Divergent Multi-Version Execution: Per-Instruction Full-State Hashing with Address-Space Decorrelation

Petro Baran
Independent Researcher
zoshytlogic@gmail.com

Uzhgorod, February 2026

Abstract

Redundancy-based fault tolerance techniques typically execute identical binaries with identical address layouts, leaving systems vulnerable to correlated control-flow faults.

This paper introduces **Divergent Multi-Version Execution (DME)**, which combines **address-space decorrelation** with **per-instruction full-state hashing**. Identical instruction bytes are preserved across replicas, while basic blocks are mapped to distinct addresses. After each instruction, replicas compute incremental state hashes and perform synchronous comparison.

We formally define the fault model, explicitly state physical independence assumptions, and derive probabilistic bounds on undetected fault likelihood. We further analyze scalability, worst-case execution overhead, and trusted computing base (TCB) constraints. A Cortex M0 - M4 prototypes demonstrates single-instruction detection latency with bounded overhead.

1 Introduction

Safety-critical systems rely on redundancy. However, identical address layouts across replicas allow deterministic control-flow faults to propagate identically, resulting in silent data corruption.

DME addresses this through:

- Address-space decorrelation
- Per-instruction state hashing
- Immediate cross-replica comparison

Unlike N-version programming, DME preserves identical semantics while enforcing structural diversity at the address level.

Optimized for 32-bit microcontrollers. DME is specifically architected for the dominant class of embedded safety-critical platforms—32-bit microcontrollers (Cortex-M, RISC-V, AVR32). The runtime system managing program replicas consumes only **4 KB of RAM**, with each additional divergent replica requiring just **512 bytes** of additional memory for its private **CPU context** (registers, stack pointer, program counter) and hash state. Program memory may be either shared between replicas or fully duplicated depending on the deployment scenario—in virtualised environments, each replica can have its own private program memory image. Replica-private data memory—such as stack and heap—must be allocated separately and scales linearly with the number of replicas.

This exceptional memory efficiency—an order of magnitude lower than conventional redundancy schemes—brings single-cycle fault detection to the broad spectrum of automotive, medical, and industrial systems built around 32-bit MCUs, where traditional redundancy approaches are often infeasible due to cost and resource constraints.

2 System Model and Assumptions

2.1 Execution Model

We consider $N \geq 2$ deterministic replicas executing identical instruction streams under fault-free conditions.

2.2 Physical Independence Assumptions

Assumption 2.1 (Independent Fault Domains)**.** *Replica-specific architectural states (PC, registers, and replica-local memory) are subject to independent transient or permanent faults.*

Assumption 2.2 (Replica-Private Address Spaces)**.** *Each replica owns a logically disjoint code and data address space. Stacks, heaps, and global variables are allocated in replica-private memory regions such that no two replicas share virtual or physical data addresses. Consequently, a single-address memory fault cannot simultaneously corrupt the same logical object across replicas.*

Assumption 2.3 (Shared Resource Constraints)**.** *Shared clocks, buses, or power domains may introduce correlated faults; such effects are outside the strict logical decorrelation guarantee.*

This explicitly bounds the model and prevents overclaiming coverage.

3 Formal Fault Model

Definition 3.1 (Fault)**.** *A fault is a perturbation f affecting architectural state components:*
$f \in \{PC, ALU, RegFile, Memory, ControlFlow, HashUnit\}$

Fault classes:

- Transient (bit flip, SEU)
- Permanent (stuck-at)
- Intermittent

We distinguish:

- **Replica-local faults**
- **Shared physical faults**

DME guarantees apply primarily to replica-local faults.

4 Architecture Overview

DME executes N replicas of an identical binary while enforcing address-space decorrelation. Although instruction bytes remain identical, basic blocks are mapped to distinct virtual addresses across replicas, as illustrated in Figure 1.

4.1 Address-Space Decorrelation

Let program CFG be $G = (V, E)$.

