

On the recently proposed $o(n \log n)$ integer multiplication algorithm

David Harvey, UNSW Sydney

20th October 2026

Abstract

I offer some thoughts about the recently proposed integer multiplication algorithm whose complexity is claimed to be $o(n \log n)$ bit operations.

1 Executive summary

I have not checked most of the details, but I think that the claimed $o(n \log n)$ bound is plausible.

2 Declaration

This note was written without AI assistance, except for (a) typesetting (i.e., TeX wizardry), (b) proofreading, and most importantly (c) extensive discussions to help me understand the new algorithms. The main system used throughout was Claude Fable 5.1.

3 Introduction and background

A certain American technology company (henceforth “the Company”) recently dumped a large pile of manuscripts on GitHub. This “math(s) dump” contains many remarkable claims and results.

I have struggled to find the words to convey the stupendousness of these discoveries to my non-mathematical associates.

One paper in particular caught my attention [TC26a]. It proposes a new algorithm for multiplying n -bit integers, claiming that it runs in $o(n \log n)$ bit operations, in either the multitape Turing model or the Boolean circuit model. The small “ o ” is not a typo. More precisely, the paper claims a complexity bound of the form $O(n(\log n)^{1-\varepsilon})$ for some small positive constant ε .

This bound is of course utterly ridiculous. Previously the best bound was $O(n \log n)$ (big “ O ”), and this was widely believed to be optimal. The dump contains many other ludicrous results, but most of them were at least supposed to be *true*. This one was not.

Nevertheless, this is a serious Company with a track record of doing serious things, so we should make an effort to take the claim seriously.

improvement made by my work with van der Hoeven (and several other papers along the way) was much, much, *much* smaller than this. The claimed $(\log n)^\varepsilon$ improvement in the new work is *exponentially larger* than any of these. The actual value of ε is irrelevant. This is a *huge* gap. English has many synonyms for “big” but not enough for what I am trying to say.

I suspect that very few people (i.e., humans) worldwide understand all the details of the 2021 algorithm, but I happen to be one of them. This puts me in the rare position of having the background to be able to comment immediately on the new work.

4 Executive summary (longer version)

The bottom line is that I think the claimed result is plausible. I have not actually read the paper. In fact, there are three closely related papers [TC26a], [TC26b], [TC26c], and I have not read any of them. I tried, but soon gave up. The math(s) dump does not appear to be strong on exposition.

Instead, I asked Fable to explain [TC26a] to me. Fable seems to understand the argument and thinks that it is correct. I expect that thousands of frontier LLM instances around the world have come to the same conclusion over the last two weeks. If not, surely we would have heard by now.

At first I did not understand Fable’s explanations. But I asked questions, and then asked more questions, and drilled down towards the core of the problem. Gradually I started to understand parts of the argument. I have seen enough now to arrive at the tentative belief that the main result is possibly correct (notice that I choose my words very carefully), and that the bound $O(n(\log n)^\alpha)$ has the right shape. I stress that I have not checked the proof in any serious level of detail, and there are many parts that I do not yet understand properly. In the sections below I will try to explain what I have learned so far, and why I lean towards thinking it might be correct.

Incidentally, when I say that the math(s) dump “does not appear to be strong on exposition”, all I mean is that I, as an human expert in this field, have difficulty understanding it (specifically [TC26a]). I have heard similar comments from other human experts regarding other papers in the dump. But as far as Fable seems to be concerned, the exposition in [TC26a] is perfectly clear. So perhaps the problem is not with the exposition, but with us humans. Before the advent of AI, it was already the case that different styles or levels of exposition were appropriate for different audiences. Perhaps we need to ask the LLMs to “explain it to me like I’m human” (ELIH).

There is one other aspect of the situation that I would like to emphasise. After two weeks of very hard work, I have managed to understand only a small fraction of [TC26a]. When I ask Fable about the paper, it seems to read and understand the whole thing in just a few minutes, even starting from a fresh context. I know that some readers may be uncomfortable with my anthropomorphic terminology, but I believe that “understand” is accurate, in the sense that the LLM seems to be able to correctly and lucidly answer all kinds of questions about the paper, including about some very fiddly technical details. That is exactly how I would tell that a human colleague has “understood” something. This chasm between human and LLM “speed of thought” is unsettling to say the least.

It would be good to have a formalised proof of the $o(n \log n)$ result, for example in Lean. As far as I can tell, one does not yet exist. Such a formalisation seems to be achievable by current technology, but it is still not easy. If the result is correct, I expect a formalisation to be available fairly soon. My confidence in the result would be increased greatly if such a formalisation existed, assuming

that the initial definitions and statement of the main theorem had been checked by a (human?) Lean expert.

I apologise that in this note I have not maintained my usual level of care and scholarship. Everything is happening so fast. In particular I apologise to those people whose work is used in this note without adequate attribution, due to my ignorance and/or negligence. Maybe one day I will make a more careful attempt, though I suspect that an LLM will beat me to it.

Before getting to the explanations, I want to mention some consequences of the main result, assuming that it is correct:

- There is now a quantity that I would suggest calling the “exponent of integer multiplication”. It is the infimum of real numbers $\alpha \geq 0$ such that there exists an integer multiplication running in time $O(n(\log n)^\alpha)$. We previously believed that $\alpha = 1$, but the Model’s claim is that $\alpha < 1$. Already some people have found improvements in the value of α . At the time of writing the best value appears to be $\alpha \approx 0.9983$, but this number will surely be out of date by the time you are reading this.

In fact, there might be two different versions of α : a “Boolean circuit” version that only takes into account the cost of arithmetic, and a “Turing machine” version that also includes the cost of data rearrangement. It is not at all clear to me that these exponents should take the same value.

Some people have asked me, how low could α go? Of course I have no idea, but I want to mention one obstruction. In [HvdH21], the Gaussian resampling step contributes altogether time $O(n(\log n)^{3/4})$. As sketched in [HvdH21, §4.4.2], this step can be improved to $O(n(\log n)^{1/2})$. So without further improvements here, or a completely different approach that does not rely on Gaussian resampling, it does not seem possible to do better than $\alpha = 1/2$. One method that does not depend on Gaussian resampling is the variant in [HvdH22], but this unfortunately requires some unproved number-theoretic hypotheses. There has been so much progress in analytic number theory in recent weeks that it might be worth taking another look. Either way, we are still very far away from the $1/2$ limit, so I would advise against thinking about it too hard yet.

- Together with the Boolean circuit lower bound result of [AFKL19], the new result disproves the Li–Li network coding conjecture [LL04].
- The new result does not directly imply new complexity bounds for polynomial multiplication over a fixed finite field, say multiplication in $\mathbb{F}_2[x]$. The state of the art for this problem is an algorithm that multiplies polynomials of degree n in time $O(n \log n 4^{\log^* n})$ [HvdH19]. I do not immediately see how to adapt the new method to that algorithm. However, the method of [HvdH22] also works for polynomials over finite fields, and achieves the better bound $O(n \log n)$ under certain number-theoretic hypotheses. The structure of that algorithm suggests that the new techniques are applicable, and that they would lower its complexity to $O(n(\log n)^\alpha)$ (still conditional however).
- My recent paper on fast enumeration of primes [Har26] relies heavily on fast multiplication in $\mathbb{F}_2[x]$. Maybe the results of that paper can be improved by plugging in the (conditional) multiplication algorithm just mentioned. I had a very quick look, and it seems that the

bottleneck becomes the computation of the “restricted products”. So I am not sure, but it is certainly tempting to investigate. Some care is needed, because the current framework in the paper more or less assumes that the cost of multiplication is $\Omega(n \log n)$; one would have to audit the entire complexity analysis from scratch without this assumption.

- When the $O(n \log n)$ integer multiplication result was first announced in March 2019, Fredrik Johansson wrote on Reddit:

The result is of extreme practical importance. Not for actually multiplying integers (as usual with these algorithms it’s probably not faster than existing algorithms for integers that can be stored in the observable universe), but for writing papers. It has always been a hassle to write down the complexity of integer multiplication or algorithms based on integer multiplication by introducing soft- O notation, little- o exponents, epsilons greater than 0, or iterated logarithms. From now on I can just write $O(n \log n)$ in my papers and be done with it!

Alas, after seven years’ sojourn in this typographical Utopia, we are now condemned to wander forever in a wasteland of superscripts and curly braces.

Or, if you prefer, you can continue to write $O(n \log n)$. No harm will come to you.

5 The 2021 algorithm

To understand the new integer multiplication algorithm, one must first understand essentially everything in the 2021 algorithm. It is impossible for me to explain all of this here. I will give instead a brief summary of the points that are relevant for understanding how the new algorithm changes the analysis. Readers interested in the details should consult [HvdH21], which by all accounts is a superbly written paper.

We start with two n -bit integers. After a series of reductions that I will not attempt to explain (Agarwal–Cooley CRT rearrangement, Gaussian resampling, etc), we reduce to the problem of computing a Nussbaumer-style DFT (discrete Fourier transform) of a certain $(d - 1)$ -dimensional array of size $r \times \cdots \times r$, whose entries lie in the ring $R = \mathbb{C}[y]/(y^r + 1)$. Here d is a *fixed* dimension parameter (independent of n) and r is a power of two, of size roughly $n^{1/d}$. The coefficient ring R contains a principal $2r$ -th root of unity, namely y , and this is the “synthetic” root of unity used to define the DFTs. Thus, the DFT may be computed by an FFT (fast Fourier transform) algorithm, which amounts to performing a series of “butterflies”, i.e., maps $(a, b) \mapsto (a + b, a - b)$ operating on pairs of complex numbers, together with multiplications by powers of y , which are just shifts and negations. The complex numbers themselves are represented as fixed point approximations with $p = O(\log n)$ bits of precision.

The number of butterflies (over $\mathbb{C}$) is easily calculated to be $\frac{1}{2}r^d \log_2(r^{d-1})$. One can already see the $O(n \log n)$ shape of the final complexity bound in this butterfly count.

These reductions lead to a recursive bound for $M(n)$ (the cost of multiplying n -bit integers) of the form

$$M(n) < \frac{Kn}{n'} M(n') + O(n \log n), \quad n' = n^{\frac{1}{d} + o(1)}.$$

The key point is that the “expansion factor” K is an absolute constant, independent of d . In the paper we show that one may take $K = 1728$; this arises from analysing the zero-padding and encoding steps that occur during the various reductions. (In §5.4 we sketch various techniques that would reduce it to $K = 8$.)

The first $(Kn/n') M(n')$ term arises from the cost of the recursive multiplications in the Nussbaumer convolution. Provided we choose $d > K$, for example $d = 1729$, then unrolling the recursion leads to a geometric progression and keeps the cost of the subsequent recursion levels under control.

For simplicity, in [HvdH21] we keep d constant, since all we need is $d > K$. In the new algorithm one actually needs to take d increasing with n . When d grows with n , one must be more careful with certain estimates, but it introduces no essential difficulties. (In fact, in the initial stages of writing [HvdH21], we had d increasing with n , and only later in the project did we decide to fix d .)



The crucial question now is where the $O(n \log n)$ term comes from. In the Boolean circuit model, it arises entirely from the cost of the butterfly arithmetic; everything else is negligible. In the Turing model, there are additional data rearrangement costs, such as transposing arrays, which also contribute $O(n \log n)$.

If you had asked me two weeks ago for my subjective estimate of the probability that either of these $O(n \log n)$ contributions could be improved, I think I would have guessed roughly 0.01% for the data rearrangement steps, and precisely zero for the arithmetic steps. In the past, I have tried, perhaps half-heartedly, to invent faster matrix transposition algorithms. But it has never, ever, even remotely occurred to me to launch an attack on the butterfly arithmetic. Since I first learned about FFTs several decades ago, this barrier has always seemed to be woven into the fabric of the universe, a hard physical limit like the speed of light.

But the Model now asserts that we can in fact go faster than light. In the following sections I will try to explain its reasoning.

Once you believe that the butterfly arithmetic can be improved, then the rearrangement side becomes less surprising. We already know that there is a close relationship between these two problems [HvdH25]. Therefore, in my study of the new algorithm so far, I have completely ignored the data rearrangement part of the problem, and focused entirely on the arithmetic side, which I think is more fundamental. If you like, I am only thinking about the Boolean circuit model, and ignoring the Turing situation for the moment. This will be the context that I work in for the rest of this note.

6 Linear algebra and amplification

I am not really an algebra person. Tensor products are hard. Deep breath and repeat after me: “growth mindset”.

Let A be a 2×2 complex matrix. It acts in a natural way on $\mathbb{C}^2$. For example, A could be the butterfly map $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Now let m be a positive integer and consider $(\mathbb{C}^2)^{\otimes m}$. This is the m -th tensor power of $\mathbb{C}^2$. You can think of an element of $(\mathbb{C}^2)^{\otimes m} \cong \mathbb{C}^{2^m}$ as an m -dimensional array of complex numbers of size

$2 \times \cdots \times 2$ (m factors).

