CORE SERIES VIII

Packet Operator Algorithms

Exact compilation and GPU realization

Finite operator structure removes work before execution. Exact compilation makes the representation, error budget, output contract, and inherited hardware observations explicit.

Komorebi P.

Modern Monograph • 9 October 2026

How to use this volume

The complete finite manuscript comes first. All six supplementary proof developments follow as Appendices A–F. A worked visual atlas closes the volume. The separate visual and Lean companions provide another reading route and an explicit evidence register.

Reading this edition

An operator representation determines what a computation means before it determines how a device executes it. Paper VIII develops that distinction through packet blocks, low rank corrections, graph support, preconditioning, precision, and finite compilation. Three exact examples introduce the main choices. The manuscript then gives their general statements and proofs.

All seven retrieved mathematical source parts are preserved byte for byte. The main manuscript retains all 85 numbered mathematical records, including all 56 explicit theorem, proposition, and corollary proof bodies. Its section and item numbers remain unchanged. Appendices A–F retain the complete substantive mathematics of six formal supplements, including constructions that strengthen the shorter manuscript arguments. Overlapping statements and derivations are retained because their hypotheses and uses differ. These counts do not measure distinct discoveries.

The selected v0.6.0 formal checkpoint records coverage of all 56 finite manuscript-proved items under its stated hypotheses. It contains 392 named theorem declarations in 40 modules, with 373 Paper VIII declarations and 19 reused quantum-packet declarations. The 66 construction audits and four classical inputs are different evidence units. This monograph carries the inherited compiler evidence and does not claim a fresh Lean compilation. The Lean companion records the actual hypotheses, declarations, hashes, and replay boundaries.

The source already includes positive precision budgets and caps, nonempty allocation sets, positive physical denominators and transform orders, valid band widths, actual positive square roots, reflexive reachability, and the empty-graph depth-zero case. Its arithmetic counts describe declared event models. Compiler stability uses solved child scores, and local recompilation assumes correct unaffected sibling caches. These hypotheses are part of the results.

The small-core recompression result starts from supplied ordered thin-SVD equations and handles arbitrary ambient competitors by compression. It does not establish SVD existence, a floating-point QR/SVD algorithm, or a canonical singular-values API identification. The other three classical inputs concern exact-arithmetic conjugate gradients, a stated floating-point model, and simultaneous spectral decomposition. A rounded action on varying inputs requires a uniform certificate before it can be treated as one fixed linear operator.

Representation, output, model, observation

A compressed action and an explicit matrix are different outputs. An additive cost model and an observed execution time are different evidence. A finite compiler optimum and unrestricted structure discovery are different problems. Keep the representation, output contract, mathematical model, and recorded measurement together when using a result.

Every hardware time and speed ratio remains an inherited observation from the archived corpus. No fresh GPU run is reported. Missing source-value hashes and explicit correctness-reference fields limit the historical comparisons. Cross-scope CPU, graph-replay, and two-stream rows retain their qualifi-

cations. In the negative-control figure, “matched baseline” means the baseline recorded for that comparison. It does not certify the full fairness contract set out in the benchmark discussion.

The historical pending statements in Appendices A–E describe their dated checkpoints. Appendix F completes the finite register. Three broader targets remain open: unrestricted discovery, compilation with nonlocal overlapping models, and a packet-native communication theorem for a specified architecture. The new exact examples and the visual atlas illustrate finite mathematics. They add no hardware performance claim and do not replace the proofs.

Contents

Reading this edition . . . i
Three exact models before the general theory . . . 2
1 Introduction . . . 5
2 Packet operator representations . . . 7
2.1 Exact semantics . . . 7
2.2 Output contracts . . . 9
3 Arithmetic, storage, and data movement . . . 10
4 Structured packet primitives . . . 13
5 Exact low rank algebra and recompression . . . 16
5.1 Products and inverses . . . 16
5.2 Optimal factor recompression . . . 18
6 Sparse transfer graphs and schedules . . . 21
6.1 Support under algebraic operations . . . 21
6.2 Block triangular solves . . . 21
6.3 Fill under elimination . . . 23
7 Krylov methods and packet preconditioning . . . 24
8 Packetwise mixed precision . . . 26
8.1 One operator . . . 26
8.2 Several operators . . . 27
8.3 Continuous allocation . . . 27
8.4 Discrete data types . . . 28
9 Discovery and exact packet compilation . . . 30
9.1 Exact spectral discovery . . . 30
9.2 Candidate packet trees . . . 30
9.3 An exact error budget compiler . . . 33
9.4 Decision stability and local updates . . . 33
10 Device scheduling and amortization . . . 34
10.1 Shape classes . . . 34
10.2 Critical path . . . 35
10.3 One time costs . . . 35
11 Benchmark contracts and measurement protocol . . . 36
12 Recorded CPU and GPU corpus . . . 37
12.1 Dense precision frontier . . . 38
12.2 Regular packet operators . . . 39
12.3 Compressed products . . . 40
12.4 Discovery and transfer . . . 40
12.5 Negative controls . . . 40
13 Implementation and verification . . . 41
14 Hardware implications and exact boundaries . . . 42
15 Conclusion . . . 44
A Exact semantics, support, and low-rank identities . . . 46
A.1 Exact packet semantics . . . 46
A.2 Support propagation and nilpotence . . . 47
A.3 Low-rank algebra and inversion . . . 48
A.4 Packet perturbation certificates . . . 49
A.5 Statement correction and verification . . . 50

B Exact compilation, continuous precision, and explicit costs . . . 51
B.1 Exact finite-tree compilation . . . 51
B.2 Frontiers and decision stability . . . 52
B.3 Cached local recompilation . . . 52
B.4 The continuous allocation theorem is complete . . . 53
B.5 Explicit storage and operation models . . . 54
B.6 Contracts, grouping, and amortization . . . 55
B.7 Verification and remaining work . . . 56
C Product chains, Krylov spaces, and integer budgets . . . 58
C.1 Low-rank factors through a product chain . . . 58
C.2 Error propagation and recompression . . . 59
C.3 Exact and approximate packet Krylov spaces . . . 60
C.4 Integer error budgets and discrete precision . . . 61
C.5 The quantized packet-tree compiler . . . 61
C.6 Cached arrays and a work certificate . . . 62
C.7 Verification, executable checks, and remaining work . . . 63
D Structured transforms and clipped precision . . . 65
D.1 Circulant Fourier diagonalization . . . 65
D.2 Toeplitz padding and column-major vectorization . . . 66
D.3 Arbitrary tensor factors . . . 66
D.4 Walsh butterfly and exact addition count . . . 67
D.5 Clipped precision: existence and multiplier uniqueness . . . 68
D.6 Clipped precision: global optimality . . . 69
D.7 Verification, regressions, and remaining work . . . 70
E Constructed graph schedules, support, and preconditioning . . . 72
E.1 Constructing the optimal level schedule . . . 72
E.2 Forward substitution with an actual stage evaluator . . . 73
E.3 Support of inverses and Schur elimination . . . 74
E.4 Exact banded addresses and modeled work . . . 74
E.5 Exact residual certificates for supplied packets . . . 75
E.6 Block-Jacobi normalization and the CG factor . . . 76
E.7 Verification record and remaining boundary . . . 77
F Global small-core recompression from ordered thin SVD data . . . 79
F.1 Compact factors from actual column spaces . . . 79
F.2 Why a small-core optimum is a global optimum . . . 80
F.3 Spectral optimality from a kernel witness . . . 80
F.4 Frobenius optimality from kernel weights . . . 81
F.5 Thin output factors and the final theorem . . . 82
F.6 Source map and verification . . . 83
F.7 Regressions, reproduction and claim boundary . . . 84
Worked visual atlas . . . 85
References . . . 117

List of Figures

1 Exact finite action of a diagonal operator plus one thin correction. The bars display the same action as the explicit matrix. . . . 2
2 The finite candidate tree has three prunings. The middle point is optimal under error budget four, although no nonnegative Lagrangian multiplier selects it. . . . 3
3 The shaded region satisfies the supplied error budget. AFF is the unique minimum-cost assignment in that region. The model protects the first edge before spending on less sensitive edges. . . . 4
4 The representation and the output contract are fixed before the device schedule is chosen. 6
5 A packet intermediate representation stores active block addresses together with dimensions, block types, tolerances, and scheduling metadata. . . . 8
6 The ideal reduction grows with dimension only when packet width and transfer rank remain controlled. . . . 12
7 A representation can move the same mathematical problem between memory limited and compute limited regimes. . . . 13
8 A regular diagonal packet structure is coupled by global low rank factors. . . . 17
9 The approximation decision is made from the singular values of a small core whose dimension is the unrecompressed rank. . . . 20
10 Composition creates possible support along directed paths. Cancellation can remove an allowed block, so the graph gives an inclusion rather than an equality. . . . 22
11 Packets in the same topological level can be processed concurrently in a triangular solve. 23
12 The block diagonal packet operator serves as a reusable local preconditioner in a weakly coupled positive system. . . . 26
13 More sensitive packet edges receive tighter continuous tolerances. . . . 29
14 The thick leaves are selected by the exact Lagrangian recurrence. . . . 32
15 The quantized dynamic program produces the exact cost and error frontier for its finite model. . . . 34
16 The corpus records timing scope and output contract as separate fields. . . . 37
17 Recorded device resident dense throughput across dimensions and scalar types. . . . 38
18 The speed and error tradeoff depends on both precision and problem size. . . . 39
19 Regular equal shapes expose both representation reduction and batched execution. . . . 40
20 Representation and launch strategy both affect BDLR application time. . . . 41
21 Compressed and explicit products answer different computational questions. . . . 42
22 A profitable repeated representation can remain unprofitable for one use. . . . 43
23 Device residence, transfer, graph replay, and a multioutput pipeline have different timing scopes. . . . 44
24 Values above one are slowdowns against the matched baseline. . . . 45
25 The best recorded gain differs strongly across suites. Regular packet structure dominates the largest measured rows. . . . 45

Abstract

A packet operator is a linear map written between the members of a finite direct sum. We study the resulting computational representation before choosing a processor or kernel. Exact formulas are given for block sparse application, block diagonal plus low rank algebra, Woodbury inversion, support propagation, block preconditioning, and packetwise perturbation. Products of low rank corrections admit optimal recompression through a small singular value problem, with explicit error propagation through a product chain. A rooted candidate decomposition supports an exact compiler when representation cost and error are additive. Dynamic programming gives the optimal Lagrangian pruning, an exact quantized error budget algorithm, and a local update rule. Discrete precision assignment becomes a multiple choice budget problem. Arithmetic, storage, data movement, launch count, discovery, transfer, and materialization are kept as separate costs. A preserved corpus of 154 recorded CPU and GPU measurements illustrates regular packet gains, precision frontiers, crossover behavior, and retained negative controls. Its largest reported structured ratio is 14.89 against a device resident dense GPU calculation, with archived matching-metadata limitations stated explicitly. The paper proves finite algebraic and optimization statements. Hardware measurements remain specific to their recorded environment, and no generic dense matrix exponent or universal discovery algorithm is claimed.

Three exact models before the general theory

These models use exact integers and rational numbers. Their costs and integer errors are declared illustrative quantities. The figures display the computed records. They do not report timings or identify actual hardware precision types.

A low rank action and its inverse

Set

$$D = \operatorname{diag}(2, 3, 4, 5), \qquad u = (1, 0, 1, 0)^T, \qquad v = (1, 1, 0, 0)^T,$$

and define $A = D + uv^T$. Its entries are shown in the figure. The same operator can be stored by its diagonal and two thin vectors rather than by every matrix entry. For $x = (1, 2, 3, 4)^T$, first form $v^T x = 3$. Then

$$Dx = (2, 6, 12, 20)^T, \qquad u(v^T x) = (3, 0, 3, 0)^T,$$

so $Ax = (5, 6, 15, 20)^T$. Only the first and third output coordinates receive the correction.

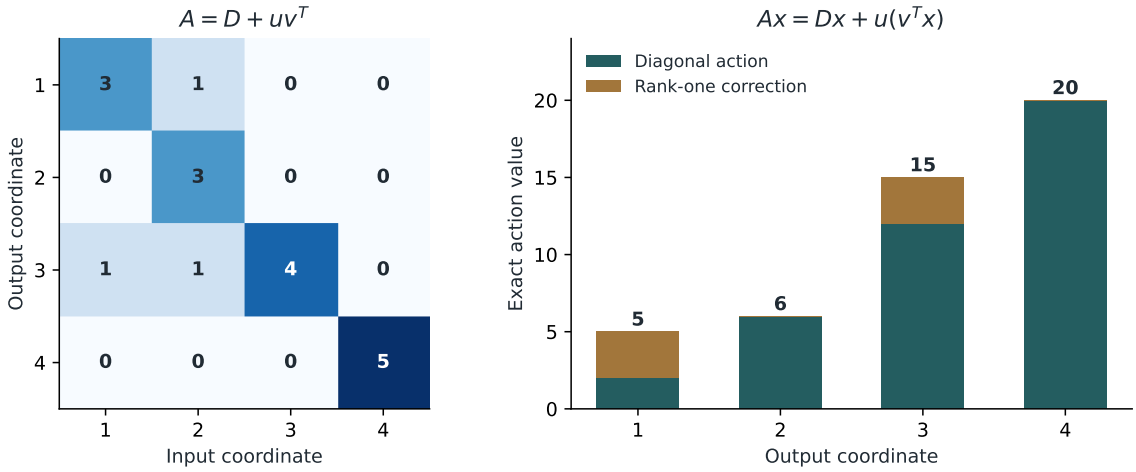


Figure 1. Exact finite action of a diagonal operator plus one thin correction. The bars display the same action as the explicit matrix.

The inverse starts with the same representation. Write $y = Ax$. Since D is invertible,

$$D^{-1}u = (1/2, 0, 1/4, 0)^T, \qquad 1 + v^T D^{-1} u = 3/2 \neq 0.$$

The scalar Woodbury core is therefore invertible. Its correction coefficient is

$$\frac{v^T D^{-1} y}{1 + v^T D^{-1} u} = \frac{9/2}{3/2} = 3.$$

Subtracting that correction recovers the original input:

$$D^{-1}y - 3D^{-1}u = (5/2, 2, 15/4, 4)^T - (3/2, 0, 3/4, 0)^T = x.$$

> **What the output contract preserves**
>
> The diagonal and thin vectors use twelve scalar slots, compared with sixteen for the explicit matrix. This small storage count assumes the full thin vectors are stored. It does not predict elapsed time. A compressed action returns Ax. Materializing all entries of A produces another output and incurs its own work. The general formulas appear in theorems 2.3 and 5.3.

A budget can select a point that no Lagrangian selects

Consider a rooted candidate tree. The root r has children a, b, and a has children a_1, a_2. Split overheads are zero. Each retained node has the following declared cost and integer error:

Node	r	a	b	a_1	a_2
Cost	1	2	1	3	1
Error	6	4	0	1	0

Exactly three prunings are possible:

Selected nodes	Total cost	Total error
r	1	6
a, b	3	4
a_1, a_2, b	5	1

For a Lagrangian multiplier $\lambda \geq 0$, their objectives are $1 + 6\lambda$, $3 + 4\lambda$, and $5 + \lambda$. The middle pruning can be optimal only if

$$3 + 4\lambda \leq 1 + 6\lambda \quad \text{and} \quad 3 + 4\lambda \leq 5 + \lambda.$$

These inequalities require $\lambda \geq 1$ and $\lambda \leq 2/3$ simultaneously. There is no such multiplier. The root and full split exchange optimality at $\lambda = 4/5$, where both have objective $29/5$.

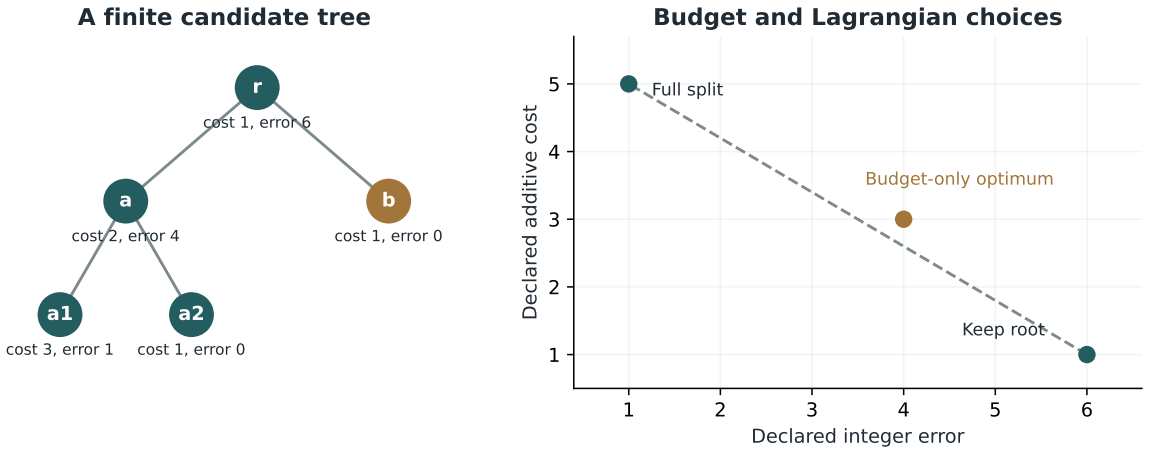


Figure 2. The finite candidate tree has three prunings. The middle point is optimal under error budget four, although no nonnegative Lagrangian multiplier selects it.

The budget problem still selects that middle pruning. Under the constraint err ≤ 4, the root is infeasible. The two feasible costs are three and five, so the minimum is three. Under budget one, only the full split is feasible. Under budget six, the root is cheapest. Thus the budget program adds something that changing λ cannot recover.

The dynamic recurrence gives the same Lagrangian choices without listing prunings. At a, it compares $2 + 4\lambda$ with $4 + \lambda$. At r, it compares $1 + 6\lambda$ with $1 + F_\lambda(a)$. Its exactness follows from the finite tree and additive model in theorems 9.4 and 9.8. Overlapping decompositions or nonlocal execution costs need another argument.

Discrete precision follows the sensitive edge

Suppose three packet edges each admit a fast option F and an accurate option A. The names describe the supplied model, rather than any actual scalar type. Exactly one option is chosen on each edge:

	F		A	
Edge	Cost	Error	Cost	Error
1	1	5	4	0
2	1	2	3	0
3	1	1	3	0

The all-fast assignment FFF has cost three and error eight. It exceeds budget three. Changing only edge one to A gives AFF, with cost $4 + 1 + 1 = 6$ and error $0 + 2 + 1 = 3$. Changing only edges two and three gives FAA, with cost seven and error five. That choice still exceeds the budget.

All eight assignments can be checked directly:

Assignment	Cost	Error	Assignment	Cost	Error
FFF	3	8	AFF	6	3
FFA	5	7	AFA	8	2
FAF	5	6	AAF	8	1
FAA	7	5	AAA	10	0

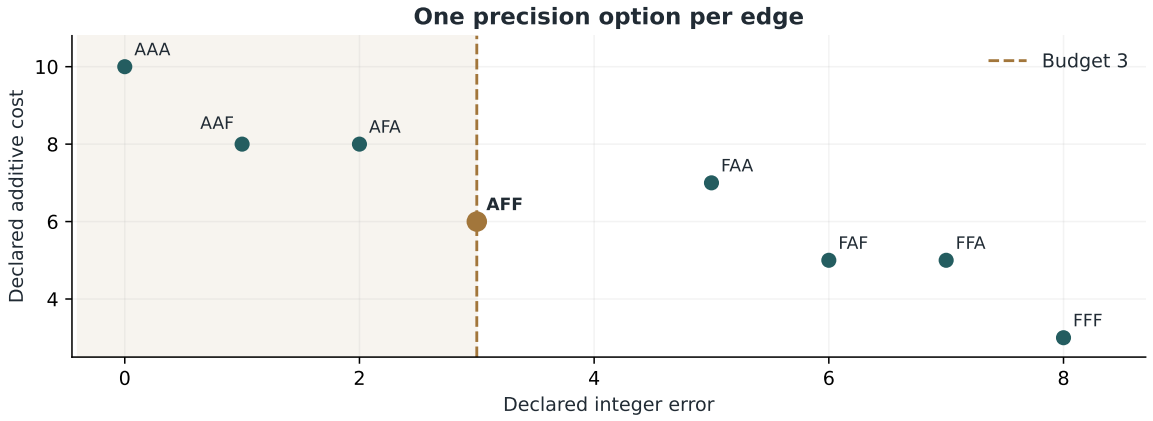


Figure 3. The shaded region satisfies the supplied error budget. AFF is the unique minimum-cost assignment in that region. The model protects the first edge before spending on less sensitive edges.

The exact recurrence reaches the same result. After edge one, only $(\mathrm{err}, \mathrm{cost}) = (5, 1)$ and $(0, 4)$ are reachable. After edge two, the reachable states are $(7, 2)$, $(5, 4)$, $(2, 5)$, and $(0, 7)$. Extending these with either option on edge three gives the final eight assignments. Among states of error at most three, the lowest cost is six.

> **The certificate belongs to the model**
>
> The integer error weights must be supplied by a stated certificate or quantization rule. The recurrence solves that finite model exactly. Kernel accumulation, overflow, underflow, scaling, and shape support require their own assumptions before an actual implementation can use the result. The precise recurrence is theorem 8.7.

The complete rational records, dynamic tables, and five rejected false controls are reproduced by `generate_monograph_examples.py`. Figure generation converts those records to plotting coordinates only after the exact checks pass.

The finite manuscript

The main reading path follows exact packet semantics, algebra, support, numerical certificates, and optimization before reaching the archived measurement corpus. The cost and error assumptions determine where each algorithm is exact.

1 Introduction

A matrix can look dense while its action is not. Independent diagonal blocks, sparse transfers, low rank couplings, tensor factors, and transform coordinates can remove most of the arithmetic and storage before a kernel is launched. The gain is mathematical first. A device can exploit only the structure that the representation exposes.

This article begins with a finite direct sum

$$X = \bigoplus_{\alpha \in I} X_\alpha$$

and a complete family of coordinate projectors

$$P_\alpha P_\beta = \delta_{\alpha\beta} P_\alpha, \qquad \sum_{\alpha \in I} P_\alpha = I.$$

An operator has packet blocks

$$A_{\beta\alpha} = P_\beta A P_\alpha.$$

The active addresses form a directed graph. Block dimensions, block types, approximation tolerances, and output requirements turn this graph into a computational intermediate representation.

The underlying linear algebra is classical. Direct and iterative methods are treated in [14, 25, 9]. Low rank approximation and updates are governed by singular value decomposition and the Sherman Morrison Woodbury identity [11, 22, 15, 4]. Floating point analysis follows the standard model in [17, 20]. Randomized discovery of low rank structure is developed in [16, 21]. Communication and attainable performance are shaped by data movement and arithmetic intensity [19, 27, 2, 12].

The first task is semantic. A compressed representation must specify the operator that it denotes. It must also specify what an algorithm returns. A product represented by block and low rank factors is not the same output as an explicit matrix. A timing comparison is valid only after the output contract and timing scope agree.

The second task is algebraic. Block diagonal plus low rank operators are closed under multiplication before recompression. Their unrecompressed rank grows additively. Woodbury inversion reduces a large inverse to packet solves and one small dense inverse. The low rank factors can then be recompressed by orthogonalizing them and taking the singular value decomposition of a small core. This produces the best truncated approximation in spectral and Frobenius norm.

The third task is combinatorial. Sparse transfer graphs predict possible support under powers, polynomials, and Schur elimination. A directed acyclic graph gives a finite path expansion. A block triangular solve can be scheduled by topological level. Graph regularity matters because equal shapes can be batched, while irregular edges require indexing and accumulation.

The fourth task is numerical. Packet perturbations can be certified by their block sensitivities. A continuous model assigns tolerances through a convex calculation. Upper bounds on available tolerances

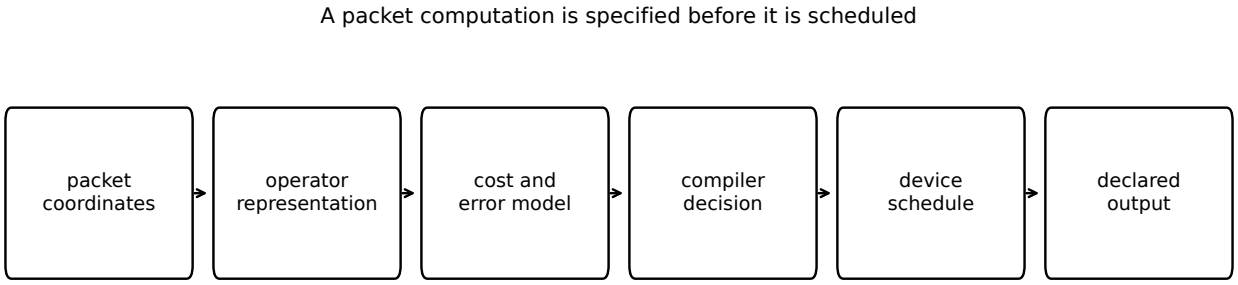


Figure 4. The representation and the output contract are fixed before the device schedule is chosen.

produce a clipped allocation. A finite set of data types gives a multiple choice budget problem that can be solved exactly after the errors are quantized.

The central computational result concerns compilation. Suppose a finite rooted tree records candidate refinements. Each node may be kept as one representation or split into its children. If cost and error are additive across the selected leaves, the optimal pruning is found exactly by dynamic programming. A quantized error budget has a second exact dynamic program. The decision is stable when the score gap exceeds the perturbation in the local models. A change inside one subtree requires recomputation only along the ancestor chain. These results prove a restricted packet compiler. They do not solve unrestricted discovery or arbitrary device scheduling.

The measured part uses a sanitized corpus collected on one recorded CPU and one NVIDIA GeForce RTX 3070 Ti Laptop GPU. The preserved corpus contains 154 recorded rows from sixteen suites. The timings are inherited observations; no GPU timing is rerun for this edition. Regular equal block operators and block diagonal plus low rank operators show large gains. Recursive HODLR, Strassen multiplication, iterative refinement, a small Walsh Hadamard transform, and an attempted two stream overlap provide negative controls. These rows are retained because structure does not guarantee speed.

The article separates four sources of gain:

$$\text{representation}, \qquad \text{device}, \qquad \text{kernel}, \qquad \text{amortization}.$$

A single speedup number can mix all four. The benchmark protocol therefore requires case identity, output contract, timing scope, source precision, correctness reference, discovery time, compilation time, and reuse count. The archived rows do not satisfy every field of this ideal contract: source-value hashes and explicit correctness-reference fields are absent; setup, discovery, and reuse metadata are sparse. Recorded ratios are consequently descriptive comparisons under their stated scopes, rather than full-provenance fair matches.

The mathematical statements are finite. The empirical statements apply only to the recorded environment and generated cases. The article does not claim a new generic matrix multiplication exponent. Fast matrix multiplication and its stability form a separate theory [26, 10]. It also does not claim that every structured matrix has a profitable GPU realization.

2 Packet operator representations

2.1 Exact semantics

Definition 2.1 (Packet layout)**.** A packet layout is a finite family of finite dimensional Hilbert spaces

$$\mathcal{P} = (X_\alpha)_{\alpha \in I}$$

with direct sum

$$X = \bigoplus_{\alpha \in I} X_\alpha.$$

Write

$$n_\alpha = \dim X_\alpha, \qquad n = \sum_{\alpha \in I} n_\alpha.$$

The coordinate inclusion and projection are $\iota_\alpha : X_\alpha \to X$ and $\pi_\alpha : X \to X_\alpha$. The ambient coordinate projector is $P_\alpha = \iota_\alpha \pi_\alpha$.

Definition 2.2 (Packet operator representation)**.** A finite packet operator representation consists of

$$\mathsf{IR} = (\mathcal{P}, E, (B_{\beta\alpha})_{(\beta,\alpha)\in E}),$$

where

$$E \subseteq I \times I$$

and

$$B_{\beta\alpha} : X_\alpha \longrightarrow X_\beta.$$

Local blocks are identified with their ambient extensions by inclusion and projection. In particular, the notation $P_\beta A P_\alpha$ denotes either the ambient block or its restriction $X_\alpha \to X_\beta$ according to the displayed domain; $\iota_\beta B_{\beta\alpha} \pi_\alpha$ is the corresponding ambient extension. Its evaluation is

$$\mathrm{eval}(\mathsf{IR}) = \sum_{(\beta,\alpha)\in E} \iota_\beta B_{\beta\alpha} \pi_\alpha,$$

where $\iota_\beta : X_\beta \to X$ is the coordinate inclusion.

Theorem 2.3 (Exact evaluation)**.** *Let $A \in \mathrm{End}(X)$. Set*

$$B_{\beta\alpha} = \pi_\beta A \iota_\alpha$$

and let E contain exactly the nonzero blocks. Then

$$\boxed{A = \sum_{(\beta,\alpha)\in E} P_\beta A P_\alpha.}$$

For every matrix of right hand sides Z,

$$AZ = \sum_{(\beta,\alpha)\in E} \iota_\beta B_{\beta\alpha} \pi_\alpha Z.$$

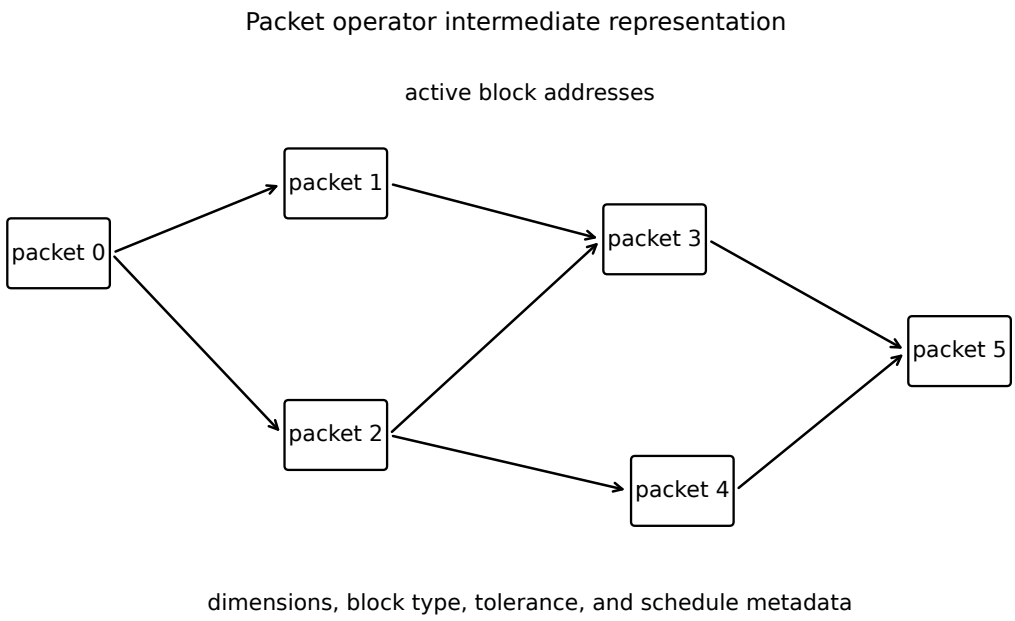


Figure 5. A packet intermediate representation stores active block addresses together with dimensions, block types, tolerances, and scheduling metadata.

Proof. Insert the identity on both sides of A:

$$A = \left(\sum_{\beta} P_\beta\right) A \left(\sum_{\alpha} P_\alpha\right) = \sum_{\alpha,\beta} P_\beta A P_\alpha.$$

The zero blocks can be removed. Applying the identity to Z gives the second formula. □

Definition 2.4 (Support graph)**.** The support graph G_A has vertex set I. It has a directed edge

$$\alpha \longrightarrow \beta$$

when $P_\beta A P_\alpha \neq 0$.

