The Vectorized Index Buy Order Execution Algorithm: A Formal Analysis of On-Chain Vector Processing

Sonia Kolasinska

November 26, 2025

Abstract

The execution of financial primitives involving baskets of underlying assets poses a significant computational challenge in constrained smart contract environments. This paper formalizes a novel, high-dimensional algorithm, implemented on a VM^2 environment, which is a Virtual Machine deployed as a smart contract (Stylus/WASM). This VM^2 is optimized for vector processing using its **Vector Intermediate Language (VIL)**, designed to execute complex index order primitives (Buy, Sell, Rebalance). The algorithm integrates four key steps: solving a vector-based quadratic pricing equation, determining a systemic margin capacity limit, performing complex market demand rebalancing, and updating net exposure (Delta). We demonstrate that this approach successfully manages a a **flexible Index structure** (e.g., a 50-asset index) against a **150-asset Market**, utilizing the VIL's labels manipulation to handle various index sizes (e.g., top 20, top 100). This capability is crucial to quantifying the operational cost and providing a blueprint for efficient, high-fidelity vector arithmetic on a blockchain.

1 Introduction to the Problem

Decentralized finance protocols increasingly rely on complex, multi-asset products. An index, composed of N_I assets (e.g., $N_I \in \{20, 50, 100\}$), must interact with a broader market environment of N_M assets (where $N_M = 150$). The specific case where $N_I = 50$ is used throughout this analysis as an **illustrative example** of an index being a portion of the total market. The core difficulty lies in processing high-dimensional vectors (up to N_M components) entirely on-chain, where computation is constrained and expensive. Traditional Ethereum Virtual Machine (EVM) operations struggle with loop-based, element-wise vector arithmetic, leading to prohibitive gas costs for products requiring continuous rebalancing.

This challenge is overcome by deploying a separate, custom VM^2 environment, which is the **Vector IL VM** running inside the Arbitrum WASM VM (implemented as a smart contract compiled via Stylus). The VM^2's core advantage lies in its specialized **Vector Instruction Set (VIS)**, providing native, low-level vector operations (e.g., $\mathbf{A} \oplus \mathbf{B}$). Critically, the VIS includes **Labels Manipulation** instructions ('JFLT', 'JUPD'). These commands are essential for correctly aligning and propagating data between the Index domain (e.g., 50 components) and the Market domain (150 components) without compromising state integrity.

It is critical to note that the internal execution complexity of these Label Manipulation instructions is linear, specifically $O(N_I + N_M)$ where N_I is the index size and N_M is the market size. However, this contained $O(N_I + N_M)$ complexity within the highly optimized VIL VM represents a transformative gas saving compared to an EVM implementation. The alternative's gas cost is dominated by the necessity of performing high-cost storage operations (SLOAD & SSTORE) on the underlying blockchain, which makes it prohibitively expensive. By performing this alignment in a single VIL operation, the overall transaction maintains a flat, low gas cost

profile. The goal is to define an algorithm that is computationally efficient and mathematically sound, ensuring the transaction satisfies pricing, collateral, and systemic risk (Margin) constraints simultaneously.

2 Mathematical Framework for Index Acquisition

The execution algorithm is an atomic, six-step process. Variables denoted in bold ($\mathbf{V}$) are vectors, while italic variables (s) are scalars. Operations $\oplus, \ominus, \odot, \oslash$ represent vector addition, saturating subtraction, component-wise multiplication, and component-wise division, respectively. $\min(\mathbf{V})$ denotes the minimum value across all vector components. **Note: When a scalar (s) is used in a vector operation (e.g., $\min(\mathbf{V}, s)$ or $s \odot \mathbf{W}$), it implies a component-wise operation where the scalar is broadcast to a vector of matching dimension.**

2.1 Step 1: Collateral Update and Index Pricing Inputs

The user's effective collateral (C_{new}) is calculated based on initial funds (C_{old}) and external transfers (C_{add}, C_{rem}).

$$C_{\text{new}} = (C_{\text{old}} + C_{add}) - C_{rem}$$

The market provides the scalar Index Price Slope (S) and Price (P), which define the marginal cost function for the total index quantity Q.