If $A_1, \dots, A_m$ are 2×2 matrices, then $A_1 \otimes \cdots \otimes A_m$ acts on $(\mathbb{C}^2)^{\otimes m}$ in a natural way:

$$A_1 \otimes \cdots \otimes A_m = (A_1 \otimes I \otimes \cdots \otimes I)(I \otimes A_2 \otimes \cdots \otimes I) \cdots (I \otimes I \otimes \cdots \otimes A_m).$$

This decomposition is just algebra, but it also expresses an *algorithm* for applying $A_1 \otimes \cdots \otimes A_m$: first act on slices in the m -th direction by A_m , then act on slices in the $(m-1)$ -th direction by A_{m-1} , and so on. And because the slice operators commute, you can do these m steps in any order. Each A_i is applied 2^{m-1} times, so the total number of 2×2 matrix applications is $m \cdot 2^{m-1}$.

Let's now focus on the case $A_1 = \cdots = A_m = A$, so we are studying the tensor power $A^{\otimes m}$ of a single A . Applying $A^{\otimes m}$ to a vector in $(\mathbb{C}^2)^{\otimes m}$ using the above algorithm requires $m \cdot 2^{m-1}$ applications of A . For example, if A is the butterfly matrix, then this is just the standard number of butterflies needed to compute the DFT of a $2 \times \cdots \times 2$ array.

Now it is conceivable that for certain A , one might be able to evaluate $A^{\otimes m}$ using fewer than $m \cdot 2^{m-1}$ applications of A . This is trivially true for example if $A = 0$ or $A = I$! Suppose however that A is not so trivial, and that someone magically hands us an algorithm for evaluating $A^{\otimes m}$ for some m , that uses only t applications of A , where t is strictly smaller than $m \cdot 2^{m-1}$. How can we use this algorithm to speed up evaluations of $A^{\otimes n}$, where n is much larger than m ?

Here is a fairly standard method. Suppose for simplicity that $n = m\ell$ for some integer $\ell \geq 1$. Then $A^{\otimes n} = A^{\otimes m\ell} \cong A^{\otimes m} \otimes \cdots \otimes A^{\otimes m}$ (ℓ factors). Essentially we have split up the original n axes of the array into ℓ groups of m axes. So what we could do is apply our fast algorithm for $A^{\otimes m}$ along the first group of axes, then along the second group, and so on. This reduces the number of applications of A from the original $n \cdot 2^{n-1}$ to $(t/(m \cdot 2^{m-1})) \cdot n \cdot 2^{n-1}$. This saving is not very interesting, because it is by a factor of only $t/(m \cdot 2^{m-1})$, regardless of the size of n .

What we really want is a saving that *grows with n* . This can be achieved by the following method, which Fable refers to as “amplification”. (The paper [TC26a] does not use this term. It is used in the companion papers. Fable points to prior art in Alman (2021); Alman, Guan and Padaki (2023); Alman and Li (2025). I have not followed up these references. The amplification trick is certainly new to me. See also comments about Strassen’s matrix multiplication method below.)

For amplification to work, we have to assume that our fast algorithm for $A^{\otimes m}$ has some additional structure. Let us denote by $(\mathbb{F}_2)^m$ the space of “addresses” for $(\mathbb{C}^2)^{\otimes m} \cong \mathbb{C}^{2^m}$, i.e., the vector $u = (u_1, \dots, u_m) \in \mathbb{F}_2^m$ refers to the $(u_1, \dots, u_m)$ -th coordinate of $(\mathbb{C}^2)^{\otimes m}$. We consider only algorithms given as sequences of the following two types of “instructions”:

- *Slice call.* For each $i = 1, \dots, m$, the operator $A_i := I \otimes \cdots \otimes A \otimes \cdots \otimes I$, i.e., A_i acts on $(\mathbb{C}^2)^{\otimes m}$ by applying A along the i -th direction. This amounts to 2^{m-1} invocations of A .
- *Relabelling.* For each matrix $M \in \text{GL}_m(\mathbb{F}_2)$, the operator P_M that acts on $(\mathbb{C}^2)^{\otimes m}$ by permuting the entries according to $x \mapsto Mx$ (for $x \in \mathbb{F}_2^m$).

For example, if M is a permutation matrix that swaps two coordinates of $\mathbb{F}_2^m$, then P_M just switches two axes of the array. But M could also involve more complicated transformations

that “mix up” the axes (i.e., bits of the addresses) in interesting (linear) ways. (Of course, this setup is not general enough to allow *arbitrary* permutations of the 2^m coordinates of $(\mathbb{C}^2)^{\otimes m}$.)

Note that in the Boolean circuit model, these sorts of relabellings are entirely free. In the Turing model we would have to account for their cost, but I am ignoring this here.

An example of such a “program” is the obvious one $A^{\otimes m} = A_1 A_2 \cdots A_m$ that we already discussed earlier. This performs exactly m slice calls, with no relabelling steps.

Suppose now that we are magically given a program for computing $A^{\otimes m}$ that performs only g slice calls, where g is strictly less than m . Without loss of generality it has the form $P_{M_0} A_{i_1} P_{M_1} A_{i_2} \cdots A_{i_g} P_{M_g}$. (Of course we could get away with just A_1 if we modified the M ’s appropriately. But it is convenient to allow ourselves the extra flexibility of the A_i ’s.)

Let us try to use this program to speed up the computation of $A^{\otimes n}$, where as before $n = m\ell$ for some $\ell \geq 1$. First observe that

$$A^{\otimes n} = (A^{\otimes m})^{\otimes \ell} = A^{\otimes m} \otimes \cdots \otimes A^{\otimes m}.$$

In terms of arrays, the formula is splitting up the n axes into ℓ groups of m axes. Now we substitute in the program:

$$A^{\otimes n} = (P_{M_0} A_{i_1} P_{M_1} A_{i_2} \cdots A_{i_g} P_{M_g})^{\otimes \ell}.$$

Keep in mind that each of the ℓ copies of (say) A_{i_1} is actually operating on a different group of m axes. We can bring the tensor product inside the composition:

$$A^{\otimes n} = (P_{M_0})^{\otimes \ell} (A_{i_1})^{\otimes \ell} \cdots (A_{i_g})^{\otimes \ell} (P_{M_g})^{\otimes \ell}.$$

I don’t know about you, but at this point my head is spinning. Let’s try to decode what is happening.

Consider first one of the factors $(P_{M_j})^{\otimes \ell}$. This is applying the same relabelling map P_{M_j} (which only acts on m axes) to each of the ℓ groups of m axes. You could do them one group at a time, or all at once; in the Boolean circuit model it makes no difference, it is free.

Now consider one of the factors $(A_{i_j})^{\otimes \ell}$. This is really $(I \otimes \cdots \otimes A \otimes \cdots \otimes I)^{\otimes \ell}$, where we are applying A in the i_j -th component. In other words, this map is applying $A^{\otimes \ell}$ to a subset of ℓ of the original n axes; it chooses one axis (the i_j -th one) out of each group of m axes, and only operates on those.

Let’s compare the total work to what would have happened if we had instead used the standard program $A_1 \cdots A_m$ for $A^{\otimes m}$. In effect, we have reduced the number of calls to $A^{\otimes \ell}$ by a factor of g/m . So if we don’t use any special techniques to evaluate $A^{\otimes \ell}$, then we save a factor of g/m in the total number of applications of A .

But of course, if ℓ is itself a multiple of m , we can (and should!) apply this strategy to $A^{\otimes \ell}$ itself to save another factor of g/m . And then of course we should do it recursively: if we take $n = m^k$ at the beginning, then overall we save a factor of $(g/m)^k$. This is the amplification: the additional

structure lets us save further factors of g/m as n grows. The total number of applications of A becomes

$$\begin{aligned} (g/m)^k \cdot n \cdot 2^{n-1} &= \exp\left(\frac{\log n}{\log m} \log(g/m)\right) \cdot n \cdot 2^{n-1} \\ &= n^\alpha \cdot 2^{n-1} \end{aligned}$$

where $\alpha = \log g / \log m$. You can see here the $N(\log N)^\alpha$ shape of the result (where $N = 2^n$). Notice that if $g < m$, then α is strictly less than 1.

(The amplification trick is reminiscent of Strassen’s matrix multiplication algorithm. But notice that in Strassen’s method, k recursion levels gets you from dimension 2 to dimension 2^k , whereas here k recursion levels gets you from dimension 2^m to dimension 2^{m^k} , i.e., doubly instead of singly exponential in k . Fable explained the difference to me as follows: in Strassen’s method, you are “tensoring a fixed gadget with the problem”, whereas here we are “substituting the problem into the gadget as a tensor power”. So I think they are algebraically not the same thing.)

Applying the preceding construction to the butterfly map $A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, we see that if we can find a program for $A^{\otimes m}$ with $g < m$ for *even one single value of m* , then we can compute DFTs on $(\mathbb{C}^2)^{\otimes n}$ for large n in essentially $O(N(\log N)^\alpha)$ amount of work ($N = 2^n$), with $\alpha < 1$.

The picture, as I have sketched it so far, is missing two main ingredients. First, as mentioned earlier, in the 2021 algorithm we don’t want to compute DFTs of size $2 \times \cdots \times 2$ over $\mathbb{C}$, but rather DFTs of size $r \times \cdots \times r$ over $R = \mathbb{C}[y]/(y^r + 1)$. I am pretty sure that the method will transfer to this situation. The latter problem can certainly be decomposed into butterflies over $\mathbb{C}$ of the right shape. I’m not sure exactly what happens to the twiddle factor multiplications (i.e., synthetic roots of unity), but these should be free in the Boolean circuit model (except possibly for chasing some signs around), and I doubt that they interfere in an essential way with the key amplification idea, which is really about how to deal with high dimensional tensor products. You can also see why we need the dimension parameter d to increase with n ; we need room to allow $n = m^k$ with growing k . All of this sounds plausible to me but I haven’t worked through it carefully yet.

The second thing we are missing is that the Model has *not* in fact found a way of obtaining a program with $g < m$. (Moreover, Fable tells me this is impossible, but I do not follow the argument yet.) Instead, what it has found, as I will discuss in the next section, is a way of doing this *on average* when evaluating $A^{\otimes m}$ on sufficiently many independent input arrays.

More precisely, taking A to be the butterfly map, and for an integer $W \geq 1$, consider the problem of evaluating $I \otimes A^{\otimes m}$ on $\mathbb{C}^W \otimes (\mathbb{C}^2)^{\otimes m}$. An element of $\mathbb{C}^W \otimes (\mathbb{C}^2)^{\otimes m}$ is just a list of W arrays of size 2^m , and evaluating $I \otimes A^{\otimes m}$ means applying $A^{\otimes m}$ to each of the W arrays. The standard number of slice calls needed to evaluate $I \otimes A^{\otimes m}$ is Wm , i.e., just transform each of the arrays separately. In the next section I will discuss how to construct a gadget that gets away with slightly less (for suitably chosen m and W). The gadget also includes some other linear operations that “mix up” the entries of the W arrays. Note that achieving even one fewer slice call, for a fixed m and W , is enough to make the whole argument go through. (Well, this is not the whole story, because we also have to track the number of “mix up” operations, which I have not thought about carefully yet. The reason I am not too worried is because of the analogy with Strassen’s matrix

multiplication; the key issue is to reduce the number of recursive calls, and the other arithmetic (like the extra additions in Strassen’s algorithm) should hopefully just sort itself out.) Once we have such a gadget available, it is not hard to modify the calling framework to take advantage of it; after all we do have to perform lots of independent butterflies, and W is fixed. (Fable tells me that the choice of how you allocate these W coefficients can affect the exponent, but I haven’t looked into this yet.)

The reader might think that it sounds crazy that one could speed up W independent evaluations of $A^{\otimes m}$ without being able to speed up a single evaluation on its own. This was certainly my first reaction. But in fact, there is a compelling precedent. Consider the problem of multiplying an $n \times n$ matrix by a single vector $v \in \mathbb{C}^n$. Assuming the matrix has no special structure, this requires $\Omega(n^2)$ operations. But if you want to multiply the matrix by n independent vectors, this is actually a matrix-matrix multiplication problem, and we learned from Strassen that we can do this faster than n^3 , i.e., $o(n^2)$ operations on average per vector. So I do not see any obvious obstruction here.

7 The gadget

Let B be the butterfly map $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ acting on $\mathbb{C}^2$. The goal in this section is to explain the construction of positive integers m and W and a “program” that computes the action of $I \otimes B^{\otimes m}$ on $\mathbb{C}^W \otimes (\mathbb{C}^2)^{\otimes m}$ in fewer than Wm slice calls. As mentioned above, there are also extra additions, but I will ignore these, I will focus only on the slice calls, which is the parameter driving the recursive gain.