Theorem 2.5 (Block composition)**.** *Let $A, B \in \mathrm{End}(X)$. Then*

$$\boxed{(AB)_{\gamma\alpha} = \sum_{\beta \in I} A_{\gamma\beta} B_{\beta\alpha}.}$$

Consequently,

$$\mathrm{supp}(AB) \subseteq \mathrm{supp}(A) \circ \mathrm{supp}(B),$$

where relational composition is defined by

$$(\gamma, \alpha) \in E \circ F$$

when some β satisfies

$$(\gamma, \beta) \in E, \qquad (\beta, \alpha) \in F.$$

Proof. Insert the packet identity between A and B:

$$P_\gamma AB P_\alpha = \sum_\beta P_\gamma A P_\beta B P_\alpha.$$

A summand can be nonzero only when both block addresses are active. □

Corollary 2.6 (Adjoint and tensor semantics)**.** *The adjoint representation has blocks*

$$(A^*)_{\alpha\beta} = A^*_{\beta\alpha}.$$

If A and B are packet resolved on X and Y, then

$$(A \otimes B)_{(\beta,\delta),(\alpha,\gamma)} = A_{\beta\alpha} \otimes B_{\delta\gamma}.$$

Proof. Both identities follow by inserting the corresponding coordinate projectors and using

$$(P_\beta A P_\alpha)^* = P_\alpha A^* P_\beta.$$

The tensor identity follows from

$$(P_\beta \otimes Q_\delta)(A \otimes B)(P_\alpha \otimes Q_\gamma) = A_{\beta\alpha} \otimes B_{\delta\gamma}.$$

□

2.2 Output contracts

Definition 2.7 (Output contract)**.** An output contract specifies the mathematical object returned by an algorithm. The contract may require an explicit vector, an explicit matrix, block factors, low rank factors, spectral factors, or an implicit operator with a declared application rule.

Definition 2.8 (Timing scope)**.** A timing scope states which costs are included. Possible costs include structure discovery, compilation, host transfer, device transfer, kernel execution, graph capture, graph replay, factor reconstruction, and explicit materialization.

Proposition 2.9 (Contract conversion)**.** *Suppose one algorithm returns a compressed representation R and another returns an explicit object Y. A direct timing comparison is valid only if*

$$\mathrm{decode}(R) = Y$$

and the time required by decode *is included, or if the downstream task consumes R directly.*

Proof. Without decoding, the two procedures return different mathematical objects. A timing ratio between different outputs does not compare implementations of the same map. Including the decoding

cost gives two procedures with the same output. If the next operation consumes R, decoding is not part of the declared computation. □

Remark 2.10. A compressed product and an explicit matrix product may both be exact. They still have different contracts. This distinction becomes decisive when materialization costs n^2 entries.

3 Arithmetic, storage, and data movement

We count one scalar multiplication and one scalar addition as two modeled floating point operations. These counts describe a declared event model for a representation. They do not predict wall time without a memory and launch model. A multiply–accumulate charges two scalar operations even for the first coefficient. Thin corrections accumulate into an already computed diagonal or sparse result. Index metadata, temporary workspace, launch overhead, discovery, transfer, and materialization are accounted for separately.

Theorem 3.1 (Dense and packet application counts)**.** *A dense n by n matrix applied to q right hand sides has modeled cost*

$$\boxed{\mathrm{flop}_{\mathrm{dense}} = 2n^2q.}$$

A packet representation with active edge set E has modeled cost

$$\boxed{\mathrm{flop}_{\mathrm{packet}} = 2q \sum_{(\beta,\alpha)\in E} n_\beta n_\alpha.}$$

It stores

$$\boxed{M_{\mathrm{packet}} = \sum_{(\beta,\alpha)\in E} n_\beta n_\alpha}$$

scalar entries.

Proof. The active block $B_{\beta\alpha}$ is an n_β by n_α matrix. Applying it to an n_α by q input costs $2n_\beta n_\alpha q$ modeled operations and stores $n_\beta n_\alpha$ scalars. Summation over the active addresses gives both formulas. □

Corollary 3.2 (Equal diagonal packets)**.** *Let $p \geq 1$ and $b \geq 1$ be integers and $n = pb$. Suppose n is divided into p equal diagonal packets of width b. Then*

$$\mathrm{flop}_{\mathrm{block}} = 2qnb, \qquad M_{\mathrm{block}} = nb.$$

The dense-to-block storage reduction factor is the following; the arithmetic reduction factor has the same value when $q \geq 1$:

$$\boxed{\frac{n}{b}.}$$

Proof. There are n/b active diagonal blocks. Each contains b^2 entries. □

Definition 3.3 (Block diagonal plus low rank operator)**.** A block diagonal plus low rank operator has the form

$$A = D + UV^*,$$

where

$$D = \bigoplus_{\alpha \in I} D_\alpha, \qquad U, V \in \mathbb{C}^{n \times r}.$$

The abbreviation BDLR will be used below.

Theorem 3.4 (BDLR application and storage)**.** *For q right hand sides,*

$$AZ = DZ + U(V^*Z)$$

has modeled cost

$$\boxed{2q\left(\sum_\alpha n_\alpha^2 + 2nr\right).}$$

The representation stores

$$\boxed{\sum_\alpha n_\alpha^2 + 2nr}$$

scalars. For $n = pb$, with integers $p, b \geq 1$, the ideal dense storage reduction factor is the following; the arithmetic reduction factor has the same value when $q \geq 1$:

$$\boxed{\frac{n}{b + 2r}.}$$

Proof. The block diagonal product costs

$$2q \sum_\alpha n_\alpha^2.$$

The products V^*Z and $U(V^*Z)$ cost $2nrq$ each. Storage consists of the diagonal blocks and the two low rank factors. For equal packets,

$$\sum_\alpha n_\alpha^2 = nb.$$

□

Definition 3.5 (One pass streaming model)**.** Fix a scalar size s bytes. For every active block, the one pass streaming model reads the block once, reads its source packet once, and reads and writes its target packet once. It does not assume cache reuse across edges.

Proposition 3.6 (Streaming byte count)**.** *Under definition 3.5, applying a packet operator to q right hand sides moves*

$$\boxed{B_{\text{stream}} = s \sum_{(\beta,\alpha) \in E} \left(n_\beta n_\alpha + n_\alpha q + 2n_\beta q\right)}$$

bytes.

Proof. The three terms count the matrix block, the source values, and the read plus write traffic for the target accumulation. The model counts these quantities independently for every active edge. □

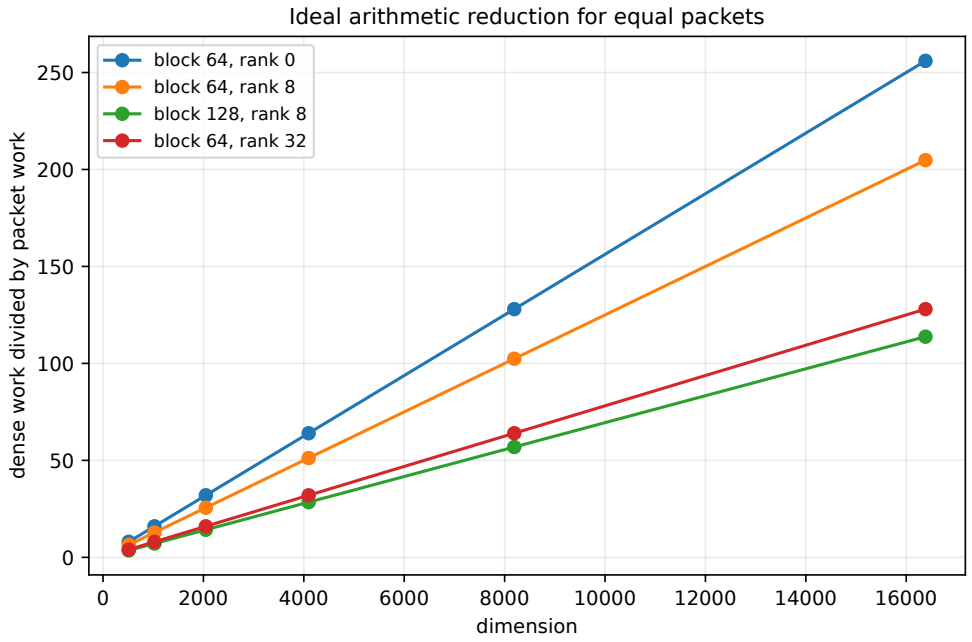


Figure 6. The ideal reduction grows with dimension only when packet width and transfer rank remain controlled.

Definition 3.7 (Arithmetic intensity)**.** When the modeled byte traffic is strictly positive, the modeled arithmetic intensity is

$$\mathcal{I} = \frac{\text{flop}}{\text{bytes}}.$$

Theorem 3.8 (Roofline lower bound)**.** *Suppose a device sustains at most $P > 0$ modeled operations per second and at most $W > 0$ bytes per second. A computation that requires F operations and B bytes satisfies*

$$\boxed{T \geq \max\left\{\frac{F}{P}, \frac{B}{W}\right\}.}$$

For positive elapsed time and positive byte traffic, its work rate is at most

$$\min\{P, W\mathcal{I}\}.$$

Proof. The operation capacity gives $T \geq F/P$. The bandwidth capacity gives $T \geq B/W$. Both restrictions hold at once. □

The roofline model is a guide rather than an exact execution law [27]. Packet structure changes both F and B. An irregular representation may remove arithmetic while lowering intensity enough to become bandwidth or launch limited.

Corollary 3.9 (Capacity threshold)**.** *Let a device have available memory $M \geq 0$ bytes and scalar size $s > 0$, and let $n \geq 0$. The matrix representation alone can fit only when*

$$n \leq \sqrt{M/s}.$$

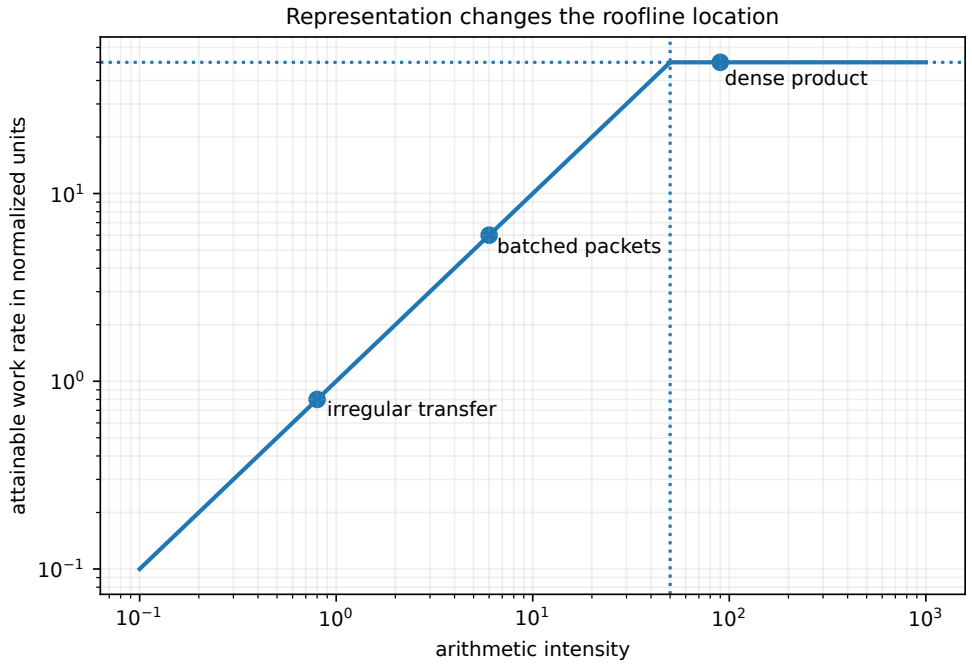


Figure 7. A representation can move the same mathematical problem between memory limited and compute limited regimes.

The equal packet BDLR representation alone fits when

$$n(b + 2r)s \leq M.$$

Proof. Substitute the dense and BDLR storage counts into the capacity inequality. □

Proposition 3.10 (Materialization cost)**.** *Materializing $D + UV^*$ as an explicit matrix requires storage for n^2 scalars and modeled arithmetic*

$$2n^2r + O(n^2)$$

when the low rank correction is formed by dense multiplication.

Proof. The product UV^* multiplies an n by r matrix by an r by n matrix. Adding the diagonal part changes only the lower order term. □

4 Structured packet primitives

The packet representation can store different exact local operator types. This section records the finite formulas used by several measured suites. The results concern representation and arithmetic. They do not assert that one implementation is fastest at every size.

Proposition 4.1 (Equal block banded count)**.** *Let $p, b \geq 1$ be integers, $n = pb$, and let $p = n/b$ equal packets have width b. Suppose a block is active precisely when*

$$|i - j| \leq w, \qquad 0 \leq w \leq p - 1.$$

The number of active blocks is

$$\boxed{N_{p,w} = p(2w+1) - w(w+1).}$$

For q right hand sides, the modeled application cost and scalar storage are

$$\boxed{2qb^2 N_{p,w} \quad \textit{and} \quad b^2 N_{p,w}.}$$

Proof. The diagonal contributes p blocks. At offset k, where $1 \le k \le w$, the upper and lower block diagonals contribute $2(p-k)$ blocks. Therefore

$$p + 2\sum_{k=1}^{w}(p-k) = p(2w+1) - w(w+1).$$

The cost and storage follow from theorem 3.1. □

Proposition 4.2 (Sparse plus low rank application)**.** *Let*

$$A = S + UV^*,$$

where S has z stored scalar nonzeros and $U, V \in \mathbb{C}^{n\times r}$. In the model that charges two operations per stored sparse entry and right hand side, applying A to q right hand sides costs

$$\boxed{2q(z + 2nr)}$$

modeled operations and stores

$$\boxed{z + 2nr}$$

scalar values, apart from sparse index metadata.

Proof. The sparse product costs $2zq$. The two dense thin products cost $2nrq$ each. The stored values are the nonzeros of S and the entries of the two factors. □

Theorem 4.3 (Circulant spectral representation)**.** *Let $n \ge 1$ be an integer and $c \in \mathbb{C}^n$, with indices taken modulo n, and define*

$$C(c)_{jk} = c_{j-k}.$$

Let F be the unitary discrete Fourier matrix

$$F_{jk} = \frac{1}{\sqrt{n}} \exp\left(-\frac{2\pi \mathrm{i} jk}{n}\right).$$

Then

$$\boxed{C(c) = F^* \operatorname{diag}(\sqrt{n}\, Fc) F.}$$

The operator is determined by n scalars and can be applied through two Fourier transforms and one pointwise product.

Proof. For each frequency k, let

$$v_j^{(k)} = \frac{1}{\sqrt{n}} \exp\left(\frac{2\pi \mathrm{i} jk}{n}\right).$$

Then

$$(C(c)v^{(k)})_j = \sum_{r=0}^{n-1} c_r v_{j-r}^{(k)} = \left(\sum_{r=0}^{n-1} c_r \exp\left(-\frac{2\pi \mathrm{i} r k}{n}\right) \right) v_j^{(k)}.$$

The eigenvalue is $\sqrt{n}(Fc)_k$. The vectors $v^{(k)}$ are the columns of F^*, which gives the formula. □

The fast Fourier transform evaluates the two transforms in $O(n \log n)$ arithmetic for the usual composite lengths [7]. This is an algorithmic factorization of the exact identity in theorem 4.3.

Corollary 4.4 (Toeplitz circulant embedding)**.** *Let $n \geq 1$ be an integer and $T \in \mathbb{C}^{n \times n}$ be Toeplitz, with first column*

$$(c_0, c_1, \dots, c_{n-1})^T$$

and first row

$$(c_0, r_1, \dots, r_{n-1}).$$

Set

$$g = (c_0, c_1, \dots, c_{n-1}, 0, r_{n-1}, \dots, r_1)^T \in \mathbb{C}^{2n}$$

and append n zeros to $x \in \mathbb{C}^n$. The first n entries of the circulant product $C(g)(x, 0)^T$ equal Tx.

Proof. For $0 \leq j < n$, the circular convolution sum separates into indices $k \leq j$, where the coefficient is c_{j-k}, and indices $k > j$, where the wrapped coefficient is r_{k-j}. The padded coordinates do not contribute. The resulting sum is exactly the jth Toeplitz row applied to x. □

Theorem 4.5 (Kronecker modewise application)**.** *Let $A \in \mathbb{C}^{m \times m}$, $B \in \mathbb{C}^{n \times n}$, and $X \in \mathbb{C}^{n \times m}$. With column major vectorization,*

$$\boxed{(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T).}$$

The right side has modeled cost

$$\boxed{2n^2m + 2nm^2}$$

and stores $n^2 + m^2$ factor entries instead of n^2m^2 entries.

Proof. The (i, j) entry of BXA^T is

$$\sum_{k=1}^{n} \sum_{\ell=1}^{m} B_{ik} X_{k\ell} A_{j\ell}.$$

This is the coordinate formula for $(A \otimes B)\,\mathrm{vec}(X)$. Multiplying first by B and then by A^T gives the stated count. □

Corollary 4.6 (Equal tensor factors)**.** *For integers $k, d \geq 0$, a product of k square factors of order d, acting on a tensor with d^k entries, can be applied by k modewise contractions, reusing intermediate slices, with modeled cost*

$$\boxed{2kd^{k+1}}$$

and factor storage kd^2.

Proof. For $k = 0$ there are no contractions and both modeled work and factor storage are zero. For $k \geq 1$, one mode has d^{k-1} fibers of length d. Multiplying each fiber by a d by d matrix costs $2d^2$, so one mode costs $2d^{k+1}$. Sum over the k modes. □

Theorem 4.7 (Walsh Hadamard recursion). *Let $N = 2^m$, and define*

$$H_1 = [1], \qquad H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}.$$

Then

$$H_N H_N^T = NI.$$

The product $H_N x$ can be evaluated in exactly $N \log_2 N$ additions or subtractions through the recursion.

Proof. The orthogonality identity follows by induction from block multiplication. One recursive stage forms $x_0 + x_1$ and $x_0 - x_1$, which uses N additions or subtractions, and then applies two transforms of size $N/2$. Thus

$$T(N) = 2T(N/2) + N, \qquad T(1) = 0,$$

whose solution is $T(N) = N \log_2 N$. □

5 Exact low rank algebra and recompression

5.1 Products and inverses

Theorem 5.1 (Exact BDLR product). *Let*

$$A = D_A + U_A V_A^*, \qquad B = D_B + U_B V_B^*$$

use the same diagonal packet layout. Then

$$AB = D_A D_B + U_C V_C^*,$$

where

$$U_C = \begin{bmatrix} D_A U_B + U_A (V_A^* U_B) & U_A \end{bmatrix}$$

and

$$V_C = \begin{bmatrix} V_B & D_B^* V_A \end{bmatrix}.$$

The unrecompressed correction rank is at most $r_A + r_B$.

Proof. Expand the product:

$$AB = D_A D_B + D_A U_B V_B^* + U_A V_A^* D_B + U_A (V_A^* U_B) V_B^*.$$

The second and fourth terms share the right factor V_B^*. The third term equals

$$U_A (D_B^* V_A)^*.$$

Concatenation gives the stated factors. □

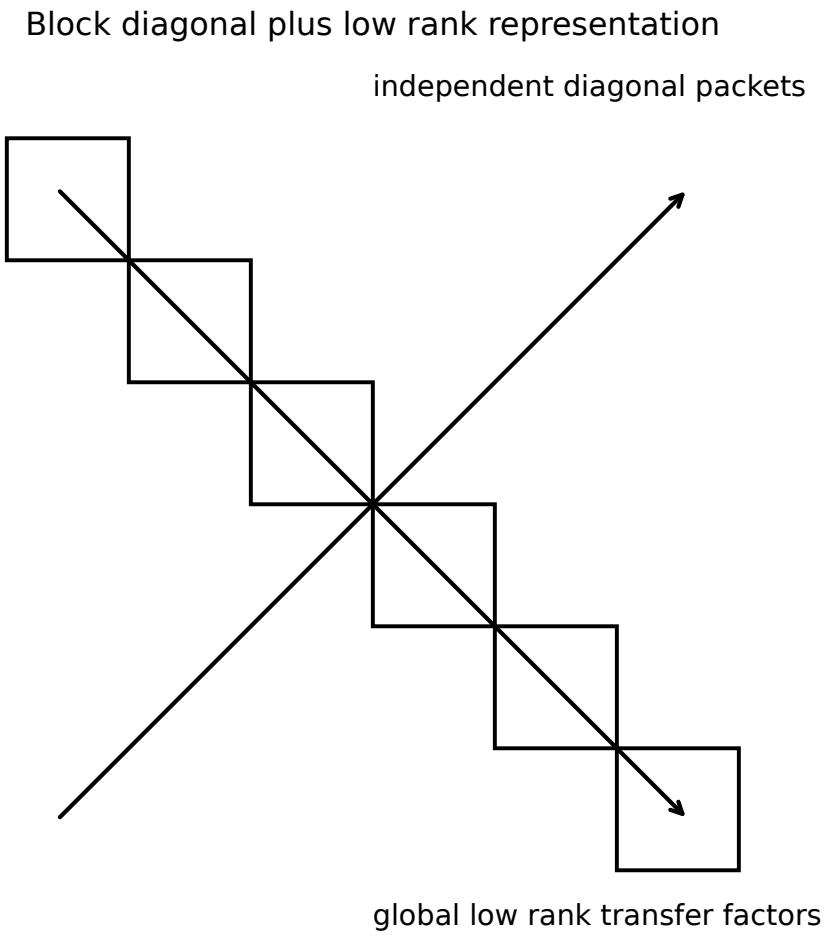


Figure 8. A regular diagonal packet structure is coupled by global low rank factors.

Corollary 5.2 (Rank growth in a product chain)**.** *Let*

$$A_j = D_j + U_j V_j^*, \qquad j = 1, \dots, m,$$

share one diagonal packet layout. Before recompression, the product

$$A_m \cdots A_1$$

has a BDLR representation whose correction rank is at most

$$\boxed{\sum_{j=1}^{m} r_j.}$$

Proof. Apply theorem 5.1 inductively. Each new factor adds at most its own correction rank. □

Theorem 5.3 (Woodbury inversion)**.** *Assume D and*

$$S = I + V^* D^{-1} U$$

are invertible. Then

$$\boxed{(D + UV^*)^{-1} = D^{-1} - D^{-1} U S^{-1} V^* D^{-1}.}$$

Proof. Multiply the proposed inverse by $D + UV^*$. The coefficient of $V^* D^{-1}$ is

$$U - U S^{-1} (I + V^* D^{-1} U) = 0.$$

The remaining term is the identity. The right inverse calculation is the same. □

The formula and its history are discussed in [15]. It reduces one large solve to packet solves with D and one dense solve with the small matrix S.

Corollary 5.4 (Inverse norm bound)**.** *Under the hypotheses of theorem 5.3,*

$$\left\| (D + UV^*)^{-1} \right\| \le \left\| D^{-1} \right\| + \left\| D^{-1} U \right\| \left\| S^{-1} \right\| \left\| V^* D^{-1} \right\|.$$

Proof. Apply the triangle inequality and submultiplicativity to the Woodbury formula. □

5.2 Optimal factor recompression

Standard Fact 5.5 (Eckart Young Mirsky theorem)**.** If M has singular values

$$\sigma_1 \ge \sigma_2 \ge \cdots,$$

pad this finite sequence with zeros. For each integer $k \ge 0$, its truncated singular value decomposition M_k is a best rank-at-most-k approximation. It satisfies

$$\left\| M - M_k \right\|_2 = \sigma_{k+1}$$

and

$$\left\| M - M_k \right\|_{\mathrm{F}} = \left(\sum_{j > k} \sigma_j^2 \right)^{1/2}.$$

This is classical [11, 22]. The next theorem shows that an n dimensional singular value decomposition is unnecessary when a correction is already factored.

Theorem 5.6 (Small core recompression)**.** *Let*

$$L \in \mathbb{C}^{m \times r}, \qquad R \in \mathbb{C}^{n \times r}.$$

Take compact QR factorizations

$$L = Q_L R_L, \qquad R = Q_R R_R,$$

where the columns of Q_L and Q_R are orthonormal. Form the small core

$$G = R_L R_R^*$$

and let

$$G = W\Sigma Z^*$$

be an ordered thin singular value decomposition, with nonnegative nonincreasing singular values. For an integer $k \geq 0$, retain $\min\{k, d\}$ columns, where d is the number of supplied thin singular directions; tail singular values beyond d are zero. Define

$$L_k = Q_L W_k \Sigma_k^{1/2}, \qquad R_k = Q_R Z_k \Sigma_k^{1/2}.$$

Then

$$L_k R_k^*$$

is a best rank at most k approximation to LR^*. Its spectral and Frobenius errors are the singular value tails in standard fact 5.5. The core dimensions equal the column ranks of the two input factors. A noncompact reduced QR may retain extra zero singular directions without changing the conclusion.

Proof. The exact factor product is

$$LR^* = Q_L G Q_R^*.$$

The two outer maps are isometries on their column spaces. They preserve the nonzero singular values and the spectral and Frobenius norms of the core. For every ambient rank-at-most-k competitor B, its compression $Q_L^* B Q_R$ has rank at most k, and compression cannot increase either norm. Thus the core best-approximation inequality also lower-bounds the error of every ambient competitor. Truncating the singular value decomposition of G and applying standard fact 5.5 gives the result. The compact QR dimensions equal the ranks of the factor column spaces. □

Remark 5.7. The large matrices enter through QR factorizations of the factors. The singular value problem has dimension at most the unrecompressed rank. Updating and recompressing low rank factors is standard numerical linear algebra [13, 4].

Theorem 5.8 (Product chain perturbation)**.** *Let $A_1, \dots, A_m$ be exact factors and let*

$$\widehat{A}_j = A_j + E_j.$$

Then

$$\boxed{\left\|\widehat{A}_m \cdots \widehat{A}_1 - A_m \cdots A_1\right\| \leq \sum_{j=1}^{m} \left(\prod_{i=j+1}^{m} \left\|\widehat{A}_i\right\|\right) \|E_j\| \left(\prod_{i=1}^{j-1} \|A_i\|\right).}$$

Empty products are one.

Proof. Use the telescoping identity

$$\widehat{A}_m \cdots \widehat{A}_1 - A_m \cdots A_1 = \sum_{j=1}^{m} \widehat{A}_m \cdots \widehat{A}_{j+1} E_j A_{j-1} \cdots A_1.$$

Take norms term by term. □

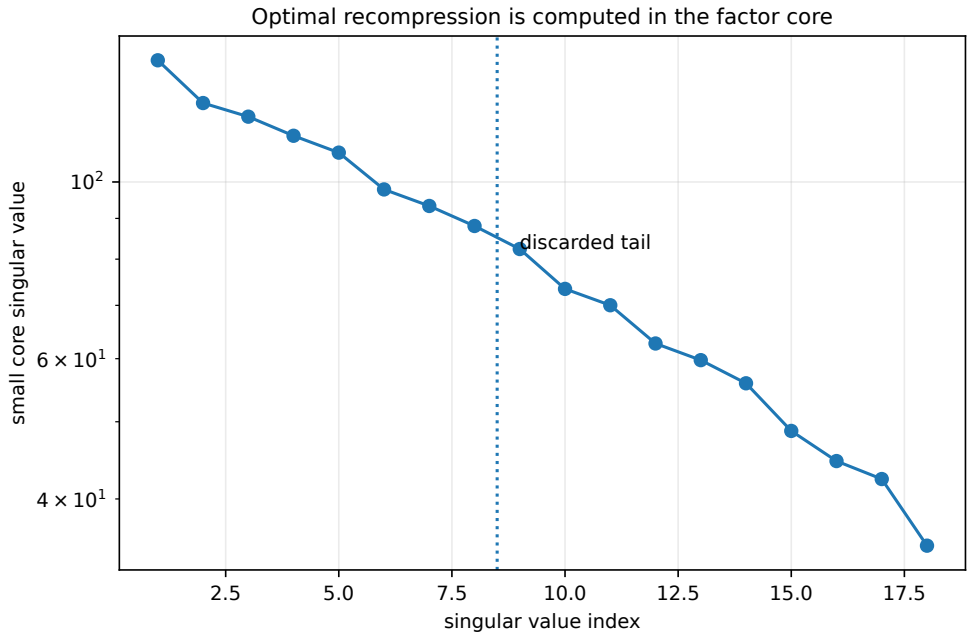


Figure 9. The approximation decision is made from the singular values of a small core whose dimension is the unrecompressed rank.

Theorem 5.9 (Certified repeated recompression)**.** *Let*

$$P_0 = I, \qquad P_j = A_j P_{j-1},$$

be exact partial products. Suppose an implemented chain satisfies

$$\widehat{P}_0 = I, \qquad \widehat{P}_j = A_j \widehat{P}_{j-1} + E_j,$$

where E_j is the additive error introduced after the jth multiplication, for example by factor recompression. Then

$$\boxed{\left\|\widehat{P}_m - P_m\right\| \le \sum_{j=1}^{m} \left(\prod_{i=j+1}^{m} \|A_i\| \right) \|E_j\|.}$$

If $\|E_j\| \le \delta_j$, the same formula with δ_j gives a certified final error.

Proof. Set $D_j = \widehat{P}_j - P_j$. Then

$$D_j = A_j D_{j-1} + E_j, \qquad D_0 = 0.$$

Repeated substitution gives

$$D_m = \sum_{j=1}^{m} A_m A_{m-1} \cdots A_{j+1} E_j.$$

Taking norms yields the result. □

6 Sparse transfer graphs and schedules

6.1 Support under algebraic operations

Theorem 6.1 (Path support of powers)**.** *For every positive integer m,*

$$P_\gamma A^m P_\alpha$$

is the sum over directed walks (vertices may repeat)

$$\alpha = \alpha_0 \longrightarrow \alpha_1 \longrightarrow \cdots \longrightarrow \alpha_m = \gamma$$

in G_A, with contribution

$$A_{\alpha_m \alpha_{m-1}} \cdots A_{\alpha_1 \alpha_0}.$$

If no such path exists, the block vanishes.

Proof. Insert the packet identity between every two consecutive factors of A. Every nonzero term determines one path of length m. □

Corollary 6.2 (Polynomial support)**.** *If*

$$p(z) = \sum_{j=0}^{d} c_j z^j,$$

then

$$\operatorname{supp}(p(A))$$

is contained in the union of the support relations of A^j for $0 \le j \le d$.

Proof. Apply theorem 6.1 to every power in the polynomial. □

Corollary 6.3 (Acyclic transfer nilpotence)**.** *Let K contain only strict transfer blocks. If its support graph is acyclic and its longest directed path has h edges, then*

$$K^{h+1} = 0.$$

Proof. There is no directed path of length $h+1$. Every block of K^{h+1} therefore vanishes by theorem 6.1. □

6.2 Block triangular solves

Definition 6.4 (Topological level)**.** Let G be a directed acyclic graph. The first level contains its source vertices. Remove them. The next level contains the new sources. Continue until every vertex has been assigned.