2.2 Step 2: Maximum Quantity Determination via Quadratic Solver

The maximum possible Index Quantity based on collateral ($Q_{\text{max, pricing}}$) is determined by solving the Index Price Function, which links the cost (C_{new}) to the quantity (Q): $C_{\text{new}} = P \cdot Q + S \cdot Q^2$.

This is reformulated into a standard scalar quadratic equation:

$$S \cdot Q^2 + P \cdot Q - C_{\text{new}} = 0$$

The maximum quantity is found by taking the positive root of the solution:

$$Q_{\text{max, pricing}} = \frac{\sqrt{P^2 + 4 \cdot S \cdot C_{\text{new}}} - P}{2 \cdot S}$$

2.3 Step 3: Margin-Based Capacity Limit (CL)

To prevent systemic risk breaches, the transaction must be capped by the market's remaining margin capacity ($\mathbf{M}$) relative to the current long (Δ_L) and short (Δ_S) exposures. The **Final Capacity Limit (CL)** for the entire index purchase is defined by the following expression, which is calculated sequentially in the VIL:

$$CL = \min\left(\frac{\mathbf{L}}{\mathbf{W}}\right)$$

1. **Margin Limit Vector (L):** The available capacity ($\mathbf{L}$) is the **sum** of the system's available resources: the **Inventory** that can be closed (Δ_L or Δ_S) plus the **Remaining Margin Headroom**, capped by the equivalent vector for the scalar Market Capacity ($\text{Cap} \odot \mathbf{W}$).

    - **For a Buy Order (increasing system Short exposure):** The capacity is the sum of Existing Long Inventory (Δ_L) which is given priority, and the Unused Short

Risk Budget ($\mathbf{M} \ominus \Delta_S$) component-wise capped by the maximum component-wise asset quantity ($\text{Cap} \odot \mathbf{W}$).

$$\mathbf{L} = \Delta_L \oplus \min(\mathbf{M} \ominus \Delta_S, \text{Cap} \odot \mathbf{W})$$

- **For a Sell Order (increasing system Long exposure):** The capacity is the sum of Existing Short Inventory (Δ_S) and the Unused Long Risk Budget ($\mathbf{M} \ominus \Delta_L$), component-wise capped by the maximum component-wise asset quantity ($\text{Cap} \odot \mathbf{W}$).

$$\mathbf{L} = \Delta_S \oplus \min(\mathbf{M} \ominus \Delta_L, \text{Cap} \odot \mathbf{W})$$

Note: M represents the maximum allowed net exposure, typically equivalent to the total Vendor-managed reserves. Since Δ_L and Δ_S are mutually exclusive (an asset cannot be net long and net short simultaneously), the additive vector operation ($\oplus$) correctly aggregates the two distinct sources of capacity.

2. **Asset Capacity Limit Vector ($\mathbf{CL_{vec}}$):** This vector is aligned via Label Manipulation ('JFLT') to the Index Asset Weights ($\mathbf{W}$) and determines the maximum *Index* quantity each asset can sustain.

$$\mathbf{CL}_{\text{vec}} = \mathbf{L} \oslash \mathbf{W}$$

3. **Final Capacity Limit (CL):** The system capacity is the minimum across all assets. This is extracted using the **VIL 'VMIN'** instruction.

$$CL = \min(\mathbf{CL}_{\text{vec}})$$

2.4 Step 4: Final Quantity Execution and Asset Calculation

The executed Index Quantity (Q_{final}) is capped by the maximum allowed by pricing ($Q_{\text{max, pricing}}$) and the Margin Capacity (CL).

$$Q_{\text{final}} = \min(Q_{\text{max, pricing}}, CL)$$

The resulting executed asset quantities ($\mathbf{A}$) are calculated by distributing Q_{final} according to the Index Weights ($\mathbf{W}$).

$$\mathbf{A} = Q_{\text{final}} \odot \mathbf{W}$$

2.5 Step 5: Market Demand Rebalancing

This is next step, where the 50-component $\mathbf{A}$ vector updates the 150-component market demand vectors ($\mathbf{D}_S, \mathbf{D}_L$).