7.1 Normalisations

First some normalisations. Let

$$H := \frac{1}{\sqrt{2}}B = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

So H (for Hadamard) is just a scaling of B to be unitary. Next let $X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and

$$C := aI + bX = \begin{pmatrix} a & b \\ b & a \end{pmatrix}, \quad a := \frac{1+i}{2}, \quad b := \frac{1-i}{2}.$$

Then $|a| = |b| = 1/\sqrt{2}$, and one checks that

$$\frac{1+i}{\sqrt{2}} \cdot H = SCS \quad \text{where } S := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

So we can think of C as just a version of H where we have changed basis by S on the domain and S^{-1} on the codomain (or maybe I have these backwards), and then thrown in a scaling by the norm-1 factor $(1+i)/\sqrt{2}$. If we can compute $C^{\otimes m}$ then we can easily recover the effect of $B^{\otimes m}$, we just have to pre- and post-multiply every coordinate by the correct powers of i , $1+i$ and $\sqrt{2}$ which is easy. So for the rest of the note I will forget about B and work with C .

Observe that C is unitary. One checks that it is diagonalised by H , and has eigenvalues 1 and i :

$$C = HSH^{-1}.$$

(Of course $H^{-1} = H$ but I prefer to write H^{-1} to make it clear that we are using the same change of basis on both sides.)

(I will just pause here to mention something I find very mysterious. The rest of the development depends crucially on the fact that the eigenvalues of C are 1 and i . But there is nothing “natural” about this; we introduced it artificially when we decided how to change bases to get from B to C . The only justification I see at the moment is that it makes everything work later on. But I have no idea why. (Update: Fable tells me this is forced by the requirements that C commutes with X , which is what makes all the C_v simultaneously diagonal in the $H^{\otimes m}$ basis (see below), and that C is the same as the butterfly up to free rescalings. I have not checked this.))

7.2 The C_v operators

As I said above, C is basically our butterfly expressed in terms of different bases on the domain and codomain. Now we want to graduate to $C^{\otimes m}$, and understand what applying C across a “slice” of $(\mathbb{C}^2)^{\otimes m}$ looks like. The simplest case is where we just apply C on the i -th coordinate. But we will actually need something more general than this.

As a notational matter, we will sometimes write elements of $(\mathbb{C}^2)^{\otimes m}$ as functions $\mathbb{F}_2^m \rightarrow \mathbb{C}$, where $\mathbb{F}_2^m$ are the addresses as before.

Let $v \in \mathbb{F}_2^m$, $v \neq 0$. Think of v as a “direction”. For example if $v = e_i$ then this is just the i -th coordinate direction.

Let X_v denote translation by v , i.e., it sends $f \in (\mathbb{C}^2)^{\otimes m}$ to $X_v f \in (\mathbb{C}^2)^{\otimes m}$ given by $(X_v f)(x) = f(x + v)$. Since $v + v = 0$, you can also think of this operation as swapping x and $x + v$, for every $x \in \mathbb{F}_2^m$. Then define

$$C_v := aI + bX_v.$$

Thus C_v acts on $(\mathbb{C}^2)^{\otimes m}$ by applying C to the 2^{m-1} pairs $(x, x + v)$. In other words it applies C in “direction v ”. For example, C_{e_i} means to apply C on all pairs in the i -th coordinate direction.

In the previous section, a “slice call” was only allowed to be in a coordinate direction, i.e., of the form C_{e_i} . But we can easily reduce a general C_v to this case by inserting appropriate relabellings before and afterwards. So from now on, “slice call” is allowed to be the more general C_v . Also, one easily checks that $(C_v)^{-1} = C_v X_v$, and since X_v is just a free rearrangement, we can also treat $(C_v)^{-1}$ as a slice call.

Although the actual algorithm works with the standard basis for $(\mathbb{C}^2)^{\otimes m}$, there is a bunch of bookkeeping we need to do in a different basis, namely the basis determined by the m -dimensional Hadamard map $H^{\otimes m}$. Our next job is to work out what X_v and C_v look like in this basis.

For X_v we have

$$(H^{\otimes m})^{-1} \cdot X_v \cdot H^{\otimes m} = \text{diag}((-1)^{v \cdot x}),$$

where $\text{diag}(f(x))$ means the diagonal matrix with x -th entry $f(x)$, for each $x \in \mathbb{F}_2^m$. Here $v \cdot x$ is the standard bilinear form on $\mathbb{F}_2^m$, i.e., $v_1 x_1 + \dots + v_m x_m \in \mathbb{F}_2$. Sketch proof: first check it on X_{e_i} by direct calculation, and then observe that translation by v amounts to translating by each e_i for

which $v_i = 1$. (The result also falls directly out of standard results about Fourier transforms on the finite group $(\mathbb{Z}/2\mathbb{Z})^m$.)

For C_v we just use the formula $C_v = aI + bX_v$. This gives

$$(H^{\otimes m})^{-1} \cdot C_v \cdot H^{\otimes m} = \text{diag}(a + b(-1)^{v \cdot x}).$$

If $v \cdot x = 0$ then the RHS is $a + b = 1$; if $v \cdot x = 1$ then we get $a - b = i$. Therefore

$$(H^{\otimes m})^{-1} \cdot C_v \cdot H^{\otimes m} = \text{diag}(i^{[v \cdot x]}), \quad (7.1)$$

where for $z \in \mathbb{F}_2$ we define $[z] \in \mathbb{Z}/4\mathbb{Z}$ by the naive lift, i.e., 0 mod 2 gets mapped to 0 mod 4 and 1 mod 2 gets mapped to 1 mod 4.

Quick numerical example as a sanity check. Take $m = 3$. Then $\mathbb{F}_2^m = \{000, 001, 010, 011, 100, 101, 110, 111\}$ and $(\mathbb{C}^2)^{\otimes m} \cong \mathbb{C}^8$. Take $v = 011$. I won't bother to write out the 8×8 matrix for C_v in the standard basis, but to work it out in the $H^{\otimes m}$ basis we first compute

$$(v \cdot 000, v \cdot 001, \dots) = (0, 1, 1, 0, 0, 1, 1, 0).$$

So the matrix for C_v is just $\text{diag}(1, i, i, 1, 1, i, i, 1)$.

A really nice feature of this setup is that *all* of the C_v 's are diagonal in this basis. So all the C_v commute with each other, and when we multiply them together, the diagonal entries just multiply. For example, since $C^{\otimes m} = C_{e_1} \cdots C_{e_m}$, we get

$$(H^{\otimes m})^{-1} \cdot C^{\otimes m} \cdot H^{\otimes m} = \text{diag}(i^{[e_1 \cdot x]}) \cdots \text{diag}(i^{[e_m \cdot x]}) = \text{diag}(i^{\text{wt}(x)}) \quad (7.2)$$

where $\text{wt}(x)$ denotes the weight of x , i.e., the number of entries equal to 1. In our $m = 3$ example, the matrix for $C^{\otimes m}$ would be $\text{diag}(1, i, i, -1, i, -1, -1, -i)$.

Having introduced the above machinery, we can now see roughly the shape of the algorithm we are aiming for. Suppose you give me an element of $\mathbb{C}^W \otimes (\mathbb{C}^2)^{\otimes m}$, i.e., a list of W arrays. I want to compute $C^{\otimes m}$ of each one. The standard algorithm is to perform $C_{e_1}, \dots, C_{e_m}$ on each array; this amounts to Wm slice calls. But I have other options. I could apply various C_v 's to various arrays in the list, and I know the effect in the $H^{\otimes m}$ basis via (7.1); I just need to keep track of how the exponents of i add up modulo 4. To mix things up a bit, I can also do things like add one array to another, which means that the various C_v are in effect going to be acting on several input arrays “in parallel”. The aim is to somehow get the right pattern of powers of i , i.e., to match those of $C^{\otimes m}$, in each coordinate, at the very end, and with all the “mixups” having been undone.

7.3 Building up nondegenerate subspaces of $\mathbb{F}_2^m$

To keep track of the effect of applying various C_v 's to $(\mathbb{C}^2)^{\otimes m}$, we introduce the following framework.

Let $U \subseteq \mathbb{F}_2^m$ be a subspace. We write $U^\perp \subseteq \mathbb{F}_2^m$ for the complement with respect to the standard dot product, i.e., $U^\perp := \{v \in \mathbb{F}_2^m : u \cdot v = 0 \text{ for all } u \in U\}$. The rank-nullity theorem shows that always $\dim(U^\perp) = m - \dim U$.

Recall that U is called *nondegenerate* with respect to the dot product if $U \cap U^\perp = 0$. This condition is equivalent to saying that $U + U^\perp = \mathbb{F}_2^m$. Note that U is nondegenerate if and only if $U^\perp$ is nondegenerate. We will work with nondegenerate subspaces throughout.

For example, when $m = 3$, the span of 100 is nondegenerate (its complement is the span of 010 and 001). But the span of 110 is degenerate (because $110 \cdot 110 = 0$, so 110 lies in both U and $U^\perp$). More generally, for any m the span of $\{e_1, \dots, e_j\}$ is nondegenerate for any j .

When U is nondegenerate, it makes sense to define a projection operator $P_U : \mathbb{F}_2^m \rightarrow U$. We just write a given $x \in \mathbb{F}_2^m$ uniquely as $y + z$ where $y \in U$ and $z \in U^\perp$, and then put $P_U(x) := y$.

Now for any nondegenerate subspace U , define $q_U : \mathbb{F}_2^m \rightarrow \mathbb{Z}/4\mathbb{Z}$ by

$$q_U(x) := \text{wt}(P_U x) \bmod 4.$$

For example, q_0 is just the zero function, and $q_{\mathbb{F}_2^m}$ is the function that returns $\text{wt}(x) \bmod 4$, i.e., exactly the exponents appearing in (7.2). Then define

$$\Phi_U := H^{\otimes m} \cdot \text{diag}(i^{q_U(x)}) \cdot (H^{\otimes m})^{-1},$$

so $\Phi_0 = I$, and $\Phi_{\mathbb{F}_2^m} = C^{\otimes m}$ by (7.2).

If U and V are nondegenerate and mutually orthogonal, then it is easy to see that $U + V$ is also nondegenerate, and that $P_{U+V} = P_U + P_V$.

We now claim that *weight modulo 4 is additive on orthogonal vectors in $\mathbb{F}_2^m$* . You can prove this by observing that $\text{wt}(x + y) = \text{wt}(x) + \text{wt}(y) - 2|\text{supp } x \cap \text{supp } y|$ (the correction term comes from the coordinates where both $x_i = 1$ and $y_i = 1$), and if $x \cdot y = 0$ then $|\text{supp } x \cap \text{supp } y|$ is even.

It follows that that $q_{U+V} = q_U + q_V$, since

$$q_{U+V}(x) = \text{wt}(P_U x + P_V x) \bmod 4 = q_U(x) + q_V(x),$$

using the fact that $P_U x \perp P_V x$. Finally this means that $\Phi_{U+V} = \Phi_U \Phi_V$.

So now we can start to see how the game works: we begin with $U = 0$, i.e., $\Phi_0 = I$, and we try to add a sequence of orthogonal subspaces to U , until we finally reach $\Phi_{\mathbb{F}_2^m} = C^{\otimes m}$. As we proceed, we can measure our “progress” towards evaluating $C^{\otimes m}$ by the dimension of the intermediate spaces U that we visit along the way.

Let us now investigate what a “single step” looks like. Let U be nondegenerate, and consider a nonzero vector orthogonal to U , say $v \in U^\perp$. We also need to assume that $\langle v \rangle$ is itself nondegenerate, i.e., we need $\text{wt}(v)$ to be *odd*. Under these conditions, $U' := U + \langle v \rangle$ is also nondegenerate, and $\Phi_{U'} = \Phi_U \Phi_{\langle v \rangle}$.

All that remains is to calculate $\Phi_{\langle v \rangle}$. Observe that (since $v \cdot v = 1$)

$$q_{\langle v \rangle}(x) = \text{wt}((v \cdot x)v) \bmod 4 = [v \cdot x] \text{wt}(v) \bmod 4.$$

Since we assumed $\text{wt}(v)$ is odd, there are two cases. If $\text{wt}(v) \equiv 1 \pmod{4}$, then we get $q_{\langle v \rangle}(x) = [v \cdot x] \pmod{4}$, and comparing with (7.1) we find that $\Phi_{\langle v \rangle} = C_v$. If $\text{wt}(v) \equiv 3 \pmod{4}$, then $q_{\langle v \rangle}(x) = -[v \cdot x] \pmod{4}$ and so $\Phi_{\langle v \rangle} = (C_v)^{-1}$. In summary,

$$\Phi_{\langle v \rangle} = \begin{cases} C_v, & \text{wt}(v) \equiv 1 \pmod{4}, \\ (C_v)^{-1}, & \text{wt}(v) \equiv 3 \pmod{4}. \end{cases}$$

So now the picture is: if we start with some nondegenerate U with corresponding matrix Φ_U , and then we add in a sequence of vectors $v_1, \dots, v_k$ of odd weight, each one orthogonal to U and all the previous vectors, then $U + \langle v_1, \dots, v_k \rangle$ is also nondegenerate, and $\Phi_{U + \langle v_1, \dots, v_k \rangle}$ is simply the product $\Phi_U C_{v_1}^{\pm 1} \dots C_{v_k}^{\pm 1}$ with appropriately chosen signs (depending on the weights of the v_i modulo 4). Each of the $C_{v_i}^{\pm 1}$ corresponds to a single slice call.