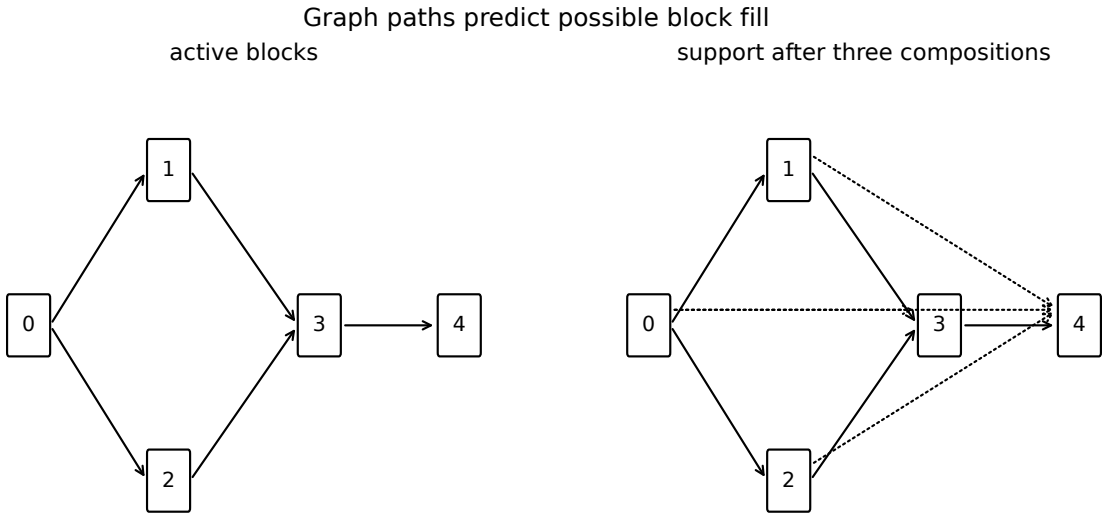


Figure 10. Composition creates possible support along directed paths. Cancellation can remove an allowed block, so the graph gives an inclusion rather than an equality.

Theorem 6.5 (Level schedule for a triangular solve)**.** *Let $A = D + K$ be block lower triangular after a topological ordering. Assume every diagonal block D_α is invertible. The solve*

$$Ax = b$$

can be completed in one sequential stage per topological level. Blocks in the same level may be solved independently after all earlier levels are known.

Proof. The equation in packet β is

$$D_\beta x_\beta = b_\beta - \sum_{\alpha \to \beta} K_{\beta\alpha} x_\alpha.$$

Every predecessor of β lies in an earlier level. Once those values are available, x_β is obtained from the local solve with D_β. Vertices in the same level have no directed edge between them, so their solves are independent. □

Corollary 6.6 (Sequential depth)**.** *For a nonempty finite graph, the number of sequential stages in the level schedule equals one plus the length of the longest directed path. The empty graph has depth zero.*

Proof. The recursive level of a vertex is one plus the largest level of its predecessors. For a nonempty graph, the largest level therefore counts the vertices on a longest path. For an empty graph, no task is scheduled and the depth is zero. □

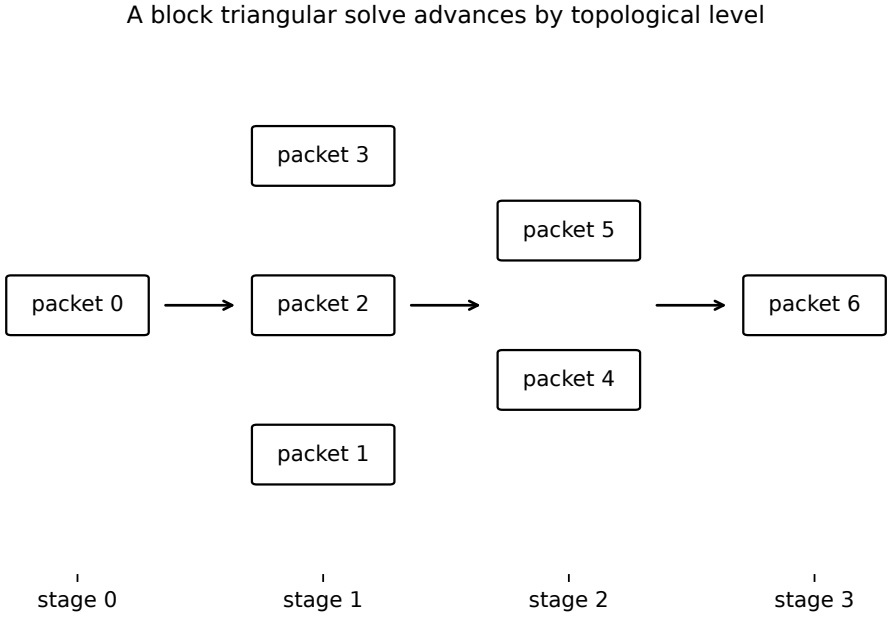


Figure 11. Packets in the same topological level can be processed concurrently in a triangular solve.

6.3 Fill under elimination

Theorem 6.7 (Inverse support lies in the transitive closure)**.** *Let A be an invertible packet operator. Reachability here includes a zero-length path from a packet to itself. If there is no directed path from packet α to packet β in G_A, then*

$$\boxed{P_\beta A^{-1} P_\alpha = 0.}$$

Proof. Let R be the set of packets reachable from α, including α, and set

$$X_R = \bigoplus_{\gamma \in R} X_\gamma.$$

No active edge leaves R, so $AX_R \subseteq X_R$. The restriction of A to the finite dimensional space X_R is injective. It is therefore onto X_R. Hence $A^{-1}X_R \subseteq X_R$. Since $\beta \notin R$, the stated block of A^{-1} vanishes. □

Proposition 6.8 (Schur support inclusion)**.** *Split the packets into retained indices R and hidden indices H. Suppose the hidden block A_{HH} is invertible. The retained Schur complement is*

$$S = A_{RR} - A_{RH} A_{HH}^{-1} A_{HR}.$$

A retained block (β, α) can be active only if it was already active in A_{RR}, or if there is a directed walk from α through hidden packets to β. The internal hidden path may have length zero, so entrance and exit may use the same hidden packet.

Proof. The correction block is

$$P_\beta A_{RH} A_{HH}^{-1} A_{HR} P_\alpha.$$

It can be nonzero only when $A_{HR}P_\alpha$ enters a hidden packet, the inverse hidden operator connects hidden packets, and $P_\beta A_{RH}$ exits to β. By theorem 6.7, the hidden inverse can connect only packets joined by a hidden directed path. Cancellation may still remove an allowed block. □

Sparse elimination and fill are treated systematically in [9]. The packet graph records the same structural issue at block scale.

7 Krylov methods and packet preconditioning

Definition 7.1 (Krylov space)**.** For a linear operator A, a vector v, and a positive integer m, define

$$\mathcal{K}_m(A, v) = \operatorname{span}\{v, Av, \dots, A^{m-1}v\}.$$

Proposition 7.2 (Invariant packet Krylov decomposition)**.** *Assume*

$$[A, P_\alpha] = 0$$

for every packet. If

$$v = \sum_\alpha v_\alpha, \qquad v_\alpha = P_\alpha v,$$

then

$$A^j v = \sum_\alpha A_\alpha^j v_\alpha$$

for every $j \geq 0$, where

$$A_\alpha = P_\alpha A P_\alpha.$$

Hence

$$\mathcal{K}_m(A, v) \subseteq \bigoplus_\alpha \mathcal{K}_m(A_\alpha, v_\alpha).$$

Proof. Commutation gives

$$A^j P_\alpha = P_\alpha A^j.$$

Apply this identity to each packet component of v. □

Theorem 7.3 (Approximate packet Krylov bound)**.** *Write*

$$A = D + K,$$

where D is packet diagonal. For every positive integer j,

$$\boxed{\left\|A^j - D^j\right\| \leq j M^{j-1} \left\|K\right\|,}$$

where

$$M = \max\{\left\|A\right\|, \left\|D\right\|\}.$$

Consequently,

$$\operatorname{dist}(A^j v, \bigoplus_\alpha \mathcal{K}_{j+1}(D_\alpha, v_\alpha)) \leq j M^{j-1} \left\|K\right\| \left\|v\right\|.$$

Proof. Use the telescoping identity

$$A^j - D^j = \sum_{k=0}^{j-1} A^{j-1-k} K D^k.$$

Each term has norm at most $M^{j-1}\|K\|$. The vector $D^j v$ lies in the direct sum of local Krylov spaces. □

Theorem 7.4 (Block Jacobi condition bound)**.** *Let*

$$A = D + K$$

act on a nonempty finite coordinate space and be Hermitian positive definite with positive definite block diagonal part D. Here $D^{-1/2}$ is the actual inverse of the positive square root of D. Suppose

$$E = D^{-1/2} K D^{-1/2}$$

is Hermitian and

$$\|E\|_2 \le \eta < 1.$$

Then

$$\boxed{\operatorname{cond}_2(D^{-1/2} A D^{-1/2}) \le \frac{1+\eta}{1-\eta}.}$$

Proof. For every unit vector x,

$$|x^* E x| \le \eta.$$

The Rayleigh quotient of $I + E$ lies in $[1-\eta, 1+\eta]$. The quotient of the extreme eigenvalue bounds gives the result. □

Standard Fact 7.5 (Preconditioned conjugate gradient convergence)**.** Let A and M be Hermitian positive definite on a nonempty finite coordinate space, and set

$$\kappa = \operatorname{cond}_2(M^{-1/2} A M^{-1/2}).$$

In exact arithmetic, the energy norm error of preconditioned conjugate gradient satisfies

$$\|x_m - x_*\|_A \le 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \|x_0 - x_*\|_A .$$

This standard estimate is discussed in [25]. Combining it with theorem 7.4 yields an explicit iteration certificate when the normalized cross packet coupling is small.

Corollary 7.6 (Certified packet preconditioning)**.** *Under the hypotheses of theorem 7.4, the convergence factor in standard fact 7.5 is bounded by the value obtained from*

$$\kappa = \frac{1+\eta}{1-\eta}.$$

Proof. Insert the condition bound from theorem 7.4 into the monotone convergence factor in standard fact 7.5. □

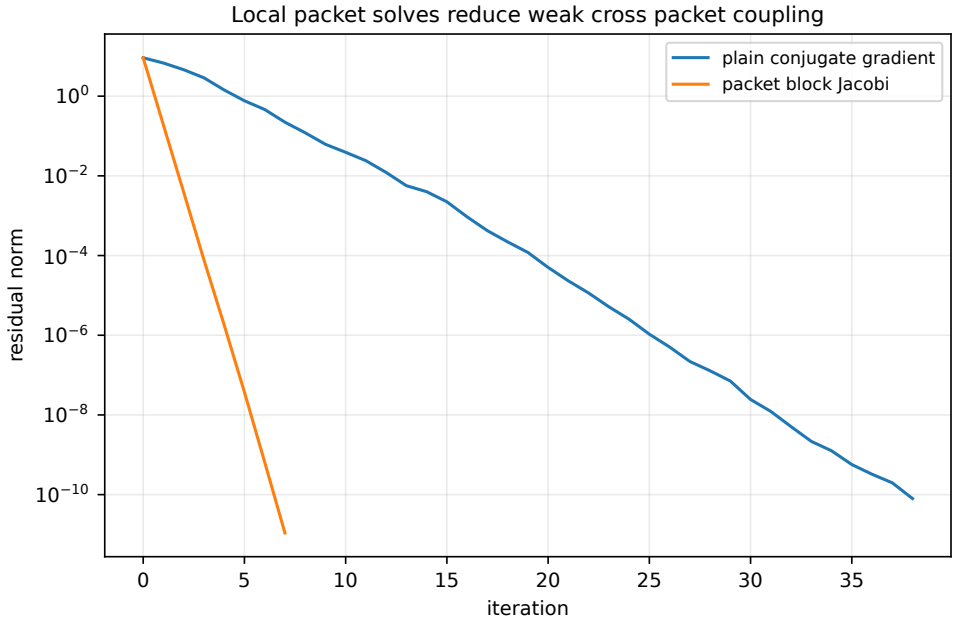


Figure 12. The block diagonal packet operator serves as a reusable local preconditioner in a weakly coupled positive system.

8 Packetwise mixed precision

8.1 One operator

Theorem 8.1 (Packet perturbation certificate). *Let*

$$A = \sum_{e \in E} A_e, \qquad \widehat{A} = \sum_{e \in E} (A_e + E_e),$$

where edge e starts at packet $\alpha(e)$, and assume

$$A_e = A_e P_{\alpha(e)}, \qquad E_e = E_e P_{\alpha(e)}.$$

Then

$$\boxed{\left\| (\widehat{A} - A)x \right\| \le \sum_{e \in E} \|E_e\| \left\| P_{\alpha(e)} x \right\|.}$$

If

$$\|E_e\| \le \varepsilon_e \|A_e\|,$$

then

$$\boxed{\left\| (\widehat{A} - A)x \right\| \le \sum_{e \in E} \varepsilon_e \|A_e\| \left\| P_{\alpha(e)} x \right\|.}$$

Proof. Exact packet evaluation gives

$$(\widehat{A} - A)x = \sum_{e} E_e P_{\alpha(e)} x.$$

Apply the triangle inequality and the operator norm bound to each term. □

Standard Fact 8.2 (Floating point matrix product model)**.** For real $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$, consider ordinary dot-product matrix multiplication in uniform precision with unit roundoff u, no overflow or underflow, and $pu < 1$. Its computed output $\widehat{C}$ satisfies the componentwise forward-error bound

$$|\widehat{C} - AB| \le \gamma_p |A||B|, \qquad \gamma_p = \frac{pu}{1-pu}.$$

Absolute values and the inequality are entrywise. Complex arithmetic, fused kernels, mixed accumulation, and tensor kernels require their own stated constants. This fact does not assert one common small input perturbation for every column, and an input-dependent rounded action is not automatically a fixed linear operator.

Detailed forms and their hypotheses are given in [17, 18]. For a fixed computed matrix, the bound can certify its error matrix. A rounded application to varying inputs needs a separately justified uniform certificate before it can be used as one fixed operator error in theorem 8.1.

8.2 Several operators

Corollary 8.3 (Mixed precision product chain)**.** *If each fixed linear factor A_j is replaced by a fixed packetwise linear approximation $\widehat{A}_j$, write $E_j = \widehat{A}_j - A_j = \sum_e \Delta_{j,e} P_{\alpha(j,e)}$. A uniform certificate is*

$$\|E_j\| \le \delta_j := \sum_e \|\Delta_{j,e}\| \, \|P_{\alpha(j,e)}\|.$$

For orthogonal packet projectors, $\delta_j \le \sum_e \|\Delta_{j,e}\|$. The final operator error is bounded by theorem 5.8 with these uniform δ_j. The corresponding vector error is this operator bound multiplied by the input norm.

Proof. Take the supremum over unit inputs in theorem 8.1, using $\|P_\alpha x\| \le \|P_\alpha\|\|x\|$. Substitute the resulting uniform operator error bounds into theorem 5.8; the action estimate follows from submultiplicativity. □

8.3 Continuous allocation

Theorem 8.4 (Continuous precision allocation)**.** *Let E be a finite nonempty edge set, let $s_e > 0$ be sensitivities and $c_e > 0$ be costs, and let $\tau > 0$. Consider*

$$\min_{\varepsilon_e > 0} \sum_e c_e \log \frac{1}{\varepsilon_e}$$

subject to

$$\sum_e s_e \varepsilon_e = \tau.$$

The unique minimizer is

$$\boxed{\varepsilon_e = \frac{\tau c_e}{s_e \sum_f c_f}.}$$

Proof. The Lagrangian derivative gives

$$-\frac{c_e}{\varepsilon_e} + \lambda s_e = 0.$$

Thus

$$\varepsilon_e = \frac{c_e}{\lambda s_e}.$$

The constraint gives

$$\lambda = \frac{\sum_f c_f}{\tau}.$$

The objective is strictly convex on the positive orthant, so the stationary point is the unique minimizer. □

Theorem 8.5 (Clipped precision allocation)**.** *Retain the finite nonempty edge set, strictly positive sensitivities and costs, and add strictly positive caps $m_e > 0$ with upper bounds*

$$0 < \varepsilon_e \le m_e.$$

Assume

$$0 < \tau \le \sum_e s_e m_e.$$

There is a unique $\lambda > 0$, except at the maximal tolerance endpoint, such that

$$\boxed{\varepsilon_e = \min\left\{m_e, \frac{c_e}{\lambda s_e}\right\}}$$

and

$$\sum_e s_e \varepsilon_e = \tau.$$

This vector minimizes the same objective over the clipped feasible set.

Proof. The Karush Kuhn Tucker conditions give the unconstrained form on every nonactive coordinate and equality with m_e on every active upper bound. The weighted sum

$$\phi(\lambda) = \sum_e s_e \min\left\{m_e, \frac{c_e}{\lambda s_e}\right\}$$

is continuous and nonincreasing. It decreases from the maximal feasible value to zero. For a strictly submaximal budget, once any coordinate is unsaturated, ϕ is strictly decreasing, which gives a unique multiplier. At maximal budget the only feasible vector is $(m_e)_e$ and every $0 < \lambda \le \min_e c_e/(s_e m_e)$ produces it. Thus the minimizing vector is unique at this endpoint while its multiplier need not be. Strict convexity gives uniqueness of the minimizer in every case. □

8.4 Discrete data types

Definition 8.6 (Discrete precision option)**.** For edge e, a discrete precision option k has a measured or modeled cost $c_{e,k}$ and a nonnegative integer error weight $w_{e,k}$. Exactly one option must be selected on every edge.

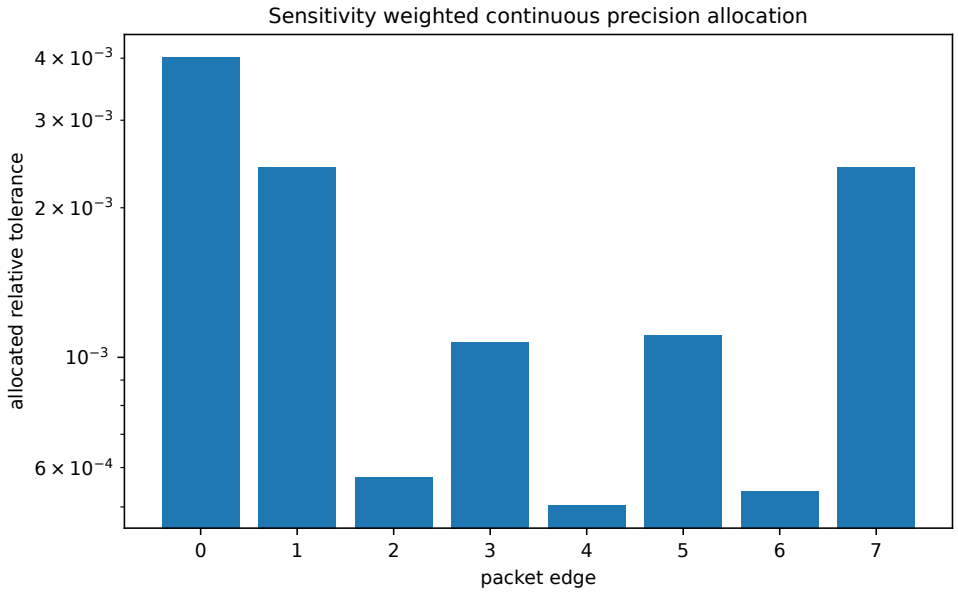


Figure 13. More sensitive packet edges receive tighter continuous tolerances.

Theorem 8.7 (Exact discrete precision program)**.** *Let the total integer error budget be B. Define*

$$F_j(b)$$

as the minimum cost for assigning precisions to the first j edges with total weight exactly b. Then

$$F_0(0) = 0, \qquad F_0(b) = +\infty \quad \text{for } b \neq 0.$$

Set $F_{j-1}(u) = +\infty$ whenever $u < 0$ or the state is unreachable. Then

$$\boxed{F_j(b) = \min_k \left\{ F_{j-1}(b - w_{j,k}) + c_{j,k} \right\}.}$$

The minimum of $F_m(b)$ over $0 \leq b \leq B$ is the exact minimum cost for the quantized model.

Proof. Every feasible assignment for the first j edges ends with one option k on edge j. Removing it leaves an optimal or suboptimal assignment for the first $j-1$ edges with residual weight $b - w_{j,k}$. Replacing a suboptimal prefix by an optimal one can only lower cost. The recurrence therefore has optimal substructure. Induction on j proves the result. □

Remark 8.8. The discrete program is exact only for the supplied options and quantized error weights. Real kernels also depend on accumulation type, scaling, overflow, underflow, tensor core availability, and shape restrictions. Mixed precision iterative refinement has its own conditioning requirements [5].

9 Discovery and exact packet compilation

9.1 Exact spectral discovery

Standard Fact 9.1 (Simultaneous spectral decomposition)**.** A finite family of pairwise commuting normal matrices admits a common orthonormal basis of joint eigenvectors. The joint eigenspaces associated with distinct eigenvalue tuples give the coarsest orthogonal packet system on which every matrix in the family acts as a scalar.

Proposition 9.2 (Exact block discovery certificate)**.** *For an orthogonal packet system (P_α), define the block diagonal compression*

$$\Pi_{\mathcal{P}}(A) = \sum_{\alpha} P_\alpha A P_\alpha.$$

Then

$$\boxed{\|A - \Pi_{\mathcal{P}}(A)\|_{\mathrm{F}}^2 = \sum_{\alpha \neq \beta} \|P_\beta A P_\alpha\|_{\mathrm{F}}^2 .}$$

In operator norm,

$$\boxed{\|A - \Pi_{\mathcal{P}}(A)\|_2 \leq \sum_{\alpha \neq \beta} \|P_\beta A P_\alpha\|_2 .}$$

Thus any proposed packet discovery has an exact finite residual certificate after its projectors are known.

Proof. Distinct matrix blocks are orthogonal in the Hilbert Schmidt inner product. Their squared Frobenius norms therefore add. The operator norm estimate follows from the triangle inequality. □

This supplies one exact discovery regime and one exact a posteriori certificate. Perturbation of spectral packets requires a gap. The Davis Kahan theorem gives quantitative control for Hermitian spectral subspaces under such a gap [8]. Randomized low rank discovery supplies another controlled regime [16, 21]. Neither result gives globally optimal packet discovery for arbitrary operators.

9.2 Candidate packet trees

Definition 9.3 (Candidate representation tree)**.** A candidate representation tree is a finite rooted tree. Every node v has

$$\kappa_v \geq 0, \qquad \epsilon_v \geq 0,$$

representing the cost and error of keeping v as one leaf. Every internal node also has a split overhead $\sigma_v \geq 0$. An admissible pruning contains exactly one selected node on every root to terminal path.

The cost and error of a pruning $\mathcal{L}$ are

$$\mathrm{cost}(\mathcal{L}) = \sum_{v \in \mathcal{L}} \kappa_v + \sum_{u \text{ split}} \sigma_u$$

and

$$\mathrm{err}(\mathcal{L}) = \sum_{v \in \mathcal{L}} \epsilon_v.$$

The model can include arithmetic, storage, launch, transfer, and precision costs inside κ_v. It remains exact only when the declared totals are additive.

Theorem 9.4 (Exact Lagrangian packet compiler)**.** *Fix $\lambda \geq 0$ and define*

$$J_\lambda(\mathcal{L}) = \mathrm{cost}(\mathcal{L}) + \lambda\,\mathrm{err}(\mathcal{L}).$$

For every node v, set

$$K_\lambda(v) = \kappa_v + \lambda\epsilon_v$$

and

$$S_\lambda(v) = \sigma_v + \sum_{w\in\mathrm{child}(v)} F_\lambda(w).$$

For a terminal node, set

$$F_\lambda(v) = K_\lambda(v).$$

For an internal node, the recurrence

$$\boxed{F_\lambda(v) = \min\{K_\lambda(v), S_\lambda(v)\}}$$

computes the exact minimum Lagrangian objective over every admissible pruning of the subtree rooted at v.

Proof. A pruning of the subtree rooted at v has two possible forms. It keeps v, in which case its objective is $K_\lambda(v)$. Otherwise it splits v and independently chooses an admissible pruning in every child subtree. By induction, the optimal child values are $F_\lambda(w)$. The split objective is therefore $S_\lambda(v)$. Taking the smaller value is exact. □

This recurrence is a computational version of best basis selection and Bellman optimality [3, 6]. The node costs here may include operator application, transfer, launch, precision, and output conversion rather than only a signal entropy.

Corollary 9.5 (Shared compilation for an operator family)**.** *Let operators $A^{(1)}, \ldots, A^{(m)}$ use the same candidate tree. Give operator j a nonnegative reuse weight r_j. Replace the local keep cost and error by*

$$\kappa_v = \kappa_v^{\mathrm{setup}} + \sum_{j=1}^{m} r_j \kappa_v^{(j)}$$

and

$$\epsilon_v = \sum_{j=1}^{m} r_j \epsilon_v^{(j)}.$$

Then theorem 9.4 gives the exact optimal shared pruning for the additive weighted objective.

Proof. The weighted family objective is still the sum of local leaf and split terms. Apply theorem 9.4 to the aggregated quantities. □

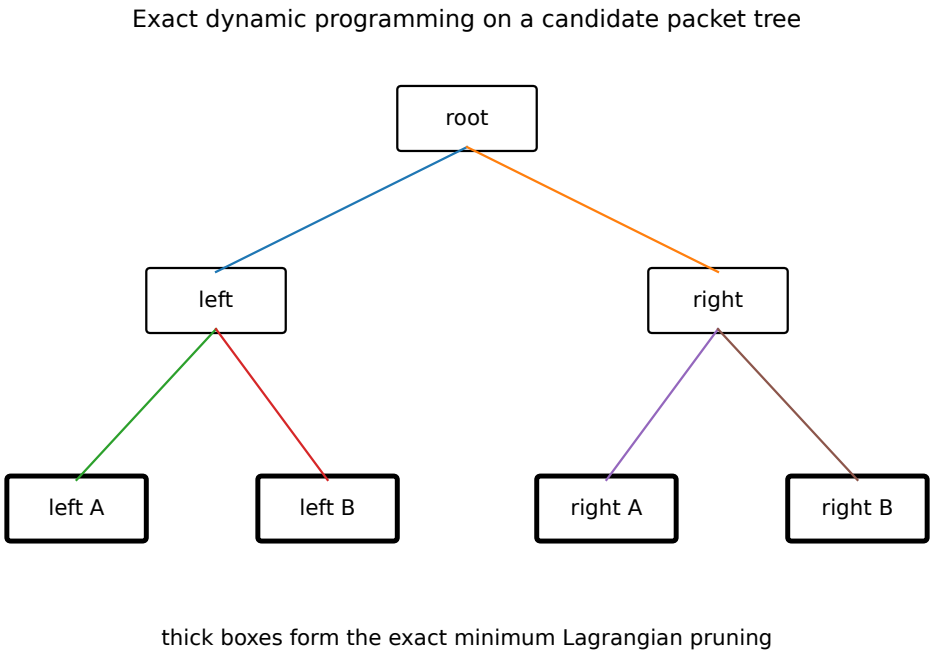


Figure 14. The thick leaves are selected by the exact Lagrangian recurrence.

Proposition 9.6 (Monotone Lagrangian frontier)**.** *Let $0 \le \lambda_1 < \lambda_2$. Choose optimal prunings $\mathcal{L}_1$ and $\mathcal{L}_2$ for the two Lagrangian parameters. Then*

$$\boxed{\operatorname{err}(\mathcal{L}_2) \le \operatorname{err}(\mathcal{L}_1)}$$

and

$$\boxed{\operatorname{cost}(\mathcal{L}_2) \ge \operatorname{cost}(\mathcal{L}_1).}$$

Proof. Optimality gives

$$\operatorname{cost}(\mathcal{L}_1) + \lambda_1 \operatorname{err}(\mathcal{L}_1) \le \operatorname{cost}(\mathcal{L}_2) + \lambda_1 \operatorname{err}(\mathcal{L}_2)$$

and

$$\operatorname{cost}(\mathcal{L}_2) + \lambda_2 \operatorname{err}(\mathcal{L}_2) \le \operatorname{cost}(\mathcal{L}_1) + \lambda_2 \operatorname{err}(\mathcal{L}_1).$$

Adding the inequalities yields

$$(\lambda_2 - \lambda_1)(\operatorname{err}(\mathcal{L}_2) - \operatorname{err}(\mathcal{L}_1)) \le 0.$$

The error ordering follows. Substitution into the first inequality gives the cost ordering. □

Corollary 9.7 (Supported Pareto points)**.** *For $\lambda > 0$, every Lagrangian optimum is Pareto optimal for the pair*

$$(\operatorname{cost}, \operatorname{err}).$$

The quantized budget compiler may recover Pareto points that are not optimal for any Lagrangian parameter.

Proof. If another pruning had no larger cost and no larger error, with one strict inequality, its Lagrangian objective would be smaller. This contradicts optimality. A finite nonconvex frontier can

contain points that are not supported by a line of slope $-\lambda$, so the budget formulation is genuinely additional. □

9.3 An exact error budget compiler

Theorem 9.8 (Quantized budget dynamic program)**.** *Assume every keep error is a nonnegative integer w_v. Let $B \in \mathbb{N}$. For each node v, define $F_v(b)$ as the minimum cost of a pruning of the subtree at v with total error exactly b. The keep option contributes*

$$F_v(w_v) \leq \kappa_v.$$

The split option is obtained by min plus convolution of the child tables, followed by addition of σ_v. The smallest value among

$$F_{\text{root}}(b), \qquad 0 \leq b \leq B,$$

is the exact minimum cost among prunings whose quantized error is at most B.

Proof. The keep case is immediate. In the split case, an admissible pruning is the union of one admissible pruning from each child. Its error is the sum of the child errors and its cost is the sum of their costs plus σ_v. Repeated min plus convolution enumerates every possible distribution of the integer budget across the child subtrees. Induction on the tree proves exactness. □

Corollary 9.9 (Pseudo polynomial complexity)**.** *If the maximum number of children is bounded and the budget is B, the quantized compiler has pseudo polynomial complexity in the number of tree nodes and B. A direct pairwise convolution implementation has quadratic dependence on B at each merge.*

Proof. Every table has at most $B+1$ entries. Merging two tables by direct convolution examines at most $(B+1)^2$ pairs. A bounded number of merges occurs at every node. □

9.4 Decision stability and local updates

Definition 9.10 (Decision gap)**.** At an internal node v, let

$$g_v = |K_\lambda(v) - S_\lambda(v)|.$$

This is the gap between the keep and split scores after the child subproblems have been solved.

Theorem 9.11 (Compiler stability)**.** *Suppose the solved keep score and split score at every internal node change in absolute value by less than $g_v/2$. These are scores after child subproblems have been solved, rather than unconstrained changes in raw local costs. Then every strict keep or split decision in the original optimal pruning remains unchanged.*

Proof. If the original keep score is lower by g_v, then after perturbation it can increase by less than $g_v/2$, while the split score can decrease by less than $g_v/2$. The order remains strict. The split case is the same. Apply the argument from the leaves upward. □

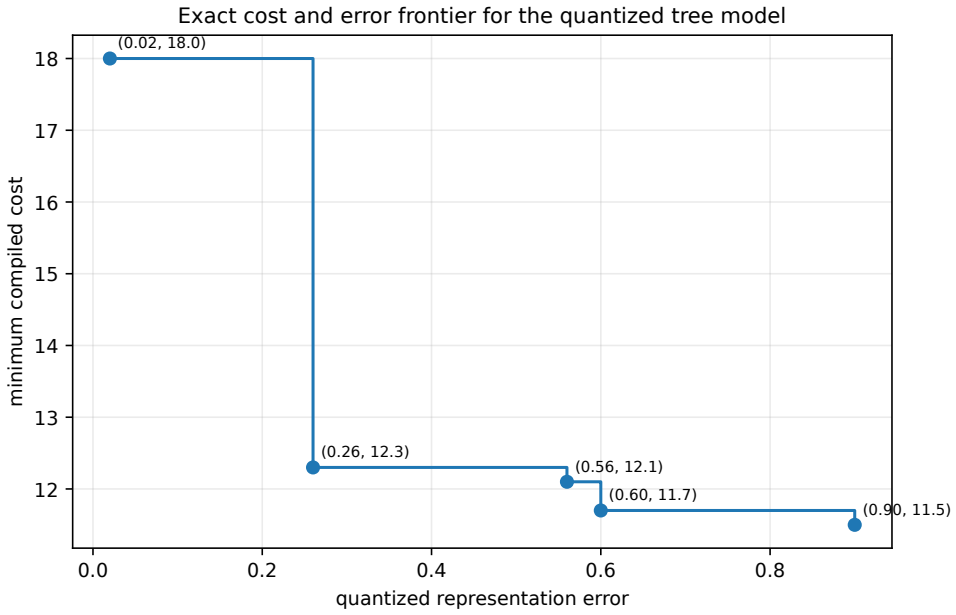


Figure 15. The quantized dynamic program produces the exact cost and error frontier for its finite model.