Mapping:

$$\phi_n : V \to \mathbb{N}$$

such that:

$$\forall n \neq m, \forall v : \phi_n(v) \neq \phi_m(v)$$

Instruction bytes remain identical.

4.2 Hash Evolution

$$H_n^{t+1} = F(H_n^t, \psi_m(exec_n^t))$$

Depth parameter:

$$m \in \{1, 2, 3\}$$

4.3 Hash Function Requirements

Required properties of F:

4.4 Hash Evolution

$$H_n^{t+1} = F(H_n^t, \psi_m(exec_n^t))$$

Depth parameter:

$$m \in \{1, 2, 3\}$$

The hash evolution function F is implemented as a per-instruction update. For each replica n, the runtime executes a fetch-execute-hash-compare cycle. A simplified implementation of this core loop is shown in Listing 1, illustrating how the incremental hash is updated based on both the instruction word and, optionally, the result of its execution to achieve greater fault coverage depth.

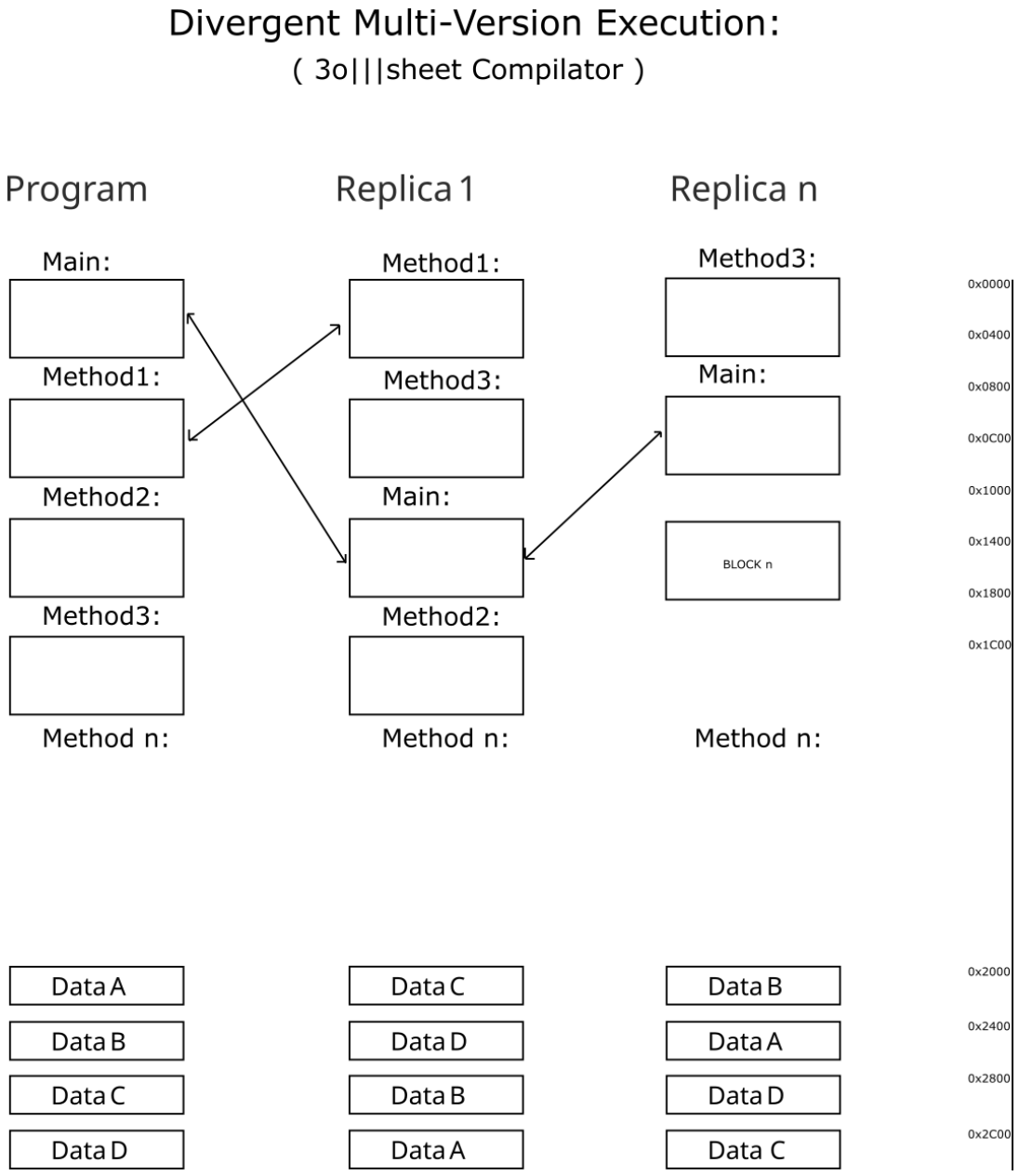

Figure 1: Address-space decorrelation produced by the 3o|||sheet Compilator. Identical program blocks are placed at distinct virtual addresses across replicas while preserving instruction bytes and control-flow semantics.

Listing 1: Per-replica execution loop with incremental state hashing and synchronous comparison

```
while(1) {
    instruction = memoryPn[PC];           // Fetch
    hashPn = updateHASH(hashPn, instruction); // Hash (m=1)

    if(Depth_parameter) {                  // m >= 2
        exe(instruction);                  // Execute
        hashPn = updateHASH(hashPn,        // Hash result
            instruction.ResultRegister);
    }

    if(hashPn != getPeerHash(n)) {          // Compare
        Arbiter.handleMismatch(n);           // Arbiter
    }
}
```

The virtualised test environment does not imply a limitation—DME can be implemented natively in assembly on physical hardware.

- Deterministic
- Collision probability $\ll$ architectural fault rate
- Constant-time execution
- Bounded state size

Cryptographic strength is sufficient but not strictly required; integrity collision resistance is primary.

5 Trusted Computing Base

The TCB includes:

- Hash function implementation
- Comparison logic
