Reliable Computations Based on Locally Decodable Codes

Andrei Romashchenko*

September 18, 2005

Abstract

We investigate the coded model of fault-tolerant computations introduced by D. Spielman. In this model the input and the output of a computation circuit is treated as words in some error-correcting code. A circuit is said to compute some function correctly if for an input which is a encoded argument of the function, the output, been decoded, is the value of the function on the given argument.

We consider two models of faults. In the first one we suppose that an elementary processor at each step can be corrupted with some small probability, and faults of different processors are independent. For this model, we prove that a parallel computation running on n elementary non-faulty processors in time $t = \mathrm{poly}(n)$ can be simulated on $\mathcal{O}(n \log n / \log\log n)$ faulty processors in time $\mathcal{O}(t \log\log n)$. Note that we get sub-logarithmic blow-up of the memory, which cannot be achieved in the classic model of faulty boolean circuit, where the input is not encoded.

In the second model, we assume that at each step some fixed fraction of elementary processors can be corrupted by an adversary, who is free to chose these processors arbitrarily. We show that in this model any computation can be made reliable with exponential blow-up of the memory.

Our method employs a sort of *mixing mappings*, which enjoy some properties of expanders. Based on mixing mappings, we implement an effective self-correcting procedure for an array of faulty processors.

1 Introduction

The problem of reliable computations with faulty elements was investigated for several types of computational models, see a survey in [14]. In the most popular models, the computation is implemented by a circuit of boolean gates; each gate can fail with a small probability. A circuit is said fault-tolerant if it returns a correct result with high probability. For the first time such a model of computation was proposed by J. von Neumann [1]. Later ideas by von Neumann was developed by Dobrushin and Ortyukov [3]. N. Pippenger in [5] presented an *effective* transformation of any boolean circuit with non-faulty gates into a fault-tolerant circuit.

*Institute for Information Transmission Problems, Moscow. e-mail:anromash@mccme.ru

The construction of Pippenger requires only logarithmic increasing of the number of gates. In general, this result cannot be improved, and logarithmic redundancy is inevitable [4, 8, 6, 7]. But this lower bound is caused by the need to encode the input with some error-correcting code and then to decode the answer. This obstacle can be eliminated if we allow to get the input and to return the result as an encoded word. Such a model was used in the work of D. Spielman [10]. Let us define this model (with minor modifications) in detail.

The computational model.

The computational array consists of N elementary processors $s_1, \ldots, s_N$. At any moment, each processor contains one bit of information (a processor is said to have an internal state 0 or 1). We fix two functions,

$$E \,:\, \{0,1\}^n \to \{0,1\}^N$$

and

$$D \,:\, \{0,1\}^N \to \{0,1\}^n$$

which are normally the encoding and decoding functions of some error-correcting code. We say that a circuit gets an input $x \in \{0,1\}^n$ if the initial state of the memory $(s_1, \ldots, s_N)$ is equal to $E(x)$.

Denote by $s_1^{(t)}, \ldots, s_N^{(t)}$ the internal states of the processors at moment t. We call by *a circuit of depth* T a list of instructions $F_i^{(t)}$, $t = 1, \ldots, T$ which define how each of the processors should update its internal state at each moment. More precisely, the state of a processor s_i at moments t evolves by the rule

$$s_i^{(t)} = F_i^{(t)}(s_{j_1}^{(t-1)}, s_{j_2}^{(t-1)}, \ldots, s_{j_r}^{(t-1)}),$$

where $F_i^{(t)}$ are boolean functions (the indexes $j_1, \ldots, j_r$ depend on i and t). The arity r is supposed to be fixed in advance. We shall always suppose that r is a constant independent of n.

We say that a circuit of depth T computes a function $f : \{0,1\}^n \to \{0,1\}^n$ if for any x

$$D(s_1^{(T)}, \ldots, s_N^{(T)}) = f(x), \quad \text{if } (s_1^{(0)}, \ldots, s_N^{(0)}) = E(x)$$

In other words, we use the encoding E to provide the circuit with an input, and then use the decoding D to retrieve the result. We shall say also that such a circuit computes f *in* T *steps*.

In the model of random faults we suppose that at any step each processor can be randomly corrupted, i.e., with some small probability ϵ it can change its internal state contrary to the rule above. Faults at different positions i and moments t (i.e., the events that a processor s_i was corrupted at moment t) are supposed to by independent. For this model, we say that some circuit correctly computes a function f with probability ϵ', if

$$\mathrm{Prob}[D(s_1^{(T)}, \ldots, s_N^{(T)}) = f(x) | (s_1^{(0)}, \ldots, s_N^{(0)}) = E(x)] = 1 - \epsilon'.$$

In another model faults are made by an malevolent adversary. This means that at any moment t any ϵN processors can be corrupted. We say that a circuit computes some function f if for any choice of the positions where processors are corrupted, the final result is correct, i.e., again

$$D(s_1^{(T)}, \ldots, s_N^{(T)}) = f(x) \text{ if } (s_1^{(0)}, \ldots, s_N^{(0)}) = E(x)$$

Note that the defined model is trivial if the functions E and D may depend on the function f. In the sequel we fix some E and D and show that *any* function f can be computed by a reliable circuit (in the model with random faults or in the model with an adversary).

The rest of the paper is organized as follows. In Section 2 we introduce *mixing mappings*, the main combinatorial tool of our proofs. In Section 3 we consider the model of random faults and prove that any circuit of depth T with n processors can be converted in a reliable circuit with $\mathcal{O}(n \log(nT)/\log\log(nT))$ processors, which computes the same function in time $\mathcal{O}(T \log\log(nT)$. This bound is a bit stronger than the result of [10], where the construction requires poly-logarithmic blow-up and poly-logarithmic slow down. If $T = \mathrm{poly}(n)$, the blow-up in our construction is equal to $\mathcal{O}(\log n/\log\log n)$, i.e., it is below the $\log n$ barrier, which is strict for the usual model of faulty boolean circuits (where the input of a circuit is not encoded). Our construction is effective, i.e., the fault-tolerant circuit can be constructed from the original one by a polynomial algorithm.

In Section 4 we deal with the model where faults are chosen by an adversary. We prove that any circuit of depth T with n processors can be converted in a reliable circuit with $2^{\mathcal{O}(n)}$ processors and depth $\mathcal{O}(nT)$.

2 Mixing functions

In this section we define mixing mappings and prove some of their properties.