Theorem 9.12 (Local recompilation)**.** *Suppose local costs change only inside the subtree rooted at u, while ancestor and sibling local models are unchanged. Use correct cached sibling values from the old solve. Then every dynamic programming value outside that subtree and outside the ancestor chain from u to the root remains unchanged. Recomputing the subtree at u and that ancestor chain gives the exact new global optimum.*

Proof. A node value depends only on its own local data and the values of its children. Nodes outside the changed subtree and outside the ancestor chain have the same local data and the same child values. Their values remain unchanged. Every affected dependency lies inside the changed subtree or on the unique path from u to the root. □

Remark 9.13 (Boundary of the compiler theorem). The exact compiler requires a finite candidate tree and additive local models. Arbitrary overlapping decompositions, nonlocal cache interactions, global rank coupling, and unrestricted structure discovery can destroy optimal substructure. No universal approximation factor is proved for those settings.

10 Device scheduling and amortization

10.1 Shape classes

Definition 10.1 (Shape class)**.** Two active packet blocks lie in the same shape class when their input dimension, output dimension, right hand side count, scalar type, accumulation type, and kernel contract agree.

Proposition 10.2 (Launch reduction by shape)**.** *Suppose B independent active blocks fall into g shape classes, and suppose the backend accepts one batched array or pointer list for each class. Grouped execution then requires at most g launches instead of B, without changing the arithmetic or the output.*

Proof. Place all blocks in one class along a batch dimension and invoke one batched kernel for that class. Every block receives the same local computation. One launch per class is sufficient. □

CUDA provides graph capture and replay for repeated dependency graphs [23]. These mechanisms reduce repeated launch work, but they do not change the operator or its arithmetic count.

10.2 Critical path

Theorem 10.3 (Acyclic schedule depth)**.** *Let a packet task graph be acyclic. Assume every task takes one normalized stage, every edge is a dependency, and an unlimited number of independent tasks may run concurrently. For a nonempty finite graph, the minimum possible number of sequential stages equals the number of vertices on a longest directed path. The empty task graph has depth zero.*

Proof. Every path gives a lower bound because its tasks must execute in order. The topological level schedule in theorem 6.5 achieves exactly this depth. With no tasks, the empty schedule has depth zero. □

10.3 One time costs

Theorem 10.4 (Amortization threshold)**.** *Let $s \geq 0$ be a one time setup cost. Let $t_b, t_c \geq 0$ be the baseline and candidate times per use and let $R \in \mathbb{N}$ be a reuse count. If $t_b > t_c$, the candidate is faster after R uses exactly when*

$$R > \frac{s}{t_b - t_c}.$$

The smallest integer reuse count is

$$\boxed{R_{\min} = \left\lfloor \frac{s}{t_b - t_c} \right\rfloor + 1.}$$

If $t_b \leq t_c$, no positive reuse count amortizes a positive setup cost.

Proof. Compare

$$s + Rt_c$$

with

$$Rt_b.$$

Rearranging gives the stated inequality. □

Corollary 10.5 (Graph replay threshold)**.** *If graph capture costs $c \geq 0$, ordinary execution costs $t_o \geq 0$, and replay costs $0 \leq t_r < t_o$, then capture is profitable when*

$$R > \frac{c}{t_o - t_r}.$$

Proof. Apply theorem 10.4 with setup cost c. □

Corollary 10.6 (Materialization threshold)**.** *Suppose a compressed representation costs $t_c \geq 0$ per downstream use, an explicit representation costs $t_e \geq 0$ per use, and one time materialization costs $m \geq 0$, with reuse count $R \in \mathbb{N}$. Materialization is profitable exactly when*

$$R > \frac{m}{t_c - t_e}$$

and $t_c > t_e$.

Proof. Compare repeated compressed use with materialization followed by repeated explicit use. □

11 Benchmark contracts and measurement protocol

Performance data are meaningful only after the mathematical and timing contracts are fixed. The protocol follows standard concerns from numerical linear algebra, floating point analysis, and machine performance modeling [17, 14, 27].

Definition 11.1 (Case signature)**.** A case signature records the dimensions, packet layout, rank, right hand side count, random seed, source values, source type, reference computation, and output contract.

Definition 11.2 (Fair match)**.** Two measured rows are a fair match when they share one case signature, return the same declared output, use the same correctness reference, and have identical timing scopes. If one row includes an extra conversion or setup cost, it is a different scope and must be reported separately or added to both rows before comparison.

Proposition 11.3 (Fair matching is an equivalence relation)**.** *On rows with complete metadata, equality of the case signature, output contract, correctness reference, and timing scope is an equivalence relation.*

Proof. Equality of each metadata field is reflexive, symmetric, and transitive. Their conjunction has the same properties. □

Definition 11.4 (Representation gain)**.** Representation gain compares dense and structured work or storage for the same operator, independently of the processor.

Definition 11.5 (Device gain)**.** Device gain compares processors for the same representation, output, and timing scope.

Definition 11.6 (Kernel gain)**.** Kernel gain compares two implementations on the same device for the same representation, output, and timing scope.

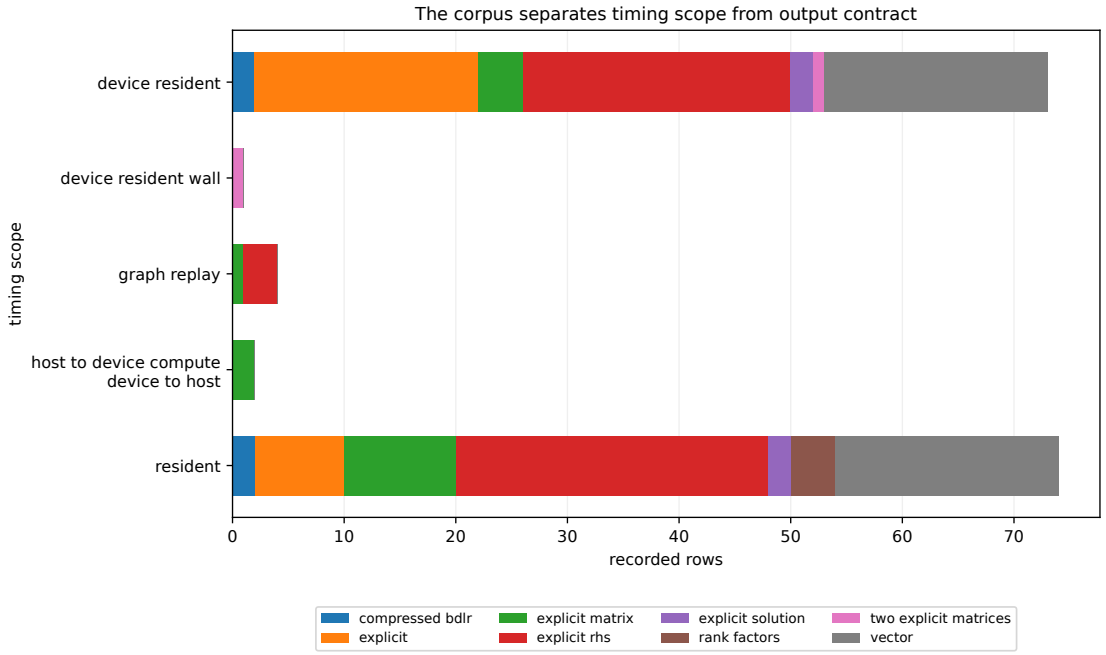


Figure 16. The corpus records timing scope and output contract as separate fields.

Suite	Rows	Suite	Rows
Dense multiplication	28	BDLR application	15
Block diagonal	12	BDLR product	12
Circulant	12	Toeplitz	12
Kronecker	12	Block banded	8
Sparse plus low rank	8	HODLR	8
Strassen	6	Transfer pipeline	5
Walsh Hadamard	4	Hidden blocks	4
Randomized low rank	4	Iterative refinement	4

Table 1. Coverage of the sanitized measured corpus.

Definition 11.7 (Amortization gain)**.** Amortization gain records the effect of one time discovery, compilation, transfer, capture, or materialization at a declared reuse count.

A measured total speedup can mix these four effects. A multiplicative decomposition is only exact when matched rows isolate the factors. Otherwise the decomposition is interpretive.

Remark 11.8 (Logical work rate). A structured method may report a dense equivalent logical work rate greater than the physical dense throughput because it does not execute the dense operations. This statistic describes avoided work. It is not a claim that the device performed those operations.

12 Recorded CPU and GPU corpus

The empirical corpus contains 154 measured rows from sixteen suites. It has 74 CPU rows and 80 GPU rows. The GPU was an NVIDIA GeForce RTX 3070 Ti Laptop GPU with compute capability 8.6 and 8 GB nominal memory. The recorded software used PyTorch 2.11.0 with a CUDA 12.8 build. These facts identify the environment. They do not predict another machine.

Method	Median time in ms	Logical GFLOP per second	Relative error
FP64	541.2803	253.91	2.18×10^{-15}
FP32	18.1309	7580.35	5.76×10^{-7}
TF32	8.6999	15797.75	2.94×10^{-4}
FP16	4.0264	34134.72	3.60×10^{-4}
BF16	4.1482	33132.00	2.88×10^{-3}

Table 2. Recorded dense accuracy and throughput at one dimension.

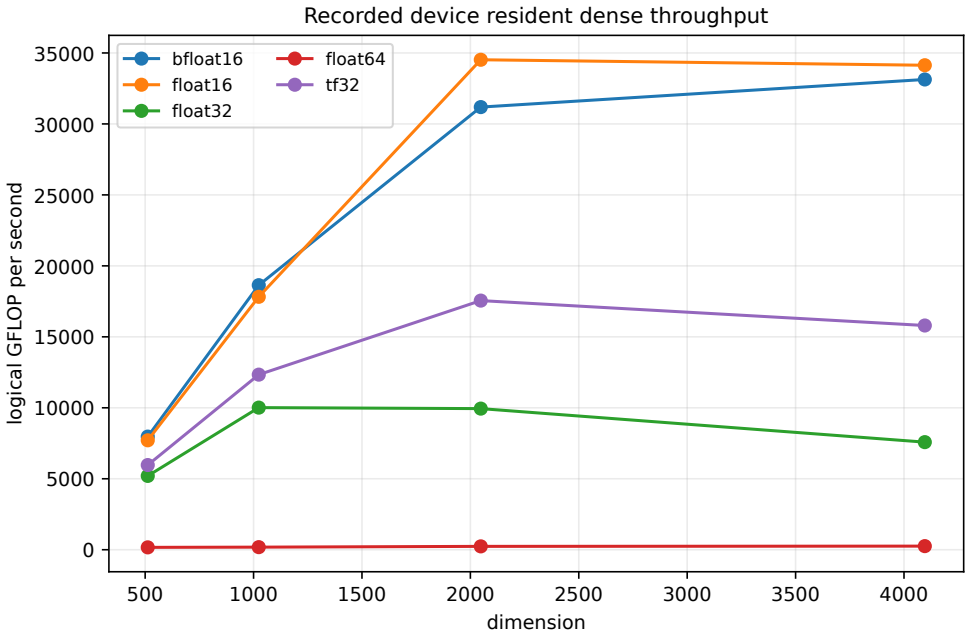


Figure 17. Recorded device resident dense throughput across dimensions and scalar types.

The 154 archived rows contain recorded case identifiers, seeds, types, output contracts, and timing scopes. Source-value hashes and explicit correctness-reference fields are absent. Setup time, discovery time, and reuse count are nonnull in only two rows each; correctness errors are present in 149 rows. Equality of available metadata cannot establish the full fair-match relation in definition 11.2. In particular, CPU-resident and GPU-device-resident scopes differ, and graph replay excludes capture while a device-resident baseline has a different raw scope. The two-stream example also compares wall time with a serial device-time scope. The values below are retained as historical observations with these limitations.

12.1 Dense precision frontier

At dimension 4096, the device resident dense rows gave the values in table 2.

The frontier is not a universal ordering of scalar types. It records one generated family. Packet precision assignment should use local sensitivities and supported kernels rather than one global type.

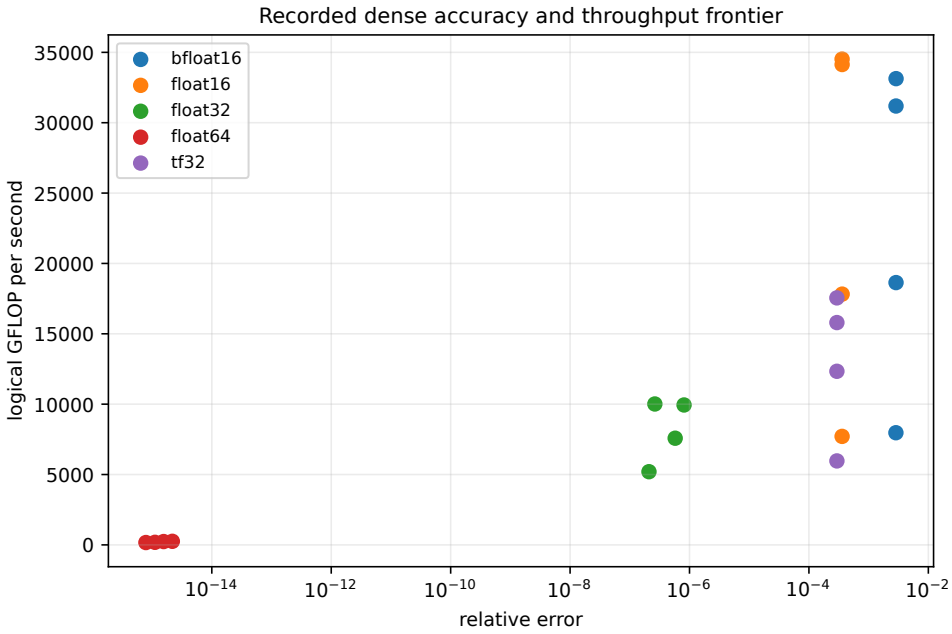


Figure 18. The speed and error tradeoff depends on both precision and problem size.

12.2 Regular packet operators

For an 8192 by 8192 equal block operator with block width 64 and 32 right hand sides, the batched packet calculation took

$$0.046032 \text{ ms}.$$

It was

$$14.8919$$

times faster than the recorded device resident dense GPU row with the same recorded case, output, and timing scope and

$$235.7404$$

times faster than the recorded CPU dense row (resident CPU scope versus device-resident GPU scope). The relative error was

$$8.17 \times 10^{-8}.$$

For BDLR application with the same dimension and right hand side count, block width 64, and rank 8, CUDA Graph replay took

$$0.079872 \text{ ms}.$$

It was

$$10.9355$$

times faster than the recorded dense GPU row (graph-replay scope versus device-resident scope) and

$$153.6527$$

times faster than the recorded CPU dense row (resident CPU scope versus device-resident GPU scope). Capture time was excluded from the replay scope and must be amortized separately.

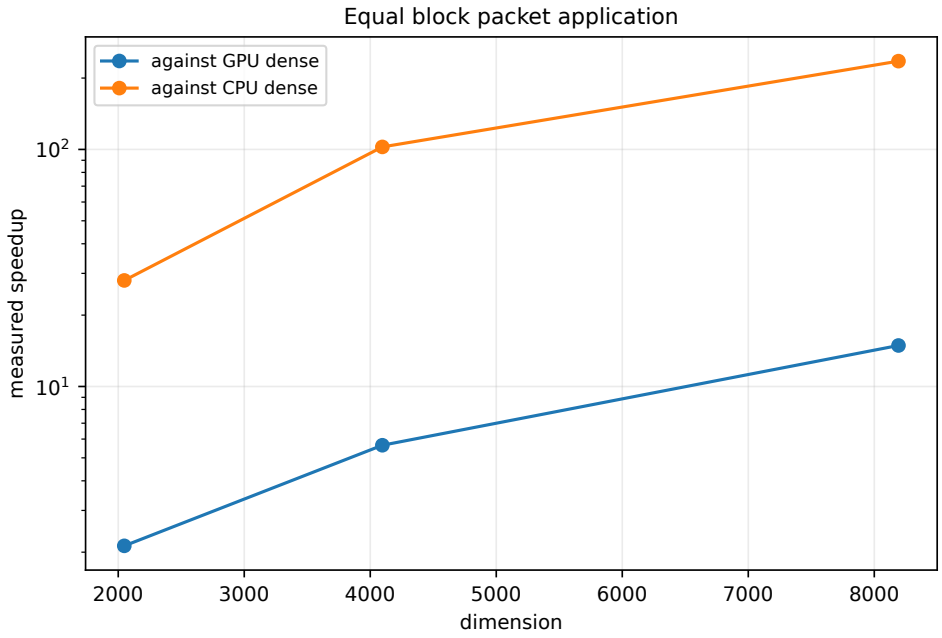


Figure 19. Regular equal shapes expose both representation reduction and batched execution.

12.3 Compressed products

A compressed BDLR product returns block and low rank factors. Materialization returns all matrix entries. The historical product rows illustrate the cost of changing the output contract; compressed and materialized outputs are separate computations. The GPU materialized BDLR rows at dimensions 512 and 1024 are sums of separately measured factor and expansion medians, as the raw notes state; they are not complete-workflow timing medians.

The algebra in theorem 5.1 explains why the compressed product can remain small. The materialization cost in proposition 3.10 explains why explicit output can erase the advantage.

12.4 Discovery and transfer

For hidden permuted blocks at dimension 512, structure discovery took 2.1102 ms, dense application took 0.0969 ms, and packet application took 0.0151 ms. The exact threshold in theorem 10.4 is 26 uses. At dimension 1024, the corresponding values were 5.2726, 0.1896, and 0.0241 ms, giving 32 uses.

For one 2048 dense FP32 product, device resident graph replay took 1.9993 ms. The end to end pageable path took 7.3036 ms. The pinned path took 6.3084 ms. Two serial device resident products took 3.6557 ms, while the tested two stream path took 4.2368 ms.

12.5 Negative controls

The corpus retains methods that lost in the recorded baseline comparisons. These are descriptive ratios; scope and complete-metadata qualifications apply.

- At dimension 512, recursive Strassen multiplication with leaf size 128 took 7.0950 ms, while the recorded OpenBLAS dense multiplication took 1.9597 ms.
- At dimension 2048, the tested recursive HODLR GPU path took 5.6079 ms, while dense GPU multiplication took 0.0860 ms.

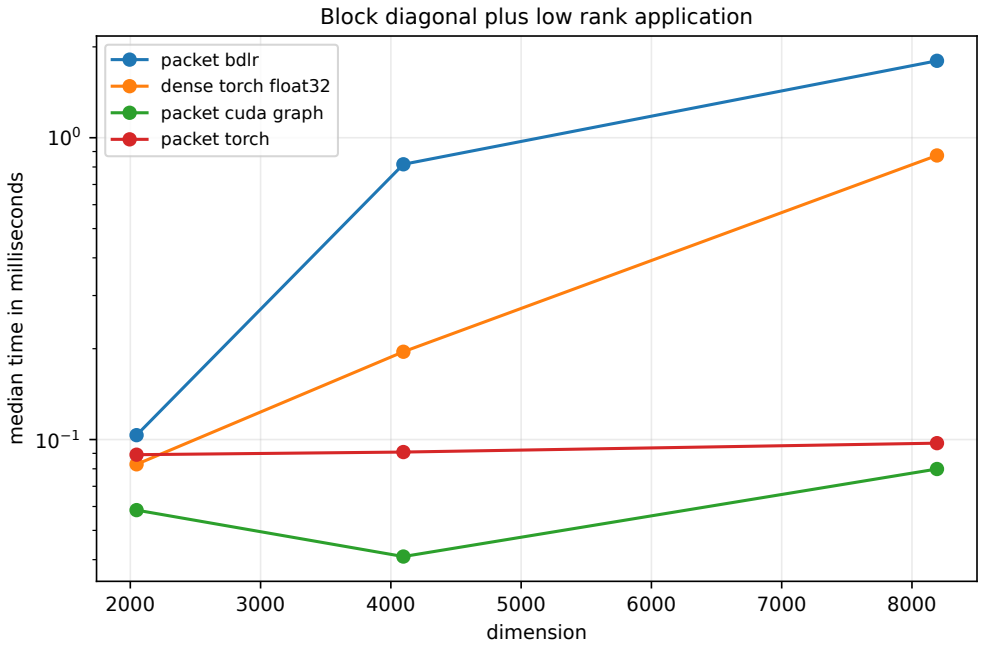


Figure 20. Representation and launch strategy both affect BDLR application time.

- The tested FP32 iterative refinement solve took 22.0848 ms, while the direct FP64 GPU solve took 12.6286 ms with a comparable final error.
- The two stream dense path was slower than the serial two product path.
- The GPU Walsh Hadamard transform was slower than the CPU implementation at length 65536, then became faster at length 262144.

Fast multiplication can be asymptotically valuable and still lose at measured sizes because of recursion, memory traffic, and kernel granularity. Hierarchical matrix methods have similar crossover issues [1]. Transform algorithms and tensor factorizations form additional structure classes [7, 24].

13 Implementation and verification

The release contains a small reference implementation of the exact operators and optimization algorithms used in the paper. The mathematical interface is deliberately narrow. This Python implementation is a numerical prototype, not extracted Lean code. Its stored-key support can retain zero-valued blocks, whereas the exact support graph uses nonzero blocks. Its reduced numerical QR is not the exact rank-compact range construction proved in Appendix F. Prototype tests and theorem proofs therefore have different targets.

1. A packet layout records dimensions and coordinate slices.
2. A packet operator stores active blocks and evaluates them without materializing inactive blocks.
3. BDLR factors support exact application, product, solve, and small core recompression.
4. Graph utilities compute support products, path closure, Schur fill inclusion, and topological levels.
5. Tree dynamic programs solve the Lagrangian and quantized budget models.

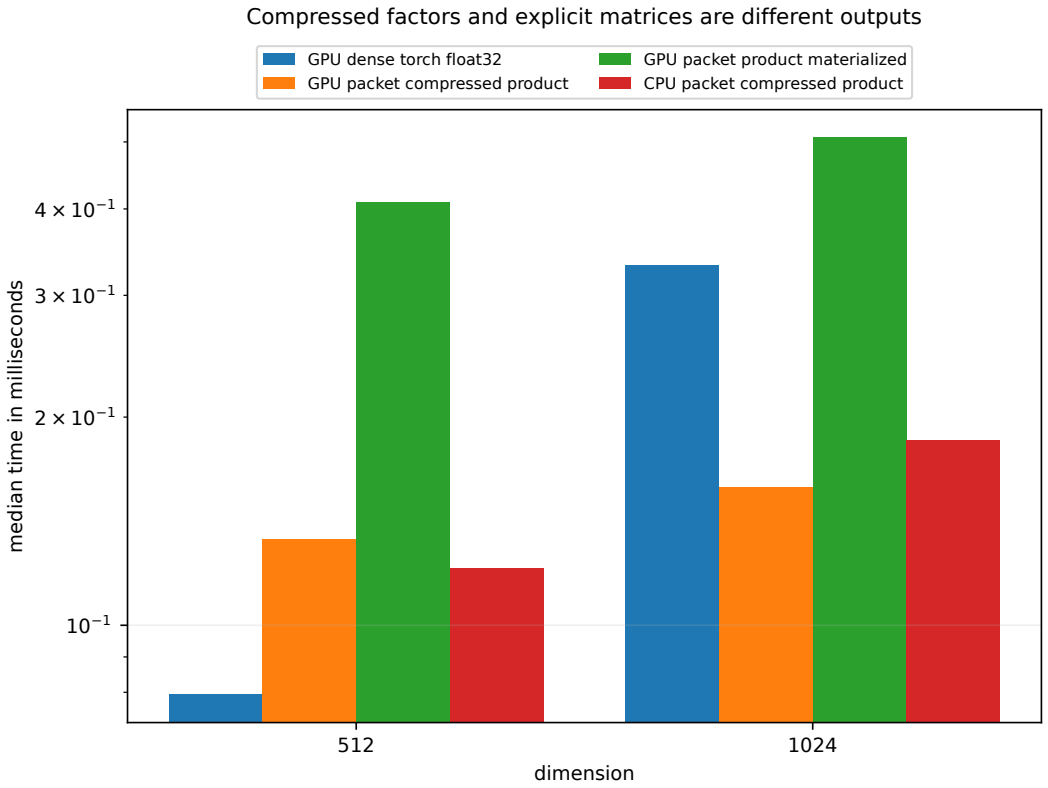


Figure 21. Compressed and explicit products answer different computational questions.

6. A multiple choice dynamic program assigns discrete precision options.
7. Benchmark utilities audit case identity, output contracts, timing scopes, and retained negative controls.

The strengthened release records a deterministic verifier with 37392 checks in nine groups. It covers packet evaluation, adjoints, composition, structured primitives, recompression optimality, product chain error, graph support, Schur fill, tree compilation, precision assignment, and the benchmark corpus. The strengthened release records eleven passing unit tests and twenty three passing original-source tests for BDLR operations, Krylov methods, mixed precision, measured data schema, and figure generation.

14 Hardware implications and exact boundaries

The finite results suggest a device interface with grouped block kernels, low rank contractions, packet metadata, a transfer graph scheduler, and per edge precision tags. Such an interface may reduce materialization and repeated launch work when the operator family remains closed under the chosen representation.

The exact results stop at a finite candidate tree and a declared device model. Three broader problems remain open.

1. A universal discovery algorithm would need to find a profitable packet structure from an arbitrary operator and data distribution. The simultaneous spectral theorem and randomized low rank approximation cover special cases only.

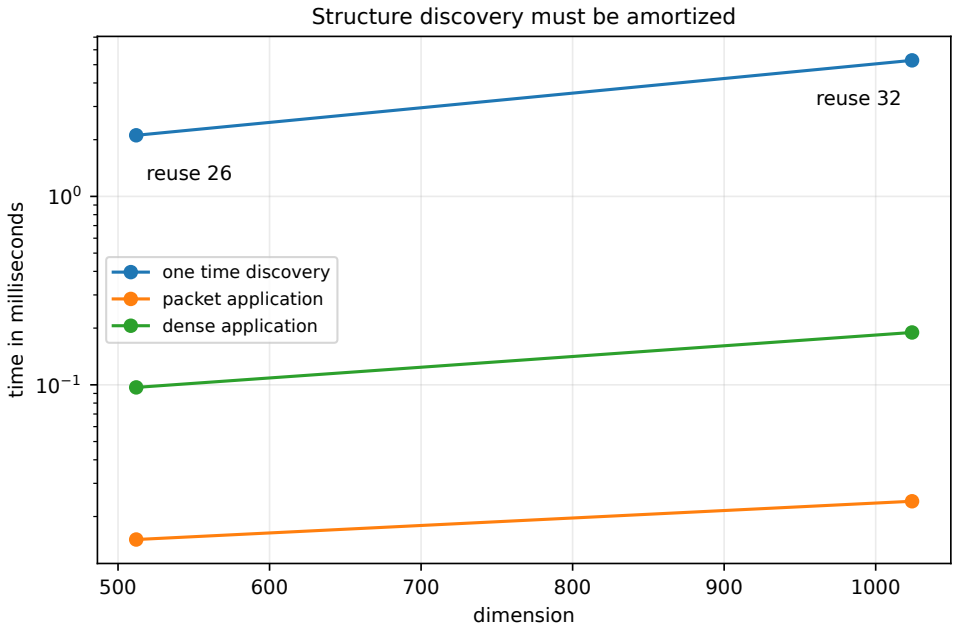


Figure 22. A profitable repeated representation can remain unprofitable for one use.

Layer	Verification target
Packet semantics	materialization, application, adjoint, composition
Low rank algebra	exact product, Woodbury solve, optimal core recompression
Graph algorithms	path support, topological levels, Schur fill inclusion
Optimization	Lagrangian tree compiler, budget compiler, precision assignment
Numerical bounds	product chain error, block perturbation, clipped allocation
Empirical corpus	row count, suite count, device split, status, error ceiling
Document	proofs, labels, citations, figures, font embedding, render audit

Table 3. Verification layers in the strengthened release.

2. A universal compiler would need a controlled approximation factor when costs are nonlocal and candidate decompositions overlap. The dynamic programs in theorems 9.4 and 9.8 do not cover that setting.

3. A packet native memory hierarchy would need a communication theorem for a family of operators and a specified architecture. The one pass byte model in proposition 3.6 is not such a lower bound.

Communication lower bounds require explicit machine models [19, 2]. CUDA Graph behavior and kernel constraints depend on the installed CUDA platform [23]. The measured results in this paper do not prove an optimal accelerator.

The article also makes no claim about the exponent of generic dense matrix multiplication. Packet algorithms answer a different question. They exploit a restricted operator class and may return a compressed output.

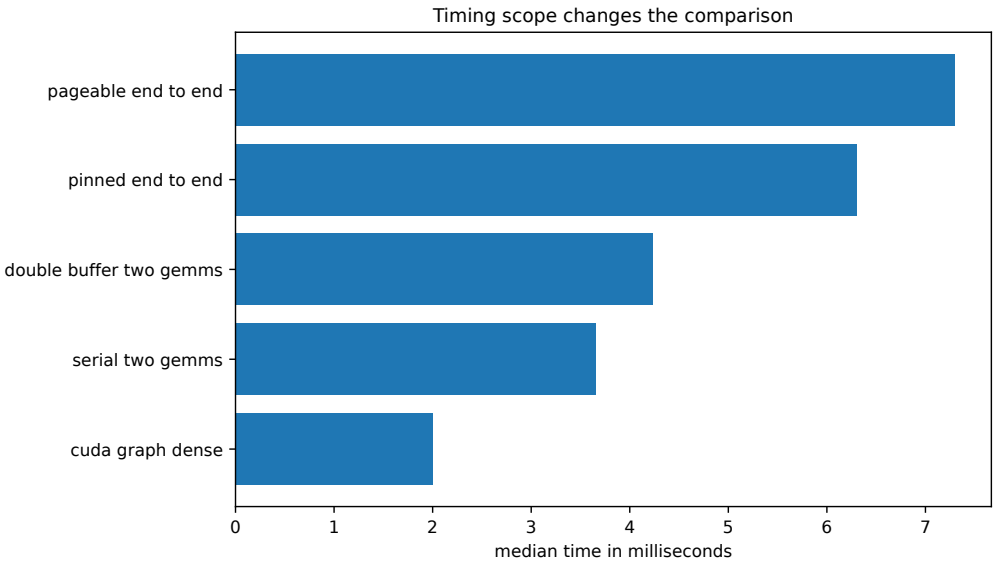


Figure 23. Device residence, transfer, graph replay, and a multioutput pipeline have different timing scopes.