- Arbiter
- Compiler mapping correctness

To avoid single-point failure:

- Distributed comparison is preferable to master-only comparison
- Byzantine majority voting may be used for $N \geq 3$

6 Diversity Divergence Theorem

Figure 2 illustrates fault-free execution under address-space decorrelation. Although each replica executes the same instruction bytes, basic blocks reside at different virtual addresses. Consequently, identical logical execution produces identical architectural state transitions and equal hash evolution across replicas.

To reason formally about detection guarantees, we define the complete replica state and the probabilistic behavior of the hash function.

Definition 6.1 (Architectural State)**.** *Let S_n^t denote the complete architectural state of replica n at time t, including program counter, register file, and all replica-private memory (stack, heap, and globals).*

Definition 6.2 (Hash State)**.** *Let H_n^t denote the incremental hash value maintained by replica n after executing t instructions.*

Assumption 6.1 (Hash Collision Bound)**.** *The hash evolution function F behaves as a uniform mapping over its state space with per-comparison collision probability at most $\epsilon \ll 1$. Hash updates execute in constant time.*

Intuitively, fault-free deterministic execution implies identical states across replicas, which implies identical hashes. Conversely, a state divergence is detected unless a hash collision occurs. Address-space decorrelation further reduces the probability that multiple replicas follow the same incorrect control-flow path after a fault.

Theorem 6.1 (Conditional Diversity Divergence)**.** *Under Assumptions 1–4 and the replica-local fault model:*

(1) Fault-free consistency. *For deterministic fault-free execution,*

$$\forall t,\ \forall i, j: \quad S_i^t = S_j^t \ \Rightarrow\ H_i^t = H_j^t.$$

(2) State-divergence detection. *If a replica-local fault causes $S_i^t \neq S_j^t$, then*

$$\Pr(H_i^t = H_j^t) \leq \epsilon,$$

i.e., divergence is detected except with hash-collision probability.

