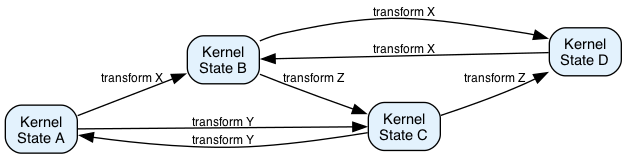
NKI Gym: A Tunable Kernel Environment for Neuron Hardware

Project Proposal - 2026 Offsite Planning

# 1. Executive Summary

NKI Gym is a tunable kernel environment designed to expose the kernel optimization search space for systematic exploration using Monte Carlo Tree Search (MCTS), Large Language Models (LLMs), or reinforcement learning agents. The project uses equality saturation via mlir-egglog to explore all semantically equivalent programs: transformations are defined as declarative rewrite rules in a new nkigym dialect, and the e-graph captures all valid optimizations. Mechanical lowering then generates multiple NISA variants for autotuning on Trainium hardware.



*Figure 1: NKI Gym as RL Environment - Kernels as States, Transforms as Actions*

The key innovation is using equality saturation for guaranteed complete exploration of the transform space. Unlike sequential pass-based compilation where transform ordering matters, equality saturation builds an e-graph representing all equivalent programs simultaneously. This enables automated discovery of high-performance kernel implementations without manual tuning or lucky ordering choices.

# 2. Motivation

Manual kernel optimization is hard; pure LLM generation is error-prone. NKI Gym provides a structured middle ground.

* **Manual optimization:** Vast search space of tile sizes, loop orders, fusion strategies—impractical to explore manually
* **LLM generation:** No correctness guarantees, hallucination risk, cannot systematically optimize
* **NKI Gym:** Correctness by construction + automated search over transformation space

# 3. Architecture Overview

NKI Gym follows a five-stage pipeline with equality saturation at its core: (1) NumPy tracing via NKIPyKernelGen produces linalg dialect IR, (2) a graph-aware tiling pass converts to nkigym dialect with independent tile subgraphs, (3) equality saturation via mlir-egglog explores all semantically equivalent programs, (4) mechanical lowering generates multiple NISA variants, and (5) autotuning benchmarks variants on Trainium to select the best.

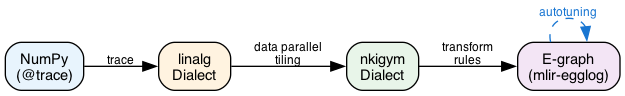


Figure 2: NKI Gym Lowering Pipeline – NumPy functions are traced to linalg, tiled into the nkigym dialect, then optimized via equality saturation (mlir-egglog) with autotuning-guided search.

The architecture uses equality saturation for systematic transform exploration. The nkigym dialect represents independent tile subgraphs where each tile is self-contained with all operations needed to compute its output slice. Transforms are defined as declarative egglog rewrite rules (data reuse, multi-buffer, tile reordering, etc.) rather than imperative code. The e-graph represents all semantically equivalent programs simultaneously, guaranteeing complete exploration of the transform space. Downstream NISA lowering is purely mechanical with no optimization decisions.

## 3.1 Design Example: Equality Saturation with Data Reuse

The following example illustrates how equality saturation explores transforms. Consider a 2x2 tiled matmul where tiles share input data. The nkigym dialect represents each tile as an independent subgraph:

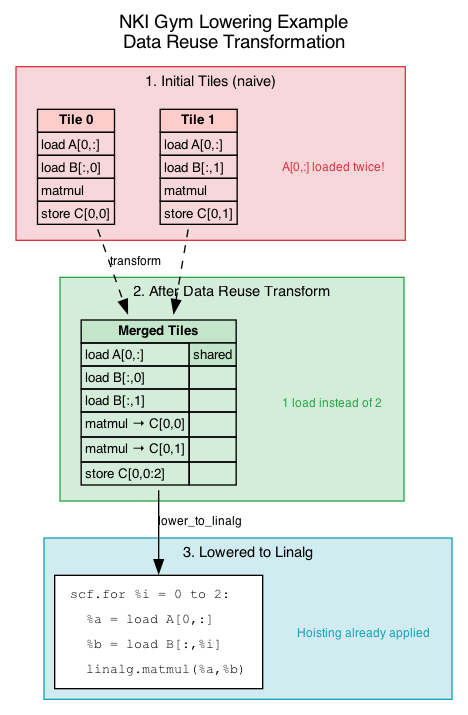


Figure 3: Equality Saturation Example - Data Reuse Discovery

Stage 1: Graph-aware tiling creates independent tile subgraphs, each containing all ops for its output slice. Stage 2: Equality saturation applies the data reuse rule: tiles loading the same HBM region (A[0,:]) are matched and rewritten to share a single SBUF buffer. The e-graph now contains both the original (redundant loads) and optimized (shared load) variants. Stage 3: All variants are extracted, lowered to NISA, and benchmarked. The shared-load variant is selected for better performance.