15 Conclusion

A packet operator representation records which block maps exist, how they are stored, and what output they denote. Exact block evaluation and composition turn the support graph into algebra. Arithmetic, storage, and one pass data movement follow from the active block dimensions. Block diagonal plus low rank operators remain closed under multiplication before recompression. Their inverse is reduced by Woodbury, and their correction can be recompressed optimally through a small singular value problem.

Sparse transfer graphs predict path support, triangular schedule depth, and possible fill under elimination. Packet diagonal structure gives local Krylov spaces and a certified block Jacobi condition bound. Packetwise perturbations give a direct mixed precision certificate. Continuous tolerances follow from a convex allocation, while finite data type choices lead to an exact multiple choice budget program.

A finite candidate refinement tree supports a genuine packet compiler. The Lagrangian optimum, the quantized error budget optimum, and local recompilation are exact under additive hypotheses. This is narrower than unrestricted automatic discovery, but it turns one broad compiler conjecture into a proved algorithmic statement.

The measured corpus shows the practical distinction between regular and irregular structure. Equal block and BDLR operators gained substantially on the recorded GPU. Recursive HODLR, Strassen, iterative refinement, a small transform, and an attempted overlap lost to recorded baselines. Discovery, transfer, capture, and materialization changed the break even point.

The computational principle is therefore precise. Remove work through a verified representation. Preserve the declared output. Compile only inside a model whose costs and errors are stated. Assign precision by sensitivity. Amortize one time costs. Keep negative controls. Treat the final timing as a property of the representation, device, kernel, and reuse regime together.

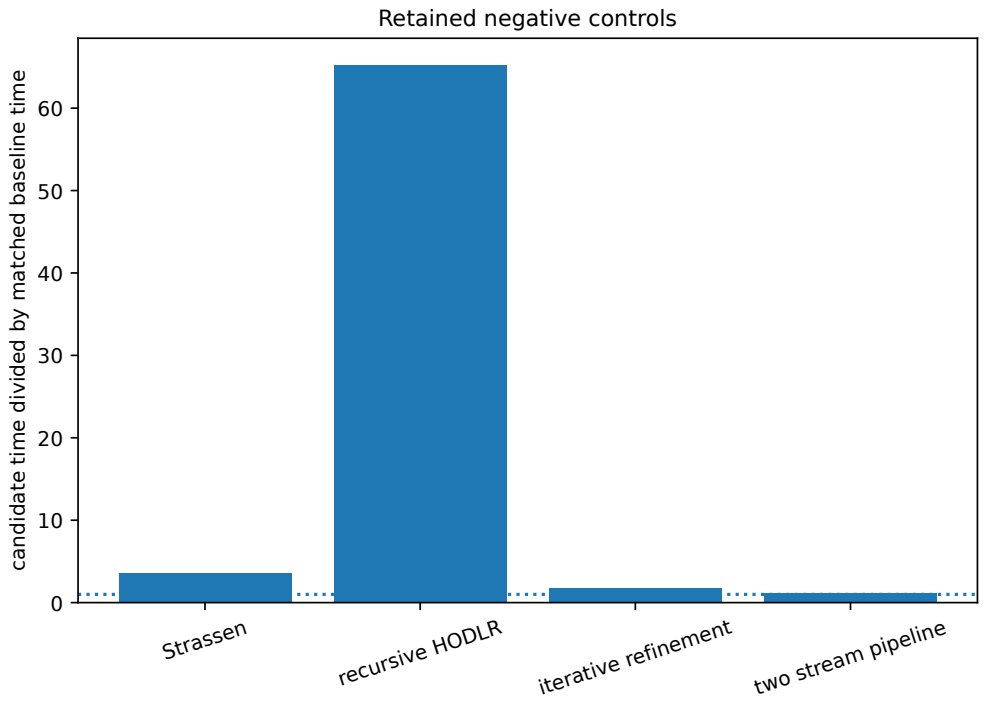


Figure 24. Values above one are slowdowns against the matched baseline.

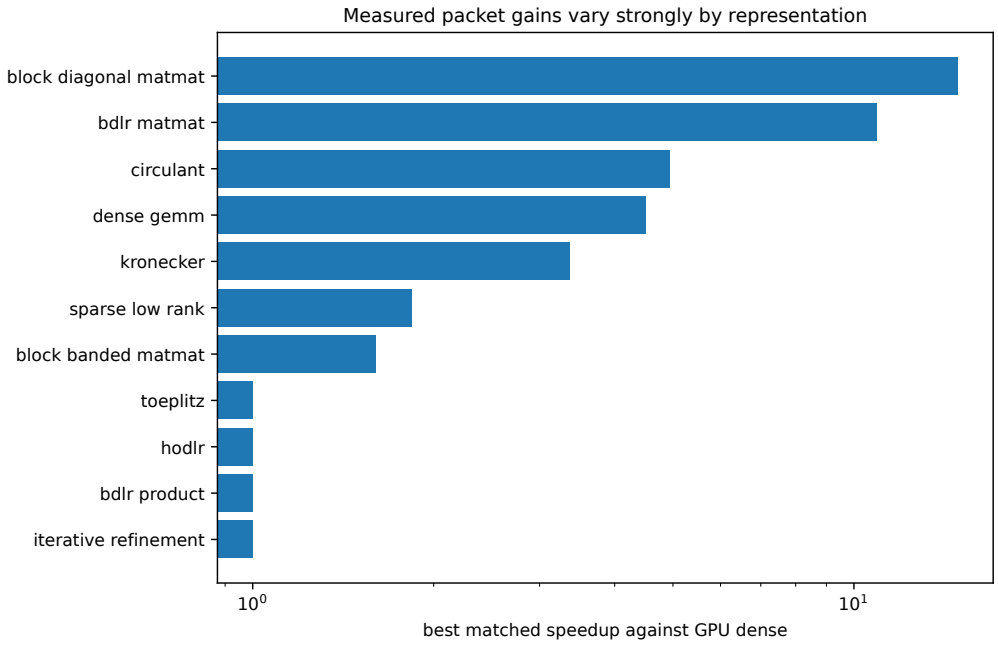


Figure 25. The best recorded gain differs strongly across suites. Regular packet structure dominates the largest measured rows.

A Exact semantics, support, and low-rank identities

This chapter retains the substantive mathematical pages of verification block 1 (v0.1.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

A.1 Exact packet semantics

Source items 2.3, 2.5, and 2.6

Let R be a ring, I a finite index type, and $(P_a)_{a\in I}$ a family satisfying

$$P_bP_a = \begin{cases} P_a & b = a, \\ 0 & b \neq a, \end{cases} \qquad \sum_a P_a = 1.$$

Write $A_{ba} = P_bAP_a$ and let $E_A = \{(b,a) : A_{ba} \neq 0\}$. These are exact algebraic zeros, not values discarded by a floating-point threshold.

Reconstruction and action. The checked finite sum is

$$\boxed{A = \sum_{(b,a)\in E_A} A_{ba}.}$$

The proof inserts the complete projector sum on both sides and removes only zero summands. Applying this identity to an arbitrary module gives exact action. A separate finite-matrix theorem establishes

$$AZ = \sum_{(b,a)\in E_A} A_{ba}Z$$

for every rectangular batch of right-hand sides Z.

Composition. Inserting the projector identity between factors yields

$$\boxed{(AB)_{ba} = \sum_c A_{bc}B_{ca}.}$$

If the left side is nonzero, at least one intermediate c has both $A_{bc} \neq 0$ and $B_{ca} \neq 0$. Consequently $\operatorname{supp}(AB) \subseteq \operatorname{supp}(A) \circ \operatorname{supp}(B)$. No equality of supports is asserted.

Adjoint and tensor products. With an involution and self-adjoint projectors, the formalization proves

$$(A_{ba})^* = (A^*)_{ab}.$$

For actual finite matrices and actual Kronecker products, it proves

$$(P_b \otimes Q_d)(A \otimes B)(P_a \otimes Q_c) = A_{ba} \otimes B_{dc}.$$

The self-adjointness hypothesis is explicit in the adjoint theorem. The tensor theorem is not a statement about an uninterpreted tensor symbol.

Concrete coordinates. Given any tag map from finite scalar coordinates to packet addresses, `coordinatePackets` constructs diagonal zero-one matrices. Their orthogonality and completeness are proved. The construction allows unequal packet sizes and unused packet addresses. Its action is checked en-

trywise: it retains exactly the coordinates carrying the selected tag.

The underlying ring-valued projector calculus is imported from the unchanged, freshly recompiled Paper VII foundation. Reuse is recorded separately from the new Paper VIII declarations.

A.2 Support propagation and nilpotence

Source items 6.1, 6.2, and 6.3

A path of length n in the Lean predicate `TransferPath` is a directed walk: vertices may repeat. This is the notion required by matrix powers. At length zero the endpoints coincide. At length $n+1$, a nonzero last block is appended to a length-n walk.

Define the active contribution sum recursively by

$$F_0(b,a) = \begin{cases} P_a & b = a, \\ 0 & b \neq a, \end{cases}$$
$$F_{n+1}(b,a) = \sum_{c:\, A_{bc} \neq 0} A_{bc} F_n(c,a).$$

The checked identity is

$$\boxed{P_b A^n P_a = F_n(b,a) \qquad (n \geq 0).}$$

Its proof is induction using the exact composition theorem, not an assumed support recurrence. When no active walk exists, the corresponding contribution sum is zero, and hence the actual block of A^n vanishes.

Nilpotence certificate. If no walk of length $h+1$ exists between any pair of packets, then

$$\boxed{A^{h+1} = 0.}$$

Every block vanishes, and reconstruction forces the whole operator to vanish. A second theorem derives the required absence of walks from a supplied level map

$$\ell : I \longrightarrow \{0, \ldots, h\}, \qquad A_{bc} \neq 0 \implies \ell(c) < \ell(b).$$

It proves that a length-n walk increases the level by at least n. This certifies the acyclic case with a bounded height. This block does not implement or verify an algorithm that constructs a topological ordering or computes the longest path.

Polynomial support. For the actual finite sum

$$p(A) = \sum_{j=0}^{d} c_j A^j,$$

the checked support lies in the union of the support relations of the powers. The formalization treats the $j = 0$ contribution explicitly, so the constant term is not accidentally omitted.

Cancellation control. Lean verifies the rational example

$$A = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \qquad A^2 = 0.$$

The coordinate block A_{00} is nonzero, and a length-two active walk from zero to zero exists. Nevertheless

the corresponding block of A^2 is zero. This is an exact counterexample to replacing support inclusion by equality.

A.3 Low-rank algebra and inversion

Source items 5.1, 5.3, and 5.4

Let $A = D_A + U_A V_A^*$ and $B = D_B + U_B V_B^*$ have rectangular factor widths r_A and r_B. The formalization constructs

$$U_C = \begin{bmatrix} D_A U_B + U_A (V_A^* U_B) & U_A \end{bmatrix},$$
$$V_C = \begin{bmatrix} V_B & D_B^* V_A \end{bmatrix}$$

and proves

$$\boxed{AB = D_A D_B + U_C V_C^*.}$$

These are actual concatenated matrices with $r_B + r_A$ columns. Expanding the product and grouping the terms with common right factors proves the identity. The separate rank theorem establishes

$$\operatorname{rank}(AB - D_A D_B) \le r_A + r_B.$$

It bounds actual matrix rank, not just the number of stored columns. Packet diagonality of $D_A D_B$ follows from diagonality of both factors and the checked block-composition formula.

Woodbury. For D and

$$S = I + V^* D^{-1} U$$

both invertible, Lean checks the ordinary matrix inverse formula

$$\boxed{(D + UV^*)^{-1} = D^{-1} - D^{-1} U S^{-1} V^* D^{-1}.}$$

The proof specializes Mathlib's classical Woodbury identity. A separate theorem derives invertibility of $D + UV^*$ from the two stated assumptions. A further theorem verifies that multiplying the proposed inverse by any batch Z solves the updated system. Invertibility of the update is not inserted as an extra premise.

Norm bound. Over the real or complex scalars, all matrix norms in the following theorem are Hilbert-space operator norms, including the rectangular factors:

$$\boxed{\left\|(D + UV^*)^{-1}\right\| \le \left\|D^{-1}\right\| + \left\|D^{-1}U\right\| \left\|S^{-1}\right\| \left\|V^* D^{-1}\right\|.}$$

After the checked inverse identity, the proof uses the triangle inequality and two rectangular operator-norm product bounds.

Not covered here. The arbitrary product-chain rank corollary, optimal small-core recompression, singular-value approximation theorem, chain perturbation bounds, and numerical stability of a particular solve implementation are not marked complete by these results. Algebraic representability is not a GPU execution-time guarantee.

A.4 Packet perturbation certificates

Source item 8.1 and two uniform consequences

Let X be a normed space over a nontrivially normed field. The formal objects are actual continuous linear maps $X \to X$, with their operator norm. Let $a(e)$ identify the source packet of edge e, and suppose

$$\Delta_e = \Delta_e P_{a(e)}.$$

Set

$$A = \sum_e A_e, \qquad \widehat{A} = \sum_e (A_e + \Delta_e).$$

The exact operator and vector identities are proved first:

$$\widehat{A} - A = \sum_e \Delta_e P_{a(e)}, \qquad (\widehat{A} - A)x = \sum_e \Delta_e (P_{a(e)} x).$$

Applying the finite triangle inequality and the defining operator-norm estimate gives

$$\boxed{\left\|(\widehat{A} - A)x\right\| \le \sum_e \|\Delta_e\| \left\|P_{a(e)} x\right\|.}$$

If $\|\Delta_e\| \le \varepsilon_e \|A_e\|$, then the formal relative certificate is

$$\boxed{\left\|(\widehat{A} - A)x\right\| \le \sum_e \varepsilon_e \|A_e\| \left\|P_{a(e)} x\right\|.}$$

The source additionally assumes that each exact block is source supported. The proof of this bound does not need that assumption. The checked statement therefore uses weaker sufficient hypotheses without weakening the conclusion.

Uniform operator estimates. Two additional checked consequences are

$$\left\|\widehat{A} - A\right\| \le \sum_e \|\Delta_e\| \left\|P_{a(e)}\right\|$$

and, when $\|P_a\| \le 1$ for every packet,

$$\boxed{\left\|\widehat{A} - A\right\| \le \sum_e \|\Delta_e\|.}$$

These estimates are uniform in the input vector and can be used as inputs to later chain analyses. The chain results themselves are not included in this block's completed inventory.

Implementation boundary. The maps Δ_e and their bounds must be supplied or justified for a numerical implementation. This formalization does not establish IEEE rounding behavior, prevent overflow or underflow, model a CUDA kernel, or certify the recorded device speedups. Those are distinct claims and require separate evidence.

A.5 Statement correction and verification

Boundary audit for item 8.4; evidence for this release

Positive budget is necessary. The source minimizes

$$\sum_e c_e \log \frac{1}{\varepsilon_e}, \qquad \varepsilon_e > 0, \qquad \sum_e s_e \varepsilon_e = \tau,$$

with positive sensitivities and costs, but it does not explicitly impose $\tau > 0$. For a nonempty edge set, Lean proves

$$s_e > 0 \text{ and } \varepsilon_e > 0 \text{ for all } e \quad \Longrightarrow \quad \sum_e s_e \varepsilon_e > 0.$$

It follows that every nonpositive budget is infeasible. A one-edge, unit-sensitivity, zero-budget instance is checked as an exact boundary example.

The amended nonempty-index statement should require $\tau > 0$. Nonemptiness also makes the sum of positive costs in the displayed optimizer's denominator positive. The empty edge set should be treated separately. **Existence, optimality, and uniqueness of the corrected optimizer are not proved in this release.** Item 8.4 remains boundary-audit-only.

Fresh Lean run. All six modules completed with exit code zero under Lean 4.34.0-rc2 and Mathlib revision

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`.

All 60 theorem-level audits and the separate coordinate-projector construction audit use only `propext`, `Classical.choice`, and `Quot.sound`. Warnings were treated as errors. The deliberately false equality and the deliberate admitted-proof placeholder were both rejected. All 17 verifier regression tests passed.

Failures are retained. Nine of the 15 captured development checks failed before correction. Their snapshots and compiler logs remain separate from the active modules and the successful final run. The initial long runtime-extraction command was also interrupted by the host limit. A second extraction completed, and the final run verified the restored runtime and dependency pins.

Reproducibility. The release includes exact sources, module-level commands, source and compiled-artifact hashes, axiom logs, the complete source-result coverage ledger, and a checkpointed verifier. The original CPU suite passed 37,392 checks. It does not reproduce GPU timings.

Remaining scope. Forty-six manuscript-proved items remain not fully covered, plus four separately tracked classical inputs. Mathlib's imported proofs were not independently rechecked, and earlier papers were not rebuilt in full. The next work can build on these semantics to address the cost models, compiler, recompression, and numerical-analysis statements without treating empirical performance as a theorem.

B Exact compilation, continuous precision, and explicit costs

This chapter retains the substantive mathematical pages of verification block 2 (v0.2.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

B.1 Exact finite-tree compilation

Source items 9.4 and 9.5

A candidate tree is either a terminal node or an internal node with a nonempty finite family of children. Arbitrary finite branching is supported. Every node carries a label from which keep cost κ_v, keep error ϵ_v, and split overhead σ_v are read.

Independent admissible objects. A legal cut is defined independently of the optimizer. It either keeps the root of its subtree or, at an internal node, chooses one legal cut in every child. A terminal node can only be kept. This is the recursive form of selecting one node on each root-to-terminal path.

The cost and error of a cut L are computed recursively. With multiplier λ, its objective is

$$J_\lambda(L) = \mathrm{cost}(L) + \lambda\,\mathrm{err}(L).$$

Lean proves that this objective equals the recursively accumulated local keep score $\kappa_v + \lambda\epsilon_v$, with σ_v charged at every split.

Recurrence. For terminal nodes,

$$F_\lambda(v) = \kappa_v + \lambda\epsilon_v.$$

For internal nodes,

$$\boxed{F_\lambda(v) = \min\left\{\kappa_v + \lambda\epsilon_v,\ \sigma_v + \sum_{w\in\mathrm{child}(v)} F_\lambda(w)\right\}.}$$

The checked result has two parts. First, $F_\lambda(v) \le J_\lambda(L)$ for every legal cut of the subtree. Second, the constructed function `compile` returns a legal cut with objective exactly $F_\lambda(v)$. Thus the minimum is attained, not merely bounded below.

The proof proceeds by induction on the actual tree and case analysis on the independently defined cut. In the split case, it sums the child lower bounds. The returned cut chooses the lower root alternative and recursively uses attaining child cuts. Ties are resolved in favor of keeping.

Shared operator families. For reuse weights r_j, the local keep model can aggregate setup and per-operator costs:

$$\kappa_v = \kappa_v^{\mathrm{setup}} + \sum_j r_j\kappa_v^{(j)}, \qquad \epsilon_v = \sum_j r_j\epsilon_v^{(j)}.$$

Lean proves the aggregation identity at the cut level, including split overhead, before applying the compiler theorem. The proof is valid over an ordered field, including $\mathbb{Q}$ and $\mathbb{R}$. The intended nonnegative model is a specialization.

No running-time theorem is asserted for the returned-cut wrapper. It can recompute subtree scores when constructing the cut. The minimum-value recurrence and the separately checked cached-update algorithm have distinct implementations and claims.

B.2 Frontiers and decision stability

Source items 9.6, 9.7, and 9.11

Let L_1, L_2 be actual optima at $0 \le \lambda_1 < \lambda_2$. Their optimality inequalities are

$$\begin{aligned} \operatorname{cost}(L_1) + \lambda_1 \operatorname{err}(L_1) &\le \operatorname{cost}(L_2) + \lambda_1 \operatorname{err}(L_2), \\ \operatorname{cost}(L_2) + \lambda_2 \operatorname{err}(L_2) &\le \operatorname{cost}(L_1) + \lambda_2 \operatorname{err}(L_1). \end{aligned}$$

Adding them and using the strict multiplier ordering gives

$$\boxed{\operatorname{err}(L_2) \le \operatorname{err}(L_1), \qquad \operatorname{cost}(L_2) \ge \operatorname{cost}(L_1).}$$

These conclusions hold for any choices among tied optima. For the cuts returned by the compiler, the required optimality premises are supplied by its constructive correctness theorem.

Supported Pareto points. For $\lambda > 0$, a scalarized optimum cannot be dominated in both cost and error with one strict improvement. Lean proves this directly from the two inequalities and positivity of λ.

The converse is not asserted. For the three cost/error points

$$(0, 2), \qquad (3/2, 1), \qquad (2, 0),$$

the middle point is undominated, but it is not selected by scalarization. A checked rational inequality gives, for every $\lambda \in \mathbb{Q}$,

$$\min\{2\lambda, 2\} < 3/2 + \lambda.$$

The quantized-budget compiler, which can recover unsupported points, remains a separate unformalized item in this release.

Strict half-gap stability. At a solved internal node write

$$K = \kappa_v + \lambda \epsilon_v, \qquad S = \sigma_v + \sum_w F_\lambda(w), \qquad g = |K - S|.$$

If the updated scores satisfy

$$|K' - K| < g/2, \qquad |S' - S| < g/2,$$

Lean proves

$$K' \le S' \quad \Longleftrightarrow \quad K \le S.$$

Applying this condition recursively at every internal node proves equality of the entire returned cuts before and after perturbation, not only equality of one numerical minimum.

The hypotheses concern *solved* split scores, including changes propagated from descendants. A bound on raw local costs alone is not substituted for this condition. At an exact tie $g = 0$, the strict inequalities cannot hold. No robustness at ties is claimed.

B.3 Cached local recompilation

Source item 9.12

The implementation represents an edit using an explicit one-hole context. Each ancestor frame stores its parent label, the untouched sibling subtrees, the child slot containing the edit, and cached sibling optimum values. The context terminates at the root.

Let v be the new optimum of the edited subtree, and let j be the edited child slot at one frame. The update performed at that frame is

$$\boxed{\mathrm{step}(v) = \min\left\{K,\ \sigma + \sum_i \begin{cases} v, & i = j, \\ \widehat{F}_i, & i \neq j. \end{cases}\right\}.}$$

This result is passed to the next ancestor. The update function does not recursively evaluate sibling subtrees. It reads their stored values $\widehat{F}_i$.

Cache validity. A context is valid when each untouched sibling cache equals the optimum for its unchanged subtree. The function `refresh` constructs such a context from the original tree model, and Lean proves its validity. This assumption concerns old, unchanged sibling computations. It is not an assumption about the desired new root optimum.

Correctness of the actual update. For any edited subtree T' and valid context C, Lean proves

$$\mathrm{updateValue}(C, F(T')) = F(\mathrm{fill}(C, T')).$$

The proof inducts through the context. At each frame, the updated child contributes the new value, and each other child contributes its unchanged cached optimum. The frame recurrence is therefore exactly the full-tree recurrence.

The result is connected back to legal cuts: the updated value is no larger than the objective of any cut of the edited whole tree, and some cut attains it. The latter witness follows from the independently proved compiler theorem.

Unchanged context. Separate theorems show that a nonedited sibling subtree is unchanged by the child replacement, and its minimum value is consequently unchanged. A localized change of keep/error/overhead data can be represented by replacing just the corresponding subtree while leaving the surrounding model fixed.

This formalizes the dependency-local claim: recompute the edited subtree, then update the ancestor frames using old sibling caches. It does not assert that a real compiler has no nonlocal cache, rank, allocation, or device-scheduling interactions. Those interactions are outside the additive tree model.

B.4 The continuous allocation theorem is complete

Source item 8.4, with an explicit positive budget

Let E be a finite nonempty edge set. Assume

$$s_e > 0, \qquad c_e > 0, \qquad \tau > 0,$$

and define $C = \sum_e c_e$. The optimization problem is

$$\min_{\varepsilon_e > 0} \sum_e c_e \log \frac{1}{\varepsilon_e} \qquad \text{subject to} \qquad \sum_e s_e \varepsilon_e = \tau.$$

Lean constructs the displayed candidate

$$\boxed{\varepsilon_e^* = \frac{\tau c_e}{s_e C}}$$

and proves strict positivity and the exact budget equality. It then proves global optimality and unique-

ness for the actual logarithmic objective.

Global comparison. For any other feasible ε, set $r_e = \varepsilon_e/\varepsilon_e^* > 0$. The budget identities give

$$\sum_e c_e r_e = C.$$

Using the standard analytic inequality $\log r \le r - 1$ yields

$$\sum_e c_e \log r_e \le \sum_e c_e(r_e - 1) = 0.$$

Lean separately proves the exact objective-difference identity

$$J(\varepsilon) - J(\varepsilon^*) = -\sum_e c_e \log r_e.$$

Therefore $J(\varepsilon) \ge J(\varepsilon^*)$.

If the feasible vectors differ, at least one $r_e \ne 1$. At that coordinate the logarithm inequality is strict. Since $c_e > 0$, the finite weighted sum is strict, giving

$$\varepsilon \ne \varepsilon^* \quad \Longrightarrow \quad J(\varepsilon) > J(\varepsilon^*).$$

The final theorem characterizes every feasible global minimizer as exactly the constructed formula. No unproved stationary-point, convexity, or KKT optimality assertion is supplied as a hypothesis.

Why the amendment matters. In the previous block, Lean proved that positive sensitivities and positive tolerances on a nonempty edge set force the budget to be positive. Nonpositive budgets are infeasible. This block completes the optimization theorem with that missing condition explicit. The source PDF is unchanged, and the amendment is recorded separately.

This is a real-valued mathematical optimizer. It is not an executable floating-point implementation. Clipped allocation and the discrete precision program remain outside this completed item.

B.5 Explicit storage and operation models

Source items 3.1, 3.2, 3.4, 3.6, 3.8–3.10, and 4.2

For a finite collection of rectangular blocks, the scalar addresses are actual elements of

$$\coprod_{e \in E} (\mathrm{Fin}(m_e) \times \mathrm{Fin}(n_e)).$$

The corresponding operation addresses include a right-hand-side index and one of two scalar-operation tags. Lean proves their cardinalities rather than merely defining a formula named “cost.”

Representation	Scalar storage	Modeled work for q right sides
Dense	n^2	$2qn^2$
Packet blocks	$\sum_e m_e n_e$	$2q\sum_e m_e n_e$
Equal diagonal blocks	nb	$2qnb$
BDLR	$\sum_i n_i^2 + 2nr$	$2q(\sum_i n_i^2 + 2nr)$
Equal-block BDLR	$n(b + 2r)$	$2qn(b + 2r)$
Sparse plus low rank	$z + 2nr$	$2q(z + 2nr)$

For equal blocks, the layout condition is $n = |I|b$. Under nonzero denominator conditions, the dense-to-block reduction is n/b and the BDLR reduction is $n/(b + 2r)$. The BDLR output thin product accumulates into the diagonal result under the declared two-operations-per-coefficient model.

Streaming. The explicit event type counts one block read, one source read, and a target read plus write for every edge. With scalar size s, its byte count is

$$B_{\text{stream}} = s \sum_e (m_e n_e + n_e q + 2m_e q).$$

No cross-edge cache reuse is assumed. Sparse index metadata and temporary workspace remain separate.

Capacity and roofline. Positive physical capacities and the inequalities $F \le PT$, $B \le WT$ give

$$T \ge \max\{F/P, B/W\}.$$

For positive bytes and elapsed time, the rate form is $F/T \le \min\{P, WF/B\}$. The dense memory bound is the exact equivalence $n^2 s \le M \iff n \le \sqrt{M/s}$ for nonnegative n, M and positive s.

Materialization. The declared dense product followed by one addition at each output entry has exactly $2n^2 r + n^2$ modeled events and n^2 output entries. The remainder is bounded by n^2 for every n, r, giving a concrete quadratic certificate for the manuscript's $O(n^2)$ term.

These theorems count a mathematical model. They do not verify CUDA instructions, wall time, memory-system behavior, or a lower bound for every implementation.

B.6 Contracts, grouping, and amortization

Source items 2.9, 10.2, 10.4–10.6, and 11.3

Output conversion. A decoded run returns the actual pair

$$(\text{decode}(\text{compressed}(x)),\ t_{\text{run}}(x) + t_{\text{decode}}(\text{compressed}(x))).$$

Under the declared output equality, Lean proves that this returns the same explicit result and charges the conversion time. It also proves the downstream composition identity when compressed output is consumed directly. Omitting a positive decode cost strictly understates the modeled explicit-output time. This formalizes the protocol; it cannot certify the truth of measured times.

Shape grouping. Active shape classes are the image of the complete shape-signature map. Their count is at most the number of active blocks. The fiberwise sum over classes equals the original block sum, both for additive outputs and for modeled work. Thus grouping neither loses nor duplicates contributions. The ability to process one class in one batched launch remains a backend capability premise.

Exact profitability. With setup s, baseline time t_b, candidate time t_c, and $t_c < t_b$, Lean proves

$$\boxed{s + Rt_c < Rt_b \quad \iff \quad R > \frac{s}{t_b - t_c}.}$$

For $s \ge 0$ and natural-number reuse counts, the smallest profitable count is

$$\boxed{R_{\min} = \left\lfloor \frac{s}{t_b - t_c} \right\rfloor + 1.}$$

It is proved both profitable and minimal. At equality there is only break-even. A checked example gives setup 6, baseline 5, candidate 3: three uses break even, and four are the first profitable count. Zero setup gives a first profitable count of one when there is a per-use advantage.

When the candidate is no faster per use, nonnegative setup and reuse cannot give strict profitability. Graph capture/replay is a direct specialization. For materialization, with nonnegative conversion cost and reuse, Lean proves the full equivalence: profitability requires both a lower explicit per-use time and reuse above the conversion threshold.

Fair matching. The benchmark signature has separate case, output-contract, correctness-reference, and timing-scope fields. Equality of all four fields is proved equivalent to signature equality, hence reflexive, symmetric, and transitive. Measured time is deliberately not a field in this equivalence. Matching metadata is not proof of measurement accuracy.

B.7 Verification and remaining work

Cumulative proof and artifact boundaries

The final run used Lean **4.34.0-rc2** with Mathlib revision

```
aeb72f1e9981b6d35b426b16dca7759be3e7ba06
```

All 11 modules returned exit code zero with warnings treated as errors. All 129 theorem-level audits and five construction audits were confined to the permitted standard axioms `propext`, `Classical.choice`, and `Quot.sound`, or subsets thereof. Active proof sources have no admitted proofs, new axioms, unsafe proofs, or native-decision bypasses. The false-equality and placeholder controls were both rejected.

Failures are retained. Eight of 13 captured development checks failed before correction. The source snapshots and diagnostics remain separate from the successful final run. The verifier's 17 regression tests passed. Archive recovery checked all seven supplied parts and their combined SHA-256 before successful extraction.

Executable checks. An independent exhaustive enumerator in Lean generated 85,320 pruning scores for 600 exact-rational cases. The recurrence was no larger than every enumerated score, attained an enumerated value, and agreed with the returned cut. A further 600 cached updates matched full recomputation. These tests do not add 600 theorems to the inventory.

Remaining items	Work not yet fully formalized
4.1, 4.3–4.7	Banded counts, circulant/Toeplitz, tensor contractions, Walsh recursion
5.2, 5.6, 5.8, 5.9	Rank growth, optimal recompression, product and repeated-recompression errors
6.5–6.8	Level solves, depth, inverse support, Schur support
7.2–7.4, 7.6	Krylov decomposition/bounds, block-Jacobi and PCG certificates
8.3, 8.5, 8.7	Mixed-precision chains, clipped allocation, discrete precision program
9.2, 9.8, 9.9	Block discovery certificate, budget compiler, its complexity
10.3	Optimal acyclic schedule depth

These are 25 proved manuscript items. The separate classical inputs are Eckart–Young–Mirsky, PCG convergence, the floating-point product model, and simultaneous spectral decomposition.