Fable points out that my earlier $m = 3$ example cannot illustrate a nontrivial vector sequence, because the only orthonormal basis for $\mathbb{F}_2^3$ is the standard one. It suggests instead an $m = 4$ example with orthonormal basis 1110, 1101, 1011, 0111. Then one has $C^{\otimes 4} = C_{1110}^{-1} C_{1101}^{-1} C_{1011}^{-1} C_{0111}^{-1}$.

7.4 Programs, registers and subspaces

In the following sections I will discuss some “programs”, which are a sequence of instructions acting on an array of W “registers”, say $(r_1, \dots, r_W)$. Each register holds an element of $(\mathbb{C}^2)^{\otimes m}$. Here are the allowed operations:

- We can apply C_v or C_v^{-1} to a register, replacing the existing value. Each of these operations costs one slice call (plus data rearrangements, which I remind the reader we are completely ignoring).
- We can add a scalar multiple of one register to another register. For example I could replace r_2 by $r_2 + \frac{1}{2}r_1$. As (vaguely) explained above, I am ignoring the cost of these operations.

Each register has an associated (nondegenerate) subspace U_i . Initially the subspaces $U_1, \dots, U_W$ are set to some prescribed values, and then they get updated during the execution of the program. The algorithm does not actually store or manipulate the subspaces; they are just there for accounting purposes. Here is how to think about it: each register r_i has an “underlying” value, say $f_i \in (\mathbb{C}^2)^{\otimes m}$. What is “physically” stored in r_i (i.e., the value actually manipulated by the algorithm) is the partially transformed value $\Phi_{U_i} f_i$. To make this work, we have to follow some rules:

- Suppose that we want to enlarge U_i to $U'_i := U_i + \langle v \rangle$. For this to be allowed, we require that v has odd weight and that $v \perp U_i$. According to the previous section, $\Phi_{U'_i}$ is $\Phi_{U_i} C_v$ if $\text{wt}(v) \equiv 1 \pmod{4}$, and $\Phi_{U_i} C_v^{-1}$ if $\text{wt}(v) \equiv 3 \pmod{4}$. So in the first case we apply C_v to r_i , in the second case we apply C_v^{-1} . In either case, the value in the register is still $\Phi_{U_i} f_i$ for the new value of U_i . The key thing is that *the underlying value f_i does not change*; we have changed both the physical representation and the associated U_i in sync.
- Suppose that we want to shrink U_i to $U'_i := U_i \cap \langle v \rangle^\perp$. (This is equivalent to enlarging $U_i^\perp$ to $U_i^\perp + \langle v \rangle$.) For this to be allowed, we require that v has odd weight and that $v \in U_i$ ($= U_i^{\perp\perp}$). Similarly to the previous bullet, if $\text{wt}(v) \equiv 1 \pmod{4}$ then we apply C_v^{-1} to r_i , if $\text{wt}(v) \equiv 3 \pmod{4}$ we apply C_v . Again the underlying value f_i does not change.

- We are only allowed to use the “add scalar multiple” instruction on registers whose U_i are currently *equal*. In other words, if registers r_i and r_j are both currently attached to the same subspace U , with physically stored values $\Phi_U f_i$ and $\Phi_U f_j$, then we are allowed to perform an operation such as replacing r_j with $r_j + \frac{1}{2}r_i$. In this example we have effectively replaced f_j with $f_j + \frac{1}{2}f_i$. Everything stays consistent because $\Phi_U(f_j + \frac{1}{2}f_i) = \Phi_U f_j + \frac{1}{2}\Phi_U f_i$.

As an example, suppose that a register begins life with $U_i = 0$. Since $\Phi_0 = I$, the physically stored value is the same as the underlying value, say $f_i \in (\mathbb{C}^2)^{\otimes m}$. If we apply in sequence $C_{e_1}, \dots, C_{e_m}$ to the register, then U_i gets updated to $\mathbb{F}_2^m$, so that $\Phi_{U_i} = C^{\otimes m}$. At this point, the underlying value is still f_i , but the value physically in the register is $\Phi_{U_i} f_i = C^{\otimes m} f_i$. So we have succeeded in computing $C^{\otimes m} f_i$.

In the program described in the next section, some registers will begin life with $U_i \neq 0$. For example, they might begin with $U_i = \langle v \rangle$ for some odd-weight vector v . In this case, the initial underlying value is *different* to the initial physical value. This does not cause any problems, and in fact the algorithm does not even need to know the initial underlying values. As we will see, the possibility of starting with $U_i \neq 0$ is central to making the whole gadget work.

7.5 The baby program

In this section we are going to describe a baby version of the main program. You should think of this as motivation/warmup, rather than as literally a subroutine of the main program (which is presented in the next section). Instead of $(\mathbb{C}^2)^{\otimes m}$ we work here with $(\mathbb{C}^2)^{\otimes h}$ for some integer $h \geq 8$, reserving “ m ” for the main program.

Let $[h]$ denote the set $\{1, \dots, h\}$.

Let $\mathcal{T}$ denote the set of subsets of $[h]$ of cardinality 3. Thus $|\mathcal{T}| = \binom{h}{3}$.

Let $\mathcal{P}$ denote the set of ordered pairs $(S, T) \in \mathcal{T} \times \mathcal{T}$ such that $|S \cap T|$ is even, i.e., such that $|S \cap T|$ is either 0 or 2. For each $S \in \mathcal{T}$ there are $3(h-3)$ choices of T with $|S \cap T| = 2$, and $\binom{h-3}{3}$ choices of T with $|S \cap T| = 0$, so altogether $|\mathcal{P}| = \binom{h}{3}(\binom{h-3}{3} + 3(h-3))$.

Initialisation

There are many registers, playing various different roles. Instead of calling them $r_1, \dots, r_W$, we will give them (slightly) more meaningful names. Here they are:

- For each $T \in \mathcal{T}$, a “data” register x_T .
- For each $S \in \mathcal{T}$, a “data” register y_S .
- For each $(S, T) \in \mathcal{P}$, a “side” register $A_{S,T}$.
- For each $i \in [h]$, a “central” register c_i .
- One last “central” register c_* .

I do not fully understand the meaning of these register names. The total number of registers is

$$W = 2|\mathcal{T}| + |\mathcal{P}| + h + 1.$$

It is perhaps worth noting that if you count the number of registers in each category, then asymptotically speaking, the side registers dominate (sextic in h), followed by the data registers (cubic in h), and then only a handful of central registers (linear in h).

The initial subspaces are all declared to be zero, except for the x_T registers, which are initialised to $\langle v_T \rangle$. Here $v_T \in \mathbb{F}_2^h$ means the vector with 1's at the indices lying in T , and zero elsewhere. Note that $\text{wt}(v_T) = 3$ which is odd.

I will use a tilde, for example $\tilde{x}_T$, to indicate the *initial underlying value* of a register.

At the beginning of the program, here is the state of play:

register	current U	underlying value
x_T	$\langle v_T \rangle$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Keep in mind that at this point, the physical values match the underlying values, except for x_T , whose underlying value is the image under $\Phi_{\langle v_T \rangle}^{-1}$ of the physical value (i.e., the value actually supplied to the program as input).

The program is now given as 8 steps, labelled step 0 to step 7. Each step consists of a *transform phase*, where we use slice calls to modify the subspaces attached to various registers, and then a *mixup phase*, where we apply linear operations to various registers sitting at the same subspace. Then we give a cost field indicating the number of slice calls (we ignore the cost of mixups and other linear operations, and any data rearrangements), and then a table showing the final state of all registers after these operations.

Step 0

Transforms: none.

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then subtract $\frac{1}{2}A_{S,T}$ from y_S , and if $|S \cap T| = 2$ then add $\frac{1}{2}A_{S,T}$ to y_S .

Cost: zero.

Final state:

register	current U	underlying value
x_T	$\langle v_T \rangle$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Step 1

Transforms: none.

Mixups: for each $S \in \mathcal{T}$, add $\frac{1}{2}c_* - \frac{1}{2}\sum_{i \in S} c_i$ to y_S .

Cost: zero.

Final state:

register	current U	underlying value
x_T	$\langle v_T \rangle$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Step 2

Transforms: for each $(S, T) \in \mathcal{P}$, apply $\Phi_{\langle v_T \rangle}$ to $A_{S,T}$. This brings its subspace up to $\langle v_T \rangle$ (to match x_T). This costs one slice call per pair (S, T) .

Mixups: for each $(S, T) \in \mathcal{P}$, add x_T to $A_{S,T}$. (This is legal, as the subspaces now match.)

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
x_T	$\langle v_T \rangle$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$\langle v_T \rangle$	$\tilde{A}_{S,T} + \tilde{x}_T$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Step 3

Transforms: for each $T \in \mathcal{T}$, we want to apply $\Phi_{\langle v_T \rangle^\perp}$ to x_T , to bring its subspace up to $\langle v_T \rangle + \langle v_T \rangle^\perp = \mathbb{F}_2^h$. To achieve this, we need to break up $\Phi_{\langle v_T \rangle^\perp}$ into a sequence of one-dimensional steps, which means we need to find an orthonormal basis for $\langle v_T \rangle^\perp$. Here is an explicit orthonormal basis. Let $T = \{p, q, r\}$, and choose any $s \notin T$ (possible since $h \geq 4$). Then take the vectors

$$e_p + e_q + e_s, \quad e_p + e_r + e_s, \quad e_q + e_r + e_s, \quad e_j \text{ for } j \in [h] \setminus \{p, q, r, s\}.$$

The cost is $h - 1$ slice calls per T . (I want to emphasise here that the reason all of this works, if you trace back through the logic, is that weight is additive mod 4 on orthogonal vectors, and the diagonal entries of the C_v are all powers of i which has order 4. Maybe this is well known to some people, but I have never seen it before and I think it's very beautiful.)

We also want to move c_i and c_* up to $\mathbb{F}_2^h$. For this we need an orthonormal basis for $\mathbb{F}_2^h$; of course we can just use the standard basis. The cost is h slice calls for each c_i, c_* .

Mixups: for each $T \in \mathcal{T}$, add x_T to every c_i for which $i \in T$, and also to c_* .

Cost: $|\mathcal{T}|(h-1) + (h+1)h$.

Final state:

register	current U	underlying value
x_T	$\mathbb{F}_2^h$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$\langle v_T \rangle$	$\tilde{A}_{S,T} + \tilde{x}_T$
c_i	$\mathbb{F}_2^h$	$\tilde{c}_i + \sum_{T: i \in T} \tilde{x}_T$
c_*	$\mathbb{F}_2^h$	$\tilde{c}_* + \sum_T \tilde{x}_T$

Step 4

Transforms: we now want to move c_i, c_* back down to $U = 0$, to match with y_S . This appears to be *undoing* some of the progress we already made, but the point is that the mixed-in values of $\tilde{x}_T$ come along for the ride. To achieve this, we need to perform $\Phi_{\mathbb{F}_2^h}$ in reverse, i.e., we do $\Phi_{\langle e_j \rangle}^{-1}$ for each $j \in [h]$. The cost is h slice calls per c_i, c_* .

Mixups: for each $S \in \mathcal{T}$, subtract $\frac{1}{2}c_* - \frac{1}{2} \sum_{i \in S} c_i$ from y_S . (This is the inverse of step 1.)

This has the following effects on y_S :

- It cancels out the $\tilde{c}_i$ and $\tilde{c}_*$ terms from y_S .
- For each $i \in S$, the c_i term contributes $\frac{1}{2}\tilde{x}_T$ for each T containing i . The total contribution from all of these i is $\frac{1}{2} \sum_T |S \cap T| \tilde{x}_T$.
- The c_* term contributes $-\frac{1}{2} \sum_T \tilde{x}_T$.