Definition 1 *We call a mapping* $F : \{0,1\}^m \times \{0,1\}^\tau \to \{0,1\}^m$ *a* (m, τ, α, β)-mixer *if for any* $A, B \subset \{0,1\}^m$ *such that* $|A| \geq \alpha 2^m$

$$\| |\{(x,u) \,:\, x \in A, F(x,u) \in B\}| - \frac{2^\tau |A| \cdot |B|}{2^m} \| < \beta 2^\tau |A|.$$

This definition was inspired by the well-known Expander Mixing Lemma, see e.g, [11]. The Expander Mixing Lemma implies that an expander is a mixer with appropriate parameters.

We need mixers with some additional structure. First of all, we consider $L = \{0,1\}^m$ as an m-dimensional linear space over $\mathbb{Z}/2\mathbb{Z}$ (for $u, v \in L$ the vector $u + v$ is just the bitwise sum of u and v modulo 2).

Definition 2 *We call a mapping* $F : \{0,1\}^m \times \{0,1\}^\tau \to \{0,1\}^m$ *a* linear (m, τ, α, β)-mixer *if (1)* F *is a* (m, τ, α, β)-mixer*, and (2) the mapping* $G : \{0,1\}^m \times \{0,1\}^\tau \to \{0,1\}^m$ *defined as* $G(x,u) = F(x,u) + x$*, is also a* (m, τ, α, β)*-mixer.*

Standard probabilistic arguments show that a linear mixer exists:

Lemma 1 *For any $\alpha, \beta \in (0,1)$ there exists a τ such that for all m there exists a linear (m, τ, α, β)-mixer.*

We prove this lemma in Appendix.

The following properties of a mixer easily follows from the definition:

Claim 1 *If $0 < \delta' < \delta < 1/8$ then for any $(m, \tau, \delta', \delta)$-mixer F, for any $B \subset \{0,1\}^m$ of size at most $2^m/8$ there are less than $\delta 2^m$ elements $x \in \{0,1\}^m$ such that for at least 25% of $u \in \{0,1\}^\tau$*

$$F(x,u) \in B.$$

Claim 2 *Let $0 < \delta' < \delta < 1/128$ and F be an $(m, \tau, \delta', \delta)$-mixer. Then for any $B \subset \{0,1\}^m$ of size at most $2^m/128$ there are less than $\delta 2^m$ elements $x \in \{0,1\}^m$ such that for at least $1/64$ of all $u \in \{0,1\}^\tau$*

$$F(x,u) \in B.$$

We prove Claim 1 in Appendix; Claim 2 follows from similar arguments.

Lemma 2 *Let F_1 be an $(m_1, \tau_1, \alpha_1, \beta_1)$-mixer and F_2 be an $(m_2, \tau_2, \alpha_2, \beta_2)$-mixer. Then the tensor product $F = F_1 \otimes F_2$*

$$F : \{0,1\}^{m_1+m_2} \times \{0,1\}^{\tau_1+\tau_2} \to \{0,1\}^{m_1+m_2}$$

is an $(m_1+m_2, \tau_1+\tau_2, \alpha, \beta)$-mixer for $\alpha = \sqrt{\alpha_2}$ and $\beta = \mathcal{O}(\frac{\alpha_1+\beta_1+\beta_2}{\alpha_2\sqrt{\alpha_2}} + \sqrt{\alpha_2})$

This lemma is interesting for $\alpha_1 \ll \beta_1 \ll \beta_2 \ll \alpha_2$. The bound in this lemma is quite rough, but it is enough for application below. Lemma 2 can be proved with quite standard combinatorial arguments, see Appendix.

Remark that if F_1 and F_2 are *linear* mixers then $F_1 \otimes F_2$ is also a *linear* mixer.

Lemma 3 *For any α, β and large enough τ there exists an algorithm which an the input m constructs a linear (m, τ, α, β)-mixer in time $\mathrm{poly}(2^{2^m})$.*

Proof: Denote $N = 2^{2^m}$. From Lemma 1 it follows that for all α', β' and a large enough τ, for all n a linear $(n, \tau', \alpha', \beta')$-mixer exists. If $2^{n2^n} = \mathrm{poly}(N)$, we can construct a linear $(n, \tau', \alpha', \beta')$-mixer in time $\mathrm{poly}(N)$ using a brute force search. In particular, for any $\alpha', \beta', \alpha'', \beta''$ we can get in polynomial time some mixers with parameters $(m/2, \tau', \alpha', \beta')$ and $(m/2, \tau'', \alpha'', \beta'')$. (degree of the polynomial may depend on $\alpha', \beta', \alpha'', \beta''$). Further, construct a tensor product of these two mixers. From Lemma 2 it follows that we obtain a linear $(m, \tau'+\tau'', \sqrt{\alpha''}, \mathcal{O}(\frac{\alpha'+\beta'+\beta'}{\alpha''\sqrt{\alpha''}} + \sqrt{\alpha''}))$-mixer. It remains to choose appropriate α', α'' and β', β''. $\square$

3 Computations resistant to random faults

In this section we show how to implement reliable computations with a circuit based on faulty processors. We assume that each processor at one step of computation is

corrupted with small enough probability $\epsilon > 0$, and that faults at different processors and at different moments of time are independent.

We use encoding based on the Hadamard code. Remind that the Hadamard code is a mapping

$$Had \,:\, \{0,1\}^n \to \{0,1\}^{2^n}$$

where $Had(a_1, \ldots, a_n)$ is the graph of the linear function of n variables

$$f(x_1, \ldots, x_n) = \sum_{1 \le i \le n} a_i x_i$$

(the coefficients a_i and the variables x_i ranges over the field $\mathbb{Z}/2\mathbb{Z}$). The code is linear, so for any $x, y \in \{0,1\}^n$ we have

$$Had(x \oplus y) = Had(x) \oplus Had(y).$$

Here and in the sequel we denote by $x \oplus y$ the bitwise sum modulo 2.

Theorem 1 *For any circuit S of depth t with n processors, for any $\epsilon_{res} > 0$ there exists a circuit $\hat{S}$ of depth $\mathcal{O}(t \log\log(tn))$, with $\mathcal{O}(n \log(nt)/\log\log(nt))$ processors that computes the same function for the encoded input so that if any processor at each step is corrupted with probability $\epsilon < 1/8$ then the result is correct with probability at least $(1 - \epsilon_{res})$. Moreover, there exists a polynomial algorithm that constructs such a circuit $\hat{S}$ given S.*

Proof: Denote by $y^{(j)} = (y_1^{(j)}, y_2^{(j)}, \ldots, y_n^{(j)})$ the state of the memory of S at the j-th step of the computation, $j = 0, \ldots, t$. We shall define an encoding $E : \{0,1\}^n \to \{0,1\}^N$ and the corresponding decoding $D : \{0,1\}^N \to \{0,1\}^n$, and a circuit $\hat{S}$ with N processors so that for any j the internal state $z^{(j)} = (z_1^{(i)}, \ldots, z_N^{(i)})$ of $\hat{S}$ with high probability is close to $E(y^{(i)})$. More precisely, we require that with high probability the equality $D(z^{(j)}) = y^{(j)}$ holds. In particular, for the final result $z^{(t)}$ we get $D(z^{(t)}) = y^{(t)}$.