The original CPU numerical suite and GPU benchmarks were not rerun in this block. Papers I–VII were not rebuilt in full, and all imported Mathlib proofs were not independently rechecked. The release

verifies the statements and models recorded in its ledger, not the entire paper or a production GPU implementation.

Exact commands, source and dependency hashes, axiom reports, scope ledger, tests, and failed attempts accompany the ZIP. The integrity manifest detects file changes; it is not an external authenticity signature.

C Product chains, Krylov spaces, and integer budgets

This chapter retains the substantive mathematical pages of verification block 3 (v0.3.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

C.1 Low-rank factors through a product chain

Source item 5.2

Let

$$A_j = D_j + U_j V_j^*, \qquad U_j, V_j \in \mathbb{C}^{n \times r_j},$$

with all D_j diagonal in one packet layout. Set

$$P_m = A_m \cdots A_1, \qquad D^{(m)} = D_m \cdots D_1.$$

The formalization constructs thin matrices L_m, R_m such that

$$\boxed{P_m = D^{(m)} + L_m R_m^*.}$$

Their column index type is a recursively constructed disjoint sum. Its cardinality is proved to equal $\sum_{j=1}^{m} r_j$.

Construction. Suppose $P_{m-1} = D^{(m-1)} + L_{m-1} R_{m-1}^*$. The new factors are

$$\begin{aligned} L_m &= \begin{bmatrix} D_m L_{m-1} + U_m (V_m^* L_{m-1}) & U_m \end{bmatrix}, \\ R_m &= \begin{bmatrix} R_{m-1} & (D^{(m-1)})^* V_m \end{bmatrix}. \end{aligned}$$

Substitution into the already checked two-factor product identity proves exactness. This is not a rank-only existence argument. The recursively defined factors are themselves audited.

Applying the ordinary matrix-rank inequality gives

$$\boxed{\operatorname{rank}(P_m - D^{(m)}) \le \sum_{j=1}^{m} r_j.}$$

The bound concerns the actual correction matrix, not just the number of allocated columns. The proof also shows that $D^{(m)}$ stays packet diagonal. The empty chain has identity base, empty column type, and zero correction. Individual $r_j = 0$ are permitted.

What is not completed here. This is an unrecompressed representation. Optimal small-core singular-value recompression, item 5.6, remains separate. The construction can accumulate redundant columns and is not a claim of minimal factor width.

C.1.1 Actual noncommutative products

The error module defines the left-accumulated product recursively, with $P_0 = I$ and $P_{j+1} = A_{j+1}P_j$. For a perturbed sequence $\widehat{A}_j$, it derives the discrepancy recursion

$$\widehat{P}_{j+1} - P_{j+1} = \widehat{A}_{j+1}(\widehat{P}_j - P_j) + (\widehat{A}_{j+1} - A_{j+1})P_j.$$

The identity follows from the actual products. It is not supplied as an assumption to a norm estimate, and it does not commute the operators. Lean uses zero-based indices internally, so index j in source code corresponds to manuscript factor $j + 1$.

C.2 Error propagation and recompression

Source items 5.8, 5.9, and 8.3

For $\widehat{A}_j = A_j + E_j$, the checked product-chain bound is

$$\boxed{\left\|\widehat{A}_m \cdots \widehat{A}_1 - A_m \cdots A_1\right\| \le \sum_{j=1}^{m} \left(\prod_{i=j+1}^{m} \left\|\widehat{A}_i\right\|\right) \|E_j\| \left(\prod_{i=1}^{j-1} \|A_i\|\right).}$$

Empty products are one. The proof first solves a scalar gain recurrence as a finite sum, then bounds the actual discrepancy recursion by it. A version with certified upper bounds $\delta_j \ge \|E_j\|$ is included. The product-norm argument uses a norm-one identity, as in operators on a nonzero normed space.

Post-product recompression is different. Define the implemented iterates by

$$\widetilde{P}_0 = I, \qquad \widetilde{P}_j = A_j\widetilde{P}_{j-1} + E_j.$$

Here E_j is a residual introduced after multiplication. Lean derives

$$\widetilde{P}_j - P_j = A_j(\widetilde{P}_{j-1} - P_{j-1}) + E_j$$

and proves

$$\boxed{\left\|\widetilde{P}_m - P_m\right\| \le \sum_{j=1}^{m} \left(\prod_{i=j+1}^{m} \|A_i\|\right) \delta_j, \quad \|E_j\| \le \delta_j.}$$

This theorem works in a normed ring without the extra norm-one condition. It includes the empty iteration. There is no exact-prefix factor in the local residual term, because the residual is already measured at the post-product output.

Packetwise mixed precision. At each stage let

$$A_j = \sum_e A_{j,e}, \qquad \widehat{A}_j = \sum_e (A_{j,e} + \Delta_{j,e}), \qquad \Delta_{j,e} = \Delta_{j,e}P_{j,a(e)}.$$

The previous packet certificate supplies the uniform operator bound

$$\left\|\widehat{A}_j - A_j\right\| \le \delta_j := \sum_e \|\Delta_{j,e}\| \left\|P_{j,a(e)}\right\|.$$

The new theorem substitutes these δ_j into the actual product-chain estimate. The resulting vector error is at most the operator certificate times $\|x\|$. A vector-specific packet bound is not silently reused as an

operator norm.

These are deterministic implications from stated residual bounds. They do not establish the rounding behavior of a floating-point kernel. The floating-point matrix-product model remains one of the separately tracked classical inputs.

C.3 Exact and approximate packet Krylov spaces

Source items 7.2 and 7.3

Write

$$\mathcal{K}_m(A, v) = \operatorname{span}\{v, Av, \dots, A^{m-1}v\}.$$

For a complete finite projector family P_i, suppose A commutes with every P_i. Let $A_i = P_i A P_i$. Lean proves the actual orbit identity

$$A^j v = \sum_i A_i^j P_i v$$

and therefore

$$\boxed{\mathcal{K}_m(A, v) \subseteq \bigoplus_i \mathcal{K}_m(A_i, P_i v).}$$

The case $j = 0$ uses $\sum_i P_i = I$. The zeroth local power is the ambient identity, applied to $P_i v$, rather than an incorrectly substituted local projector power.

The direct sum is justified. Each local space is proved to lie in the actual range of P_i. Projecting a finite sum of local vectors selects its ith summand. Consequently a zero sum of local vectors has every summand zero. The proof does not assert equality between the global Krylov space and the full sum of local Krylov spaces.

For approximate invariance let $A = D + K$, where D is packet diagonal, and set

$$M = \max\{\|A\|, \|D\|\}.$$

For every positive integer j, the operator estimate is

$$\boxed{\left\|A^j - D^j\right\| \le j M^{j-1} \|K\|.}$$

The proof is noncommutative and uses actual powers. Its positive-index version works even in a trivial normed ring. No division by M is used, so $M = 0$ causes no exception.

Define the comparison space

$$\mathcal{L}_{j+1} = \bigoplus_i \mathcal{K}_{j+1}(P_i D P_i, P_i v).$$

Packet diagonality is first shown to imply commutation. The exact theorem then proves $D^j v \in \mathcal{L}_{j+1}$. Using that vector as a witness in the infimum distance gives

$$\boxed{\operatorname{dist}(A^j v, \mathcal{L}_{j+1}) \le j M^{j-1} \|K\| \|v\|.}$$

This statement uses Lean's actual metric infimum distance to the subspace. It does not assume that the comparison vector lies there. The block-Jacobi condition bound and the preconditioned conjugate-gradient corollary are not completed by these Krylov results.

C.4 Integer error budgets and discrete precision

Source item 8.7 and common minimum machinery

The finite budget programs use integer weights and exact ordered-field costs. They apply to rational costs for execution and to real costs for mathematical statements. Unreachable states are represented by `none`, not by a large finite sentinel.

For a predicate S on costs, the common minimum specification means:

$$\begin{aligned} &\texttt{none}: && \text{no cost satisfies } S,\\ &\texttt{some}(v): && S(v) \text{ and } v \le z \text{ whenever } S(z). \end{aligned}$$

Both attainment and minimality are included. Optional minimum and optional addition are proved to implement union and independent sum of feasible sets.

For exact-weight tables F, G, min-plus convolution is

$$(F \star G)(b) = \min_{0 \le k \le b} \{F(k) + G(b-k)\},$$

with an infeasible sum whenever either operand is infeasible. Its correctness theorem identifies every finite answer with an actual feasible pair of component weights and costs. Repeated convolution handles arbitrary finite lists of components.

Independent precision assignments. For every edge, a finite list contains its permitted pairs (w, c) of quantized error and cost. The legal assignment type chooses exactly one listed pair per edge. It is defined independently of the dynamic program. Costs and weights are then summed over that assignment.

If D is the already-solved tail table, the next edge has exact-weight recurrence

$$F(b) = \min_{(w,c) \text{ allowed},\ w \le b} \{D(b-w) + c\}.$$

The final answer is $\min_{0 \le b \le B} F(b)$. Lean proves that this is the exact minimum over legal assignments with total error at most B, or `none` exactly when no such assignment exists.

The subtraction guard matters. Natural subtraction truncates at zero. Without the condition $w \le b$, an option of weight one could be accepted at budget zero. The implemented program checks the condition before subtraction. Dedicated regressions also cover a zero-weight option, an empty option set, and the empty collection of edges. The last case has cost and error zero.

This result does not prove that an unquantized physical error model is accurately represented by integer weights. The formal optimum concerns the declared quantized model. Clipped continuous allocation, item 8.5, remains separate.

C.5 The quantized packet-tree compiler

Source item 9.8

The new program uses the previous block's actual finite `CandidateTree` and independent legal `Cut` type. An internal node has any nonempty finite family of children. A cut either keeps its node or chooses a legal cut in every child. A terminal node can only be kept.

Keeping node v costs κ_v and incurs integer error w_v. Splitting costs overhead σ_v in addition to the

selected child cuts. The exact-weight recurrence is

$$F_v(b) = \min \begin{cases} \kappa_v & \text{if } b = w_v, \\ \sigma_v + \min\limits_{\sum_i b_i = b} \sum\limits_i F_{v_i}(b_i) & \text{if } v \text{ is internal.} \end{cases}$$

Nonexistent alternatives are infeasible. The child sum is implemented by the proved finite min-plus convolution, not by assuming an optimal allocation.

Exact semantics. For every tree and every integer weight b, Lean proves that $F_v(b)$ is a minimum of costs of independent legal cuts of exact weight b. Thus each finite answer has an attaining cut and is a lower bound for every other legal cut at that weight. Infeasibility is equivalent to the absence of such a cut.

The root scan

$$\boxed{\min_{0 \le b \le B} F_{\text{root}}(b)}$$

is therefore the exact budget-constrained optimum. Increasing the budget cannot increase a finite optimum. The theorem retains feasibility hypotheses where necessary rather than treating `none` as an ordinary real cost.

Why the budget program adds something. Consider the three admissible cost/error pairs

$$(0, 4), \qquad (3, 2), \qquad (4, 0).$$

At budget two the answer is cost three. The middle pair cannot minimize cost plus λ times error: comparison with the first pair would require $\lambda \ge 3/2$, while comparison with the third would require $\lambda \le 1/2$. The executable suite checks the budget answer on an actual unary candidate tree with these cuts. It is not obtained merely by varying the previous Lagrangian compiler's multiplier.

Output boundary. This budget implementation returns the minimum cost or `none`. Attainment is proved, but executable backtracking metadata for a selected cut is not yet returned. Universal packet discovery, overlapping decompositions, nonadditive cache costs, and arbitrary device scheduling remain outside the theorem.

C.6 Cached arrays and a work certificate

Source item 9.9 and implemented forms of 8.7 and 9.8

A recursive value equation alone need not be an efficient program. Repeated calls can recompute the same subtree table. The new implementation therefore stores each table as an array-backed `Vector` with exactly $B + 1$ optional entries.

Each child table is evaluated and cached before the parent merges it. A merge reads two existing tables and tabulates their convolution. The discrete precision program likewise computes its tail table once before filling the next edge's table. The final scan reads the stored root or final-edge table.

Entrywise implementation theorem. For every $b \le B$, Lean proves

$$\text{lookup}(\text{storedTree}(B, t), b) = F_t(b).$$

There is a parallel theorem for stored discrete precision. The proof establishes first that an output entry at b depends only on input entries at most b, then proceeds through cached convolution and the tree recursion. Consequently the optimality, attainment, and infeasibility specification applies to the actual

stored result.

Counted candidates. One full convolution table examines indices

$$(b, k), \qquad 0 \le b \le B, \quad 0 \le k \le b.$$

These form a concrete finite dependent-sum type. Its cardinality is proved to be

$$\sum_{b=0}^{B}(b+1) \le (B+1)^2.$$

For a nonempty finite tree, the number of child merges plus one equals the node count N. This identity is proved for arbitrary finite branching.

Let $W_B(t)$ be the declared unit-cost count of table-cell constructions and tested min-plus candidates. The model charges the empty child-combination table as well as the output cells and merge candidates. Lean proves

$$W_B(t) \le 2(B+1)N + \big((B+1) + (B+1)^2\big)(N-1)$$

and hence

$$\boxed{W_B(t) \le N\big(3(B+1) + (B+1)^2\big).}$$

The root minimum scan adds a linear pass. Thus the declared tabulation model is pseudo-polynomial in N and the numeric budget B.

Scope of this certificate. These are table and candidate counts, not a measured execution trace or a theorem about every instruction emitted by the compiler. Rational bit complexity, allocation overhead, memory traffic, and hardware timing are not included. The certificate is not polynomial in the binary input length of B.

C.7 Verification, executable checks, and remaining work

Release evidence and precise completion boundary

The fresh cumulative run used **Lean 4.34.0-rc2** and pinned Mathlib revision

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`.

All 18 modules returned exit code zero with warnings treated as errors. Every one of the 197 theorem audits used only `propext`, `Classical.choice`, and `Quot.sound`, or subsets thereof. Twelve constructions were separately audited. The active proof files contain no admitted proofs or added axiom declarations. The deliberately false equality and admitted-proof control were both rejected.

Exact-rational executable suite. Independent enumeration, rather than the DP recurrence, produces the candidate costs used for comparison.

New test category	Count
Tree instances with unary, binary, ternary, and mixed branching	384
Exact-weight table comparisons for trees	4,992
Budget-minimum comparisons for trees	4,992
Independently enumerated legal cuts	19,152
Additional functional-specification comparisons	1,440
Discrete precision instances	288
Exact-weight / budget comparisons for precision	3,744 each
Independently enumerated legal assignments	1,453
Explicit boundary regressions	5

All passed. The preceding block's 600 compiler cases, 85,320 enumerated pruning scores, and 600 cached-update comparisons were also rerun and passed. These counts describe executable tests, not additional theorem declarations.

Failures retained. Eleven of 18 captured development compilation checks failed before correction. Their source snapshots and diagnostics are retained separately from the successful final run. They principally concern type elaboration, module visibility, finite indices, and simplification. An unsupported interactive runtime-recovery request failed before assembly started. The later noninteractive restoration succeeded.

Sixteen manuscript-proved items remain: 4.1, 4.3–4.7, 5.6, 6.5–6.8, 7.4, 7.6, 8.5, 9.2, and 10.3. These concern structured spectral and tensor primitives, optimal recompression, graph scheduling and inverse support, preconditioning, clipped allocation, and block-discovery certificates. Four classical inputs are still separate.

No GPU timings, original CPU numerical suite, full Papers I–VII rebuild, or independent recheck of imported Mathlib proofs was performed in this block. The JSON verification report, execution reports, preservation report, source ledger, logs, and scripts provide the auditable release record.

D Structured transforms and clipped precision

This chapter retains the substantive mathematical pages of verification block 4 (v0.4.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

D.1 Circulant Fourier diagonalization

Source item 4.3; module CirculantFourier

Let $n > 0$, with coordinates indexed by $\mathbb{Z}/n\mathbb{Z}$. Define the actual circulant by

$$C(c)_{jk} = c_{j-k}.$$

The formal Fourier matrix is

$$F_{jk} = \frac{1}{\sqrt{n}}\,\chi(-jk), \qquad \chi(a) = \exp\!\left(\frac{2\pi i a}{n}\right).$$

The standard character is not left abstract: a checked theorem identifies these entries with

$$\frac{1}{\sqrt{n}}\exp\!\left(-\frac{2\pi i\ \mathrm{val}(j)\,\mathrm{val}(k)}{n}\right).$$

The value representatives enter only this explicit exponential formula. All modular algebra uses the residue classes themselves.

Normalization and inversion. The proof identifies matrix application with the ordinary counting-measure discrete Fourier transform:

$$Fx = n^{-1/2}\,\mathrm{DFT}(x).$$

It separately identifies F^*x with the appropriately scaled inverse transform. Mathlib's checked Fourier inversion theorem and the proved scalar normalization yield both identities

$$\boxed{F^*F = I, \qquad FF^* = I.}$$

No unitarity assumption is inserted.

Convolution and reconstruction. An explicit bijective change of index proves

$$\mathrm{DFT}(C(c)x)_k = \mathrm{DFT}(c)_k\,\mathrm{DFT}(x)_k.$$

This gives the matrix intertwining relation $FC(c) = \mathrm{diag}(\mathrm{DFT}\,c)F$. Multiplication by the proved inverse reconstructs the circulant:

$$\boxed{C(c) = F^*\,\mathrm{diag}(\sqrt{n}\,Fc)F.}$$

The two-transform application formula is also checked for every input vector. The result includes $n = 1$ and arbitrary complex coefficients. Hermiticity is not required.

Claim boundary. This verifies the exact Fourier representation, not an $O(n \log n)$ implementation of the transform. The latter is an additional algorithmic factorization of the same linear map.

D.2 Toeplitz padding and column-major vectorization

Source items 4.4 and 4.5

Toeplitz embedding. For $n > 0$, define

$$T_{ij} = \begin{cases} c_{i-j}, & j \le i, \\ r_{j-i}, & i < j. \end{cases}$$

The diagonal is supplied by c_0; the coordinate r_0 is unused. The constructed generator of length $2n$ is

$$g = (c_0, \dots, c_{n-1}, 0, r_{n-1}, \dots, r_1),$$

and the padding map is the actual finite-index sum eliminator that appends n zeros.

Lean first proves the upper-left block identity

$$C(g)_{ij} = T_{ij} \qquad (0 \le i, j < n).$$

For $j \le i$, finite modular subtraction has its ordinary value $i - j$. For $i < j$, it has wrapped value $2n + i - j$, whose reversed generator coordinate is $j - i$. Both branches are checked with the actual finite-index arithmetic.

Splitting the product sum into the original and padded halves then gives

$$\boxed{\left(C(g)(x, 0)^T\right)_{0:n} = Tx.}$$

The generator's first half and its middle zero are checked separately. Neither symmetry nor conjugation is involved.

Kronecker application. For $A \in \mathbb{C}^{m\times m}$, $B \in \mathbb{C}^{n\times n}$ and $X \in \mathbb{C}^{n\times m}$, Lean proves

$$\boxed{(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T).}$$

The formal vectorization is explicitly column-major: its pair coordinate (j, i) stores X_{ij}. The right factor is the *ordinary transpose*, not the conjugate transpose. This distinction is essential over $\mathbb{C}$.

Finite types of multiplication/accumulation events and stored scalar addresses give

$$\boxed{2n^2m + 2nm^2 \text{ modeled operations}, \qquad n^2 + m^2 \text{ factor entries}.}$$

The explicit Kronecker matrix has n^2m^2 entries. As in the manuscript, the operation model assigns two scalar operations to each multiply-accumulate term.

D.3 Arbitrary tensor factors

Source item 4.6; module StructuredTensor

Let $A_0, \dots, A_{k-1}$ be square factors of order d. Tensor coordinates are represented by a nested product of k copies of $\{0, \dots, d-1\}$. The empty product is a singleton. Lean proves that the coordinate type has cardinality d^k.

The represented matrix is defined recursively using the actual Kronecker product:

$$M_0 = [1], \qquad M_{k+1} = A_0 \otimes M_k(A_1, \dots, A_k).$$

The algorithm is defined independently. On each slice with fixed first coordinate, it recursively contracts the trailing modes. It then applies A_0 to the resulting first-coordinate fibers:

$$y_{i,u} = \sum_{j=0}^{d-1} (A_0)_{ij} \ \mathrm{Apply}_k(A_1, \dots, A_k; x_{j,\cdot})_u.$$

Induction, the Kronecker entry formula, and finite-sum distributivity prove

$$\boxed{\mathrm{Apply}_k(A; x) = M_k(A)x.}$$

Thus correctness is not inferred merely from agreeing dimension or storage formulas.

Operation events. The event type for $k+1$ modes is a disjoint union of d copies of the trailing event type and the events for the final fiber contraction. Its cardinality obeys

$$W_0 = 0, \qquad W_{k+1} = dW_k + 2d^{k+2}.$$

Lean proves the closed form

$$\boxed{W_k = 2kd^{k+1}.}$$

The factor-address type has cardinality

$$\boxed{kd^2.}$$

These statements include $k = 0$ and $d = 0$. With no factors, the action is the identity on the singleton coordinate type and incurs zero modeled arithmetic. With $d = 0$ and at least one factor, the coordinate type is empty.

Implementation boundary. These are recursive mathematical contractions with exact complex arithmetic. The event model assumes each intermediate slice is computed once and reused. It is not an instrumented execution trace or a cost bound for every evaluator of the raw functional definition. The count excludes indexing, allocation, data movement, fused instruction effects, and the real-operation cost of a complex scalar operation. Those require a separately declared backend model.

D.4 Walsh butterfly and exact addition count

Source item 4.7; module WalshTransform

The two-point matrix is constructed as

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

and the full family is defined by $H_1 = [1]$ and

$$H_{2N} = H_2 \otimes H_N = \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix}.$$

The two-point orthogonality identity is checked explicitly. Kronecker multiplication and induction

then prove

$$\boxed{H_N H_N^T = NI, \qquad N = 2^k.}$$

The ordinary transpose is sufficient because all constructed entries are real, even though the scalar field is $\mathbb{C}$.

Actual butterfly. For the two halves x_0, x_1 of the input, the independently defined procedure returns

$$\mathrm{Walsh}_{2N}(x_0, x_1) = (\mathrm{Walsh}_N(x_0 + x_1), \mathrm{Walsh}_N(x_0 - x_1)).$$

Lean proves that this vector equals $H_{2N}x$. The argument expands the two first-coordinate cases and applies finite-sum distributivity. It does not assume that the recursive algorithm already represents the matrix.

Exact arithmetic count. Each pair of slices forms one sum and one difference at every trailing coordinate. The event type therefore gives

$$A_0 = 0, \qquad A_{k+1} = 2A_k + 2^{k+1}.$$

The closed form and its logarithmic re-expression are both checked:

$$\boxed{A_k = k2^k = N \log_2 N.}$$

The logarithm in the formal count is the natural-number binary logarithm, with its power-of-two identity proved by Mathlib. The case $N = 1$ has zero arithmetic events.

A further action theorem gives

$$H_N(H_N^T x) = Nx,$$

which identifies the inverse after division by the nonzero scalar N. This result concerns exact arithmetic, not floating-point error accumulation through a butterfly implementation.

D.5 Clipped precision: existence and multiplier uniqueness

Source item 8.5; module ClippedPrecision

Let E be a finite nonempty set and suppose

$$s_e > 0, \qquad c_e > 0, \qquad m_e > 0, \qquad 0 < \tau \le \sum_e s_e m_e.$$

The feasible set is

$$0 < \varepsilon_e \le m_e, \qquad \sum_e s_e \varepsilon_e = \tau,$$

and the objective is the same logarithmic objective used in item 8.4:

$$J(\varepsilon) = \sum_e c_e \log(1/\varepsilon_e).$$

Strictly positive caps are stated explicitly. A zero or negative cap cannot admit a positive tolerance at that coordinate.

Constructive parameterization. Write $t = 1/\lambda$ and set

$$u_e(t) = \min\{m_e, tc_e/s_e\}, \qquad \Phi(t) = \sum_e s_e u_e(t).$$

Lean proves that Φ is continuous and nondecreasing, with $\Phi(0) = 0$. At the finite positive scale

$$T = \sum_e \frac{m_e s_e}{c_e},$$

every coordinate is saturated, so $u(T) = m$ and $\Phi(T) = \sum_e s_e m_e$. The intermediate value theorem produces $t > 0$ with $\Phi(t) = \tau$. The multiplier is not assumed to exist.

Strictness below saturation. If $t < u$ and $\Phi(t)$ is below the maximum, at least one coordinate is unsaturated at t. That coordinate strictly increases; every other coordinate is nondecreasing. Hence $\Phi(t) < \Phi(u)$. This proves uniqueness of the scale and, by reciprocation, the exact manuscript statement

$$\boxed{\exists!\lambda > 0 : \quad \sum_e s_e \min\left\{m_e, \frac{c_e}{\lambda s_e}\right\} = \tau}$$

when $0 < \tau < \sum_e s_e m_e$.

At the maximal budget, the minimizing vector remains unique but the multiplier need not be. The one-coordinate case $s = c = m = 1$ is checked: both $\lambda = 1$ and $\lambda = 1/2$ produce the same tolerance 1.

D.6 Clipped precision: global optimality

Source item 8.5, continued

Existence of a multiplier alone does not prove optimality. The formal proof supplies a global supporting inequality, rather than invoking an assumed Karush–Kuhn–Tucker optimality oracle.

Fix the feasible vector $u = u(t)$ obtained above and any feasible competitor ε. For each coordinate, Lean proves

$$\boxed{c_e\left(\frac{\varepsilon_e}{u_e} - 1\right) \le \frac{s_e}{t}(\varepsilon_e - u_e).}$$

When $u_e = tc_e/s_e$, this is equality. When $u_e = m_e$, the capped condition implies $s_e/t \le c_e/m_e$, while $\varepsilon_e - m_e \le 0$. Multiplying by this nonpositive difference gives the required inequality in the correct direction.

The equal-budget constraint then yields

$$\sum_e c_e\left(\frac{\varepsilon_e}{u_e} - 1\right) \le \frac{1}{t}\sum_e s_e(\varepsilon_e - u_e) = 0.$$

Applying the checked inequality $\log z \le z - 1$ for $z > 0$ gives

$$\sum_e c_e \log(\varepsilon_e/u_e) \le 0.$$

Since

$$J(\varepsilon) - J(u) = -\sum_e c_e \log(\varepsilon_e/u_e),$$

we obtain $J(u) \le J(\varepsilon)$ for every feasible competitor.

If $\varepsilon \neq u$, one coordinate has ratio different from 1. The logarithmic inequality is strict there, and its positive weight preserves strictness in the sum. Consequently,

$$\boxed{\varepsilon \neq u \quad \Longrightarrow \quad J(u) < J(\varepsilon).}$$

This proves the unique global minimizing vector for every positive feasible budget, including the maximal endpoint.

Numerical boundary. Existence uses real continuity and the intermediate value theorem. This is a mathematical construction, not an executable floating-point multiplier search with a certified stopping tolerance. The companion exact-rational water-filling routine is an independent regression mirror and is not itself the compiled Lean construction.

D.7 Verification, regressions, and remaining work

Cumulative release evidence

Lean verification. All 23 modules were rebuilt in a fresh output directory under Lean 4.34.0-rc2 and Mathlib revision

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`.

Every final module returned exit code zero, with warnings treated as errors. The 247 theorem audits and 22 construction audits used only `propext`, `Classical.choice`, and `Quot.sound`, or subsets thereof. Active proof sources contain no admitted proofs or added axioms. The false-equality and admitted-proof negative controls were rejected. All 17 verifier regression tests passed.

Development failures. Of 20 captured development checks in this block, 12 failed and eight passed. Failures included incomplete development lemmas, import visibility, finite-index instance reduction, and warning diagnostics. Failed source snapshots and compiler logs are retained separately from the successful fresh cumulative run. The first request for interactive runtime recovery was unsupported; noninteractive archive assembly and extraction succeeded, with every part hash and the combined hash verified.

Independent regressions. The Python mirrors passed 180 exact-rational Kronecker cases, 120 tensor cases, 96 Walsh cases, 240 Toeplitz cases, and 600 clipped-allocation cases. The latter include 60 maximal-budget cases and 1,800 rejected infeasible budgets. These are finite checks, not additional universal theorems or executions of the Lean definitions.

A separate complex128 Fourier regression covered 96 cases at lengths 1 through 24. Maximum absolute errors were approximately 3.34×10^{-14} for diagonalization, 2.98×10^{-15} for unitarity, and 2.04×10^{-13} for application, all below the declared 10^{-10} threshold. These numerical checks are not substituted for the symbolic Lean proof.

Ten manuscript-proved items remain:

Item	Remaining result
4.1	Equal block banded count
5.6	Small-core optimal recompression
6.5, 6.6	Triangular level solve and sequential depth
6.7, 6.8	Inverse and Schur support inclusion
7.4, 7.6	Block-Jacobi condition bound and certified preconditioning
9.2	Exact block-discovery certificate
10.3	Acyclic schedule depth

Four classical inputs remain separately recorded. This release does not rerun the original numerical corpus or GPU benchmarks, rebuild Papers I–VII in full, or independently recheck every imported Mathlib proof.

E Constructed graph schedules, support, and preconditioning

This chapter retains the substantive mathematical pages of verification block 5 (v0.5.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

E.1 Constructing the optimal level schedule

Source items 6.6 and 10.3; module GraphSchedules

Let V be finite and let $u \to v$ mean that task v depends on task u. Assume there is no nonempty directed cycle. A topological order or valid level function is *not* supplied as an additional hypothesis.

For each vertex, define its strict ancestor set

$$\mathrm{Anc}(v) = \{u : \text{there is a positive-length path } u \rightsquigarrow v\}.$$

For every edge $u \to v$, the proof establishes

$$\mathrm{Anc}(u) \subsetneq \mathrm{Anc}(v).$$

The inclusion follows by appending the edge. It is strict because u is an ancestor of v but cannot be its own strict ancestor in an acyclic graph. Cardinality therefore decreases when following predecessors. This constructs a well-founded relation on which the level recursion is defined:

$$\boxed{\ell(v) = 1 + \max_{u \to v} \ell(u), \qquad \max \emptyset = 0.}$$

Validity and attainment. Lean proves $\ell(v) > 0$ and $\ell(u) < \ell(v)$ on every edge. A path of n edges from a to b satisfies

$$\ell(a) + n \le \ell(b).$$

Conversely, predecessor-max attainment and induction produce an actual path with $\ell(v) - 1$ edges ending at each vertex v. The result is an equality with a longest path, not merely an upper bound.

A valid schedule $s : V \to \mathbb{N}$ assigns positive stages and strictly increases on each edge. Induction proves the stronger pointwise statement

$$\ell(v) \le s(v) \quad \text{for every } v.$$

Consequently the constructed schedule achieves the least possible makespan,

$$\boxed{d = \max_{v \in V} \ell(v).}$$

For a nonempty graph, d equals the number of vertices on an attained longest path. Vertices of the same level have no dependency edge in either direction.

Empty graph and interpretation. When $V = \emptyset$, the checked depth is zero. The phrase "one plus the longest edge-path length" applies to nonempty graphs. These are ideal discrete scheduling statements. Weighted task costs, a finite processor budget, launch latency and communication are not part of this model. The definitions are mathematical constructions, not a certified hardware scheduler.