Figure 2: Fault-free execution under address-space decorrelation

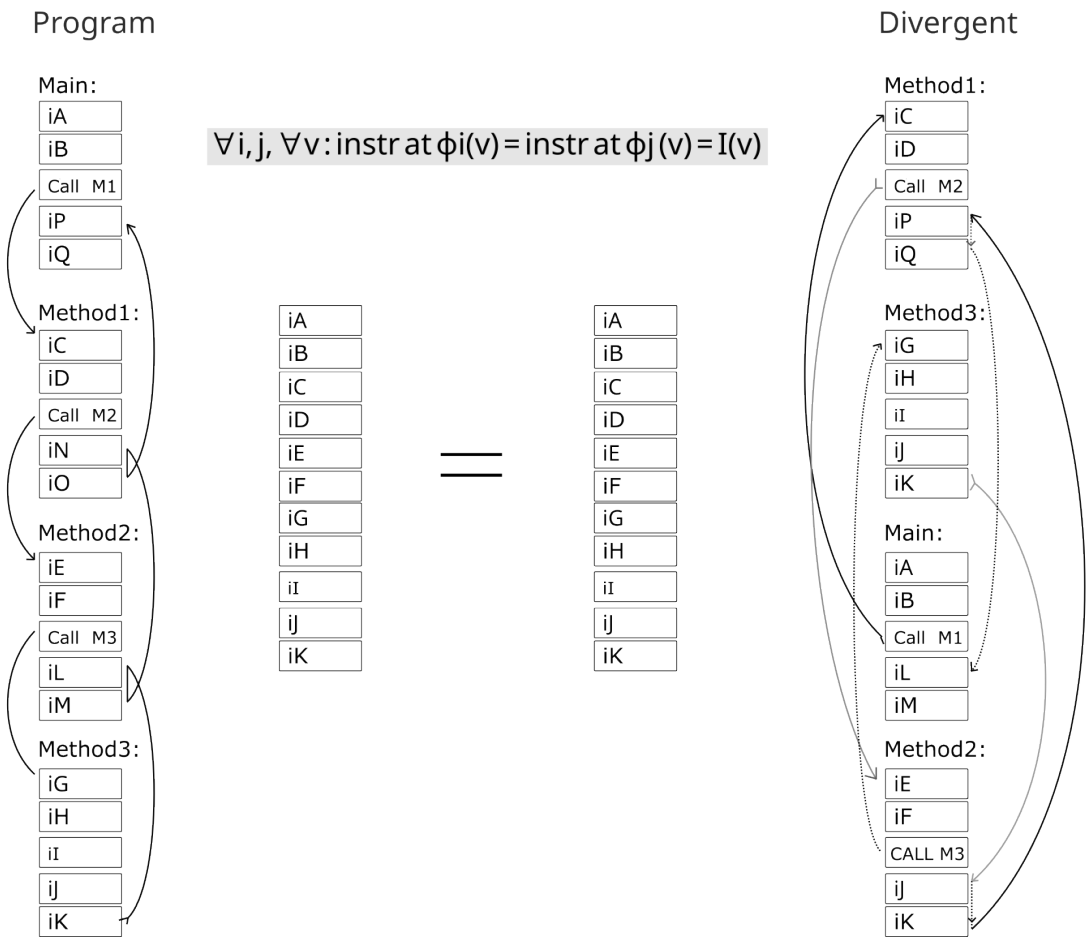

Figure 2: Fault-free execution under address-space decorrelation. Identical instruction bytes are fetched in each replica, while addresses differ. Architectural states and hashes remain equal.