Let us implement the plan presented above. We split the set of variables $(x_1, \ldots, x_n)$ into blocks of size $k = \log(\log n + \log t) + C$:

$$\begin{array}{lcl} b_1 & = & x_1, \ldots, x_k, \\ b_2 & = & x_{k+1}, \ldots, x_{2k}, \\ \ldots & & \ldots \end{array}$$

(the constant C will be chosen below). The total number of blocks b_i is $r = \lceil n/k \rceil$.

A Restriction on Parallelism: Let us restrict the power of the parallelism in S. We shall assume that in the circuit S at each step in every block b_i only one processor changes its internal state. Moreover, we assume that every time when a processor changes its state, its new state is a boolean function of internal states of *two* processors from the previous step. In the sequel we show that a circuit that satisfies these restrictions can be simulated by a fault-tolerant circuit $\hat{S}$ with blow-up of the memory $\mathcal{O}(2^k/k)$ in real time, i.e., without any slow down.

It is easy to see that an arbitrary circuit can be transformed so that the assumptions above hold. The price for this transformation is a constant blow-up of the memory and slow down $\mathcal{O}(k) = \mathcal{O}(\log\log(nt))$. Thus, to prove the theorem it remains to explain how to construct a reliable circuit for an S satisfying these restrictions.

From this moment, we assume that S satisfy the *Restriction on Parallelism*. First of all, we specify encoding and decoding. Define encoding $E : \{0,1\}^n \to \{0,1\}^N$ as follows:

$$E(x_1, \ldots, x_n) = Had(b_1) \ldots Had(b_r).$$

Denote $\hat{b}_i = Had(b_i)$. Note that the length of each $\hat{b}_i$ is $2^k = 2^C \log(tn)$ and the length of the codewords is $N = r2^k = \mathcal{O}(n\log(tn)/\log\log(tn))$.

Define manipulations with encoded data. Denote by

$$z^{(j)} = (z_1^{(j)}, z_2^{(j)}, \ldots, z_N^{(j)})$$

the state of memory of the circuit at j-th stage of computation. We split $z^{(j)}$ into blocks of length 2^k and denote them $\hat{b}_1^{(j)}, \ldots, \hat{b}_r^{(j)}$.

Further we define the transition rule: how $z^{(j+1)}$ is computed from $z^{(j)}$. We define it so that with high probability for all j each block $\hat{b}_i^{(j)}$ differs from $Had(b_i^{(j)})$ in a fraction at most $1/8$ of all bits.

In our model, at each step $j = 1, \ldots, t$ each value $z_1, \ldots, z_N$ is computed as a function of the internal states of $\mathcal{O}(1)$ processors at the previous step. Remind that some of cells can be corrupted by random faults. Faults at different cells are independent and each one occurs with probability ϵ. We say that the random perturbation is ϵ_0-*normal* if for any block $\hat{b}_i$ at any stage of computation, there are at most $\epsilon_0 \cdot 2^k$ faults.

Lemma 4 *For any $\epsilon_0 \in (\epsilon, 1/8)$ and large enough constant C (which defines the length of blocks b_i) the perturbation is ϵ_0-normal with probability greater than $(1 - \epsilon_{res})$.*

Proof of the lemma: For each block $\hat{b}_i$ at each step the average number of faults is equal to $\epsilon 2^k$. From the Chernoff bound it follows that

$$\mathrm{Prob}[\text{number of faults} > \epsilon_0 2^k] < e^{-(\epsilon-\epsilon_0)^2 2^k}.$$

Sum up this probability for all $i = 1, \ldots, r$ and all step $j = 1, \ldots, t$. The sum is less than ϵ_{res} if the constant C is large enough. □

Let us fix some $\epsilon_0 \in (\epsilon, 1/8)$; in the sequel we assume that the perturbation is ϵ_0-normal, and construct a circuit which returns a correct result under this assumption. Also we fix some $\delta_0 > 0$ such that $8\delta_0 + \epsilon_0 < 1/8$. Let Mix be a linear $(k, \tau, \delta_0/2, \delta_0)$-mixer. As we showed in Lemma 3, such a mixer can be found in time $\mathrm{poly}(n, t)$.

In the sequel we prove by induction the following property: if at step j each $\hat{b}_i^{(j)}$ differs from the corresponding $Had(\hat{b}_i^{(j)})$ in at most $1/8$ of bits, then each $\hat{b}_i^{(j+1)}$ also differs from the corresponding $Had(\hat{b}_i^{(j+1)})$ in at most $1/8$ of bits (we assume that the random perturbation is ϵ_0-normal).

We define the computation step in two stages. First, we define $\hat{b}'_i, i = 1, \ldots, r$; each bit of $\hat{b}'_i$ is a function of $\mathcal{O}(1)$ bits from $\hat{b}_i^{(j)}$. At the second stage we define

$\hat{b}''_i, i = 1, \ldots, r$, each bit of a $\hat{b}''_i$ is a function of $\mathcal{O}(1)$ bits from $(\hat{b}'_1, \ldots, b'_r)$. Then we set $\hat{b}_i^{(j+1)} = \hat{b}''_i, i = 1, \ldots, r$.

Now we describe the first stage of the construction. Fix a number of a block i, number of a step j, and let $\hat{b}_i^{(j)} = (u_1, \ldots, u_{2^k})$. We identify an integer $q \in \{1, \ldots, 2^k\}$ and its k-digit binary representation (with leading zeros). Thus, we can use q as the first argument of the mixer Mix. Moreover, for an integer $q \leq 2^k$ and $\eta \in \{0,1\}^\tau$ we can consider $u_{Mix(q,\eta)\oplus q}$, where $Mix(q,\eta)\oplus$ is the bitwise sum of two k-bit words: $Mix(q,\eta)$ and the binary representation of q.

Assume that $\hat{b}_i^{(j)}$ differs from $E(b_i^{(j)})$ in at most $2^k/8$ bits. For any q the bit u'_q is computed as

$$u'_q = majority\{(u_{Mix(q,\eta)\oplus q} - u_{Mix(q,\eta)}) \mid \eta \in \{0,1\}^\tau\}$$

(here we identify the integer $q \leq 2^k$ and its binary representation; thus, $u_{Mix(q,\eta)\oplus q}$ is u_j, where the k-digit binary representation of j is the bitwise sum modulo 2 of the binary representation of q and $Mix(q,\eta)$).)