# 4. Key Benefits of NKI Gym

NKI Gym provides several key advantages for kernel development and optimization:

Explicit search space: All optimization choices (data reuse, multi-buffer, fusion, etc.) are captured in the e-graph via equality saturation, enabling systematic exploration via MCTS, RL, or exhaustive extraction.

Complete exploration: Equality saturation applies all valid rewrite rules until saturation, guaranteeing that all reachable equivalent programs are represented. No transform ordering decisions—the e-graph captures everything.

**Per-tile optimization:** Different optimization strategies can be applied to individual tiles, enabling fine-grained tuning impossible with uniform loop transformations.

Clean separation of concerns: Optimization decisions are explored in the e-graph via equality saturation; downstream NISA lowering is purely mechanical. This makes the search space explicit and the final selection empirical (based on benchmarks).

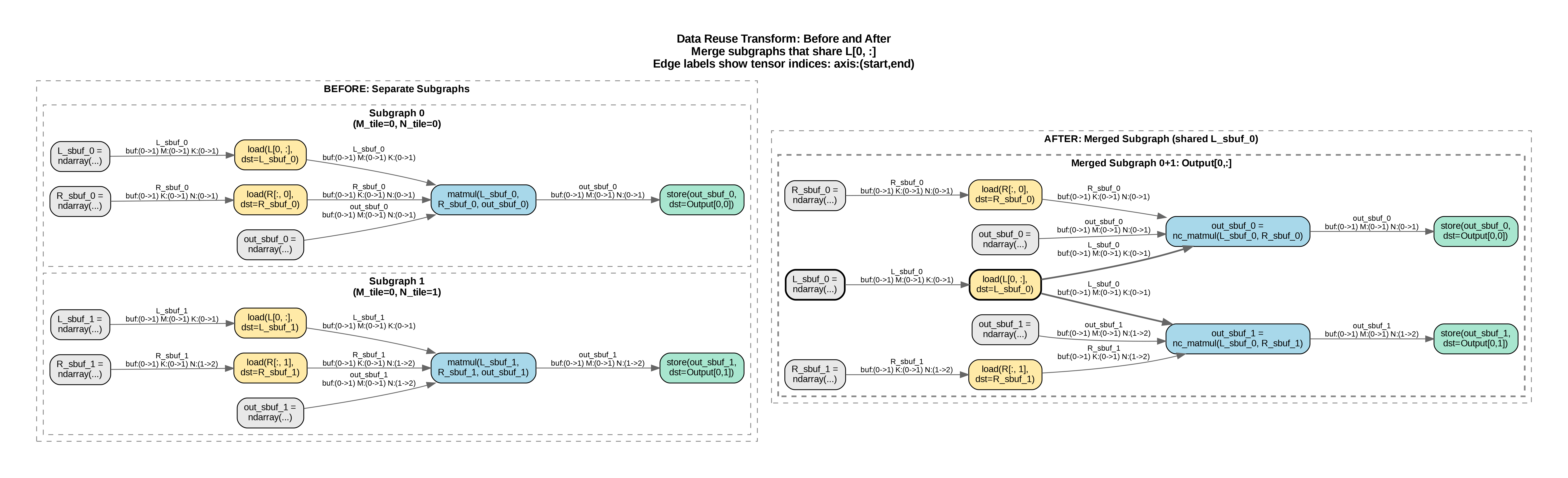
**Built-in operator fusion:** Computation is broken into parallel tiles from the start, with producer-consumer relationships tracked in the IR. Fusion happens naturally—intermediate data stays on-chip without explicit fusion passes.