Figure 3: Fault-induced divergence under address-space decorrelation

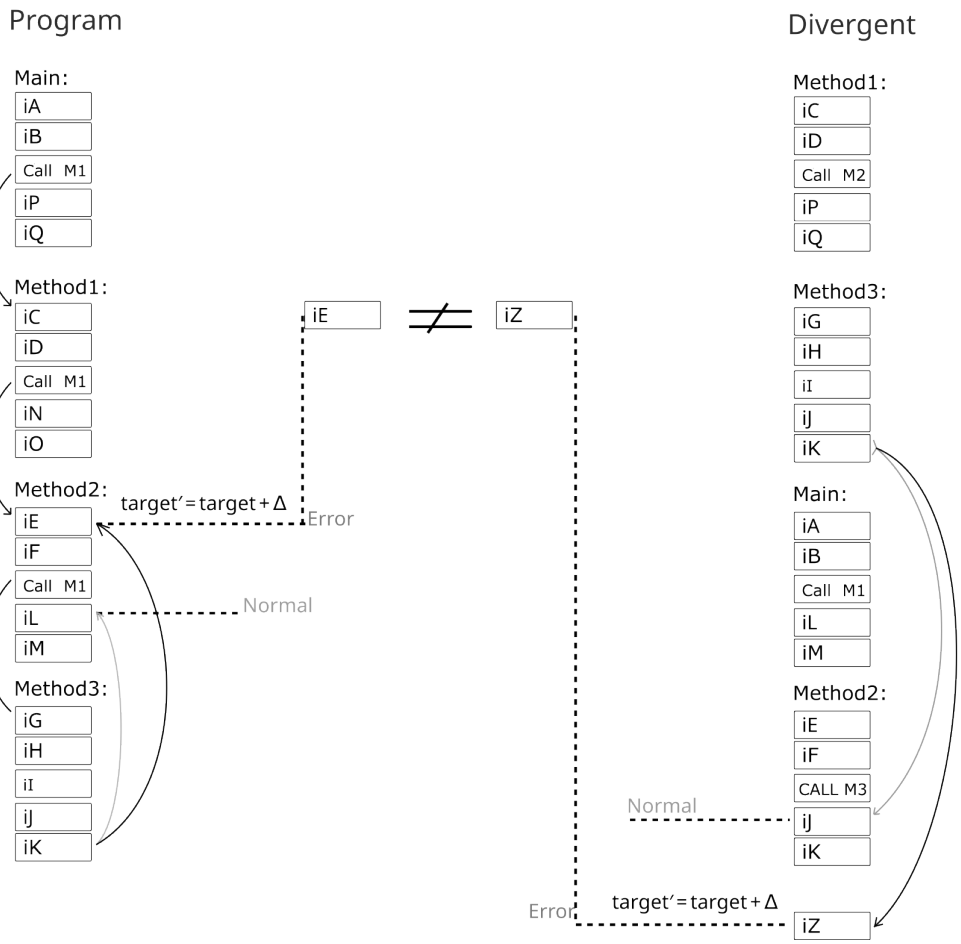

Figure 3: Fault-induced divergence under address-space decorrelation. A corrupted program counter resolves to different basic blocks across replicas, producing distinct state transitions and immediate hash mismatch.

(3) Control-flow decorrelation bound. *For faults that redirect execution to an unintended basic block, let ρ denote code density, L the number of blocks, and R the address-space range. Assuming independent uniform mappings ϕ_n, the probability that all N replicas resolve to the same incorrect logical block is bounded by*

$$P \leq (\rho L/R)^{N-1}.$$

Proof sketch. Property (1) follows from deterministic execution and identical instruction bytes. Property (2) follows from the collision bound of F. Property (3) follows from independence of address mappings: each additional replica must independently alias to the same erroneous logical block, yielding a multiplicative probability. □

The theorem formalizes that DME converts replica-local faults into detectable hash divergence with probability approaching one, while address-space decorrelation exponentially suppresses correlated control-flow aliasing as the number of replicas increases.

7 Physical Correlation and Environmental Limits

While DME provides probabilistic detection guarantees under the independence assumptions stated in Section 2 (see Theorem 6.1), it operates strictly at the level of the instruction set architecture (ISA) and the processor's architectural state. The scheme assumes that the underlying hardware platform is functionally correct, apart from the modeled transient or permanent faults affecting individual replicas.

