Sumcheck with low memory throughput

1 Introduction

Let's imagine we want to run the sumcheck protocol in a world where the prover's machine is memory bounded (i.e. read / writes are the main bottleneck compared to finite field operations). Note that this is very different from the case where the prover has limited memory space. Here the memory is unlimited, but slow.

In our scenario, the prover wants to convince the verifier of the following value:

$$\sum_{i \in \{0,1\}^n} F(M_1(i), \dots, M_k(i))$$

Where $M_1, \dots, M_k$ are k multilinear polynomials in n variables (that the verifier can query at some point) and F is a multivariate polynomial in k variables of degree d.

We will assume the prover's machine has a small cache on which memory reads / writes are highly efficient (which is typically the case on CPU).

2 Classical algorithm

At round $r \in \{1, \dots, n\}$, the prover computes the sumcheck polynomial, of degree d, by evaluating F at d points (we can skip one evaluation using section 3.1 of [1]). Cost $= k.2^{n+1-r}$ memory reads.

Then, after receiving a random challenge, the prover must fold each multilinear polynomial. Cost $= k.2^{n+1-r}$ memory reads, $k.2^{n-r}$ memory writes.

The total cost is: $k.2^{n+2}$ memory reads, $k.2^n$ memory writes.

3 Trivial improvement

Except for the first round, we can batch together the folding of the multilinears and the computation of the next sumcheck polynomial. This saves $k.2^{n-1} + k.2^{n-2} + \cdots = k.2^n$ memory reads (25%). Note that from an implementation perspective, this approach is very friendly to the optimizations for small-field sumcheck of [2].

4 Trading memory throughput for compute

The first sumcheck polynomial the prover must compute is the following:

$$P_1(X) = \sum_{i \in \{0,1\}^{n-1}} F(M_1(X, i), \dots, M_k(X, i))$$

Classically, the prover starts by computing $P_1(X)$ and wait for the the random random challenge α_1 before folding / computing $P_2(X)$.

We could instead compute, in advance, the next s sumcheck polynomials $P_1(X), \dots, P_s(X)$. In reality, $P_2(X)$ depends on the random challenge α_1, $P_3(X)$ depends on α_1 and α_2 etc. So, what the prover must actually compute is: $P_1(X), P_2(X, \alpha_1), P_3(X, \alpha_1, \alpha_2), \dots, P_s(X, \alpha_1, \dots, \alpha_{s-1})$.

The last sumcheck polynomial to compute here, $P_{1+s}(X, \alpha_2, \alpha_3, \dots, \alpha_s)$, has s variables, and degree $s.d$, meaning we need $\binom{s+s.d}{s}$ evaluation points (to perform a generalized Lagrange interpolation, in multiple variables). That's why we cannot take a too big value for s, otherwise compute would likely become a bottleneck again.

Computing these s sumcheck polynomials can be done in $k.2^n$ memory reads. After receiving the random challenges $\alpha_1, \dots, \alpha_s$, we can fold, and at the same time compute the next sumcheck polynomial $P_{s+1}(X)$, as suggested by 3. But instead, we can, again, use the same technique to compute in advance $P_{s+1}(X), \dots, P_{2s}(X)$.

Total cost:

- $2.k.2^n + k.2^{n-s} + k.2^{n-2s} + k.2^{n-3s} + \dots = k.2^n(1 + \frac{2^s}{2^s-1})$ memory reads.
- $k.2^{n-s} + k.2^{n-2s} + \dots = k.2^{n-s} . \frac{2^s}{2^s-1}$

Asymptotically, we can save up tp 33% of memory reads and 100% of memory writes compared with 3.

The computational overhead is (TODO do the maths carefully):

$$\approx \frac{\binom{s.(d+1)}{s} \cdot 2^{n-s}}{\sum_{r=1}^{s} d.2^{n-r}}$$

In practice, with $s = 2$, we spare 22% of memory reads, 66% of memory writes, at the cost of a factor increase in compute of:

$$\frac{(d+1).(2.d+1)}{3d}$$

For instance, in WHIR's sumcheck, $d = 2$ gives a compute overhead of 2.5x.

5 Unintended consequence

If some the multilinears $M_1, \dots, M_k$ are initially in a small field, the 2nd, 3rd, ..., s-th sumcheck polynomials are much easier to compute than with the classical algorithm, because all the computations happen in the small field. In WHIR for instance, with an extension degree 4 or 5, the computational overhead is potentially bellow 2x (TODO be more precise) (This is the case when the committed polynomial is the small field, in which case one of the 2 multilinears of our sumcheck (the weights) is in the big field, and the other (the evals) in the small field).

References

[1] A. Gruen, “Some improvements for the PIOP for ZeroCheck,” 2024. [Online]. Available: https://eprint.iacr.org/2024/108

[2] S. Bagad, Q. Dao, Y. Domb, and J. Thaler, “Speeding up sum-check proving,” Cryptology ePrint Archive, Paper 2025/1117, 2025. [Online]. Available: https://eprint.iacr.org/2025/1117