Thus, after this step, the underlying value of y_S is given by

$$\tilde{y}_S - \frac{1}{2} \sum_{T: |S \cap T|=0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: |S \cap T|=2} \tilde{A}_{S,T} + \frac{1}{2} \sum_T (|S \cap T| - 1) \tilde{x}_T.$$

We can simplify the last sum a bit. The only case where $|S \cap T| = 3$ is when $T = S$; this term gives $\frac{1}{2}(3-1)\tilde{x}_S$ which is simply $\tilde{x}_S$. Also, the terms with $|S \cap T| = 1$ all vanish. So we are left with

$$\tilde{y}_S + \tilde{x}_S - \frac{1}{2} \sum_{T: |S \cap T|=0} (\tilde{A}_{S,T} + \tilde{x}_T) + \frac{1}{2} \sum_{T: |S \cap T|=2} (\tilde{A}_{S,T} + \tilde{x}_T).$$

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_T	$\mathbb{F}_2^h$	$\tilde{x}_T$
y_S	0	$\tilde{y}_S + \tilde{x}_S - \frac{1}{2} \sum_{T: S \cap T =0} (\tilde{A}_{S,T} + \tilde{x}_T)$ $+ \frac{1}{2} \sum_{T: S \cap T =2} (\tilde{A}_{S,T} + \tilde{x}_T)$
$A_{S,T}$	$\langle v_T \rangle$	$\tilde{A}_{S,T} + \tilde{x}_T$
c_i	0	$\tilde{c}_i + \sum_{T: i \in T} \tilde{x}_T$
c_*	0	$\tilde{c}_* + \sum_{T \in \mathcal{T}} \tilde{x}_T$

Step 5

Transforms: for each $S \in \mathcal{T}$ we bring y_S up to $\langle v_S \rangle^\perp$. We use the same orthonormal basis as in step 3. The cost is $h - 1$ slice calls per S .

Next, for each $(S, T) \in \mathcal{P}$ we want to bring $A_{S,T}$ up to $\langle v_S \rangle^\perp$, to match y_S . The crucial fact here is that $v_T \cdot v_S = 0$ (since $|T \cap S| = 0$ or 2), so $v_T \in \langle v_S \rangle^\perp$. (This seems to me to be a really critical point in the argument; it feels like everything else in the gadget might have been designed around this one tiny nugget.) So $\langle v_S \rangle^\perp = \langle v_T \rangle + \langle v_S, v_T \rangle^\perp$ (orthogonal sum), and what we need is an orthonormal basis for $\langle v_S, v_T \rangle^\perp$. We can construct one as follows:

- Suppose that $|S \cap T| = 2$, say $S = \{p, q, r\}$ and $T = \{p, q, s\}$ for $r \neq s$. Then we can use

$$e_p + e_r + e_s, \quad e_q + e_r + e_s, \quad e_j \text{ for } j \notin \{p, q, r, s\}.$$

This is $2 + (h - 4) = h - 2$ vectors altogether.

- Suppose that $|S \cap T| = 0$, say $S = \{p, q, r\}$ and $T = \{p', q', r'\}$, all six indices distinct. Choose distinct $s, s' \notin S \cup T$ (possible since $h \geq 8$). Then we can use

$$\begin{array}{lll} e_p + e_q + e_s, & e_p + e_r + e_s, & e_q + e_r + e_s, \\ e_{p'} + e_{q'} + e_{s'}, & e_{p'} + e_{r'} + e_{s'}, & e_{q'} + e_{r'} + e_{s'}, \end{array}$$

together with e_j for $j \notin \{p, q, r, s, p', q', r', s'\}$. This is $6 + (h - 8) = h - 2$ vectors altogether.

The cost is $h - 2$ slice calls per (S, T) .

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then add $\frac{1}{2}A_{S,T}$ to y_S , and if $|S \cap T| = 2$ then subtract $\frac{1}{2}A_{S,T}$ from y_S . This is the inverse of step 0, and has the effect of removing the $\tilde{A}_{S,T} + \tilde{x}_T$ terms from y_S .

Cost: $|\mathcal{T}|(h - 1) + |\mathcal{P}|(h - 2)$.

Final state:

register	current U	underlying value
x_T	$\mathbb{F}_2^h$	$\tilde{x}_T$
y_S	$\langle v_S \rangle^\perp$	$\tilde{y}_S + \tilde{x}_S$
$A_{S,T}$	$\langle v_S \rangle^\perp$	$\tilde{A}_{S,T} + \tilde{x}_T$
c_i	0	$\tilde{c}_i + \sum_{T: i \in T} \tilde{x}_T$
c_*	0	$\tilde{c}_* + \sum_T \tilde{x}_T$

Step 6

Transforms: we bring each c_i, c_* back up to $\mathbb{F}_2^h$, at a cost of h slice calls for each register.

Mixups: for each $T \in \mathcal{T}$, subtract x_T from every c_i for which $i \in T$, and also from c_* . This is the inverse of step 3.

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_T	$\mathbb{F}_2^h$	$\tilde{x}_T$
y_S	$\langle v_S \rangle^\perp$	$\tilde{y}_S + \tilde{x}_S$
$A_{S,T}$	$\langle v_S \rangle^\perp$	$\tilde{A}_{S,T} + \tilde{x}_T$
c_i	$\mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h$	$\tilde{c}_*$

Step 7

Transforms: for each $(S, T) \in \mathcal{P}$, bring $A_{S,T}$ up to $\mathbb{F}_2^h$ by a single application of $\Phi_{\langle v_S \rangle}$. The cost is one slice call per (S, T) .

Mixups: for each $(S, T) \in \mathcal{P}$, subtract x_T from $A_{S,T}$. This is the inverse of step 2.

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
x_T	$\mathbb{F}_2^h$	$\tilde{x}_T$
y_S	$\langle v_S \rangle^\perp$	$\tilde{y}_S + \tilde{x}_S$
$A_{S,T}$	$\mathbb{F}_2^h$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h$	$\tilde{c}_*$

What did the baby program achieve?

At the end of the baby program, the registers $A_{S,T}$, c_i and c_* contain their original underlying values, and the attached subspaces have moved from $U = 0$ to $U = \mathbb{F}_2^h$. Thus we have succeeded in applying $C^{\otimes h}$ to each of them.

As for the data registers, we have computed the map $(x, y) \mapsto (x, x + y)$ on the underlying values. But we have not fully transformed the data, due to the initial subspace on x_T having one extra dimension, and the final subspace on y_S being one dimension short.

Notice however that the $\tilde{x}_S$ part of the final y_S register *has* been fully transformed. Its initial physical value was $\Phi_{\langle v_S \rangle}(\tilde{x}_S)$, and its final physical value is $\Phi_{\langle v_S \rangle^\perp}(\tilde{x}_S)$, so it has undergone an overall transformation by

$$\Phi_{\langle v_S \rangle^\perp} \Phi_{\langle v_S \rangle}^{-1} = \Phi_{\langle v_S \rangle^\perp + \langle v_S \rangle} \Phi_{\langle v_S \rangle}^{-2} = \Phi_{\mathbb{F}_2^h} \Phi_{\langle v_S \rangle}^{-2}.$$

The $\Phi_{\mathbb{F}_2^h}$ factor is just the desired transformation $C^{\otimes h}$. Since $\text{wt}(v_S) = 3 \pmod{4}$ we have $\Phi_{\langle v_S \rangle} = C_{v_S}^{-1}$, so the $\Phi_{\langle v_S \rangle}^{-2}$ factor becomes $C_{v_S}^2 = X_{v_S}$, a harmless data rearrangement. This observation is crucial for understanding how the main program in the next section can possibly work out.

The total number of slice calls was

$$\begin{aligned} |\mathcal{P}| + |\mathcal{T}|(h-1) + (h+1)h + (h+1)h + |\mathcal{T}|(h-1) + |\mathcal{P}|(h-2) + (h+1)h + |\mathcal{P}| \\ = |\mathcal{P}|h + 2|\mathcal{T}|(h-1) + 3(h^2 + h). \end{aligned}$$

Recalling that $W = 2|\mathcal{T}| + |\mathcal{P}| + (h+1)$ (the total number of registers), we can rewrite the slice call count as

$$Wh - 2|\mathcal{T}| + 2(h^2 + h).$$

Here is how to think about this formula: the Wh term is the “expected” number of slice calls (h per register), the $2|\mathcal{T}|$ term is the savings coming from the fact that we didn’t quite fully transform the data registers (we only did $h-1$ per data register), and the $2(h^2 + h)$ term is the loss coming from the extra work (“undoing progress”) on the central registers ($2h$ per register). Since $|\mathcal{T}|$ is cubic in h , for large enough h the savings term beats the loss term, and the total becomes strictly less than the expected Wh .

During Fable’s proofreading of this section, it proactively decided to implement the whole baby program as a sanity check. It confirmed the formulas in the state tables given above for some random inputs, and verified the above formula for the number of slice calls, for a few small values of h . I have not checked the code carefully. Notice that when $h = 8$ the size of the vectors is $2^8 = 256$, still quite manageable for actual computations. For the main program in the next section, such experiments will become infeasible due to combinatorial explosion.

7.6 The main program

If you already had a headache at the end of the last section, I must warn you that it’s about to get worse.

Let h be as above. We are going to construct a new program. The registers are now going to be elements of $(\mathbb{C}^2)^{\otimes m}$ where $m := h^3$. The address space becomes $\mathbb{F}_2^m = (\mathbb{F}_2^h)^{\otimes 3}$. You can think of an address as being an $h \times h \times h$ array of bits.

(The combinatorial explosion is already clear. Even for $h = 8$ we have $m = 512$ so the vectors have length 2^{512} , well beyond the size of any physically realisable computer in the observable universe.)

The symbol $\mathcal{T}$ has the same meaning as in the previous section, i.e., size-3 subsets of $[h] := \{1, \dots, h\}$. Thus $|\mathcal{T}| = \binom{h}{3}$. Similarly $\mathcal{P}$ has the same meaning as before.

The main program uses the following registers:

- Data registers x_{a_1, a_2, a_3} and y_{a_1, a_2, a_3} for $(a_1, a_2, a_3) \in \mathcal{T}^3$. So the number of data registers is $2|\mathcal{T}|^3$.

It might help to think of x as a big $|\mathcal{T}| \times |\mathcal{T}| \times |\mathcal{T}|$ cube, where each entry is a register holding an array in $(\mathbb{C}^2)^{\otimes m}$. And sitting next to it is a similar cube y .

- As we will see below, the main program invokes some variant of the baby program $3|\mathcal{T}|^2$ times. Each invocation needs its own set of scratch registers $A_{S,T}$, c_i and c_* . So altogether we need $3|\mathcal{T}|^2(|\mathcal{P}| + h + 1)$ scratch registers.

The total number of registers is thus

$$W := 2|\mathcal{T}|^3 + 3|\mathcal{T}|^2(|\mathcal{P}| + h + 1).$$

Before we go through the steps of the program, we will set up some further notation and recall a few useful facts about tensor products.

If we have subspaces $U \subseteq \mathbb{F}_2^a$ and $V \subseteq \mathbb{F}_2^b$, then the subspace $U \otimes V \subseteq \mathbb{F}_2^a \otimes \mathbb{F}_2^b = \mathbb{F}_2^{ab}$ is the span of the vectors $u \otimes v$ for $u \in U$, $v \in V$. A frequently used special case is the fact that $\langle u \rangle \otimes \langle v \rangle = \langle u \otimes v \rangle$.

We have

$$(u \otimes v) \cdot (u' \otimes v') = (u \cdot u')(v \cdot v'), \quad u, u' \in \mathbb{F}_2^a, \quad v, v' \in \mathbb{F}_2^b.$$

In particular if $\{u_i\}$ and $\{v_j\}$ are orthonormal bases for U and V , then $\{u_i \otimes v_j\}$ is an orthonormal basis for $U \otimes V$.

In the rest of this section, whenever we write $U \oplus U'$ we always mean *orthogonal* direct sum. Also, when we write $U^\perp$, the ambient space should always be clear from context.