Let us bound the number of bits in $\hat{b}' = (u'_1, \ldots, u'_{2^k})$ that differ from the corresponding bits of $Had(b_i^{(j)})$. There may be two reasons why a u'_q differs from the corresponding bit of $Had(b_i^{(j)})$:

1. in the sequence $u_{Mix(q,0)\oplus q}, \ldots, u_{Mix(q,\tau)\oplus q}$ at least 25% of values differ from the corresponding values of $E(b_i^{(j)})$;
2. in the sequence $u_{Mix(q,0)}, \ldots, u_{Mix(q,\tau)}$ at least 25% of values differ from the corresponding values of $Had(b_i^{(j)})$;

From Claim 1 it follows that there are at most $2\delta_0 2^k$ positions q where at least one of these two conditions hold. Thus, we proved the following bound:

Claim 3 *There are at most $2\delta_0 2^k$ positions $q = 1, \ldots, 2^k$, where u'_q differs from the corresponding bit of $E(b_i^{(j)})$.*

Remind that we assume that at each step of the computation in the original circuit S exactly one bit of every block is updated. Let on the j-th step in a block $\hat{b}_{i_0}$ the internal state of a processor c_0 was updated: $x_{c_0}^{(j+1)} = F_j(x_{c_1}^{(j)}, x_{c_2}^{(j)})$, where F_j is some boolean function. Let the bits x_{c_1} and x_{c_2} be from the blocks $\hat{b}_{i_1}$ and $\hat{b}_{i_2}$ respectively ($c_i \in \{1, \ldots, N\}$). The difference between $Had(b_{i_0}^{(j)})$ and $Had(b_{i_0}^{(j+1)})$ is a function of $x_{c_0}, x_{c_1}, x_{c_2}$. More exactly, if $\xi = F_j(x_{c_1}^{(j)}, x_{c_2}^{(j)}) - x_{c_0}^{(j)}$, then the bitwise sum modulo 2

$$Had(b_{i_0}^{(j)}) \oplus Had(b_{i_0}^{(j+1)})$$

is equal to $Had(0, \ldots, 0, \xi, 0, \ldots, 0)$.

Let us explain the second stage of the construction and define u''_q. First of all, for all $i \neq i_0$ we let $\hat{b}''_i = \hat{b}'_i$. For $\hat{b}''_{i_0}$ make the computations as follows. Let $\hat{b}'_{j_0} =$

$(u'_1, \ldots, u'_{2^k})$, $\hat{b}'_{j_1} = (v'_1, \ldots, v'_{2^k})$ and $\hat{b}'_{j_2} = (w'_1, \ldots, w'_{2k})$. For each $q = 1, \ldots, 2^k$ we estimate the value x_{c_0} as

$$\tilde{x}_{c_0} = majority\{(u'_{Mix(q,\eta)\oplus c_0} - u'_{Mix(q,\eta)}) \mid \eta \in \{0,1\}^\tau\},$$

and the values x_{c_1} and x_{c_2} as

$$\tilde{x}_{c_1} = majority\{(v'_{Mix(q,\eta)\oplus c_1} - v'_{Mix(q,\eta)}), \mid \eta \in \{0,1\}^\tau\}$$

and

$$\tilde{x}_{c_2} = majority\{(w'_{Mix(q,\eta)\oplus c_2} - w'_{Mix(q,\eta)}), \mid \eta \in \{0,1\}^\tau\}$$

respectively. Set $\tilde{\xi} = F_j(\tilde{x}_{c_1}, \tilde{x}_{c_2}) - \tilde{x}_{c_0}$ and add the q-th digit of the value

$$Had(0, \ldots, 0, \tilde{\xi}, 0, \ldots, 0)$$

to the bit u'_q (as usual, all operations are in the field $\mathbb{Z}/2\mathbb{Z}$). Note that $\tilde{x}_{c_0}$ is estimated correctly unless at least 25% of the values $v_{Mix(q,0)+c_0}, \ldots, v_{Mix(q,\tau)+c_0}$ are corrupted or at least 25% of the values $v_{Mix(q,0)}, \ldots, v_{Mix(q,\tau)}$ are corrupted. From Claim 1, there are at most $2\delta_0 2^k$ positions q where one of these two conditions holds. The same is true for $\tilde{x}_{c_1}$ and $\tilde{x}_{c_2}$.

Let us bound the number of positions q where u''_q differs from the q-th bit of $Had(b_1^{(j+1)})$. All but $2\delta_0 2^k$ positions u'_q are equal to the corresponding bits of the (2^k)-bit string $Had(b_1^{(j)})$. All three values x_{c_0}, x_{c_1}, and x_{c_2} are estimated correctly for at least $6\delta_0 2^k$ positions. Further, at most $\epsilon_0 2^k$ bits of one block can be corrupted due to random faults. In total, the fraction of position where u''_q is not equal to the corresponding bit of $Had(b_0^{(i+1)})$ is at most $8\delta_0 + \epsilon_0 < 1/8$. $\square$

We proved that any computation can be made reliable so that the result is correct with some fixed probability $(1 - \epsilon_{res})$. We might want to get a circuit which fails with exponentially small probability. To decrease the probability of a failure, we can increase the size k of blocks b_i used in the proof of Theorem 1. If we let $k = C' \log\log(tn)$ for some $C' > 1$, our construction results in a circuit which fails with probability $e^{-\Omega(\log^{C'}(tn))}$; blow-up of the memory in this circuit is $\mathcal{O}(2^k/k) = \mathcal{O}(\log^{\mathcal{O}(1)}(tn))$. To implement this construction, we need a linear mixer with parameters $(\log(\log^{\mathcal{O}(1)}(tn)), \tau, \alpha, \beta)$. Such a mixer can be constructed in time $\mathrm{poly}(t, n)$; really, it is enough to get the tensor product of $\mathcal{O}(1)$ linear mixers with parameters $(\frac{1}{2}\log\log(nt), \tau, \alpha', \beta')$.