# 5. Performance Search Space Coverage

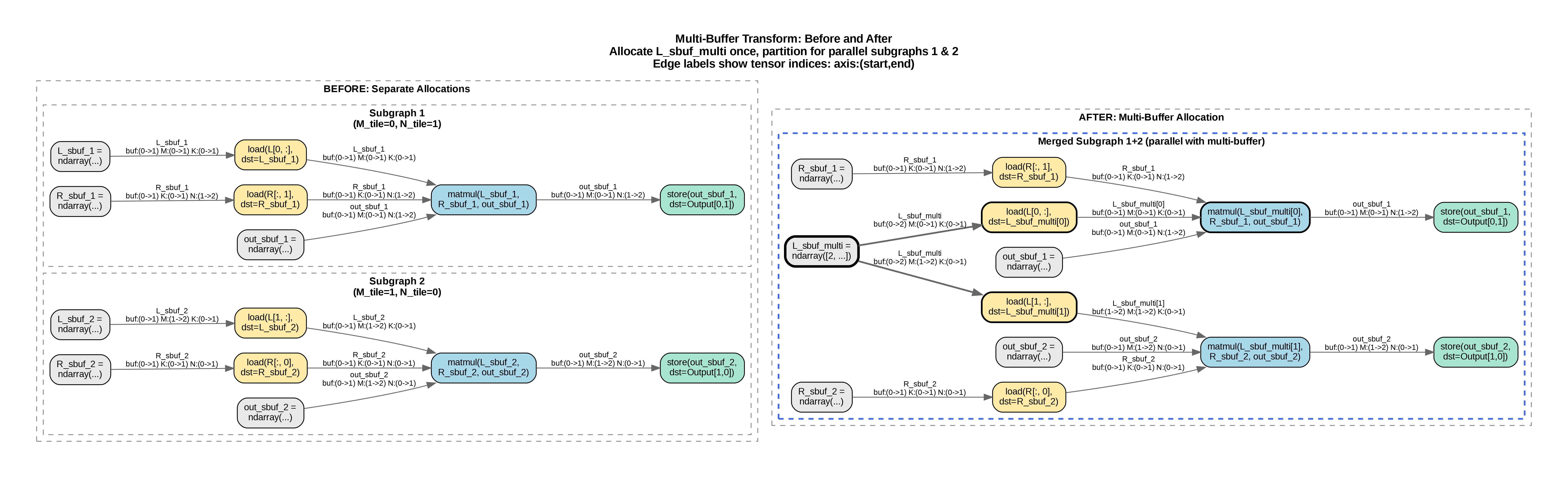
NKI Gym's nkigym dialect fully covers the performance-critical transformations for Neuron kernels. Each transform addresses a specific performance bottleneck and can be modeled as MLIR rewrite patterns.

## 5.1 Memory Bandwidth Optimizations

Data Reuse: Reduces redundant DMA by sharing loaded data across tiles. Impact: Up to 2× fewer loads for shared input tiles. Proposed MLIR approach: Pattern matches tiles with identical load sources, merges into single load with multiple consumers.



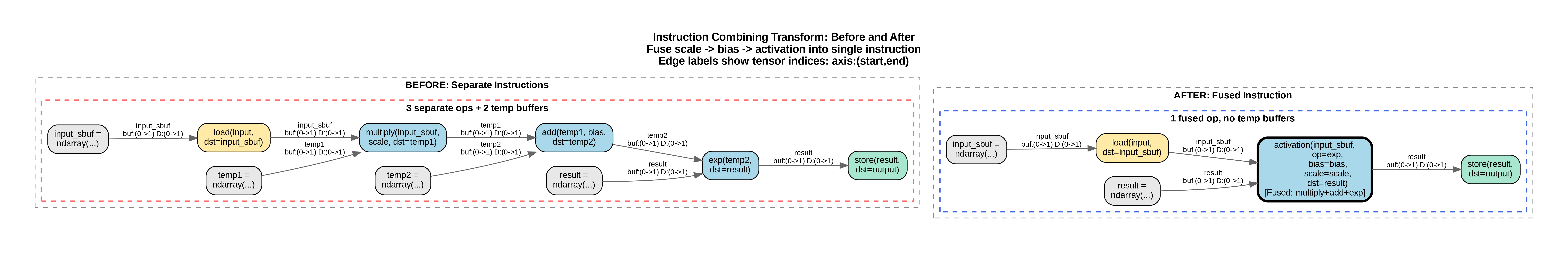
Multi-Buffer: Consolidates allocations with spatial partitioning for parallel access. Impact: Reduces allocation overhead, enables parallel tile execution. Proposed MLIR approach: Combines separate alloc ops into single multi-buffer alloc with partition indices.