1. **New Short Demand ($\mathbf{D}_{S,new}$):** The executed assets reduce the standing short demand, saturating at zero.

$$\mathbf{D}_{S,new} = \mathbf{D}_S \ominus \mathbf{A}$$

2. **Residual Quantities ($\mathbf{R_A}$):** Any asset quantities that could not be matched by existing short demand become residual.

$$\mathbf{R}_{\text{A}} = \mathbf{A} \ominus \mathbf{D}_S$$

3. **New Long Demand ($\mathbf{D}_{L,new}$):** The residual quantities are added to the long demand.

$$\mathbf{D}_{L,new} = \mathbf{D}_L \oplus \mathbf{R}_\text{A}$$

2.6 Step 6: Delta Update and Financial Commit

The final net exposures (Δ_L, Δ_S) are updated based on the new demand/supply state $(\mathbf{D}_S, \mathbf{D}_L, \mathbf{S}_L, \mathbf{S}_S)$.

$$\mathbf{T}_L = \mathbf{S}_L \oplus \mathbf{D}_{S,new} \quad ; \quad \mathbf{T}_S = \mathbf{S}_S \oplus \mathbf{D}_{L,new}$$

$$\Delta_{L,new} = \mathbf{T}_L \ominus \mathbf{T}_S \quad ; \quad \Delta_{S,new} = \mathbf{T}_S \ominus \mathbf{T}_L$$

The order's spent collateral is calculated: $C_\text{spent} = Q_\text{final} \cdot (S \cdot Q_\text{final} + P)$. All market and order state vectors are then committed.

3 Performance Analysis and Efficiency

The efficiency of this vectorized execution, even with the complex $50 \to 150$ vector alignment, is quantified by the observed gas cost.

The analyzed transaction required **3,381,746 gas** to execute, based on the provided transaction receipt. This cost is incurred for the following operations:

- Solving the scalar quadratic equation.
- Performing exactly 11 vector arithmetic operations $(\oplus, \ominus, \odot, \oslash)$ on vectors up to 150 components.
- Executing exactly 4 Label Manipulation instructions ('JFLT', 'JUPD', 'JADD') to align and commit data between the Index and Market domains.

Table 1: Transaction Cost and Scale Summary (Updated)

Metric	Value	Cost Equivalency (at 100 Gwei)
Total Gas Used	$\mathbf{3,381,746}$	Actual on-chain consumption
Asset Scale	$N_I = 50$ vs. $N_M = 150$	High dimensional processing
ETH Cost	$3,381,746 \cdot 100 \cdot 10^{-9}$ ETH	**0.0003381746** ETH
USD Cost	0.0003381746 ETH $\times$ \$3,600/ETH*	$\approx$ \$**1.22** USD (or **121.74** cents)

*Assuming an ETH price of \$3,600 for illustrative purposes.

The performance demonstrates that even complex, risk-controlled financial logic—which would typically require dozens of expensive storage reads and writes in standard Solidity—can be executed efficiently using native vector processing. The reduced gas cost is a testament to the VIS's ability to minimize expensive external memory operations and instead rely on fast, in-VM vector arithmetic.

4 Conclusion

The Vectorized Index Buy Order Execution Algorithm provides a mathematically sound and highly performant mechanism for executing high-dimensional financial derivatives on-chain. The formalism highlights the intricate balancing act between pricing constraints ($Q_{\text{max, pricing}}$), systemic stability (CL), and market dynamics ($\mathbf{D}_{S,new}, \mathbf{D}_{L,new}$). By offloading vector operations

and data alignment to the optimized VIS layer, the system achieves a transaction cost of $\approx \$\mathbf{1.22}$ for a complex 50/150 asset trade, confirming the viability of vector-based financial primitives in decentralized ecosystems.