement in space, that we achieve (as compared to [VSBW13]) is lost. Still, we believe that even in these contexts, our approach may be concretely beneficial both due to the concrete space saving and, in contexts in which it is a bottleneck, also the saving in the number of additions and/or multiplications.

In particular, proof-systems based on multilinear techniques, usually use an efficient implementations of the sumcheck protocol [Tha13, XZZ+19]. This implementation uses 2^m space regardless of the computation of the multilinear extension. Recent works by Setty *et al.* [STW23, Appendix G] and Chiesa *et al.* [CFFZ24] show a time-space tradeoff for the sumcheck protocol ([STW23] actually focus on the harder case of sparse polynomials). Still, even combining their techniques with our approach it is unclear how to construct a sumcheck prover that runs simultaneously in linear-time and logarithmic space. Recent works show other optimizations to the sumcheck prover [Che23, Gru24, BDT24, DT24], focusing on use cases that arise in practice. Their techniques could potentially be combined with our approach.

A recent line of work, [BTVW14, BHR+20, BHR+21, BCHO22, FPP24, NDC+24] has studied proof-systems in which the prover is space-efficient. In this context we believe that our algorithm may be a helpful also as a step towards achieving new asymptotic results.

---

[1] Actually, the description in [VSBW13] uses $3 \cdot 2^m$ multiplications and 2^m additions, but a known optimization (described, for example, in [DT24, Algorithm 1]) converts 2^m of the multiplications to be additions.

Setty *et al.* [STW23, Page 13] propose a way to obtain time-space tradeoff for computing the multilinear extension, that is geared towards sparse multilinear polynomials. In the general case though, it requires either a polynomial amount of space or super-linear time.

2 Computing the Multilinear Extension Efficiently

Proposition 1. *Given as input $z \in \mathbb{F}^m$, the sequence of values $(eq(z,b))_{b\in\{0,1\}^m}$ can be generated in time $O(2^m)$ and space $O(m)$. In more detail, the algorithm performs exactly 2^m field multiplications and an additional $O(m)$ additions, multiplications and* inversions.

Proof. Assume first that all of the entries of z are not in $\{0,1\}$ (later we shall show how to handle the general case).

We generate the sequence according to the *Gray code* ordering of the integers between 0 to $2^m - 1$. Recall that the Gray code has the property that the binary representation of each integer in the ordering only differs by a single bit from the previous one – i.e., it can be produced by XORing the current index with a unit vector. For any $b \in \{0,1\}^m$ and $i \in [m]$, using e_i to denote the i-th unit vector we have that:

$$eq(z, b \oplus e_i) = eq_1(z_i, b_i \oplus 1) \cdot \prod_{j \neq i} eq_1(z_j, b_j) = \frac{eq_1(z_i, b_i \oplus 1)}{eq_1(z_i, b_i)} \cdot eq(z,b), \tag{2}$$

where we use our assumption that $z_i \notin \{0,1\}$ so as not to divide by 0.[2] By precomputing the $2m$ values $\left(\frac{eq_1(z_i,\sigma\oplus 1)}{eq_1(z_i,\sigma)}\right)_{i\in[m],\sigma\in\{0,1\}}$ before the enumeration starts, using Eq. (2) we can therefore compute $eq(z, b\oplus e_i)$ from $eq(z,b)$ using a single multiplication.

Thus, we can generate the sequence of values $(eq(z,b))_{b\in\{0,1\}^m}$, by enumerating over $b \in \{0,1\}^m$, according to the Gray code order, and in each step store only the previous $eq(z,b)$ and update it using a single multiplication. Overall, following the precomputation phase, the process requires $2^m - 1$ multiplications.

To handle general vectors $z \in \mathbb{F}^m$, we partition z according to coordinates $S \subseteq [m]$ that are Boolean valued vs. those that are not (i.e., coordinates in $\bar{S}$ are in $\mathbb{F}\backslash\{0,1\}$). Recall that for every $b \in \{0,1\}^m$ it holds that $eq(z,b) = eq(z_S, b_S) \cdot eq(z_{\bar{S}}, b_{\bar{S}})$. Observe that $eq(z_S, b_S) = 0$ whenever $b_S \neq z_S$ and otherwise (i.e., $z_S = b_S$)) it holds that $eq(z_S, b_S) = 1$. Thus, in our enumeration we can skip over all vectors b for which $b_S \neq z_S$ and for the remaining vectors, perform the enumeration only over the coordinates in $\bar{S}$. □

Combining Proposition 1 with Eq. (1) we immediately obtain the following corollary.

Corollary 2. *There exists a time $O(2^m)$ and space $O(m)$ algorithm that, given as input $z \in \mathbb{F}^m$ and oracle access to a function $f : \{0,1\}^m \to \mathbb{F}$, outputs $\hat{f}(z)$. In more detail, the algorithm performs exactly 2^{m+1} multiplications, 2^m additions and an additional $O(m)$ field operations.*

Note that in case the function f is Boolean-valued, the number of multiplications reduces to 2^m.

Remark 3. *Our approach can be parallelized by partitioning the index set 2^m into k equal sized parts and handling each part separately. This will reduce the parallel time by a factor of k but requires an additional $k \cdot m$ bits of space, due to the k running indices.*

---

[2] Observe that $eq(z_i, b_i) = 0$ if and only if $z_i = NOT(b_i)$.