E.2 Forward substitution with an actual stage evaluator

Source item 6.5; module TriangularSolve

Use the graph and levels constructed on the previous page. A vertex v carries an arbitrary vector space X_v over a field. The spaces may have different dimensions. Let

$$D_v : X_v \xrightarrow{\sim} X_v, \qquad T_{vu} : X_u \longrightarrow X_v$$

be an invertible diagonal map and linear transfer maps. The operator being solved is defined by its actual block action:

$$(Ax)_v = D_v x_v + \sum_{u \to v} T_{vu} x_u.$$

No solution, solve equation or precomputed topological certificate is assumed.

Constructed solution. Well-founded recursion defines

$$\boxed{x_v = D_v^{-1}\left(b_v - \sum_{u \to v} T_{vu} x_u\right).}$$

Applying D_v proves the original equation at v. If another vector satisfies all equations, induction over predecessors makes each previously used coordinate identical, and invertibility of D_v forces agreement at v. Thus the construction gives existence and uniqueness, and the block operator is bijective.

Parallel stages. A second, independent definition is an explicit staged evaluator. Set $x^{(0)} = 0$ and

$$x_v^{(k+1)} = \begin{cases} D_v^{-1}\left(b_v - \sum_{u \to v} T_{vu} x_u^{(k)}\right), & \ell(v) = k+1, \\ x_v^{(k)}, & \ell(v) \neq k+1. \end{cases}$$

It reads the previous-stage vector, not partially updated coordinates. The invariant is

$$\ell(v) \leq k \quad \Longrightarrow \quad x_v^{(k)} = x_v.$$

Every predecessor of a level-$k+1$ vertex has level at most k, so the recurrence uses only coordinates already proved correct. At the canonical depth d this yields

$$\boxed{Ax^{(d)} = b.}$$

The local independence theorem separately proves that changing coordinates outside the strict earlier levels cannot change a vertex's local solve.

Scope. Diagonal invertibility is expressed by genuine linear equivalences. The task graph may include declared transfers whose values happen to be zero; optimality refers to that declared dependency graph. Exact field arithmetic and exact local inverses are used. No floating-point stability or implementation-specific parallel speed is inferred.

E.3 Support of inverses and Schur elimination

Source items 6.7 and 6.8; modules InverseSupport and SchurSupport

Let (P_a) be a complete idempotent packet family in a finite-dimensional algebra. Define $a \rightsquigarrow b$ by an actual finite path through nonzero blocks P_bAP_a. **Zero-length paths are included**, so every packet reaches itself.

Fix the source packet a. In the regular representation, define the submodule

$$W_a = \{x : P_b x = 0 \text{ whenever } a \not\rightsquigarrow b\}.$$

It contains P_a. Completeness of the projectors expands each component of Ax into packet transfers. If a transfer into an unreachable packet were nonzero from a reachable packet, it would extend a path, a contradiction. Thus $AW_a \subseteq W_a$.

Suppose $BA = I$. Left multiplication by A is injective, including on W_a. Finite-dimensionality makes that restricted map surjective. Given $x \in W_a$, choose $y \in W_a$ with $Ay = x$. Then $Bx = BAy = y \in W_a$. Applying this to P_a proves

$$\boxed{a \not\rightsquigarrow b \quad \Longrightarrow \quad P_bBP_a = 0.}$$

The matrix specialization takes B to Mathlib's actual nonsingular matrix inverse, with an invertible determinant. No inverse-sparsity hypothesis is inserted.

Hidden elimination. For the actual Schur complement

$$S = A_{RR} - A_{RH}A_{HH}^{-1}A_{HR},$$

the hidden inverse is expanded through the hidden projectors. A nonzero retained correction block requires indices j, k with nonzero entrance and exit blocks and a hidden path $j \rightsquigarrow k$. Therefore a nonzero block S_{ba} requires either a direct retained block, or

$$a \longrightarrow j \ \rightsquigarrow_H \ k \longrightarrow b.$$

The entrance and exit matrices may be rectangular; retained and hidden coordinate sizes and packet systems are independently typed. The hidden matrix must be invertible. A zero-length hidden path permits entrance and exit through the same hidden packet.

Inclusion is not equality. Cancellation can remove a block permitted by a graph route. For example,

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{has} \quad (A^{-1})_{13} = 0$$

despite a direct nonzero $(1, 3)$ block. The implementation proves support inclusion and the corresponding no-route zero certificate, not a converse.

E.4 Exact banded addresses and modeled work

Source item 4.1; module BandedCount

For p packets and half-bandwidth w, the actual address set is

$$\mathcal{B}_{p,w} = \{(i, j) \in \{0, \dots, p-1\}^2 : i \le j + w \text{ and } j \le i + w\}.$$

The predicate is the integer version of $|i-j| \le w$. The count is first defined as a double sum of indicators, then identified with the cardinality of this subtype. It is not defined by the target closed form.

Under $0 \le w < p$, the formal proof establishes

$$\boxed{N_{p,w} = |\mathcal{B}_{p,w}| = p(2w+1) - w(w+1).}$$

The proof uses finite prefix counting, the newly exposed boundary row and column when p increases, and the saturated initial band. An intermediate subtraction-free identity avoids relying on natural-number subtraction without the required inequality. The diagonal case $w = 0$ gives exactly p blocks, and the saturated case $w = p - 1$ gives p^2.

Stored scalars. With equal block width b, the storage-address type is

$$\mathcal{S} = \mathcal{B}_{p,w} \times \{0, \dots, b-1\}^2.$$

Its cardinality is the actual number of represented scalar entries:

$$\boxed{|\mathcal{S}| = b^2 N_{p,w}.}$$

Arithmetic events. For q right-hand sides, the declared event type adds a right-hand-side index and one of two operation roles, multiplication or accumulation:

$$\mathcal{O} = \{0,1\} \times \{0, \dots, q-1\} \times \mathcal{S}.$$

Hence

$$\boxed{|\mathcal{O}| = 2qb^2 N_{p,w}.}$$

These identities include zero right-hand-side count. The abstract storage formula also admits zero block width. To use the manuscript's substitution $p = n/b$, one must separately require $b > 0$ and $n = pb$.

Boundary. Two events per coefficient and right-hand side are a declared arithmetic convention. These counts do not identify the number of CUDA instructions, account for fused multiply-add lowering, price complex arithmetic in real operations, or include allocation, indexing, communication and launch overhead. They are also not a performance comparison with dense multiplication.

E.5 Exact residual certificates for supplied packets

Source item 9.2; module DiscoveryCertificate

Let (P_a) be a complete orthogonal projector system on a finite complex coordinate space. The projectors need not be coordinate masks. For an actual matrix A, define

$$\Pi(A) = \sum_a P_a A P_a, \qquad E = A - \Pi(A).$$

Reconstruction gives the exact off-diagonal residual

$$E = \sum_{b \ne a} P_b A P_a.$$

Hilbert–Schmidt orthogonality. For distinct ordered packet pairs $(b,a) \ne (d,c)$, adjoint multiplication

and projector orthogonality give

$$\operatorname{tr}((P_b A P_a)^* P_d B P_c) = 0.$$

If $b \neq d$, the central projector product vanishes. Otherwise $a \neq c$, and cyclicity of trace brings $P_c P_a$ together. This is proved for arbitrary input matrices A, B.

Lean identifies the actual Frobenius norm with its trace expression,

$$\|M\|_{\mathrm{F}}^2 = \operatorname{Re}\operatorname{tr}(M^* M),$$

then expands the finite sum and removes each cross term. The result is

$$\boxed{\|A - \Pi(A)\|_{\mathrm{F}}^2 = \sum_{b \neq a} \|P_b A P_a\|_{\mathrm{F}}^2 .}$$

No Pythagorean equality for the target residual is assumed.

Operator norm. In a separate scoped matrix norm, the triangle inequality gives

$$\boxed{\|A - \Pi(A)\|_2 \leq \sum_{b \neq a} \|P_b A P_a\|_2 .}$$

Here $\|\cdot\|_2$ is the Euclidean induced operator norm, not the Frobenius norm. The formal declarations use distinct norm instances, preventing the two certificates from being conflated. Self-adjointness is used for Frobenius orthogonality; the sum estimate itself requires only the complete packet decomposition.

Discovery boundary. This result certifies the residual of a *supplied* decomposition. It does not construct useful projectors from an arbitrary matrix, prove that a searched decomposition is optimal, or guarantee that its application is profitable on a GPU. The simultaneous spectral-decomposition input remains separately recorded as classical.

E.6 Block-Jacobi normalization and the CG factor

Source items 7.4 and 7.6; module BlockJacobi

Work on a nonempty finite complex coordinate space with Euclidean operator norms. Assume D and $A = D + K$ are positive definite. Define

$$S = (\sqrt{D})^{-1}, \qquad B = SAS, \qquad E = SKS.$$

The square root is the actual positive matrix square root supplied by continuous functional calculus; the inverse is the actual matrix inverse. Positive definiteness makes $\sqrt{D}$ a unit. Its square identity and inverse identities prove

$$SDS = I, \qquad B = I + E.$$

A direct inverse bound. If $C = (I + E)^{-1}$ and $\|E\|_2 \leq \eta < 1$, then $C = I - EC$. Consequently

$$(1 - \eta)\,\|C\|_2 \leq 1, \qquad \|I + E\|_2 \leq 1 + \eta.$$

The constructed normalization is invertible because S and A are. Combining the estimates gives the

actual condition number bound

$$\boxed{\kappa_2(B) := \|B\|_2 \left\|B^{-1}\right\|_2 \leq \frac{1+\eta}{1-\eta}.}$$

This norm argument does not require a separately assumed Hermitian perturbation. Nonnegative η follows from the norm bound. Nonemptiness is explicit because the condition-number and norm-of-identity convention uses a nonzero coordinate space.

The preconditioning corollary. Define

$$q(\kappa) = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}.$$

Monotonicity of square root and positive denominators prove that q is monotone. Lean also proves $\kappa_2(B) \geq 1$, $q(\kappa) \geq 0$ for $\kappa \geq 1$, and $q(\kappa) < 1$. Thus

$$\boxed{q(\kappa_2(B)) \leq q\left(\frac{1+\eta}{1-\eta}\right).}$$

For $1 \leq \kappa$ and $e_0 \geq 0$, an explicit classical premise $e_m \leq 2q(\kappa)^m e_0$ and a certified upper bound $\kappa \leq \bar{\kappa}$ yield $e_m \leq 2q(\bar{\kappa})^m e_0$.

What is not claimed. The classical conjugate-gradient convergence theorem, item 7.5, is not newly formalized. The new corollary proves its factor comparison and bound substitution. It does not replace the original theorem's algorithm hypotheses with a new axiom or claim a floating-point CG convergence proof.

E.7 Verification record and remaining boundary

Cumulative fifth-block evidence

Fresh Lean checks. A separate baseline rebuild accepted the unchanged 23-module, 247-declaration fourth block. The final run rebuilt all 30 active modules into a new output directory under Lean 4.34.0-rc2 and Mathlib revision

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`.

All modules returned exit code zero with warnings treated as errors. All 313 theorem and 36 construction audit targets matched the inventory. Dependencies are contained in `propext`, `Classical.choice` and `Quot.sound`. Both deliberately invalid controls were rejected. Seventeen existing verifier tests and twelve new release-contract tests passed.

Preservation and corrections. All 23 upstream proof files and the original manuscript assets are unchanged. The source-to-result ledger is updated, including the explicit conventions for empty graphs, reflexive reachability and nonempty condition-number spaces. Two stale fourth-ledger increment fields were corrected: they had retained older "new items" metadata even though its cumulative count was correct. The new regression tests check these fields against this block's actual targets.

Development failures. Seventeen captured fifth-block development checks contain ten failed drafts and seven accepted drafts, one per new module. Failures included declaration scoping, finite-type instances, rewrite direction, parser syntax and warnings. The source snapshots and logs are retained

separately. They are not substituted for the successful fresh cumulative run. Every final active source has a matching successful audit.

Independent mirrors. Exact rational tests passed 820 band-address cases, 200 DAG/staged-solve cases, 60 inverse-support cases, 50 Schur-support cases and 72 non-coordinate projector cases. Of the DAG cases, 125 also exhausted all candidate schedules on at most four vertices. Eight cyclic graphs were rejected. The inverse and Schur tests checked 390 and 97 absent-route blocks respectively, and retained a cancellation control.

Separate numerical tests passed 64 complex projector cases and 120 positive-matrix normalization/CG-factor cases. The largest squared-Frobenius identity residual was 1.42×10^{-13}; the largest positive condition-bound violation was 6.66×10^{-15}, within the declared tolerance. These are independent Python mirrors, not Lean execution, new universal proofs or GPU measurements.

Remaining item. Manuscript theorem 5.6, small-core optimal recompression, remains unformalized. Four external inputs remain separately tracked: Eckart–Young–Mirsky, CG convergence, the floating-point product model and simultaneous spectral decomposition. The register's 55/56 therefore must not be read as verification of every assertion or all classical algorithms in the paper.

Reproduction and limits. Run `scripts/verify.py` with the separately supplied portable kit. The current evidence is `verification/fifth_final/SUMMARY.json`. The original CPU/GPU corpus, earlier compiler execution suites and Papers I–VII in full were not rerun. The pinned Mathlib cache was reused, not independently rebuilt or rechecked from source. No manuscript-to-formal semantic-equivalence theorem is claimed.

F Global small-core recompression from ordered thin SVD data

This chapter retains the substantive mathematical pages of verification block 6 (v0.6.0). Its dated audit and pending-status passages describe that historical checkpoint; the final status is 56 of 56 registered finite items under the recorded hypotheses. Later chapters supersede earlier pending statements. All compiler and regression counts in this chapter are inherited records, not fresh runs for this edition.

F.1 Compact factors from actual column spaces

Modules CompactRangeFactors and SmallCoreTransfer

Let

$$L \in \mathbb{C}^{m\times t}, \qquad R \in \mathbb{C}^{n\times t}, \qquad A = LR^{*}.$$

Use the ordinary Euclidean inner products on the coordinate spaces. Define

$$S_L = \operatorname{range}(L), \quad r_L = \dim S_L, \qquad S_R = \operatorname{range}(R), \quad r_R = \dim S_R.$$

The implementation constructs an orthonormal basis of each actual range subspace. Let Q_L and Q_R be the matrices of the corresponding isometric inclusions. Express the range-restricted input maps in these bases to obtain R_L, R_R.

The checked identities are

$$\boxed{Q_L^{*}Q_L = I, \quad L = Q_L R_L, \quad R_L = Q_L^{*}L,}$$

with the analogous identities on the right. The dimension theorem is not an input:

$$\boxed{r_L = \operatorname{rank} L, \qquad r_R = \operatorname{rank} R.}$$

The coordinate factors have full row rank. The construction therefore handles rank-deficient input factors without retaining artificial column directions.

Define the small core by actual multiplication,

$$G = R_L R_R^{*} \in \mathbb{C}^{r_L\times r_R}.$$

Finite matrix associativity and adjoint identities give

$$\boxed{A = Q_L G Q_R^{*}, \qquad \operatorname{rank} G = \operatorname{rank} A.}$$

The rank identity follows from both multiplication inequalities and the left inverse obtained by compression. It is not inferred from agreeing dimensions.

Zero factors and empty spaces. No positive-rank or nonempty-coordinate hypothesis is imposed. If a range is zero, its orthonormal basis has an empty index type. The factorization uses the resulting rectangular empty matrices. A separate theorem proves that the compact dimension is zero exactly when the original factor is zero.

Relation to QR. The argument only needs orthonormal columns and exact reconstruction. A compact QR factorization supplies these properties, but triangularity of its coordinate factor is irrelevant to the recompression theorem. The formal construction uses an exact orthonormal range basis; it does not claim Householder steps, pivot selection or floating-point stability. A supplied noncompact orthonormal factorization is also covered by the general transfer theorem, including extra zero singular

directions.

F.2 Why a small-core optimum is a global optimum

Module SmallCoreNorms; arbitrary ambient competitors

For rectangular isometries Q, V define

$$\mathcal{L}(G) = QGV^*, \qquad \mathcal{C}(B) = Q^*BV, \qquad Q^*Q = I, \quad V^*V = I.$$

The following statements are proved for the actual matrix rank and actual matrix norms:

$$\mathcal{C}(\mathcal{L}(G)) = G, \qquad \operatorname{rank}\mathcal{L}(G) = \operatorname{rank} G, \qquad \operatorname{rank}\mathcal{C}(B) \le \operatorname{rank} B,$$

$$\boxed{\|\mathcal{L}(G)\|_\nu = \|G\|_\nu, \qquad \|\mathcal{C}(B)\|_\nu \le \|B\|_\nu, \quad \nu \in \{2, \mathrm{F}\}.}$$

For the spectral norm, the proof uses the Gram matrix identity

$$(QG)^*(QG) = G^*G$$

and the C^*-norm square identity, together with right-side adjoint symmetry. Compression follows from submultiplicativity and the isometry norm bound, including empty spaces.

For the Frobenius norm, trace cyclicity proves left and right isometric invariance. The proof also establishes the projection identity

$$\|B - QQ^*B\|_{\mathrm{F}}^2 = \|B\|_{\mathrm{F}}^2 - \|Q^*B\|_{\mathrm{F}}^2.$$

Nonnegativity of the left side gives left compression contractivity; the right version follows by adjoints. No norm-preservation or contractivity hypothesis is inserted.

Suppose G_k is a best rank-at-most-k approximation to G in either norm. Take an *arbitrary* ambient matrix B with $\operatorname{rank} B \le k$. Compression is an admissible small-core competitor and

$$\begin{aligned}\|\mathcal{L}(G) - \mathcal{L}(G_k)\|_\nu &= \|G - G_k\|_\nu \\ &\le \|G - \mathcal{C}(B)\|_\nu \\ &= \|\mathcal{C}(\mathcal{L}(G) - B)\|_\nu \\ &\le \|\mathcal{L}(G) - B\|_\nu.\end{aligned}$$

This is the key global bridge: B is not required to lie in the supplied row and column spaces. The next two pages discharge the intermediate small-core optimality obligation, rather than leaving it as a hypothesis in the final theorem.

F.3 Spectral optimality from a kernel witness

Modules DiagonalTruncation and SpectralOptimality

Let $\sigma_1 \ge \cdots \ge \sigma_d \ge 0$ and

$$D = \operatorname{diag}(\sigma_1, \dots, \sigma_d), \qquad D_k = \operatorname{diag}(\sigma_1, \dots, \sigma_{\min(k,d)}, 0, \dots, 0).$$

Lean uses indices beginning at zero; the formulas on this page use the conventional one-based notation. The rank of D_k is proved to be at most k, even when retained singular values are zero.

For $k < d$, take any $d \times d$ matrix B of rank at most k. Restrict its domain to the first $k+1$ coordinates using the actual coordinate inclusion J. Since

$$\operatorname{rank}(BJ) \leq \operatorname{rank} B \leq k < k+1,$$

rank-nullity supplies a nonzero vector z with $BJz = 0$. The vector $y = Jz$ is nonzero, satisfies $By = 0$ and is supported in the first $k+1$ coordinates. The injection and the support property are checked separately.

Monotonicity of the diagonal entries gives

$$\|Dy\|^2 = \sum_{i=1}^{k+1} \sigma_i^2 |y_i|^2 \geq \sigma_{k+1}^2 \|y\|^2 .$$

Because $(D-B)y = Dy$, the operator-norm inequality implies

$$\sigma_{k+1} \|y\| \leq \|(D-B)y\| \leq \|D-B\|_2 \|y\| .$$

Cancellation of the strictly positive norm of y proves

$$\boxed{\sigma_{k+1} \leq \|D-B\|_2 .}$$

The norm of the actual diagonal residual $D - D_k$ equals σ_{k+1}, so D_k attains the lower bound. If $k \geq d$, the residual is zero and optimality follows from norm nonnegativity.

No optimality oracle. The nonzero kernel vector, its coordinate support and the diagonal norm bound are all derived. No Eckart–Young inequality, best-rank property or singular-tail formula is assumed. The final rectangular SVD result follows by the isometric transfer proved on the preceding page.

F.4 Frobenius optimality from kernel weights

Modules KernelFrame and FrobeniusOptimality

For an arbitrary competitor B with rank $B \leq k$, construct an orthonormal matrix Q spanning its actual kernel. Then

$$BQ = 0, \qquad Q^*Q = I, \qquad q = \dim\ker B = d - \operatorname{rank} B.$$

Define the row weights

$$w_i = \sum_{j=1}^{q} |Q_{ij}|^2.$$

The proof derives

$$\boxed{0 \leq w_i \leq 1, \qquad \sum_i w_i = q \geq \max(d-k, 0).}$$

The upper bound uses contraction of Q^* on a coordinate vector. The weight sum follows from the trace of Q^*Q and rank-nullity.

For any nonnegative decreasing sequence a_i, weights with these properties satisfy

$$\sum_i a_i w_i \geq \sum_{i>k} a_i.$$

This finite weighted inequality is itself proved. For $k < d$, use $\mu = a_{k+1}$ and compare the summands

to tail weights $v_i = 0$ for $i \le k$ and $v_i = 1$ otherwise:

$$a_i v_i + \mu(w_i - v_i) \le a_i w_i.$$

For a retained coordinate the inequality uses $a_i \ge \mu$ and $w_i \ge 0$. For a tail coordinate it uses $a_i \le \mu$ and $w_i \le 1$. Summation and $\sum w_i \ge \sum v_i$ give the result.

Set $a_i = \sigma_i^2$. Right compression cannot increase the Frobenius norm, and $BQ = 0$, hence

$$\begin{aligned}\|D - B\|_{\mathrm{F}}^2 &\ge \|(D - B)Q\|_{\mathrm{F}}^2 \\ &= \|DQ\|_{\mathrm{F}}^2 = \sum_i \sigma_i^2 w_i \ge \sum_{i>k} \sigma_i^2.\end{aligned}$$

Direct calculation of $D - D_k$ attains equality. Thus the best-rank Frobenius assertion is proved from kernel geometry and finite sums, independently of the spectral lower-bound proof. Repeated and zero singular values require no uniqueness claim.

F.5 Thin output factors and the final theorem

Modules CompactTruncation and SmallCoreRecompression

For the actual core $G = R_L R_R^*$, supply ordered thin SVD data

$$G = W \operatorname{diag}(\sigma) Z^*, \qquad W^*W = I, \qquad Z^*Z = I, \qquad \sigma_1 \ge \cdots \ge \sigma_d \ge 0.$$

These are precisely the fields of the formal SVD structure. It contains no rank-optimality or tail-error field. A derived theorem checks the Gram eigenvector equation

$$G^*GZ = Z \operatorname{diag}(\sigma_1^2, \dots, \sigma_d^2).$$

Let $h = \min(d, k)$ and retain exactly the first h column indices. The rectangular square-root mask has entries

$$M_{ij} = \begin{cases} \sqrt{\sigma_j}, & i = j, \\ 0, & i \ne j, \end{cases} \qquad 1 \le i \le d, \quad 1 \le j \le h.$$

Lean proves its index cardinality and $MM^* = D_k$. Define

$$\boxed{L_k = Q_L W M, \qquad R_k = Q_R Z M.}$$

The product equals $Q_L W D_k Z^* Q_R^*$ and has rank at most k. The principal theorem proves, for every ambient B with rank $B \le k$,

$$\boxed{\|LR^* - L_k R_k^*\|_\nu \le \|LR^* - B\|_\nu, \quad \nu \in \{2, \mathrm{F}\}.}$$

It also gives the exact errors

$$\|LR^* - L_k R_k^*\|_2 = \begin{cases} \sigma_{k+1}, & k < d, \\ 0, & k \ge d, \end{cases} \qquad \|LR^* - L_k R_k^*\|_{\mathrm{F}} = \left(\sum_{i>k} \sigma_i^2 \right)^{1/2}.$$

Both the squared and square-root Frobenius formulas are checked. All these statements are combined in

`KomorebiPaperVIII.SmallCore.small_core_optimal_recompression`.

Endpoints. Rank zero produces the zero output; $k \geq d$ reconstructs the original product. An empty SVD certificate is constructed for a zero core, and diagonal SVD certificates are constructed explicitly. Zero singular values are never inverted. At repeated singular values the result asserts optimality, not uniqueness of the minimizer.

F.6 Source map and verification

One completed manuscript item; 79 supporting declarations

Module	Main purpose	Theorems
SmallCoreNorms	Isometric norms, compression, rank	19
CompactRangeFactors	Actual compact column factors	7
SmallCoreTransfer	Global best-rank transfer	7
KernelFrame	Actual kernel and row weights	8
DiagonalTruncation	Rank and exact diagonal errors	9
FrobeniusOptimality	Weighted-tail lower bound	5
SpectralOptimality	Prefix-kernel lower bound	7
CompactTruncation	Thin square-root factors	5
SmallCoreRecompression	Main theorem and Gram relation	6
RecompressionEndpoints	Zero and full-retention cases	6
Total	Theorem 5.6 and its support	79

Cumulative Lean run. All 40 local modules were compiled in a newly created output directory under Lean 4.34.0-rc2. Each exited with code zero and warnings were fatal. The 392 named theorem reports and 66 separately counted construction reports match the inventory exactly. Dependencies are confined to `propext`, `Classical.choice` and `Quot.sound`, or subsets thereof. False equality and admitted-proof controls were both rejected.

Pinned dependency. The actual Mathlib Git revision and manifest revision both match

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. The supplied compiled Mathlib cache was reused. It was not independently rebuilt or rechecked. Recovery verified all seven offline archive parts and the combined archive hash.

Infrastructure. Forty-three unit tests passed: 17 verifier tests, 12 fifth-block compatibility tests updated to the current totals, and 14 sixth-block tests. The original fifth-block fixed-count test file and metadata are preserved. Exact audit coverage, dependency order, no-bypass scans, classical-input boundaries and absence of optimality fields in the SVD structure are tested.

Failures retained. Of 31 captured development checks, 21 failed before correction and ten accepted drafts supplied the new modules. Failures concerned proof elaboration, finite-index/cardinality APIs, dependent rewrites, norm scopes and warning diagnostics. Their inactive source snapshots and logs are retained separately from the successful fresh cumulative run. An initial interactive runtime launch was unsupported; noninteractive assembly succeeded.

F.7 Regressions, reproduction and claim boundary

Exact mathematical guarantees versus numerical implementations

Independent numerical checks. A deterministic NumPy complex128 mirror tested 182 input-factor pairs at 638 truncation cases. The cases include rectangular matrices, rank-deficient and zero factors, empty spaces, repeated singular values, rank zero and ranks beyond the core width. Comparison used both the small core and an independently computed ambient SVD reference. There were 3,190 arbitrary rank-bounded competitors, nine invalid-input rejection controls and two checks demonstrating failure when essential assumptions are removed.

The largest relative tail/reconstruction discrepancy was below 3.7×10^{-15}, against a declared 2×10^{-9} tolerance. The mirror uses numerical rank thresholds and a numerical SVD to construct column bases. It is not compiled or extracted Lean code, a proof for all inputs, or a GPU benchmark. The formal result uses exact ranks and exact factorization equations.

Reproduce from the extracted source root.

```
python3 scripts/verify_integrity.py
python3 -m unittest discover -s scripts -p 'test_*.py' -v
python3 scripts/verify.py --kit /path/to/portable-kit \
    --out verification/recheck --timeout 240
python3 scripts/run_sixth_regressions.py \
    --out verification/recheck_numerical.json
```

The unit tests and numerical mirror require NumPy. The Lean verification path needs Python 3.10 or later and the separately supplied pinned Linux x86-64 kit. Local compiled interfaces are regenerated. The runner also supports bounded batches with `--max-checks` and hash-checked `--resume`. A partial checkpoint is not a pass.

What is complete. The 56-item manuscript-proved register is covered under the recorded hypotheses. The original theorem says to take orthonormal factors and a core SVD. The new construction and proof discharge the recompression conclusion from those exact factorization equations. In particular, no best-approximation assertion or error formula is hidden in an input certificate.

What is not claimed. General SVD existence or a numerical SVD implementation is not newly formalized. The four classical entries remain separately tracked; the ordered-SVD-data optimality argument now has checked support, while the CG algorithm theorem, floating-point product model and simultaneous-decomposition input are not newly proved in full. No new generic matrix-multiplication exponent, floating-point stability theorem or hardware speedup follows from this release.

The original manuscript, bibliography, source package and all five earlier supplements are preserved. Historical CPU/GPU measurements, earlier compiler execution suites and Papers I–VII in full were not rerun. Prose-to-formal correspondence is an explicit mathematical review, not an automated theorem equating a sentence with a Lean type.

Worked visual atlas

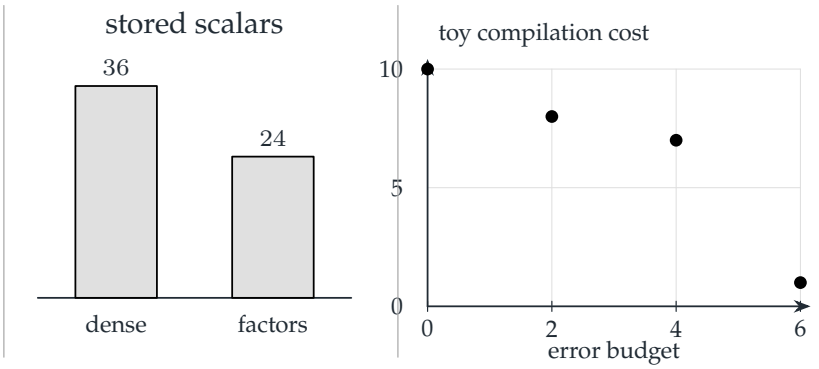
V1 / THREE VIEWS OF ONE COMPUTATION

Compute the representation first

A dense array, a factored operator, and a compiled execution plan are different objects. They may denote the same linear map, but they need not store the same data, return the same output, or incur the same costs. This guide makes those distinctions visible through calculations rather than a pipeline of boxes.

A: 36 visible entries

4	2	1	1	1	1
2	4	1	1	1	1
1	1	4	2	1	1
1	1	2	4	1	1
1	1	1	1	4	2
1	1	1	1	2	4



Three models to carry through the paper

The first is a six-dimensional operator $A = D + uv^T$. Every entry of A is nonzero, yet three 2×2 diagonal blocks and two thin vectors suffice to apply it exactly. Pages V3, V9, V10, and V27 reuse this operator or its factors.

The second starts with redundant factors L and R. Actual range coordinates reduce a 4×3 product to a 2×3 core. Pages V11–V12 connect rank, singular tails, and the geometric reason a small-core optimum is also an ambient optimum.

The third is a seven-node candidate tree with exactly five prunings. Its budget optimum is not selected by any Lagrangian penalty. Pages V21–V23 let readers inspect every feasible choice and every changed subproblem.

How to use the guide

The standalone visual companion supplies the traditional reading references. Its source crosswalk records each result label and both its printed and physical PDF page. The preserved examples use rational, Gaussian-rational, symbolic, and finite-enumeration arithmetic. Portable checks use exact rational and Gaussian-rational arithmetic. The original SymPy suite is replayed for this edition, with its assertions and rejected controls recorded separately. Smooth drawing coordinates are illustrations of stated formulas, not additional proof evidence.

> **Read this as:** An exact representation can remove work before a device is chosen. It does not by itself supply a fast kernel, a good approximation, or a fair timing comparison.

V2 / A READER'S ATLAS

The objects and their contracts

The same letter can refer to a mathematical operator or to a particular stored representation. Keep those levels separate when reading an algorithm or a benchmark row.