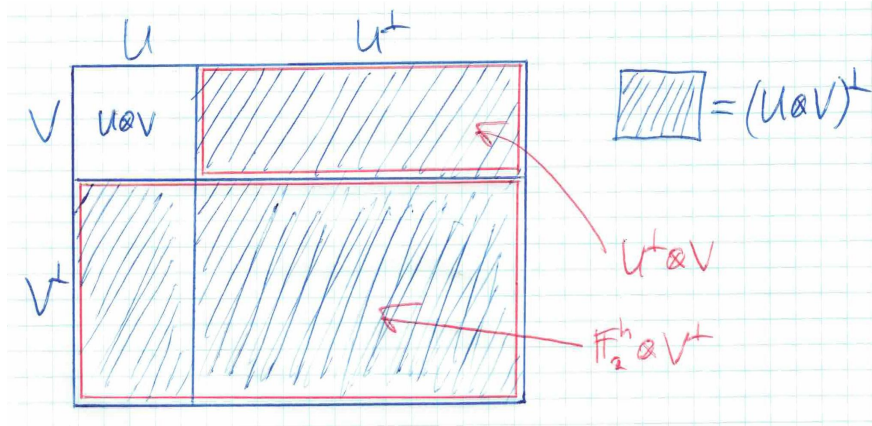
If U, V are nondegenerate subspaces of $\mathbb{F}_2^h$, we have the useful formula

$$(U \otimes V)^\perp = (\mathbb{F}_2^h \otimes V^\perp) \oplus (U^\perp \otimes V). \quad (7.3)$$

This follows from the calculation

$$(\mathbb{F}_2^h \otimes V^\perp) \oplus (U^\perp \otimes V) \oplus (U \otimes V) = (\mathbb{F}_2^h \otimes V^\perp) \oplus (\mathbb{F}_2^h \otimes V) = \mathbb{F}_2^h \otimes \mathbb{F}_2^h.$$

The formula (7.3) also holds with the tensor factors switched around; we will sometimes use it this way without further comment. Personally, when I am verifying formulas involving (7.3), instead of doing things algebraically I find it much easier to grab a piece of paper and draw diagrams like this:



We will only need the following types of orthonormal bases:

- The standard basis $e_1, \dots, e_h$ for $\mathbb{F}_2^h$.
- For $T \in \mathcal{T}$, let $\mathcal{B}_T$ denote the orthonormal basis for $\langle v_T \rangle^\perp \subseteq \mathbb{F}_2^h$ from step 3 of the baby program. It has $h - 1$ vectors. (This basis is not uniquely determined, it depends on the choice of $s \notin T$. Any choice will do. Similar remarks apply for the bases defined below.)
- For $(S, T) \in \mathcal{P}$, let $\mathcal{B}_{S,T}$ be the orthonormal basis for $\langle v_S, v_T \rangle^\perp \subseteq \mathbb{F}_2^h$ from step 5 of the baby program. It has $h - 2$ vectors. (We continue to assume that $h \geq 8$.)
- For any $S, T \in \mathcal{T}$, not necessarily coming from $\mathcal{P}$, we will need an orthonormal basis for $\langle v_S \otimes v_T \rangle^\perp \subseteq \mathbb{F}_2^h \otimes \mathbb{F}_2^h$. We will denote it by $\mathcal{B}_{S \otimes T}$. It may be constructed as follows. From (7.3) we have

$$(\langle v_S \rangle \otimes \langle v_T \rangle)^\perp = (\langle v_S \rangle^\perp \otimes \mathbb{F}_2^h) \oplus (\langle v_S \rangle \otimes \langle v_T \rangle^\perp).$$

The first component has orthonormal basis $\{b \otimes e_k : b \in \mathcal{B}_S, k \in [h]\}$. The second component has orthonormal basis $\{v_S \otimes b : b \in \mathcal{B}_T\}$. We take $\mathcal{B}_{S \otimes T}$ to be the union of these two bases. It has $h^2 - 1$ vectors. (Check: $|\mathcal{B}_S| |h| + |\mathcal{B}_T| = (h - 1)h + (h - 1) = h^2 - 1$.)

The subspaces for the main program are initialised as follows: the register x_{a_1, a_2, a_3} starts at $U = \langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$, and all other registers start at $U = 0$. The symbols $\tilde{x}_{a_1, a_2, a_3}$ and $\tilde{y}_{a_1, a_2, a_3}$ refer to the *underlying* values of these registers at the beginning of the program.

The main program has three “stages”, which we now describe in turn.

Stage 1 — overview

For each pair $(a_2, a_3) \in \mathcal{T} \times \mathcal{T}$ we perform 9 steps.

The first 8 steps are basically the same as the baby program, acting on the data registers x_{T, a_2, a_3} and y_{S, a_2, a_3} , where $T, S \in \mathcal{T}$. So you can think of stage 1 as operating on one-dimensional “slices” of the $|\mathcal{T}| \times |\mathcal{T}| \times |\mathcal{T}|$ cubes x and y , along the first dimension. We use a new set of scratch registers $(A_{S,T}, c_i, c_*)$ for each pair (a_2, a_3) . Strictly speaking we should add subscripts a_2, a_3 to the scratch registers, but in the interests of readability we will omit them.

Actually, it doesn’t quite make sense to apply the baby program in its original form, because it was stated for arrays in $(\mathbb{C}^2)^{\otimes h}$ (with addresses in $\mathbb{F}_2^h$), whereas the registers here take values in $(\mathbb{C}^2)^{\otimes m}$ (with addresses in $\mathbb{F}_2^m = (\mathbb{F}_2^h)^{\otimes 3}$). Instead, we have to modify the slice calls: whenever we would have applied $\Phi_{\langle v \rangle}$ to some register, for some $v \in \mathbb{F}_2^h$ of odd weight, we replace this by $\Phi_{\langle v \otimes v_{a_2} \otimes v_{a_3} \rangle}$ (and similarly for $\Phi_{\langle v \rangle}^{-1}$). In other words, every subspace $U \subseteq \mathbb{F}_2^h$ in the baby program gets replaced by $U \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle \subseteq \mathbb{F}_2^m$, and the latter has the same dimension as U . The replacement $v \mapsto v \otimes v_{a_2} \otimes v_{a_3}$ preserves all orthonormality relations, so all arguments about legality of slice call sequences still hold.

(Incidentally, in the baby program, recall that we only ever used v with weight 1 or 3. Since $\text{wt}(v_{a_2}) = \text{wt}(v_{a_3}) = 3$, we see that $\text{wt}(v \otimes v_{a_2} \otimes v_{a_3}) = 9 \text{wt}(v)$. Thus the modified version only ever performs slice calls in directions $v \otimes v_{a_2} \otimes v_{a_3}$ with weight 9 or 27.)

Finally, in step 8 we apply some extra $\Phi_{\langle v \rangle}$ ’s to the scratch registers to bump their subspaces up from $\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$ to the full $\mathbb{F}_2^m$.

We now briefly summarise these steps. All 9 steps are performed once for each pair (a_2, a_3) . The first 8 steps are basically copied from the baby program with all spaces/vectors tensored by $v_{a_2} \otimes v_{a_3}$ appropriately, so we won't bother writing out the transform and mixup phases. The costs are literally the same as in the baby program.

Stage 1, initialisation

Here is the initial state:

register	current U	underlying value
x_{T,a_2,a_3}	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3}$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 1, step 0

Cost: zero.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 1, step 1

Cost: zero.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 1, step 2

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$ $+ \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 1, step 3

Cost: $|\mathcal{T}|(h-1) + (h+1)h$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$ $+ \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3}$
c_i	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i + \sum_{T:i \in T} \tilde{x}_{T,a_2,a_3}$
c_*	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_* + \sum_T \tilde{x}_{T,a_2,a_3}$

Stage 1, step 4

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	0	$\tilde{y}_{S,a_2,a_3} + \tilde{x}_{S,a_2,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} (\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3})$ $+ \frac{1}{2} \sum_{T: S \cap T =2} (\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3})$
$A_{S,T}$	$\langle v_T \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3}$
c_i	0	$\tilde{c}_i + \sum_{T:i \in T} \tilde{x}_{T,a_2,a_3}$
c_*	0	$\tilde{c}_* + \sum_T \tilde{x}_{T,a_2,a_3}$

Stage 1, step 5

Cost: $|\mathcal{T}|(h-1) + |\mathcal{P}|(h-2)$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{S,a_2,a_3} + \tilde{x}_{S,a_2,a_3}$
$A_{S,T}$	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3}$
c_i	0	$\tilde{c}_i + \sum_{T:i \in T} \tilde{x}_{T,a_2,a_3}$
c_*	0	$\tilde{c}_* + \sum_T \tilde{x}_{T,a_2,a_3}$

Stage 1, step 6

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{S,a_2,a_3} + \tilde{x}_{S,a_2,a_3}$
$A_{S,T}$	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{x}_{T,a_2,a_3}$
c_i	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 1, step 7

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{S,a_2,a_3} + \tilde{x}_{S,a_2,a_3}$
$A_{S,T}$	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 1, step 8

(This step does not come from the baby program.)

Transforms: we want to move each scratch register to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h = \mathbb{F}_2^m$, so we need to add the space $\mathbb{F}_2^h \otimes (\langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle)^\perp$. We may use the orthonormal basis $\{e_j \otimes b \otimes c : j \in [h], b \otimes c \in \mathcal{B}_{a_2 \otimes a_3}\}$. This costs $h(h^2 - 1)$ slice calls per scratch register.

Mixups: none.

Cost: $(|\mathcal{P}| + h + 1) \cdot h(h^2 - 1)$.

Final state:

register	current U	underlying value
x_{T,a_2,a_3}	$\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{T,a_2,a_3}$
y_{S,a_2,a_3}	$\langle v_S \rangle^\perp \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{S,a_2,a_3} + \tilde{x}_{S,a_2,a_3}$
$A_{S,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

At this point we are done with this set of scratch registers; we put them aside and bring in a new set for the next pair (a_2, a_3) (starting again of course at $U = 0$ for each register).

Stage 1 — conclusion

As for the data registers, we have computed the map $(x, y) \mapsto (x, x + y)$ on the underlying values. The first component of the subspace for y is one dimension short of the full $\mathbb{F}_2^h \otimes \langle v_{a_2} \rangle \otimes \langle v_{a_3} \rangle$ (the current “target”), so the $\tilde{y}$ part has not been fully transformed. But, as explained in the baby program, the $\tilde{x}$ part *has* been fully transformed (i.e., up to the current target).

The scratch registers were transformed completely, i.e., we have applied $C^{\otimes m}$ to each of them.

The total number of slice calls for Stage 1 is just the baby program count, plus the cost of step 8, all multiplied by the number of pairs (a_2, a_3) :

$$\begin{aligned}
|\mathcal{T}|^2 (|\mathcal{P}|h + 2|\mathcal{T}|(h - 1) + 3(h^2 + h) + (|\mathcal{P}| + h + 1) \cdot h(h^2 - 1)) \\
= 2|\mathcal{T}|^3(h - 1) + |\mathcal{T}|^2|\mathcal{P}|h^3 + |\mathcal{T}|^2(h^2 + 2)(h^2 + h).
\end{aligned}$$

Stage 2 — overview

The aim of this stage is to compute the function $(x, y) \mapsto (x - y, y)$ on each pair of data registers. This maps the current values $(\tilde{x}_a, \tilde{y}_a + \tilde{x}_a)$ (see last state table) to

$$(\tilde{x}_a - (\tilde{y}_a + \tilde{x}_a), \tilde{y}_a + \tilde{x}_a) = (-\tilde{y}_a, \tilde{y}_a + \tilde{x}_a).$$

The strategy is essentially to run the steps of the baby program in reverse order, with each mixup operation inverted, and with the roles of x and y switched. However, we cannot run the $\Phi_{\langle v \rangle}$ ’s backwards, because we are still trying to make progress in the forward direction on the subspaces. As in the baby program, the main requirement is to ensure that the subspaces agree whenever we want to perform a linear operation on two registers.

For each pair $(a_1, a_3) \in \mathcal{T} \times \mathcal{T}$, we perform the following steps. (The modified baby program is going to act on the second coordinate.)

Stage 2, initialisation

The data registers in the following table are copied from the final state of Stage 1. For each (a_1, a_3) we use a new set of scratch registers, initialised to $U = 0$.

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \langle v_S \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{a_1,S,a_3}$
y_{a_1,T,a_3}	$\langle v_{a_1} \rangle^\perp \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	0	$\tilde{A}_{S,T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 2, step 0 (inverse of baby program step 7)

Transforms: we bring $A_{S,T}$ up to $\langle v_{a_1} \rangle^\perp \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$ (to match y_{a_1,T,a_3}) via the orthonormal basis $\{b \otimes v_T \otimes v_{a_3} : b \in \mathcal{B}_{a_1}\}$. This needs $h-1$ slice calls for each (S, T) .

Mixups: for each $(S, T) \in \mathcal{P}$, we add y_{a_1,T,a_3} to $A_{S,T}$.

Cost: $|\mathcal{P}|(h-1)$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \langle v_S \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{a_1,S,a_3}$
y_{a_1,T,a_3}	$\langle v_{a_1} \rangle^\perp \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$\langle v_{a_1} \rangle^\perp \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 2, step 1 (inverse of baby program step 6)

Transforms: we want to bring y_{a_1,T,a_3} and c_i, c_* up to $\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$.

For y_{a_1,T,a_3} we use the orthonormal basis $\{b \otimes c \otimes v_{a_3} : b \in \mathcal{B}_{a_1}, c \in \mathcal{B}_T\}$. This costs $(h-1)^2$ slice calls for each $T \in \mathcal{T}$.

For c_i, c_* we use the orthonormal basis $\{b \otimes e_j \otimes v_{a_3} : b \in \mathcal{B}_{a_1}, j \in [h]\}$. This costs $(h-1)h$ slice calls for each c_i, c_* .

Mixups: for each $T \in \mathcal{T}$, we add y_{a_1,T,a_3} to every c_i for which $i \in T$, and to c_* .