To get a circuit that fails with probability $e^{-\Omega(n)}$, we need a linear $(\log(tn), \tau, \alpha, \beta)$-mixer. Such a mixer exists, though we have no *effective* algorithm to construct it. But if we omit the condition that $\hat{S}$ can be received from S effectively then we can apply the same arguments as in Theorem 1 and get the following result:

Theorem 2 *For any circuit S of depth t with n processors there exists a circuit $\hat{S}$ of depth $\mathrm{poly}(t, n)$ with $\mathrm{poly}(t, n)$ processors that computes the same function for the encoded input so that if any processor at each step is corrupted with probability $\epsilon < 1/8$ then the result is correct with probability at least $(1 - e^{-\Omega(n)})$. The circuit $\hat{S}$ can be constructed from S in time $2^{\mathrm{poly}(t,n)}$.*

4 Computations resistant to an adversary

Now we consider the model where at each step an adversary chooses arbitrarily the fraction ϵ of all processors and corrupts them. We show that any computation circuit can made resistant to such an adversary with exponential blow-up of the memory.

Theorem 3 *For a small enough $\epsilon > 0$, for any circuit S of depth t with n processors there exists a circuit $\hat{S}$ of depth tn with $N = 2^{\mathcal{O}(n)}$processors such that $\hat{S}$ computes correctly the same function (for the encoded input) if at each step the fraction at most ϵ of all processors are corrupted.*

The circuit $\hat{S}$ can be constructed from S quasi-effectively: there exists a polynomial algorithm which gets S, the number of a processor $i \leq 2^{2n}$, and the number τ, and returns (1) the boolean function required to update the state of the i-th processor at the τ-th step, and (2) the list of $\mathcal{O}(1)$ processors that are queried by the processor number i at this step of computation. Note that the binary representation of i is $\mathcal{O}(n)$, so this algorithm runs in time $\mathrm{poly}(n, t)$.

Proof: We assume that in the original circuit at each step of computations only one processor can update its internal state (any circuit can be transformed to this form, for the price of n-time slow down and constant blow-up of the memory). Further we prove that a circuit S which satisfy this assumption can be simulated by a fault-tolerant $\hat{S}$ in real time.

Denote by $y^{(j)} = (y_1^{(j)}, \ldots, y_n^{(j)})$ the state of the memory of S at the j-th step of computation, $j = 0, \ldots, t$. Define an encoding function $E : \{0,1\}^n \to \{0,1\}^{2^{2n}}$ as follows:

$$E(x) = Had(x), \ldots, Had(x),$$

i.e., E is just the Hadamard code repeated 2^n times. The decoding function $D : \{0,1\}^{2^{2n}} \to \{0,1\}^n$ is defined as follows: to get $D(x)$ split x into 2^n blocks of length 2^n; decode each block using the Hadamard decoding; then for each position $i = 1, \ldots, n$ take the majority of the i-th bits in all 2^n results.

We shall define the computation process so that at each step j the memory of $\hat{S}$ contains a value $z^{(j)} = (z_1^{(j)}, \ldots, z_N^{(j)})$ which differers from $E(y^{(j)})$ in at most δN positions for $\delta = 1/2^{14}$. Note that this condition implies $D(z^{(t)}) = y^{(t)}$.

Let us split the internal states of the processors $(z_1^{(j)}, \ldots, z_N^{(j)})$ into 2^n blocks of size 2^n and denote them $b_1^{(j)}, \ldots, b_{2^n}^{(j)}$.

We shall employ an $(n, \tau, \delta_0/2, \delta_0)$-mixer Mix, where $\delta_0 = (\delta - \epsilon)/7$. Such a mixer exists for large enough $\tau = \tau(\delta_0)$. We don't need it to be a linear mixer, so we can employ an expander with appropriate parameters. There are known constructions of *effective* expanders with required parameters, e.g. [12].

We describe the transformation from $z^{(j)}$ to $z^{(j+1)}$ in four stages.

Stage 1. Fix $i \in \{1, \ldots, 2^n\}$. For each $\eta \in \{0,1\}^\tau$ get the block $b_{Mix(i,\eta)}^{(j)} = (u_1(\eta), \ldots, u_{2^k}(\eta))$ and compute the vector

$$w_i(\eta) = (u_{i\oplus 1}(\eta) - u_1(\eta), \ldots, u_{i\oplus 2^n}(\eta) - u_{2^n}(\eta))$$

(here for $i, s \in \{1, \dots, 2^n\}$ we denote by $i \oplus s$ the bitwise sum of n-bits binary representations of i and s). For each position $r = 1, \dots, 2^n$ get the majority of the r-th bits in all $w_i(\eta)$, $\eta \in \{0,1\}^\tau$. Denote by $c_i^{(j)}$ the obtained result (which is a vector in $\{0,1\}^{2^n}$).

Let us call a block $z_i^{(j)}$ *harmed* if it differs from $Had(y^{(j)})$ in more than $\sqrt{\delta}2^n$ positions. We assumed that $z^{(j)}$ differs from $E(y^{(j)})$ in at most δN position. Hence, there are at most $\sqrt{\delta}2^n$ harmed locks $z_i^{(j)}$.

From Claim 2 it follows that for the fraction at least $(1 - \delta_0)$ of all indexes i $(1 - 1/64) \cdot 2^\tau$ of blocks $b^{(j)}_{Mix(i,\eta)}$ ($\eta \in \{0,1\}^\tau$) are not harmed. Note that if for some i at least the fraction $(1 - 1/64)$ of blocks $b^{(j)}_{Mix(i,\eta)}$ are not harmed then all but $4(1/64 + \sqrt{\delta}) \cdot 2^n = 2^n/8$ bits in the resulted block $c_i^{(j)}$ are equal to $y_i^{(j)}$.

Stage 2. Fix $i \in \{1, \dots, 2^n\}$. Let the block $c_i^{(j)}$ consists of bits $(v_1, \dots, v_{2^k})$. For each $r = 1, \dots, 2^n$ calculate the majority in the sequence

$$v_{Mix(r,\eta)},\ \eta \in \{0,1\}^\tau,$$

Denote the result v'_r. Set $d_i^{(j)} = (v'_1, \dots, v'_{2^k})$.

From Claim 1 it follows that if a block $c_i^{(j)}$ contains at least $7/8 \cdot 2^n$ bits equal to $y_i^{(j)}$ then in the corresponding block $d_i^{(j)}$ at least $(1 - \delta_0)2^n$ bits are equal to $y_i^{(j)}$. Of course, we guarantee nothing for a block $d_i^{(j)}$ if in the corresponding $c_i^{(j)}$ more than $1/8$ of all bits differ from $y_i^{(j)}$.

Stage 3. This stage is trivial: we just make a permutation of bits in $(d_1^{(j)}, \dots, d_{2^n}^{(j)})$. For each l, m we get the l-th bit from $c_m^{(j)}$ and put it to the m-th position in the l-th block. Denote the result $(f_1^{(j)}, \dots, f_{2^n}^{(j)})$.