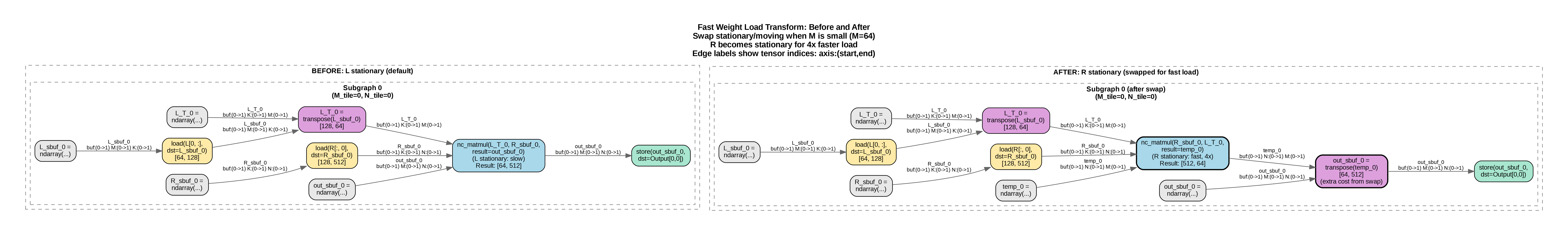
DMA Coalescing: Combines small contiguous transfers into large DMA operations. Impact: Saturates memory bandwidth (small DMAs are inefficient). Proposed MLIR approach: Detects contiguous load/store patterns, replaces with single coalesced operation.

## 5.2 Compute Optimizations

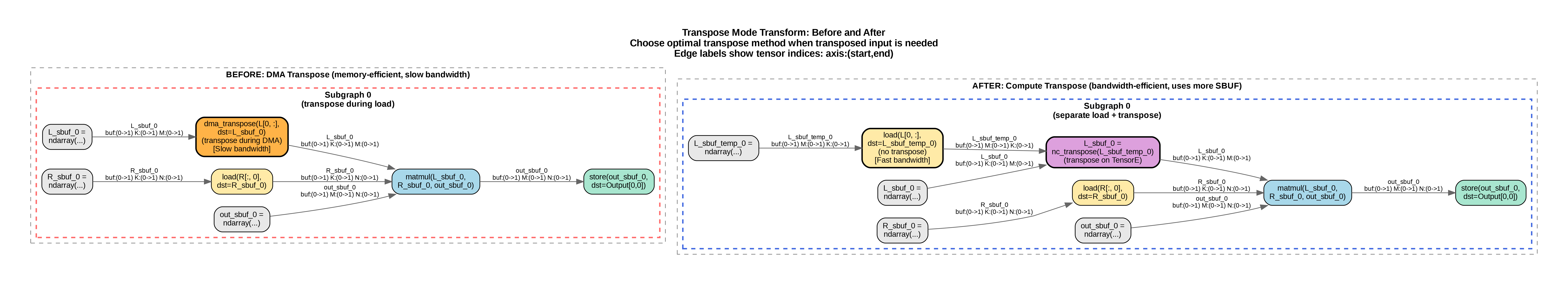
Instruction Combining: Fuses elementwise sequences (scale→bias→activation) into single hardware instruction. Impact: Eliminates intermediate buffers and memory traffic. Proposed MLIR approach: Matches fusible op sequences, replaces with combined nisa instruction.



Fast Weight Load: Swaps stationary/moving tensor roles when beneficial (stationary has 4× faster load). Impact: Significant speedup when M>N in matmul. Proposed MLIR approach: Cost model selects optimal tensor assignment, emits appropriate nc\_matmul variant.