Cost: $|\mathcal{T}|(h-1)^2 + (h-1)h(h+1)$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \langle v_S \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{a_1,S,a_3}$
y_{a_1,T,a_3}	$\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$\langle v_{a_1} \rangle^\perp \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
c_i	$\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i + \sum_{T:i \in T} (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$
c_*	$\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_* + \sum_T (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$

Stage 2, step 2 (inverse of baby program step 5)

Transforms: we want to bring both x_{a_1,S,a_3} and $A_{S,T}$ up to the subspace

$$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \otimes \langle v_{a_3} \rangle.$$

By (7.3) this is the same as

$$(\mathbb{F}_2^h \otimes \langle v_S \rangle \otimes \langle v_{a_3} \rangle) \oplus (\langle v_{a_1} \rangle^\perp \otimes \langle v_S \rangle^\perp \otimes \langle v_{a_3} \rangle).$$

For x_{a_1,S,a_3} we just need to add on the $\langle v_{a_1} \rangle^\perp \otimes \langle v_S \rangle^\perp \otimes \langle v_{a_3} \rangle$ part; we can use the orthonormal basis $\{b \otimes c \otimes v_{a_3} : b \in \mathcal{B}_{a_1}, c \in \mathcal{B}_S\}$, which requires $(h-1)^2$ slice calls.

For $A_{S,T}$, we first add $\langle v_{a_1} \rangle^\perp \otimes \langle v_S, v_T \rangle^\perp \otimes \langle v_{a_3} \rangle$ using the orthonormal basis $\{b \otimes c \otimes v_{a_3} : b \in \mathcal{B}_{a_1}, c \in \mathcal{B}_{S,T}\}$, which gets us to $\langle v_{a_1} \rangle^\perp \otimes \langle v_S \rangle^\perp \otimes \langle v_{a_3} \rangle$. (The key fact here is that $v_S \cdot v_T = 0$.) We then add the $\mathbb{F}_2^h \otimes \langle v_S \rangle \otimes \langle v_{a_3} \rangle$ part using the orthonormal basis $\{e_j \otimes v_S \otimes v_{a_3} : j \in [h]\}$. The total number of slice calls is $(h-1)(h-2) + h = h^2 - 2h + 2$.

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then subtract $\frac{1}{2}A_{S,T}$ from x_{a_1,S,a_3} , and if $|S \cap T| = 2$ then add $\frac{1}{2}A_{S,T}$ to x_{a_1,S,a_3} .

Cost: $|\mathcal{T}|(h-1)^2 + |\mathcal{P}|(h^2 - 2h + 2)$.

Final state (taking the opportunity to rewrite all of the subspaces in terms of complements for the first two components):

register	current U	underlying value
x_{a_1,S,a_3}	$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{x}_{a_1,S,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} (\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$ $+ \frac{1}{2} \sum_{T: S \cap T =2} (\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
c_i	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i + \sum_{T:i \in T} (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$
c_*	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_* + \sum_T (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$

Stage 2, step 3 (inverse of baby program step 4)

Transforms: we want to bring x_{a_1,S,a_3} and c_i, c_* up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$.

For x_{a_1,S,a_3} this requires adding on $\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp \otimes \langle v_{a_3} \rangle$, for which we use the orthonormal basis $\{v_{a_1} \otimes b \otimes v_{a_3} : b \in \mathcal{B}_S\}$. This requires $h-1$ slice calls.

For c_i, c_* this requires adding on $\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$. We use the orthonormal basis $\{v_{a_1} \otimes e_j \otimes v_{a_3} : j \in [h]\}$. This requires h slice calls.

Mixups: for each $S \in \mathcal{T}$ we add $\frac{1}{2}c_* - \frac{1}{2} \sum_{i \in S} c_i$ to x_{a_1,S,a_3} . A similar argument to step 4 of the baby program shows that most $\tilde{x}$ and $\tilde{y}$ terms drop out, and the $\tilde{x}_{a_1,S,a_3}$ gets replaced by $-\tilde{y}_{a_1,S,a_3}$.

Cost: $|\mathcal{T}|(h-1) + (h+1)h$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1,S,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$ $+ \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i + \sum_{T:i \in T} (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_* + \sum_{T \in \mathcal{T}} (\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3})$

Stage 2, step 4 (inverse of baby program step 3)

Transforms: we want to move c_i, c_* down to $(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$, to line up with y_{a_1,T,a_3} . This is the only backwards move in Stage 2. We need to subtract out the subspace $\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$, so we use the orthonormal basis $\{v_{a_1} \otimes e_j \otimes v_{a_3} : j \in [h]\}$. The number of slice calls is h .

Mixups: for each $T \in \mathcal{T}$ we subtract y_{a_1,T,a_3} from every c_i with $i \in T$, and also from c_* .

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1,S,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$ $+ \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T} + \tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
c_i	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 2, step 5 (inverse of baby program step 2)

Transforms: we want to bring both y_{a_1,T,a_3} and $A_{S,T}$ up to $(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$.

For y_{a_1,T,a_3} this requires adding on the subspace $\langle v_{a_1} \rangle \otimes \langle v_T \rangle^\perp \otimes \langle v_{a_3} \rangle$. We use the orthonormal basis $\{v_{a_1} \otimes b \otimes v_{a_3} : b \in \mathcal{B}_T\}$. The number of slice calls is $h-1$.

For $A_{S,T}$, first observe that by (7.3),

$$(\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp = (\langle v_{a_1} \rangle \otimes \langle v_S \rangle) \oplus (\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h).$$

This implies (by $v_S \cdot v_T = 0$ and (7.3) again) that

$$\begin{aligned} (\langle v_{a_1} \rangle \otimes \langle v_S \rangle^\perp)^\perp \oplus (\langle v_{a_1} \rangle \otimes \langle v_S, v_T \rangle^\perp) &= (\langle v_{a_1} \rangle \otimes \langle v_T \rangle^\perp) \oplus (\langle v_{a_1} \rangle^\perp \otimes \mathbb{F}_2^h) \\ &= (\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp. \end{aligned}$$

Thus, to get $A_{S,T}$ up to $(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$, it suffices to add the subspace $\langle v_{a_1} \rangle \otimes \langle v_S, v_T \rangle^\perp \otimes \langle v_{a_3} \rangle$. We use the orthonormal basis $\{v_{a_1} \otimes b \otimes v_{a_3} : b \in \mathcal{B}_{S,T}\}$. The number of slice calls is $h-2$.

Mixups: for each $(S, T) \in \mathcal{P}$ we subtract y_{a_1,T,a_3} from $A_{S,T}$.

Cost: $|\mathcal{T}|(h-1) + |\mathcal{P}|(h-2)$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1,S,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$ $+ \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T}$
c_i	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$(\langle v_{a_1} \rangle \otimes \mathbb{F}_2^h)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 2, step 6 (inverse of baby program step 1)

Transforms: we bring c_i, c_* back up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$. As in step 4, the number of slice calls is h .

Mixups: for each $S \in \mathcal{T}$ we subtract $\frac{1}{2}c_* - \frac{1}{2} \sum_{i \in S} c_i$ from x_{a_1,S,a_3} .

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1,S,a_3} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T}$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 2, step 7 (inverse of baby program step 0)

Transforms: we bring $A_{S,T}$ up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$. Only one slice call is needed here, to add the subspace $\langle v_{a_1} \rangle \otimes \langle v_T \rangle \otimes \langle v_{a_3} \rangle$.

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then add $\frac{1}{2}A_{S,T}$ to x_{a_1,S,a_3} , and if $|S \cap T| = 2$ then subtract $\frac{1}{2}A_{S,T}$ from x_{a_1,S,a_3} .

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
x_{a_1,S,a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1,S,a_3}$
y_{a_1,T,a_3}	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1,T,a_3} + \tilde{x}_{a_1,T,a_3}$
$A_{S,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$\tilde{c}_*$

Stage 2, step 8

Transforms: as in step 8 of Stage 1, we need to finish transforming the scratch registers, to get to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$. This requires adding the subspace $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle^\perp$. We use the orthonormal basis $\{e_j \otimes e_k \otimes b : j, k \in [h], b \in \mathcal{B}_{a_3}\}$. This costs $h^2(h-1)$ slice calls.

Mixups: none.

Cost: $(|\mathcal{P}| + h + 1)h^2(h-1)$.

Final state:

register	current U	underlying value
x_{a_1, S, a_3}	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$	$-\tilde{y}_{a_1, S, a_3}$
y_{a_1, T, a_3}	$(\langle v_{a_1} \rangle \otimes \langle v_T \rangle)^\perp \otimes \langle v_{a_3} \rangle$	$\tilde{y}_{a_1, T, a_3} + \tilde{x}_{a_1, T, a_3}$
$A_{S, T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{A}_{S, T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

As in stage 1, we are now done with these scratch registers, and we bring in a fresh set for the next pair (a_1, a_3) .

Stage 2 — conclusion

Again, the subspace for y_{a_1, T, a_3} is still one dimension short of the stage 2 target, which is $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_{a_3} \rangle$. So the $\tilde{y}$ component of y is not quite fully transformed. But the $\tilde{x}$ part is fully transformed (just as in stage 1), and the important new feature is that the $\tilde{y}$ value in the x register is *also* fully transformed up to the current target.

The total slice call count for each pair (a_1, a_3) is

$$\begin{aligned}
& |\mathcal{T}|((h-1)^2 + (h-1)^2 + (h-1) + (h-1)) \\
& + |\mathcal{P}|((h-1) + (h^2 - 2h + 2) + (h-2) + 1 + h^2(h-1)) \\
& + ((h-1)h(h+1) + (h+1)h + (h+1)h + (h+1)h + (h+1)h^2(h-1)) \\
& = 2|\mathcal{T}|(h^2 - h) + |\mathcal{P}|h^3 + (h^2 + 2)(h^2 + h).
\end{aligned}$$

Thus the total cost of Stage 2 is

$$2|\mathcal{T}|^3(h^2 - h) + |\mathcal{T}|^2|\mathcal{P}|h^3 + |\mathcal{T}|^2(h^2 + 2)(h^2 + h).$$

The combined cost of Stage 1 and Stage 2 is

$$2|\mathcal{T}|^3(h^2 - 1) + 2|\mathcal{T}|^2|\mathcal{P}|h^3 + 2|\mathcal{T}|^2(h^2 + 2)(h^2 + h).$$

Stage 3 — overview

In this stage, we want to compute the original baby program map $(x, y) \mapsto (x, y+x)$ again, to clear out the $\tilde{y}$ part of the y registers. We are going to run the baby program in forward order, now

operating on the third components. The $\Phi_{\langle v \rangle}$'s need to be modified, but at least this time they are all going in the right direction.

For each $(a_1, a_2) \in \mathcal{T} \times \mathcal{T}$ we perform the following steps. (The numbering matches that of the baby program.)

Stage 3, initialisation

The data registers in the following table are copied from the final state of Stage 2. For each (a_1, a_2) we use a new set of scratch registers, initialised to $U = 0$.

register	current U	underlying value
$x_{a_1, a_2, T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_T \rangle$	$-\tilde{y}_{a_1, a_2, T}$
$y_{a_1, a_2, S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle$	$\tilde{y}_{a_1, a_2, S} + \tilde{x}_{a_1, a_2, S}$
$A_{S, T}$	0	$\tilde{A}_{S, T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 3, step 0

Transforms: we bring $A_{S, T}$ up to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle$ (to match $y_{a_1, a_2, S}$), using the orthonormal basis $\{b \otimes c \otimes v_S : b \otimes c \in \mathcal{B}_{a_1 \otimes a_2}\}$. The number of slice calls is $h^2 - 1$.

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then subtract $\frac{1}{2}A_{S, T}$ from $y_{a_1, a_2, S}$, and if $|S \cap T| = 2$ then add $\frac{1}{2}A_{S, T}$ to $y_{a_1, a_2, S}$.

Cost: $|\mathcal{P}|(h^2 - 1)$.

Final state:

register	current U	underlying value
$x_{a_1, a_2, T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_T \rangle$	$-\tilde{y}_{a_1, a_2, T}$
$y_{a_1, a_2, S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle$	$\tilde{y}_{a_1, a_2, S} + \tilde{x}_{a_1, a_2, S} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S, T} + \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S, T}$
$A_{S, T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle$	$\tilde{A}_{S, T}$
c_i	0	$\tilde{c}_i$
c_*	0	$\tilde{c}_*$

Stage 3, step 1

Transforms: we want to move $y_{a_1, a_2, S}$ and c_i, c_* up to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$.

For $y_{a_1, a_2, S}$ this involves adding the subspace $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle^\perp$. We use the orthonormal basis $\{b \otimes c \otimes w : b \otimes c \in \mathcal{B}_{a_1 \otimes a_2}, w \in \mathcal{B}_S\}$. The number of slice calls is $(h^2 - 1)(h - 1)$.

For c_i, c_* we have to add on all of $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$. We use the orthonormal basis $\{b \otimes c \otimes e_j : b \otimes c \in \mathcal{B}_{a_1 \otimes a_2}, j \in [h]\}$. The number of slice calls is $(h^2 - 1)h$.