Stage 4. Assume that at the j-th step of the computation in the original circuit S the bit y_{i_0} is modified:

$$y_{i_0}^{(j+1)} = F(y_{i_1}^{(j+1)}, y_{i_2}^{(j+1)}),$$

where F is some boolean function. (W.l.o.g. we may assume that that the boolean function F has only two arguments.)

Fix $i \in \{1, \dots, 2^n\}$ and denote $f_i^{(j)} = (u_1, \dots, u_{2^k})$. For each $q = 1, \dots, 2^n$ calculate

$$\begin{aligned} \tilde{y}_{i_0} &= u_{q \oplus i_0} - u_q \\ \tilde{y}_{i_1} &= u_{q \oplus i_1} - u_q \\ \tilde{y}_{i_2} &= u_{q \oplus i_2} - u_q \end{aligned}$$

Then set $\xi_q = F(\tilde{y}_{i_1}, \tilde{y}_{i_2}) - \tilde{y}_{i_0}$, and calculate $\triangle_q = Had(0, \dots, 0, \xi_q, 0, \dots, 0)$ (the value ξ_q is placed at the i_0-th position). Further, get the q-th position of $\triangle_q$ and add it to the value u_q. Denote the resulted block $z^{(j+1)}$.

Note that if $d^{(j)}$ differs from $Had(y^{(j)})$ in $\gamma 2^n$ positions (for some fraction $\gamma \in (0,1)$) then $z^{(j+1)}$ differs from $Had(y^{(j)})$ in at most $6\gamma 2^n$ positions. Hence, the whole vector $z^{(j+1)}$ differs from $E(y^{(j+1)})$ in at most

$$(\delta_0 + 6\delta_0 + \epsilon)2^n$$

positions. Note that $7\delta_0 + \epsilon < \delta$, and we are done.

In our construction each bit $z_i^{(j)}$ depends on $\mathcal{O}(1)$ bit from $z^{(j)}$. Thus, we have well defined the transition rule $z^{(j)} \mapsto z^{(j+1)}$.

□

5 Conclusion

We proved that any parallel computation fulfilled on memory n in time t can be simulated by a reliable circuit with memory $\mathcal{O}(n \log(nt)/\log\log(nt))$ in time $\mathcal{O}(t \log\log(tn))$. Such a reliable circuit returns a correct result with high probability even if it is based on faulty elements (i.e., each element faults at any step with some small probability ϵ, and faults of different elements are independent). Our construction employs encoding based on the Hadamard code. Actually similar arguments can be applied for a code base on any other *linear locally decodable code*. For example, we can use the Reed-Muller code instead of the Hadamard code; then essentially the same construction provides a bit stronger bound: any computation which was done on non-faulty circuit with memory n in time t, can be simulated on faulty elements with memory $\mathcal{O}(n \log(nt)/\log^C \log(nt))$, where the constant C can be made arbitrarily large. By this method, we cannot obtain much better bounds (like $\mathcal{O}(n)$), because for any linear locally decodable error correcting code the codeword length must be exponential in the block length [13]. Thus, the main question, which remains open, is if reliable polynomial computations can be fulfilled on memory $\mathcal{O}(n)$.

Our second result, which concerns computations resistant to an adversary who can corrupt at each step some fraction of memory cells, seems quite weak. We presented a construction with exponential blow-up of the memory. Again, the proved bound cannot be essentially improved with our method, because it is based on a linear locally decodable error correcting code. Remind that if we want just to *store* some information (without computations), this can be done with a constant blow-up of the memory, even if at each step an adversary corrupts some fraction of memory cells [2]. To our knowledge, there are no results achieving polynomial blow-up of the memory for circuits computing an arbitrary function and tolerating a constant fraction of processors being corrupted at every step. In [9] this problem was solved only for a special class of boolean functions. Thus, the second important open question is if *any computation* can be made resistant to an adversary, with linear or at least polynomial increasing of the memory.

Another interesting question is if a linear (m, τ, α, β)-mixer can be effectively constructed in time $\mathrm{poly}(2^m)$. If such a construction exists, the proof of Theorem 2 can be made effective.

References

[1] J. von Neuman. *Probabilistic logics and the synthesis of reliable organisms from unreliable components.* In C. Shannon and J. McCarthy, editors, Automata Studies. Princeton University Press, 1956.

[2] A.V. Kuznetsov. *Information storage in a memory assembled from unreliable components*. Problems of Information Transmission. vol. 9(3), 1973, pp. 254–264.

[3] R. L. Dobrushin, S. L. Ortyukov. *Upper bound for the redundancy of self-correcting arrangement of unreliable functional elements*. Problems for Information Transmission, vol. 13(1), 1977, pp. 203-218.

[4] R. L. Dobrushin, S. L. Ortyukov. *Lower bound on the redundancy of self-correcting arrangement of unreliable functional elements*. Problems for Information Transmission, vol. 13(1), 1977, pp. 201-208.

[5] N. Pippenger. *On Networks of Noisy gates*. In Proc. of the 26-th IEEE FOCS Symposium, 1985, pp. 30–38.

[6] N. Pippenger, G.D. Stamoulis, J.N. Tsitsikilis. *On a lower bound on for the redundancy of reliable networks with noisy gates*. IEEE Trans. Inform. Theory, vol. 37(3), 1991, pp. 639-643.

[7] R. Reischuk, B. Schmeltz. *Reliable computation with noisy circuits and decision trees – a general $n \log n$ lower bound*. In Proc. of the 32-th IEEE FOCS Symposium, 1991, pp. 602-611.

[8] P. Gács, A. Gál. *Lower Bounds for the Complexity of Reliable Boolean Circuits with Noisy Gates*. IEEE Transactions Information Theory. vol. 40, 1994, pp. 579–583.

[9] A. Gál, M. Szegedy. *Fault Tolerant Circuits and Probabilistically Checkable proofs*. In Proc. of 10th Annual Structure in Complexity Theory Conference, 1995, pp. 65–73.

[10] D. A. Spielman. *Highly fault-Tolerant parallel Computation*. Proc. of the 37-th IEEE FOCS Symposium, 1996, pp. 154-163.

[11] A. Goldreich, A. Wigderson. *Tiny Families of Functions with Random Properties: a Quality-Size Trade-off for Hashing*. Random Struct. Algorithms 11(4), 1997, pp. 315-343.

[12] O. Reingold, S. Vadhan, and A. Wigderson. *Entropy waves, the zig-zag product, and new constant degree expanders*. Annals of Mathematics, 155(1), 2002, pp. 157–187.

[13] O. Goldreich, H.J. Karloff, L.J. Schulman, L. Trevisan. *Lower Bounds for Linear Locally Decodable Codes and Private Information Retrieval.* IEEE Conference on Computational Complexity, 2002, 175-183.