2.1 Generating $eq(z, \cdot)$ in Lexicographic Order

In some applications it may be important to generate the stream of values $eq(z, \cdot)$ in lexicographic order rather than the Gray code order used in Proposition 1. For example, if the function f is given as a stream of values in lexicographic order.

We describe two ways to adapt our algorithm to this setting:

1. We can use a similar algorithm to the one in Proposition 1 while observing that (1) the number of multiplications needed per update is equal to the Hamming distance from the (binary representation of the) previous index, and (2) that the *amortized* Hamming distance between the binary representation of consecutive integers (in lexicographic order) is roughly 2. However, this approach doubles the number of multiplications as compared to the Gray code order. As noted above, an approach similar to this was made as an unpublished observation by Vu, see Appendix A.

2. If $\mathbb{F}$ is a binary extension field (i.e., it has characteristic 2), we can handle the lexicographic order without doubling the number of multiplications, as follows.

   Consider the function $G : \{0,1\}^m \to \{0,1\}^m$ that maps an index $i \in \{0,1\}^m$ to its Gray code encoding (i.e., the i-th index in the Gray code order). This function is *linear* over $\mathbb{GF}(2)$ (see, e.g., [BCC+10, Section 2]). Observe that $eq(z, G(b)) = eq(G^{-1}(z), b)$ clearly holds for all $z \in \{0,1\}^m$. But since G is linear, both sides of the equation are multilinear (in z), and therefore the equation must hold for all $z \in \mathbb{F}^m$.

   Thus, when using a binary extension field, to stream the values of $eq(z, b)$ in lexicographic order, we can simply use the algorithm described in the proof of Proposition 1 to stream the values of $eq(G^{-1}(z), b))$ in the Gray code order.

Acknowledgements

The author is grateful to Binyi Chen, Ben Fisch, Justin Thaler and Hadas Zeilberger for extremely useful discussions. We thank Giacomo Fenzi for pointing out the connection to [CFFZ24, Section 4.1].

References

[BCC$^+$10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in F_2. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010. 4

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May*

30 - June 3, 2022, Proceedings, Part II, volume 13276 of *Lecture Notes in Computer Science*, pages 427–457. Springer, 2022. 2

[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`. 2

[BHR+20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 168–197. Springer, 2020. 2

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152. Springer, 2021. 2

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. Cryptology ePrint Archive, Paper 2014/846, 2014. `https://eprint.iacr.org/2014/846`. 2

[CFFZ24] Alessandro Chiesa, Elisabetta Fedele, Giacomo Fenzi, and Andrew Zitek-Estrada. A time-space tradeoff for the sumcheck prover. *IACR Cryptol. ePrint Arch.*, page 524, 2024. 2, 4

[Che23] Binyi Chen. Hardware-optimizations for sumcheck, 2023. `https://hackmd.io/@bychen92/H1JDAAMoo`. 2

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 90–112. ACM, 2012. 2

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, 2011. 1, 2

[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`. 2, 6

[FPP24] Cody Freitag, Omer Paneth, and Rafael Pass. Public-coin, complexity-preserving, succinct arguments of knowledge for NP from collision-resistance. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part IV*, volume 14654 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2024. 2

[Gru24] Angus Gruen. Some improvements for the PIOP for ZeroCheck. Cryptology ePrint Archive, Paper 2024/108, 2024. `https://eprint.iacr.org/2024/108`. 2

[NDC$^+$24] Wilson D. Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. *IACR Cryptol. ePrint Arch.*, page 416, 2024. 2

[STW23] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with lasso. *IACR Cryptol. ePrint Arch.*, page 1216, 2023. 2, 3

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013. 2

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. 1, 2

[Tha24] Justin Thaler. Personal Communication, 2024. 2

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237. IEEE Computer Society, 2013. 2, 6

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. *IACR Cryptol. ePrint Arch.*, page 317, 2019. 2

A Vu's Approach

In this appendix we describe Vu's approach for a space efficient implementation of the [VSBW13] algorithm for computing the sequence of values $eq(z, \cdot)$.

For every vector $b \in \{0,1\}^j$, where $j \in \{0, \dots, m\}$, let $\chi_b = \prod_{i=1}^{j} eq_1(z_i, b_i)$. Consider a full binary tree with 2^m leaves, where each vertex is labeled by a string of length at most m in the following way: the root is labeled with the empty string, and the two children of a vertex labeled by b, are labeled by $b0$ and $b1$. We associate with every vertex b the value χ_b defined above.

Observe that for $b \in \{0,1\}^j$ and $\sigma \in \{0,1\}$, it holds that $\chi_{b\sigma} = \chi_b \cdot eq_1(z_{i+1}, \sigma)$. Thus, the value associated with each vertex can be computed from its parent using a single multiplication (or using the optimization in [DT24, Algorithm 1], both children can be computed using one multiplication and one addition).

The idea underlying [VSBW13] is to generate the entire tree, and observe that the desired values are associated with the leaves. Vu's observation is that the values associated in the leaves can be generated by scanning the leaves via a depth-first search of the tree, which requires storing at most m field elements associated with the current position in the scan.