Mixups: for each $S \in \mathcal{T}$ we add $\frac{1}{2}c_* - \frac{1}{2} \sum_{i \in S} c_i$ to $y_{a_1, a_2, S}$.

Cost: $|\mathcal{T}|(h^2 - 1)(h - 1) + (h^2 - 1)h(h + 1)$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \langle v_T \rangle$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{y}_{a_1,a_2,S} + \tilde{x}_{a_1,a_2,S} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T}$ $+ \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle$	$\tilde{A}_{S,T}$
c_i	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

Stage 3, step 2

Transforms: we want to bring $x_{a_1,a_2,T}$ and $A_{S,T}$ up to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp$.

For $x_{a_1,a_2,T}$ this requires adding the subspace $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_T \rangle^\perp$ (by (7.3)). We use the orthonormal basis $\{b \otimes c \otimes w : b \otimes c \in \mathcal{B}_{a_1 \otimes a_2}, w \in \mathcal{B}_T\}$. This needs $(h^2 - 1)(h - 1)$ slice calls.

For $A_{S,T}$ we must first add the subspace $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \langle v_S \rangle^\perp$ to get up to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$. We use the orthonormal basis $\{b \otimes c \otimes w : b \otimes c \in \mathcal{B}_{a_1 \otimes a_2}, w \in \mathcal{B}_S\}$. This requires $(h^2 - 1)(h - 1)$ slice calls. Then we add the one-dimensional space $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle$, which costs only one slice call.

Mixups: for each $(S, T) \in \mathcal{P}$, add $x_{a_1,a_2,T}$ to $A_{S,T}$.

Cost: $|\mathcal{T}|(h^2 - 1)(h - 1) + |\mathcal{P}|((h^2 - 1)(h - 1) + 1)$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{y}_{a_1,a_2,S} + \tilde{x}_{a_1,a_2,S} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T}$ $+ \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp$	$\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T}$
c_i	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

Stage 3, step 3

Transforms: we want to move $x_{a_1,a_2,T}$ and c_i, c_* up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$ (the fully transformed state).

For $x_{a_1,a_2,T}$ we need to add the subspace $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle^\perp$. We use the orthonormal basis $\{v_{a_1} \otimes v_{a_2} \otimes b : b \in \mathcal{B}_T\}$, at a cost of $h - 1$ slice calls.

For c_i, c_* we need to add the subspace $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \mathbb{F}_2^h$. We use the orthonormal basis $\{v_{a_1} \otimes v_{a_2} \otimes e_j : j \in [h]\}$. This costs h slice calls.

Mixups: for each $T \in \mathcal{T}$, add $x_{a_1,a_2,T}$ to every c_i for which $i \in T$, and also to c_* .

Cost: $|\mathcal{T}|(h-1) + (h+1)h$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{y}_{a_1,a_2,S} + \tilde{x}_{a_1,a_2,S} - \frac{1}{2} \sum_{T: S \cap T =0} \tilde{A}_{S,T}$ $+ \frac{1}{2} \sum_{T: S \cap T =2} \tilde{A}_{S,T} + \frac{1}{2} \tilde{c}_* - \frac{1}{2} \sum_{i \in S} \tilde{c}_i$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp$	$\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_i - \sum_{T:i \in T} \tilde{y}_{a_1,a_2,T}$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_* - \sum_T \tilde{y}_{a_1,a_2,T}$

Stage 3, step 4

Transforms: we bring c_i, c_* back down to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$. This is the only backwards move in Stage 3. We use the same orthonormal basis as step 3. This costs h slice calls.

Mixups: for each $S \in \mathcal{T}$, subtract $\frac{1}{2}c_* - \frac{1}{2} \sum_{i \in S} c_i$ from $y_{a_1,a_2,S}$.

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{x}_{a_1,a_2,S} - \frac{1}{2} \sum_{T: S \cap T =0} (\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T})$ $+ \frac{1}{2} \sum_{T: S \cap T =2} (\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T})$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp$	$\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T}$
c_i	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_i - \sum_{T:i \in T} \tilde{y}_{a_1,a_2,T}$
c_*	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_* - \sum_T \tilde{y}_{a_1,a_2,T}$

Stage 3, step 5

Transforms: we want to bring $y_{a_1,a_2,S}$ and $A_{S,T}$ up to $(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$.

For $y_{a_1,a_2,S}$ we need to add the subspace $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle^\perp$. We use the orthonormal basis $\{v_{a_1} \otimes v_{a_2} \otimes b : b \in \mathcal{B}_S\}$, at a cost of $h-1$ slice calls.

For $A_{S,T}$ we need only add the subspace $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S, v_T \rangle^\perp$, as

$$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp = (\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle) \oplus ((\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h)$$

together with $v_S \cdot v_T = 0$ implies that

$$\begin{aligned} & (\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_T \rangle)^\perp \oplus (\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S, v_T \rangle^\perp) \\ &= (\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle^\perp) \oplus ((\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h) = (\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp. \end{aligned}$$

Thus we may use the orthonormal basis $\{v_{a_1} \otimes v_{a_2} \otimes b : b \in \mathcal{B}_{S,T}\}$, at a cost of $h-2$ slice calls.

Mixups: for each $(S, T) \in \mathcal{P}$, if $|S \cap T| = 0$ then add $\frac{1}{2}A_{S,T}$ to $y_{a_1,a_2,S}$, and if $|S \cap T| = 2$ then subtract $\frac{1}{2}A_{S,T}$ from $y_{a_1,a_2,S}$.

Cost: $|\mathcal{T}|(h-1) + |\mathcal{P}|(h-2)$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$	$\tilde{x}_{a_1,a_2,S}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$	$\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T}$
c_i	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_i - \sum_{T:i \in T} \tilde{y}_{a_1,a_2,T}$
c_*	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle)^\perp \otimes \mathbb{F}_2^h$	$\tilde{c}_* - \sum_T \tilde{y}_{a_1,a_2,T}$

Stage 3, step 6

Transforms: bring c_i, c_* up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$ by adding the subspace $\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \mathbb{F}_2^h$. We use the orthonormal basis $\{v_{a_1} \otimes v_{a_2} \otimes e_j : j \in [h]\}$, at a cost of h slice calls.

Mixups: for each $T \in \mathcal{T}$, subtract $x_{a_1,a_2,T}$ from every c_i for which $i \in T$, and also from c_* .

Cost: $(h+1)h$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$	$\tilde{x}_{a_1,a_2,S}$
$A_{S,T}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$	$\tilde{A}_{S,T} - \tilde{y}_{a_1,a_2,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

Stage 3, step 7

Transforms: bring $A_{S,T}$ up to $\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$ by a single step in the direction $v_{a_1} \otimes v_{a_2} \otimes v_S$. This costs one slice call.

Mixups: for each $(S, T) \in \mathcal{P}$, subtract $x_{a_1,a_2,T}$ from $A_{S,T}$.

Cost: $|\mathcal{P}|$.

Final state:

register	current U	underlying value
$x_{a_1,a_2,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$-\tilde{y}_{a_1,a_2,T}$
$y_{a_1,a_2,S}$	$(\langle v_{a_1} \rangle \otimes \langle v_{a_2} \rangle \otimes \langle v_S \rangle)^\perp$	$\tilde{x}_{a_1,a_2,S}$
$A_{S,T}$	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{A}_{S,T}$
c_i	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_i$
c_*	$\mathbb{F}_2^h \otimes \mathbb{F}_2^h \otimes \mathbb{F}_2^h$	$\tilde{c}_*$

At this point the scratch registers are fully transformed, and we bring in new ones for the next pair (a_1, a_2) .

Stage 3 — conclusion

Let us analyse what has happened to the physical values over the course of all three stages.

- All $3|\mathcal{T}|^2(|\mathcal{P}| + h + 1)$ scratch registers have been fully transformed.
- The initial values $\tilde{y}_{a_1, a_2, a_3}$, which started out in the y registers, have ended up in the x registers, fully transformed, and with a harmless minus sign.
- The initial values $\tilde{x}_{a_1, a_2, a_3}$, which started out in the x registers, have ended up in the y registers. They have been fully transformed, up to a rearrangement, by the same argument used in the baby case (see the section “What did the baby program achieve?”).

Therefore, the net result is that every single register has been fully transformed (modulo free rearrangements and sign changes).

The number of slice calls in Stage 3 for each pair (a_1, a_2) is

$$\begin{aligned} |\mathcal{T}|((h^2 - 1)(h - 1) + (h^2 - 1)(h - 1) + (h - 1) + (h - 1)) \\ + |\mathcal{P}|((h^2 - 1) + (h^2 - 1)(h - 1) + 1 + (h - 2) + 1) \\ + ((h^2 - 1)h(h + 1) + (h + 1)h + (h + 1)h + (h + 1)h) \\ = 2|\mathcal{T}|(h^3 - h^2) + |\mathcal{P}|h^3 + (h^2 + 2)(h^2 + h). \end{aligned}$$

Thus the total number of slice calls in Stage 3 over all pairs is

$$2|\mathcal{T}|^3(h^3 - h^2) + |\mathcal{T}|^2|\mathcal{P}|h^3 + |\mathcal{T}|^2(h^2 + 2)(h^2 + h).$$

Summing over all three stages, the grand total is

$$2|\mathcal{T}|^3(h^3 - 1) + 3|\mathcal{T}|^2|\mathcal{P}|h^3 + 3|\mathcal{T}|^2(h^2 + 2)(h^2 + h).$$

Recalling that the number of registers is $W = 2|\mathcal{T}|^3 + 3|\mathcal{T}|^2(|\mathcal{P}| + h + 1)$, after a bit of algebra we can rewrite the grand total as

$$Wh^3 - |\mathcal{T}|^2(2|\mathcal{T}| - 6h(h + 1)).$$

Since $|\mathcal{T}|$ is cubic in h , for sufficiently large h the bracketed term is positive (according to Fable, for $h \geq 22$), and we see that the total number of slice calls is strictly less than $Wh^3 = Wm$. And this is exactly what the tensor amplification wants.

Unbelievable.

8 Conclusion

As far as I can see, the gadget does what it’s supposed to do. I have worked through all the steps pretty carefully, and I think it checks out. (Where it came from, I have no idea.)

I also think I believe the tensor amplification machinery.

Parts I still don’t understand properly:

- How the linear operation count and the synthetic roots of unity interact with the tensor amplification, and how to go from $2 \times \cdots \times 2$ to $r \times \cdots \times r$. All of this feels like it should be okay, but I need to think about it some more to be certain.
- How the rest of the 2021 algorithm behaves when d is allowed to grow at the required rate. Again this feels like it should be okay, but I haven't checked.
- The big one: data rearrangement costs in the Turing model. I haven't even begun to look into this yet.

Some final thoughts about the $\log n$ exponent. To improve it, one would need to find a more efficient gadget. I gather that this is what people (and their LLM helpers) have been doing. The impression I got from a very quick glance at their repositories is that they are trying to come up with clever optimisations of the existing construction. But one other thing that might be interesting to try (if they haven't already done so) is to simply do some kind of (clever?) brute-force search for a very small gadget. Who knows, maybe it's possible to write down a non-trivial gadget for $I \otimes B^{\otimes 3}$ acting on $\mathbb{C}^4 \otimes (\mathbb{C}^2)^{\otimes 3}$ or something silly like that. I mean, no-one has ever actually tried, right?

Still a lot of work to do. But I need a break now.

References

- [AFKL19] P. Afshani, C. B. Freksen, L. Kamma, and K. G. Larsen, *Lower bounds for multiplication via network coding*, 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (Dagstuhl, Germany) (Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 10:1–10:12.
- [Har26] D. Harvey, *Faster enumeration of primes*, <https://arxiv.org/abs/2606.22851>, 2026.
- [HvdH19] D. Harvey and J. van der Hoeven, *Faster polynomial multiplication over finite fields using cyclotomic coefficient rings*, J. Complexity **54** (2019), 101404, 18. MR 3983215
- [HvdH21] ———, *Integer multiplication in time $O(n \log n)$* , Ann. of Math. (2) **193** (2021), no. 2, 563–617. MR 4224716
- [HvdH22] ———, *Polynomial multiplication over finite fields in time $O(n \log n)$* , J. ACM **69** (2022), no. 2, Art. 12, 40. MR 4433320
- [HvdH25] D. Harvey and J. van der Hoeven, *Integer multiplication is at least as hard as matrix transposition*, <https://arxiv.org/abs/2503.22848>, 2025.
- [LL04] Zongpeng Li and Baochun Li, *Network coding in undirected networks*, CISS, 2004.
- [TC26a] The Company, *Integer multiplication below $n \log n$* , GitHub, 2026.
- [TC26b] ———, *Finite tensor savings and exact Fourier circuits*, GitHub, 2026.
- [TC26c] ———, *An explicit power saving for the exact discrete Fourier transform*, GitHub, 2026.