[14] P. Gacs. *Lectures on Reliable Cellular Automata.*

http://www.cs.bu.edu/ gacs/papers/iv-eng.pdf

6 Appendix

Proof of Lemma 1: Chose a random mapping $F : \{0,1\}^m \times \{0,1\}^\tau \to \{0,1\}^m$ and check that with high probability F is a linear (m, τ, α, β)-mixer. We assume that for all $x \in \{0,1\}^m$ and $u \in \{0,1\}^\tau$ the value $F(x,u) \in \{0,1\}^m$ is chosen at random (uniformly), and values for different points are independent. Let us bound the probability that a random F is *not* a linear mixer.

Fix some $A, B \subset \{0,1\}^m$. For each pair $(x,u) \in A \times \{0,1\}^\tau$ the value $F(x,u)$ belongs to B with probability $|A|/2^m$. From the Chernoff bound we get

$$\mathrm{Prob}[\|\{(x,u) \ : \ x \in A, F(x,u) \in B\}| - \frac{2^\tau |A| \cdot |B|}{2^m}\| > \beta 2^\tau |A|] < 2e^{-\beta^2 |A| 2^\tau}$$

If $|A| \geq \alpha 2^m$, we get

$$\mathrm{Prob}[\|\{(x,u) \ : \ x \in A, F(x,u) \in B\}| - \frac{2^\tau |A| \cdot |B|}{2^m}\| > \beta 2^\tau |A|] < 2 \cdot e^{-\alpha\beta^2 2^{m+\tau}}$$

As the values of $F(x,u)$ are chosen at random independently, the values of $G(x,u)$ are also random and independent. Hence, for the function G the same bound holds:

$$\mathrm{Prob}[\|\|\{(x,u) \ : \ x \in A, G(x,u) \in B\}| - \frac{2^\tau |A| \cdot |B|}{2^m}\| > \beta 2^\tau |A|] < 2e^{-\alpha\beta^2 2^{m+\tau}}$$

Now sum up the probabilities above for all A, B. The total number of subsets in $\{0,1\}^m$ is 2^{2^m}. Thus, a randomly chosen F is not a linear mixer with probability less than

$$2 \cdot (2^{2^m})^2 4e^{-2\alpha\beta^2 2^{m+\tau}},$$

which is less than 1 for large enough τ. □

Proof of Claim 1: Let us fix some B of size at most $2^m/8$. Assume the claim is false: suppose there exist $\delta 2^m$ different elements $x \in \{0,1\}^m$ such that for a randomly chosen $u \in \{0,1\}^\tau$

$$\mathrm{Prob}_u[F(x,u) \in B] \geq 0.25$$

Let A be a set that contains exactly $\delta 2^m$ points x as above. As F is a mixer, we have

$$|\{(x,u) \ : \ x \in A, F(x,u) \in B\}| < \frac{\delta 2^\tau 2^m \cdot |B|}{2^m} + \delta^2 2^\tau 2^m < (\delta/4) \cdot 2^{m+\tau}$$

On the other hand, we assumed that for each $x \in A$, for at least $0.25 \cdot 2^\tau$ points $u \in \{0,1\}^\tau$ the value $F(x,u)$ is in B. Hence,

$$|\{(x,u) \ : \ x \in A, F(x,u) \in B\}| \geq 0.25 \cdot \delta 2^{m+\tau},$$

and we get a contradiction. □

Proof of Lemma 2: Denote $X_1 = \{0,1\}^{m_1}$ and $X_2 = \{0,1\}^{m_2}$. Let us fix some $A, B \subset X_1 \times X_2$. Our aim is to evaluate the number of pairs (x,u) such that $x \in A$, $u \in \{0,1\}^{\tau_1+\tau_2}$, and $F_1 \otimes F_2(x,u) \in B$.

Let $\delta = (\alpha_2)^2$. We use the following notation:

- for any $u \in X_1$ let $A_u = \{u \otimes v | v \in X_2,\ u \otimes v \in A\}$;
- for any $v \in X_2$ let $B_v = \{u \otimes v | u \in X_1,\ u \otimes v \in B\}$;
- $A_i = \{x \in X_1 | i\delta \le |A_x|/2^{m_2} < (i+1)\delta\}$;
- $B_i = \{x \in X_1 | i\delta \le |B_x|/2^{m_2} < (i+1)\delta\}$;
- $\hat{A}_i = \{u \otimes v \in A | u \in A_i\} = \bigcup\limits_{x \in A_i} A_x, i = 0, \dots, \lceil 1/\delta \rceil$;
- $\hat{B}_i = \{u \otimes v \in B | v \in B_i\} = \bigcup\limits_{y \in B_i} B_y, i = 0, \dots, \lceil 1/\delta \rceil$.

Step 1. First, we count the number of pairs $(x, u) \in A \times \{0,1\}^{\tau_1+\tau_2}$ such that $x \in A$ and $F(x, u) \in B$. To evaluate the number of such pairs, we calculate for each i, j the number of pairs (x, u) such that $x \in \hat{A}_i$ and $F(x, u) \in \hat{B}_j$, and then sum up these values.

The first case: $i, j \ge 1/\sqrt{\delta}$, $|A_i| \ge \alpha_1 2^{m_1}$.

Fix some integer i, j as above. At first, count the number of pairs $(x, u) \in X_1 \times \{0,1\}^{\tau_1}$ such that $x \in A_i$ and $F_1(x, u) \in B_j$.

As F_1 is a mixer,

$$\| |\{(x,u) \in A_i \times \{0,1\}^{\tau_1} | F_1(x,u) \in B_j\}| - \frac{2^{\tau_1}|A_i| \cdot |B_j|}{2^{m_1}} \| \le \beta_1 2^{\tau_1} |A_i|.$$

Further, for any $x \in A_i$, $y \in B_j$ we have

$$\| |A_x| - i\delta 2^{m_2} \| < \delta 2^{m_2}$$

and

$$\| |A_y| - j\delta 2^{m_2} \| < \delta 2^{m_2}$$

Note that the condition $i > 1/\sqrt{\delta}$ implies $|A_x| \ge \alpha_2 2^{m_2}$. As F_2 is a mixer, we have

$$\begin{aligned} &\| |\{(y,v) \in A_x \times \{0,1\}^{\tau_2} | F_2(y,v) \in B_y\}| - ij2^{m_2+\tau_2}\delta^2 \| \\ &\le \beta_2 (i+1)\delta 2^{m_2+\tau_2} + \mathcal{O}(1/i + 1/j) \cdot ij2^{m_2+\tau_2}\delta^2 \\ &= \beta_2 (i+1)\delta 2^{m_2+\tau_2} + \mathcal{O}(\sqrt{\delta}) \cdot ij2^{m_2+\tau_2}\delta^2 \end{aligned}$$

Thus, the number of pairs (x, u) such that $x \in \hat{A}_i$, $u \in \{0,1\}^{\tau_1}$, and $F(x, u) \in \hat{B}_j$ is equal to

$$\frac{|A_i|}{2^{m_1}} \cdot \frac{|B_j|}{2^{m_1}} \cdot ij2^{m_1+m_2+\tau_1+\tau_2}\delta^2(1 + \mathcal{O}(\sqrt{\delta})) + \mathcal{O}(\beta_1 + \beta_2) \cdot 2^{m_1+m_2+\tau_1+\tau_2}$$

The second case: $|A_i| < \alpha_1 2^{m_1}$.