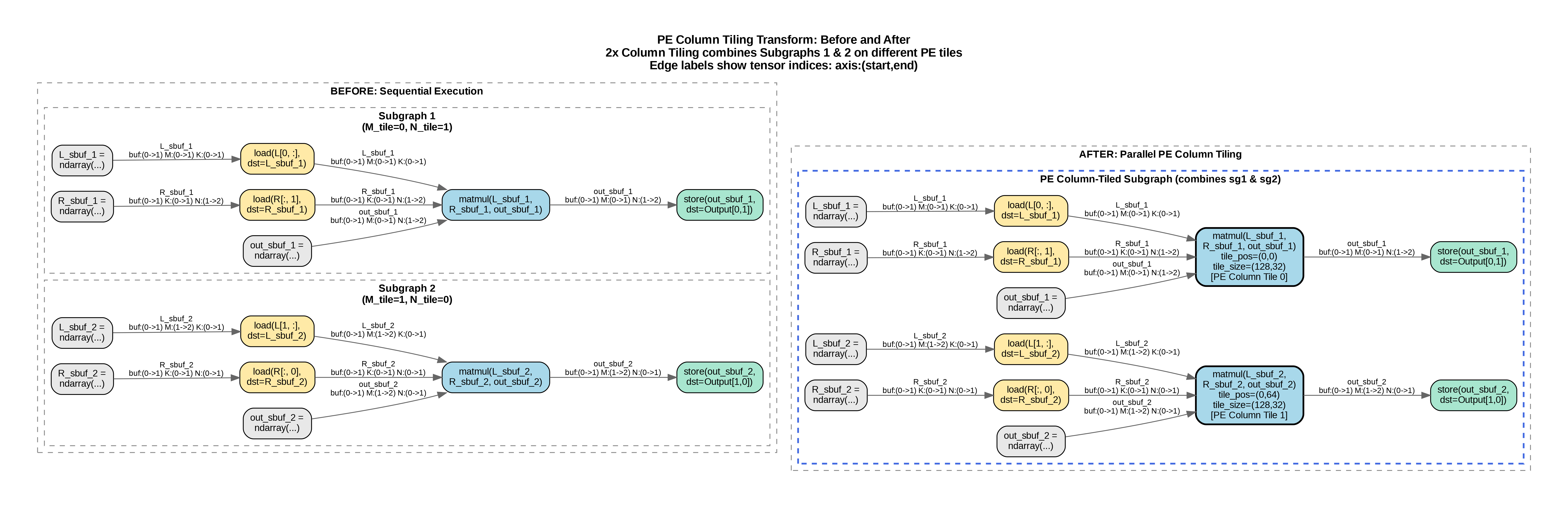


Transpose Mode: Selects between DMA transpose vs compute transpose based on kernel characteristics. Impact: DMA transpose saves SBUF; compute transpose has better bandwidth. Proposed MLIR approach: Pattern matches transpose requirements, selects load\_transpose2d or separate transpose op.

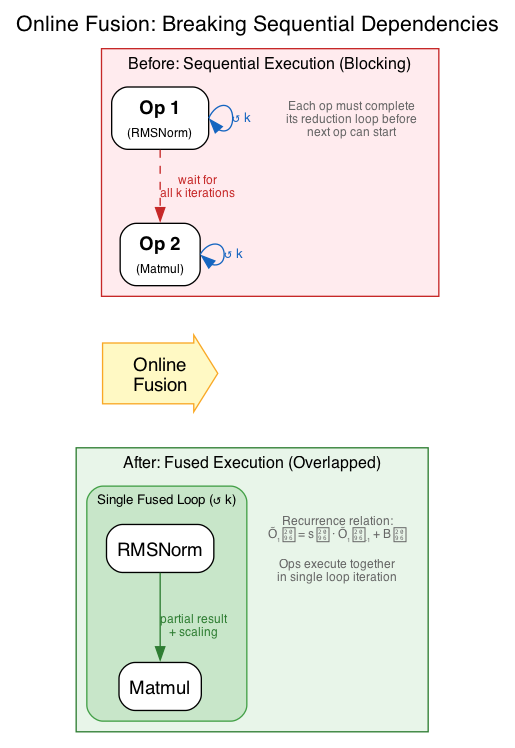


## 5.3 Parallelism Optimizations

PE Column Tiling: Executes independent tiles across multiple PE columns in parallel. Impact: Up to 2× throughput with 2-way column tiling. Proposed MLIR approach: Analyzes tile independence, partitions work across PE columns.



Online Fusion: Generalizes Flash Attention to arbitrary blocking operator + accumulation patterns (RMSNorm+Matmul, Softmax+Matmul, etc.) via recurrence relations with scaling coefficients. Impact: Enables overlapped execution and heterogeneous engine parallelism by breaking sequential dependencies. Proposed MLIR approach: Auto-detect patterns and generate fused kernels with overlapped scheduling.



These eight transformations span the complete performance optimization space for NKI kernels: memory bandwidth (data reuse, multi-buffer, DMA coalescing), compute efficiency (instruction combining, fast weight load, transpose mode), and parallelism (PE tiling, online fusion).