Consequently, DME does **not** protect against failure modes that uniformly affect all replicas due to shared physical resources or design flaws. These explicitly excluded classes include:

- **Clock and power distribution failures:** Global clock generator faults, frequency drift, or power supply brownouts that simultaneously disrupt all processing elements.
- **Systematic design errors:** Common-mode hardware bugs in the microarchitecture (e.g., flawed divider logic) or in the toolchain (e.g., compiler miscompilation affecting all generated binaries identically).
- **Environmental extremes:** Electromagnetic interference (EMI) or temperature variations that exceed design specifications and cause correlated behavior across replicated cores on the same die.

In such scenarios, all replicas experience the same physical disturbance, and the logical decorrelation of addresses does not yield divergent execution. These failure modes must be addressed by orthogonal techniques, such as physical separation or diverse hardware design.

7.1 Deployment Scenarios and Hardware Diversity

DME is designed to be agnostic to the underlying hardware organization and can be deployed across a range of platforms, offering increasing resilience with greater physical independence:

- **Logical partitioning on a single core:** Replicas are time-multiplexed on one processor (as in the prototype). This offers minimal physical separation but still protects against many transient faults.
- **Homogeneous multi-core:** Replicas execute on identical, but physically distinct, cores on the same chip. This leverages core-level independence against many local faults (e.g., a single-core power glitch).
- **Heterogeneous multi-core:** Replicas run on cores with different microarchitectures but a compatible ISA. This adds protection against certain classes of common-mode microarchitectural failures.
- **Spatially distributed nodes:** Replicas are placed on physically separate chips with independent power and clock domains. This maximizes fault-domain separation and mitigates the environmental limits discussed above.

7.2 Orthogonality and Composition

It is crucial to position DME as an **orthogonal** reliability layer. It does not compete with, but rather complements, lower-level hardware redundancy techniques. The strongest resilience is achieved by composing DME with:

- Spatial separation (distributed nodes),
- Power- and clock-domain separation (isolated voltage islands),
- Hardware-level diversity (heterogeneous cores).

By combining architectural decorrelation (DME) with physical decorrelation, the system can provide a defense-in-depth against both random architectural faults and correlated environmental upsets.

8 Scalability Analysis

Memory cost:

$$M_{total} = N \cdot (M_{code} + M_{data})$$

Communication cost per cycle:

$$C = (N-1) \cdot |H|$$

Detection probability improves exponentially in N, while memory scales linearly.

9 Worst-Case Execution Time (WCET)

Hash update adds deterministic latency:

$$WCET_{DME} = WCET_{base} + WCET_{hash}$$

Since hashing occurs per instruction and is constant-time, real-time bounds remain analyzable.

Jitter remains bounded under deterministic scheduling.

10 Experimental Evaluation

We categorize injections by:

- PC corruption
- ALU bit flips
- Memory corruption
- Control-flow redirection

Detection latency distribution and false-positive rate must be reported per category.

11 Comparison with Related Work

Explicit comparison should include:

- Hardware lockstep
- TMR (Triple Modular Redundancy)
- N-version programming
- CFI (Control-Flow Integrity)
- EDDI / SWIFT (software-based fault tolerance)

DME uniquely combines identical semantics with structural address diversity.

12 Limitations

- Linear memory growth in N
- Overhead proportional to hashing depth
- Dependence on compiler correctness
- No guarantee against fully correlated physical faults

13 Conclusion

DME reduces the probability of undetected replica-local faults via structural decorrelation and per-instruction state comparison. Guarantees are conditional on explicitly stated independence assumptions. The proposed approach offers a practical path toward enhanced reliability in embedded safety-critical systems with bounded overhead and formal guarantees.

[1] 3o|||sheet Compilator https://zoshytlogic.github.io/

[2] Petro Baran: 10.5281/zenodo.18733852