For every i such that $|A_i| < \alpha_1 2^{m_1}$

$$\| |\{(x,u) \in A_i \times \{0,1\}^{\tau_1} | F_1(x,u) \in B\}| \le \alpha_1 2^{m_1+m_2+\tau_1+\tau_2}$$

The index i ranges over $1, \ldots, \lceil 1/\delta \rceil$. Hence,

$$\sum_{i,j:\ A_i < \alpha_1 2^{m_1}} |\{(x,u) \in \hat{A}_i \times \{0,1\}^{\tau_1+\tau_2} | F_1(x,u) \in \hat{B}_j\}| \leq \mathcal{O}(\alpha_1/\delta) \cdot 2^{m_1+m_2+\tau_1+\tau_2}.$$

The third case: $i < 1/\sqrt{\delta}$.
If $i < 1/\sqrt{\delta}$ and $x \in A_i$ we have $|A_x| \leq \sqrt{\delta} 2^{m_2}$. Hence,

$$\sum_{0 \leq i < 1/\sqrt{\delta}, j \geq 0} |\{(x,u) \in \hat{A}_i \times \{0,1\}^{\tau_1+\tau_2} | F(x,u) \in \hat{B}_j\}| \leq |X_1| \cdot \sqrt{\delta} 2^{m_2+\tau_1+\tau_2},$$

which less than $\sqrt{\delta} 2^{m_1+m_2+\tau_1+\tau_2}$

The fourth case: $j < 1/\sqrt{\delta}$ and $i > 1/\sqrt{\delta}$.
For i, j under those conditions and any $x \in A_i$ and $y \in B_j$ there are at most

$$\sqrt{\delta}|A_x| 2^{\tau_2} + \beta_2 |A_x| 2^{\tau_2} \leq (\sqrt{\delta} + \beta_2) 2^{m_2+\tau_2}$$

pairs $(x,u) \in A_x \times \{0,1\}^{\tau_2}$ such that $F_2(x,u) \in B_y$. Thus,

$$\sum_{0 \leq j < 1/\sqrt{\delta},\ i > 1/\sqrt{\delta}} |\{(x,u) \in \hat{A}_i \times \{0,1\}^{\tau_1+\tau_2} | F_1(x,u) \in \hat{B}_j\}| \leq (\beta_2 + \sqrt{\delta}) \cdot 2^{m_1+m_2+\tau_1+\tau_2}$$

Sum up the four cases above:

$$\begin{array}{l} |\{(x,u) \in A \times \{0,1\}^{\tau_1+\tau_2} | F_1(x,u) \in B\}| = \\ 2^{\tau_1+\tau_2+m_1+m_2} \cdot (1 + \mathcal{O}(\sqrt{\delta})) \cdot \sum_{i,j > 1/\sqrt{\delta} \text{ and } |A_i| \geq \alpha_1 2^{m_1}} \frac{|A_i|}{2^{m_1}} \frac{|B_j|}{2^{m_2}} ij\delta^2 \\ +2^{\tau_1+\tau_2+m_1+m_2} \cdot (\mathcal{O}((\alpha_1 + \beta_1 + \beta_2)/delta + \sqrt{\delta}))) \end{array}$$

Step 2. Count the product $|A| \cdot |B|$:

$$\begin{array}{l} |A| \cdot |B| = \sum_{i,j} |\hat{A}_i||\hat{B}_j| = \\ \sum_{i,j > 1/\sqrt{\delta}} |\hat{A}_i||\hat{B}_j| + \mathcal{O}(\sqrt{\delta}) \cdot 2^{2(m_1+m_2)} = \\ 2^{2(m_1+m_2)} \cdot \sum_{i,j > 1/\sqrt{\delta}} \frac{|A_i|}{2^{m_1}} \frac{|B_j|}{2^{m_1}} ij + \mathcal{O}(\sqrt{\delta}) \cdot 2^{2(m_1+m_2)} = \\ 2^{2(m_1+m_2)} \cdot \sum_{i,j > 1/\sqrt{\delta}, |A_i| \geq \alpha_1 2^{m_1}} \frac{|A_i|}{2^{m_1}} \frac{|B_j|}{2^{m_1}} ij + 2^{2(m_1+m_2)} \cdot \mathcal{O}(\alpha_1/\delta + \sqrt{\delta}) \end{array}$$

Step 3. From the bounds obtained in step 1 and step 2 we get

$$\begin{array}{rcl} |\{(x,u) \in A \times \{0,1\}^{\tau_1+\tau_2} | F_1(x,u) \in B\}| & = & |A||B| \cdot 2^{(\tau_1+\tau_2)-(m_1+m_2)} \\ & + & \mathcal{O}((\alpha_1 + \beta_1 + \beta_2)/\delta + \sqrt{\delta})) \cdot 2^{m_1+m_2+\tau_1+\tau_2} \end{array}$$

Obviously, if $|A| \geq \sqrt{\alpha_2} 2^{m_1+m+2+\tau_1+\tau_2}$, we have

$$\mathcal{O}((\alpha_1+\beta_1+\beta_2)/\delta+\sqrt{\delta})) \cdot 2^{m_1+m_2+\tau_1+\tau_2} = \mathcal{O}(((\alpha_1+\beta_1+\beta_2)/\delta+\sqrt{\delta}))/\alpha_2) \cdot |A|$$

Recall that $\delta = (\alpha_2)^2$, and we get that $F_1 \otimes F_2$ is a $(m_1+m_2, \tau_1+\tau_2, \sqrt{\alpha_2}, \beta)$-mixer for $\beta = \mathcal{O}(\frac{\alpha_1+\beta_1+\beta_2}{\alpha_2\sqrt{\alpha_2}} + \sqrt{\alpha_2})$. $\square$