# 6. RL/LLM Integration

NKI Gym is designed for integration with automated search methods:

* States: E-graph representations containing all equivalent kernel variants discovered so far
* Actions: Egglog rewrite rules (data reuse, multi-buffer, tile reordering, etc.) applied to expand the e-graph
* Rewards: Performance of extracted NISA variants measured by benchmarking on Trainium
* Search Methods: Exhaustive saturation for complete exploration, MCTS/LLM for guiding rule application order or extraction selection

Equality saturation guarantees that all reachable equivalent programs are found. The RL/LLM role shifts from discovering transform sequences to efficiently guiding saturation (which rules to apply first) and extraction (which variants to benchmark). This hybrid approach combines exhaustive correctness guarantees with learned efficiency.

# 7. Milestone Planning

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Description** | **Effort** | **Dependencies** |
| M1: nkigym Dialect | Define MLIR operations for IR nodes (alloc, load, compute, store) and transformation attributes | 2 weeks | - |
| M2: Initial Tile Generation | Implement naive tile generation from numpy workload spec | 1 week | M1 |
| M3: Memory Transforms | Data reuse, multi-buffer, DMA coalescing transformations | 3 weeks | M1, M2 |
| M4: Compute Transforms | Instruction combining, transpose mode, fast weight load | 2 weeks | M1, M2 |
| M5: Parallelism Transforms | PE column tiling implementation | 1 week | M1, M2 |
| M6: Online Fusion | Integrate online fusion transformation for X+Accumulation patterns | 2 weeks | M3, M4 |
| M7: Lowering Pass | nkigym → linalg lowering, connect to existing MLIR pipeline | 2 weeks | M3-M6 |
| M8: RL/LLM Integration | Search space API, MCTS/LLM agent interface | 2 weeks | M7 |
| M9: Evaluation | Benchmark suite, performance validation against manual kernels | 2 weeks | M8 |

**Total estimated effort: ~17 weeks**

# Appendix: NKI Gym's Design for Searchable Kernel Optimization

## The Challenge with Traditional Lowering

MLIR-based pipelines spread optimization decisions across multiple passes: tile sizes in tiling, fusion strategies in tile-and-fuse, and layout choices in NISA conversion. The loop-based IR (scf.for) applies transformations uniformly across all tiles—per-tile optimization requires unrolling first. Sequential pass dependencies further complicate matters: bufferization converts tensors to memrefs, obscuring producer-consumer relationships and making late fusion difficult. The result is no single point where search agents can enumerate and explore optimization choices.

## NKI Gym's Solution: Equality Saturation

NKI Gym uses equality saturation via mlir-egglog to systematically explore the transform space. Rather than applying transforms sequentially and hoping to find a good order, equality saturation builds an e-graph that represents all semantically equivalent programs simultaneously. This guarantees complete exploration of reachable optimizations.

## Transforms as Egglog Rewrite Rules

Every optimization is defined as a declarative egglog rewrite rule with explicit preconditions. For example, the data reuse rule pattern-matches tiles loading the same HBM region and rewrites them to share a single SBUF buffer. The e-graph captures both original and rewritten forms as equivalent—all variants are correct by construction. Search agents can enumerate valid rewrites at any e-graph state, apply rules until saturation, then extract and benchmark multiple variants.

## Tile-Level Granularity

Each parallel tile is a distinct node with explicit data dependency edges. Per-tile optimization is natural—different tiles can use different strategies without requiring loop unrolling. This avoids the uniform-behavior constraint of loop-based representations.

## Early Decisions, Mechanical Lowering

The nkigym dialect captures all performance-relevant choices in the e-graph. After saturation, multiple equivalent variants are extracted and lowered to NISA. This lowering is purely mechanical—no optimization heuristics, just translation. Finally, autotuning benchmarks each NISA variant on real Trainium hardware to select the best performer. This separation ensures the search space is explicit and the selection is empirical.

## Why Equality Saturation?

Traditional compiler passes apply transforms in a fixed order, making the final result dependent on ordering choices. Equality saturation solves this by building an e-graph that represents all equivalent programs simultaneously. When a rewrite rule matches, both the original and rewritten forms are kept—nothing is lost. After saturation (when no new rewrites apply), the e-graph contains all reachable optimizations. This approach has proven successful in compiler research (TASO for tensor graphs, egg for general-purpose optimization, Diospyros for DSP kernels). NKI Gym uses mlir-egglog (https://github.com/sdiehl/mlir-egglog) to bring equality saturation to MLIR.