Symbol	Object	Question to ask
P_α	Coordinate projector	Which input or output coordinates does it select?
$A_{\beta\alpha}$	A block map	What are its source and target dimensions?
E	Active block addresses	Is this exact support, or only a structural upper bound?
$D + UV^*$	Factored operator	Is the desired output a vector, factors, or a dense array?
Q_L, G, Q_R	Range factors and core	Are the outer columns orthonormal, with exact reconstruction?
E_j	An error operator	Is it a factor perturbation or an additive error after a product?
s_e, ϵ_e	Sensitivity and tolerance	Is the budget input-dependent or uniform over unit inputs?
$(\kappa_v, \epsilon_v, \sigma_v)$	Candidate-tree data	Do costs, errors, and split overheads really add?
$F_v(b)$	A budget table	Does b mean exact weight, or an upper bound?
R	Reuse count	Which one-time costs have been included?

Conventions used in the examples

Matrices act on column vectors. A block $A_{\beta\alpha}$ goes from source α to target β. Products act from right to left. Real examples use transpose; complex examples use adjoint unless a vectorization identity explicitly requires transpose. Column-major vectorization stacks the columns of a matrix.

An arithmetic count follows the paper's modeled multiply/add convention. It is not a count of executed machine instructions. The tiny scheduling and amortization examples use declared synthetic units, not measured GPU times. The original benchmark corpus remains historical data in the preserved source tree.

Evidence levels

The original theorem ledger records its inherited finite formal scope. The new exact checks validate these particular examples. A plot samples a formula. A historical timing describes its recorded case and machine. None of these evidence types can silently substitute for another.

> **Read this as:** Before interpreting a number, identify its object, norm, output contract, and timing scope. Most apparent contradictions disappear once those four fields are explicit.

V3 / THE RUNNING SIX-DIMENSIONAL OPERATOR

A dense-looking matrix, one scalar coupling

Let D have three identical blocks $\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$ and put $u = v = \mathbf{1} \in \mathbb{Q}^6$. Then $A = D + uv^T$ has diagonal entries four, within-block off-diagonal entries two, and every cross-block entry one. All 36 entries are nonzero.

$$x = (1, -1, 2, 0, -2, 3)^T, \qquad v^T x = 3, \qquad Ax = Dx + 3u.$$

x		Dx		$u(v^T x)$		Ax
1		2		3		5
−1		−2		3		1
2		6		3		9
0	+	2	+	3	=	5
−2		−3		3		0
3		7		3		10

$v^T x = 3$ is shared by all six output coordinates.

What was avoided?

The global part does not require 36 independent coefficients. It depends on the input through one scalar, $v^T x$, which is broadcast through u. The diagonal part is computed by three separate local matrix products. Nothing is approximated and no channel is discarded.

The sixth output is $7 + 3 = 10$, while the fifth is $-3 + 3 = 0$. A zero result can arise through cancellation even when the corresponding row contains no zero coefficient. The output vector is therefore not a support map of the input matrix.

What has not been established?

This computation proves neither that the supplied factors are the cheapest representation nor that six small tasks outperform one dense kernel. For example, the repeated diagonal blocks and all-one vectors admit further special handling. The stated BDLR model deliberately treats the blocks as dense and the factors as stored vectors so that the paper's general formulas apply literally.

The checker constructs the complete matrix independently, multiplies it by x, and compares all six entries with the factored route. The same input returns in the product calculation on V27.

> **Read this as:** A matrix can have full entrywise support and still admit a small exact representation. Entry sparsity and algebraic compressibility are different properties.

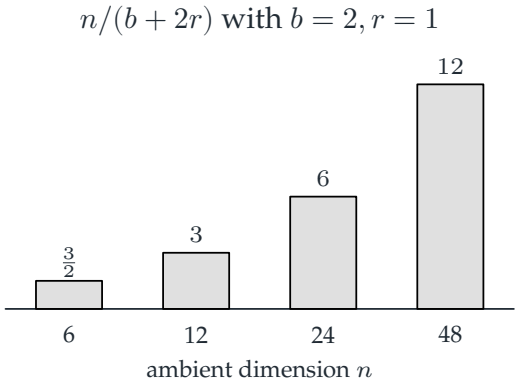
V4 / ARITHMETIC, STORAGE, AND CAPACITY

Work removed is not time measured

For q right-hand sides, the paper models dense application by $2n^2q$ operations. A block-diagonal-plus-rank-r representation with equal block width b uses

$$F = 2qn(b + 2r), \qquad M = n(b + 2r) \text{ stored scalars.}$$

These formulas count the dense block products and two thin products. They follow the manuscript convention rather than adding every scalar accumulation or index operation.



Six-dimensional running operator

dense: 36 scalars, $72q$ work

BDLR: 24 scalars, $48q$ work

ratio: $3/2$, not a timing result

For the running example, $n = 6, b = 2, r = 1$. The model reduces storage from 36 to 24 scalars and application work from $72q$ to $48q$. For fixed b, r, the ratio $n/(b + 2r)$ grows with n. If the rank instead grows proportionally with dimension, that apparent advantage need not grow.

Memory is another constraint

With scalar size s, the representation alone requires sM bytes. Inputs, outputs, workspace, allocator overhead, and stored indices are additional. A matrix-only capacity inequality is not a complete device-memory fit test.

For a block band of p packets and half-bandwidth w, the exact active-block count is

$$p(2w + 1) - w(w + 1).$$

The subtraction removes overcounted edge diagonals. The checker enumerates all addresses for $1 \le p \le 6$ and every admissible w, including $w = 0$ and $w = p - 1$.

> **Read this as:** An arithmetic or storage ratio explains a representation advantage. Kernel launch cost, traffic, utilization, and conversion determine whether that advantage becomes a time advantage.

V5 / CIRCULANT MULTIPLICATION

Four coefficients, four exact frequencies

Take $c = (2, 1, 0, -1)^T$ and define $C(c)_{jk} = c_{j-k \bmod 4}$. The unitary Fourier matrix is $F_{jk} = \mathrm{i}^{-jk}/2$. Its normalization is visible in the exact diagonalization:

$$C(c) = F^* \operatorname{diag}(2Fc) F, \qquad 2Fc = (2, 2-2\mathrm{i}, 2, 2+2\mathrm{i})^T.$$

$C(c)$

2	−1	0	1
1	2	−1	0
0	1	2	−1
−1	0	1	2

$\sqrt{4}\,Fc$

2
$2-2i$
2
$2+2i$

$$F_{jk} = \tfrac{1}{2}\,\mathrm{i}^{-jk}$$

$$C = F^* \operatorname{diag}(\sqrt{4}Fc) F$$

$\sqrt{4} = 2$ is essential.

A unitary transform changes the normalization of the convolution eigenvalues.

Follow an actual input

For $x = (1, 2, -1, 0)^T$, dense multiplication gives $(0, 6, 0, -2)^T$. Transforming x, multiplying by the four displayed eigenvalues, and applying F^* gives exactly the same vector. Every entry belongs to $\mathbb{Q}(\mathrm{i})$, so the check needs no approximate trigonometric values.

There is a common normalization trap. With an unnormalized discrete transform, the convolution multiplier has one convention. With the unitary transform used here, the eigenvalue vector is $\sqrt{n}\,Fc$, not Fc. Dropping the factor two reconstructs $C/2$ rather than C in this example. The invalid control deliberately tries that change and rejects it.

Where the fast algorithm enters

The identity specifies the represented operator. An FFT is an additional factorization that evaluates the transforms efficiently. The algebraic diagonalization does not certify a particular floating-point FFT, data layout, or launch policy. For a tiny four-point example, direct multiplication may be the simpler execution path even though the frequency representation is exact.

> **Read this as:** Transform coordinates change both the storage pattern and the normalization conventions. Verify the inverse transform and convolution scale before interpreting a frequency-domain product.

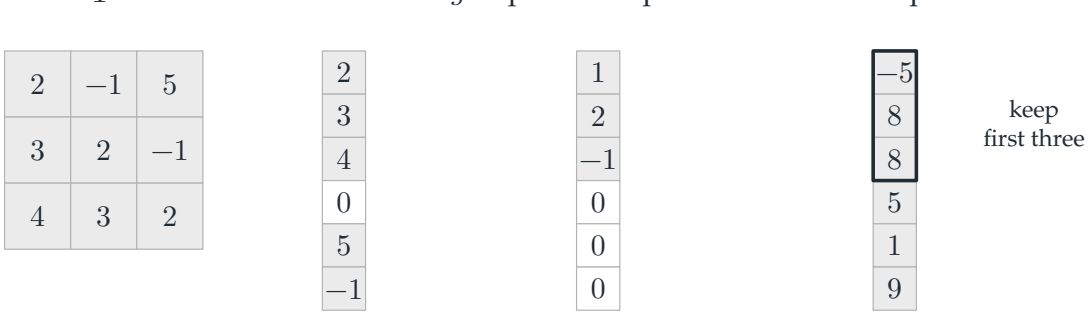
V6 / TOEPLITZ EMBEDDING WITHOUT WRAPAROUND

Padding is part of the theorem

Consider

$$T = \begin{pmatrix} 2 & -1 & 5 \\ 3 & 2 & -1 \\ 4 & 3 & 2 \end{pmatrix}, \qquad x = (1, 2, -1)^T.$$

The diagonal pattern is Toeplitz, not circulant. To use a circular convolution, embed it into dimension six with first column $g = (2, 3, 4, 0, 5, -1)^T$ and pad the input with three zeros.



Padding is part of the operator identity, not optional workspace.

Read the wrapped indices

For output index $j < 3$, an input index $k \le j$ uses the first-column coefficient c_{j-k}. An input index $k > j$ wraps to the stored first-row coefficient r_{k-j}. The padded input positions contribute nothing. Keeping the first three outputs therefore gives

$$Tx = (-5, 8, 8)^T.$$

The remaining circular outputs are workspace consequences of the embedding, not additional components of Tx.

A concrete failure mode

Using only the first column $(2, 3, 4)$ as a three-dimensional circulant changes the upper-triangular coefficients. That is a different matrix. The checker compares its output with Tx and rejects the unpadded shortcut.

The middle zero in g is not a discarded Toeplitz coefficient: it separates the lower and reversed upper diagonals in the chosen $2n$ embedding. Other embeddings require their own index identities. It is not sufficient for an array merely to have the right length or to produce a plausible plot.

> **Read this as:** Zero padding prevents the wrong circular interactions. It is an algebraic part of the operator representation, not an optional performance tweak.

V7 / KRONECKER ACTION AND ARRAY ORDER

A transpose, not an adjoint

With column-major vectorization, the tensor identity is

$$(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T).$$

Here the transpose records index order, not a Hermitian inner product. Use the explicitly complex matrix $A = \begin{pmatrix} 1 & \mathrm{i} \\ 2 & -1 \end{pmatrix}$, together with $B = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$ and $X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

X

1	2
3	4

$\mathrm{vec}(X)$

1
3
2
4

BXA^T

$5 + 8i$	2
$9 + 12i$	6

A^T not A^*

$$(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^T)$$

Check the coordinates

The stacked input is $(1, 3, 2, 4)^T$. Direct tensor multiplication agrees with stacking the columns of

$$BXA^T = \begin{pmatrix} 5 + 8\mathrm{i} & 2 \\ 9 + 12\mathrm{i} & 6 \end{pmatrix}.$$

Replacing A^T with A^* changes the signs of the imaginary terms. A real-only test would conceal that error, which is why the worked model uses an imaginary coefficient.

Why modewise contraction matters

For A of order m and B of order n, the explicit tensor matrix stores m^2n^2 entries. Its factors store $m^2 + n^2$. Applying through BXA^T costs $2n^2m+2nm^2$ modeled operations, rather than forming the full tensor matrix first.

A different flattening convention has a different coordinate formula. The statement is not independent of how an array is reshaped. In a software implementation, shape, stride, and flattening order belong to the correctness contract as well as the cost model.

> **Read this as:** Tensor structure gives exact savings only when the address convention is preserved. A silent transpose, conjugation, or reshape change implements a different map.

V8 / WALSH RECURSION IN EIGHT COORDINATES

See every butterfly value

The unnormalized Walsh matrix satisfies $H_1 = [1]$ and

$$H_{2n} = \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix}, \qquad H_n H_n^T = nI.$$

Apply the three add/subtract stages to $(1, 2, 3, 4, 5, 6, 7, 8)^T$. The full intermediate ledger is displayed below.

input	1	2	3	4	5	6	7	8
stage 1	3	-1	7	-1	11	-1	15	-1
stage 2	10	-2	-4	0	26	-2	-4	0
stage 3	36	-4	-8	0	-16	0	0	0

$8 \log_2 8 = 24$ additions/subtractions; $H_8 H_8^T = 8I$.

Arithmetic can be counted exactly

Each stage visits eight coordinates, producing one sum or difference per coordinate. Three stages therefore use exactly 24 additions or subtractions. No multiplication by a dense 8×8 matrix is necessary.

The result $(36, -4, -8, 0, -16, 0, 0, 0)^T$ still reconstructs the input: the inverse is $H_8^T/8$. Because the displayed transform is unnormalized, its squared output norm is eight times the squared input norm. The normalized transform $H_8/\sqrt{8}$ would instead be orthogonal.

What the zeros mean

The zero coefficients arise from exact cancellation among the input values. They do not imply that the transform's defining matrix is sparse. The butterfly structure, not the final coefficient support, explains the arithmetic reduction.

The checker verifies the recursion, orthogonality, dense-versus-butterfly output, and operation count for sizes one, two, four, eight, and sixteen. These bounded tests exercise the implementation and boundary cases. The general recursion theorem remains the manuscript's mathematical statement.

> **Read this as:** Factorization can turn a dense matrix into a short exact arithmetic circuit. Keep its normalization separate from its recursive evaluation rule.

V9 / CLOSED LOW-RANK PRODUCT ALGEBRA

The cross term has somewhere to go

Let $A = D_A + u_a v_a^T$ be the running operator. Take $D_B = \mathrm{diag}(1,2,3,4,5,6)$, $u_b = (1,0,1,0,1,0)^T$, and $v_b = (0,1,0,1,0,1)^T$. Define $B = D_B + u_b v_b^T$.

$$U_C = [D_A u_b + u_a(v_a^T u_b)\ \ u_a], \qquad V_C = [v_b\ \ D_B^T v_a].$$

U_C

6	1
4	1
6	1
4	1
6	1
4	1

V_C

0	1
1	2
0	3
1	4
0	5
1	6

$$AB = D_A D_B + U_C V_C^T$$

$$U_A(V_A^T U_B)V_B^T$$

must be retained

Two input rank-one corrections produce at most rank two, not three.

Why there are only two columns

Expanding AB gives three correction terms, but two share the right factor v_b^T:

$$D_A u_b v_b^T + u_a(v_a^T u_b)v_b^T = (D_A u_b + u_a(v_a^T u_b))v_b^T.$$

The remaining term is $u_a v_a^T D_B$. Thus $AB = D_A D_B + U_C V_C^T$ with at most two correction directions. Counting the three expanded summands as independent ranks would overestimate this representation.

The scalar $v_a^T u_b$ is three. Removing the quadratic term would therefore remove $3u_a v_b^T$, which is nonzero. The checker compares the complete product to the factor construction and rejects a calculation that omits this contribution.

Before and after recompression

A rank bound on stored factors is not always the actual matrix rank. Repeated products may introduce dependent columns. Recompression can remove those dependencies exactly, or remove small singular directions approximately under an explicit error budget.

> **Read this as:** Closure of a representation class is an algebraic statement about all product terms. It is not permission to drop an interaction merely because it is second order in the factors.

V10 / WOODBURY WITH BOTH INVERTIBILITY CHECKS

A large inverse, a small denominator

For the running operator, each diagonal block has inverse $\frac{1}{8}\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}$. Hence $D^{-1}u = u/4$ and

$$S = 1 + u^T D^{-1} u = \frac{5}{2}.$$

The Woodbury identity reduces the entire inverse to

$$A^{-1} = D^{-1} - D^{-1} u S^{-1} u^T D^{-1} = D^{-1} - \frac{1}{40} u u^T.$$

one block of D^{-1}

$\frac{3}{8}$	$-\frac{1}{8}$
$-\frac{1}{8}$	$\frac{3}{8}$

$D^{-1}u = \frac{1}{4}u$

$S = 1 + u^T D^{-1} u = \frac{5}{2}$

$A^{-1} = D^{-1} - \frac{1}{40} u u^T$

$I - e_1 e_1^T$

0	0
0	1

invertible D
singular update

Two hypotheses, not one

Invertibility of D alone does not suffice. For $D = I_2$, $u = e_1$, and $v = -e_1$, the update is $\mathrm{diag}(0, 1)$. The small matrix S is zero, exactly where the correction formula would require an invalid division.

For the running model, the checker multiplies the proposed inverse by A in both orders and obtains the identity. It also checks the determinant relation $\det A = \det D \det S$. These tests verify the actual factors and sign of the correction, not just a numerical solve residual.

Conditioning survives the reduction

A small S can still be ill-conditioned. Reducing its dimension does not bound $\|S^{-1}\|$. The manuscript's inverse estimate keeps that term explicit, along with the norms of $D^{-1}u$ and $v^T D^{-1}$.

> **Read this as:** The hidden coupling has not vanished. Its solvability and sensitivity are concentrated in a smaller operator that still has to be invertible and numerically controlled.

V11 / ACTUAL RANGE COORDINATES

Stored width is not intrinsic rank

Choose the isometry

$$Q_L = \begin{pmatrix} 3/5 & 0 \\ 4/5 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad R_L = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad R_R = \begin{pmatrix} 3 & 0 & 2 \\ 0 & 2 & 0 \\ 1 & 0 & -1 \end{pmatrix}.$$

Set $L = Q_L R_L$ and $R = R_R$. The two equal columns of L do not create a third range direction. In contrast, R has rank three.

L: rank 2

$\frac{3}{5}$	0	$\frac{3}{5}$
$\frac{4}{5}$	0	$\frac{4}{5}$
0	1	0
0	0	0

R: rank 3

3	0	2
0	2	0
1	0	-1

G: 2×3

5	0	0
0	2	0

$$LR^T = Q_L G$$

The stored width is three; the left range has only two independent directions.

The actual core is

$$G = R_L R_R^T = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix}, \qquad LR^T = Q_L G.$$

Its dimensions are the two actual input ranks, two and three. This is not inferred by merely discarding a column with a small norm. The range factor Q_L has orthonormal columns and reconstructs L exactly.

The preserved formal boundary

The sixth release constructs orthonormal bases of the actual column spaces. It then assumes exact ordered thin-SVD data for the core, as the manuscript instructs the reader to take an SVD. It does not assume an optimality or tail-error conclusion. Nor does it certify a floating-point QR or SVD implementation simply because those routines use similar notation.

This example has a diagonal rectangular core, so its singular directions are explicit. No iterative numerical factorization is needed to identify the two nonzero singular values, five and two.

> **Read this as:** Compactness is a statement about actual range spaces. The number of stored factor columns can exceed the dimensions that matter for recompression.

V12 / COMPRESSION, KERNEL WITNESSES, AND TAILS

Why a core optimum is global

The product from the previous page is

$$A_0 = LR^T = \begin{pmatrix} 3 & 0 & 0 \\ 4 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Keeping only its first singular direction gives a rank-one approximation with spectral and Frobenius error exactly two. Keeping none gives errors five and $\sqrt{29}$; keeping both gives zero.

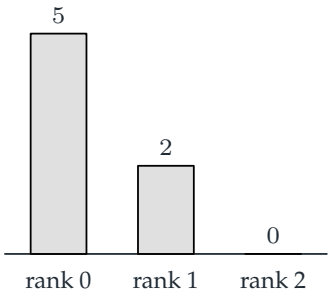
LR^T

3	0	0
4	0	0
0	2	0
0	0	0

rank-one output

3	0	0
4	0	0
0	0	0
0	0	0

spectral error



Frobenius errors: $\sqrt{29},\ 2,\ 0$. The rank-one choice loses only the second mode.

Why an outside competitor cannot do better

For isometric outer factors, compression $C(B) = Q_L^* B Q_R$ cannot increase rank or either matrix norm. Every ambient rank-k competitor therefore supplies an admissible core competitor. The lifted core residual has exactly the core residual norm. These two facts transfer a core optimum to the entire ambient matrix space, not just to competitors already lying in the supplied ranges.

For a d-dimensional diagonal core and $0 \le k < d$, the spectral lower bound has a concrete witness. A rank-at-most-k competitor has a nonzero kernel vector in the first $k+1$ coordinates. On that vector the diagonal matrix acts with norm at least σ_{k+1} times the vector norm. The competitor cannot cancel that direction.

The Frobenius proof uses an orthonormal basis of the competitor's kernel. Its row weights lie in $[0, 1]$ and sum to the kernel dimension. Weighting the decreasing squared singular values forces at least the discarded tail. The two norm arguments are distinct; one is not a graph of the other.

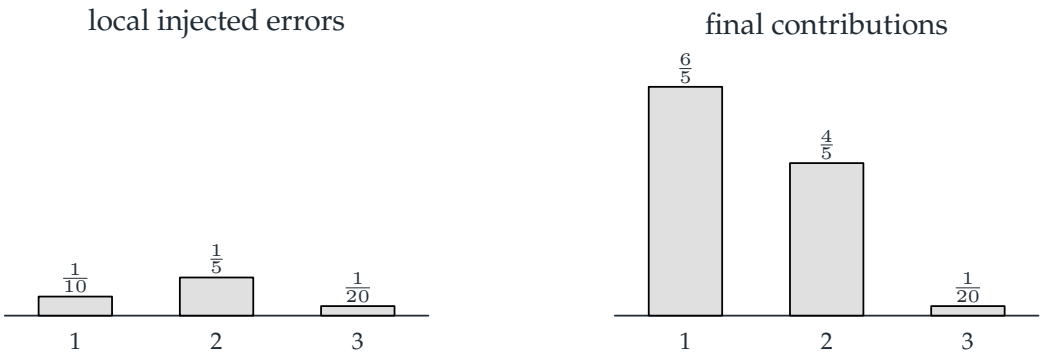
> **Read this as:** Optimality is not inferred from the appearance of the singular-value plot. It follows from compression and a lower bound valid against every rank-bounded competitor.

V13 / FACTOR PERTURBATION VERSUS RECOMPRESSION

Errors enter at different places

Let the exact scalar factors be $(2, 3, 4)$ and start from $p_0 = 1$. If additive errors $(1/10, 1/5, 1/20)$ enter *after* each partial multiplication, then

$$\hat{p}_j = a_j \hat{p}_{j-1} + e_j, \qquad \hat{p}_3 - p_3 = 12e_1 + 4e_2 + e_3 = \frac{41}{20}.$$



$\hat{p}_j = a_j \hat{p}_{j-1} + e_j$, with $(a_1, a_2, a_3) = (2, 3, 4)$.

The same local numbers, a different model

If instead the factors themselves are perturbed, the computed product is $(4+e_3)(3+e_2)(2+e_1)$. Its error is $402/125$, not $41/20$. The factor-perturbation telescoping formula contains already-perturbed later factors; the additive recompression formula contains the original later factors.

The visual comparison uses the additive model. The first error is multiplied by both remaining factors. The last error is not amplified at all. Summing only the local error magnitudes gives $7/20$, which is smaller than the actual final error and therefore not a valid bound here.

In a matrix chain

The corresponding formulas replace scalar magnitudes with operator norms and respect multiplication order. A bound can be conservative when contributions cancel, but it must not omit possible amplification. Singular-tail errors from a recompression step become useful only after they are inserted into the correct global error model.

> **Read this as:** An error tolerance needs an entry point. A bound for perturbing factors and a bound for compressing partial products are not interchangeable.

V14 / SUPPORT IS AN INCLUSION

Two routes can cancel exactly

Let K have transfers $0 \to 1$, $0 \to 2$, $1 \to 3$, and $2 \to 3$, with coefficients $1, 1, 1, -1$. There are two length-two routes from packet zero to packet three, but their contributions sum to zero.

K

0	0	0	0
1	0	0	0
1	0	0	0
0	1	−1	0

path counts at length two

0	0	0	0
0	0	0	0
0	0	0	0
2	0	0	0

$$1 + (-1) = 0$$

$$K^2 = 0$$

Structural support is an upper bound. It does not encode destructive cancellation.

$$(K^2)_{30} = K_{31}K_{10} + K_{32}K_{20} = 1 - 1 = 0, \qquad K^2 = 0.$$

The matrix of path counts sees the two possibilities and forgets the signs. It therefore predicts possible support but not actual support. Replacing the inclusion $\operatorname{supp}(AB) \subseteq \operatorname{supp}(A) \circ \operatorname{supp}(B)$ by equality would be wrong even in this four-dimensional model.

What an acyclic graph does guarantee

If the longest strict directed path has h edges, then no term of K^{h+1} can survive. Thus $K^{h+1} = 0$. This is an upper bound on the nilpotence index, not an assertion that $K^h \neq 0$. Here the graph has paths of length two, while coefficient cancellation already gives $K^2 = 0$.

A structurally permitted block may also vanish because a product of rectangular maps is zero before any sum is taken. A support graph cannot distinguish that mechanism from cancellation among separate routes. The actual block coefficients are therefore part of a proof-bearing operator representation.

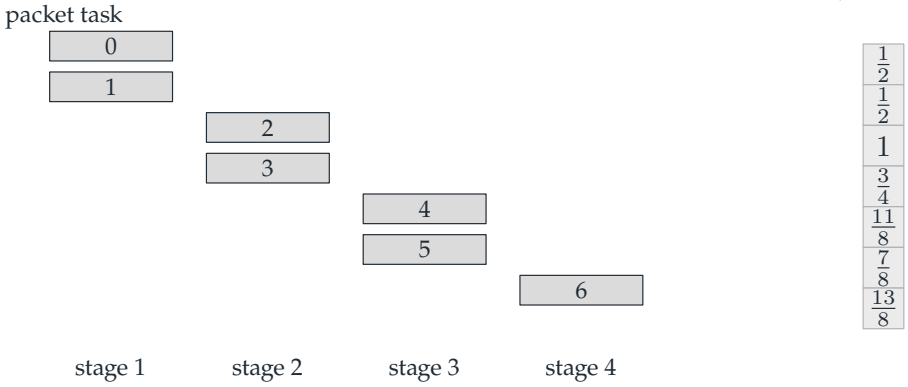
> **Read this as:** Use graph reachability to exclude impossible transfers. Use the algebra to determine which permitted transfers actually survive.

V15 / TOPOLOGICAL LEVELS WITH EXACT OUTPUTS

A solve has a dependency clock

For seven scalar packets, put two on the diagonal and -1 on each declared incoming edge. The edges are $0 \to 2$, $1 \to 2$, $1 \to 3$, $2 \to 4$, $3 \to 4$, $3 \to 5$, $4 \to 6$, and $5 \to 6$. Solve $(2I + K)x = \mathbf{1}$.

$$x = (2I + K)^{-1}\mathbf{1}$$



Read the stages as dependencies

Packets zero and one are ready first. Packets two and three use those values. Packets four and five then become available, followed by packet six. For instance,

$$x_4 = \frac{1 + x_2 + x_3}{2} = \frac{11}{8}, \qquad x_6 = \frac{1 + x_4 + x_5}{2} = \frac{13}{8}.$$

All entries in the displayed solution are checked by multiplying the original triangular matrix by the resulting vector.

When is four a minimum?

There are four vertices on a longest dependency path. Under the paper's unit-duration, unlimited-concurrency model, four stages are both necessary and attainable. If only one execution slot is available, seven tasks require seven slots of work. If task durations differ, the relevant critical path is weighted.

The theorem is not a universal GPU scheduling result. Transfers, synchronization, contention, and nonunit task costs change the model. The exact benefit here is that independent rows in one level need not wait for one another's mathematical results.

> **Read this as:** A topological schedule separates dependencies from resources. Its optimal depth is conditional on the task-duration and concurrency model that defines a stage.

V16 / SCHUR FILL AND ITS CANCELLATION

Elimination creates new interactions

Retain the first two coordinates of

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

The third coordinate is hidden. Its invertible block is $[2]$, so eliminating it produces

$$S = A_{RR} - A_{RH}A_{HH}^{-1}A_{HR} = \begin{pmatrix} 3/2 & -1/2 \\ -1/2 & 3/2 \end{pmatrix}.$$

A before elimination

2	0	1
0	2	1
1	1	2

$S = A_{RR} - A_{RH}A_{HH}^{-1}A_{HR}$

$\frac{3}{2}$	$-\frac{1}{2}$
$-\frac{1}{2}$	$\frac{3}{2}$

cancelling variant

$\frac{3}{2}$	0
0	$\frac{3}{2}$

A hidden route creates $-1/2$; adding $+1/2$ in the original block cancels it.

The hidden route has a sign

The retained coordinates had no direct coupling, yet both touch the hidden coordinate. The resulting correction couples them by $-1/2$. This is fill: a new active retained block produced by an eliminated route.

Now change both original retained off-diagonal entries to $+1/2$. The same hidden correction cancels them, leaving the diagonal matrix $\frac{3}{2}I$. Thus a structural route can create or remove visible coupling depending on the direct coefficient already present.

A stronger inverse statement

For an invertible finite packet matrix, inverse support lies in the directed transitive closure. A reachable-coordinate subspace is invariant under the matrix. Injectivity of its restriction implies surjectivity in finite dimension, so the inverse preserves it as well. This proof uses finite-dimensional invertibility, not a convergent geometric series.

The checker verifies both Schur complements and, in each invertible example, equality between S^{-1} and the retained block of A^{-1}.

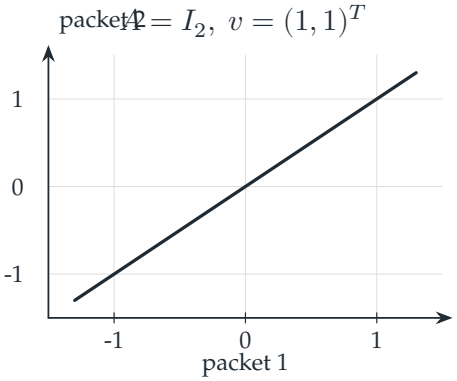
> **Read this as:** Removing coordinates does not remove their influence. It changes the retained operator, with possible fill, cancellation, and new conditioning constraints.

V17 / A STRICT KRYLOV INCLUSION

Invariant does not mean independently addressable

Take $A = I_2$, split into the two coordinate packets, and start with $v = (1,1)^T$. Every packet is invariant. Nevertheless every polynomial applies the *same* scalar to both coordinates:

$$p(A)v = p(1)(1,1)^T, \qquad \dim \mathcal{K}_m(A,v) = 1 \quad (m \geq 1).$$



$$p(A)v = p(1)(1,1)^T$$

$$\dim \mathcal{K}_m(A,v) = 1$$

Independent local spans have dimension 2.

The direct sum of the two local Krylov spans is all of $\mathbb{Q}^2$. It permits independent coefficients in the two packets. The global Krylov space does not. This is why the manuscript states an inclusion, not an unconditional equality.

What breaks the linkage?

With $A = \mathrm{diag}(1,2)$, the vectors v and $Av = (1,2)^T$ are independent. Two polynomial samples now distinguish the packet labels. Alternatively, allowing the packet projectors as extra address operators separates the coordinates even when $A = I_2$. Both changes enlarge the addressing power, but in different ways.

When invariance is only approximate

If $A = D + K$ with D packet diagonal, then

$$\|A^j - D^j\| \leq jM^{j-1}\|K\|, \qquad M = \max\{\|A\|, \|D\|\}.$$

This controls departure from the decoupled iterate. It is not a claim that the dimension or numerical conditioning of a Krylov basis varies continuously. Near dependence can remain difficult even when the operator error is small.

> **Read this as:** A single global polynomial can lock otherwise invariant packets together. Independence of subspaces is not the same as independence of the allowed actions.

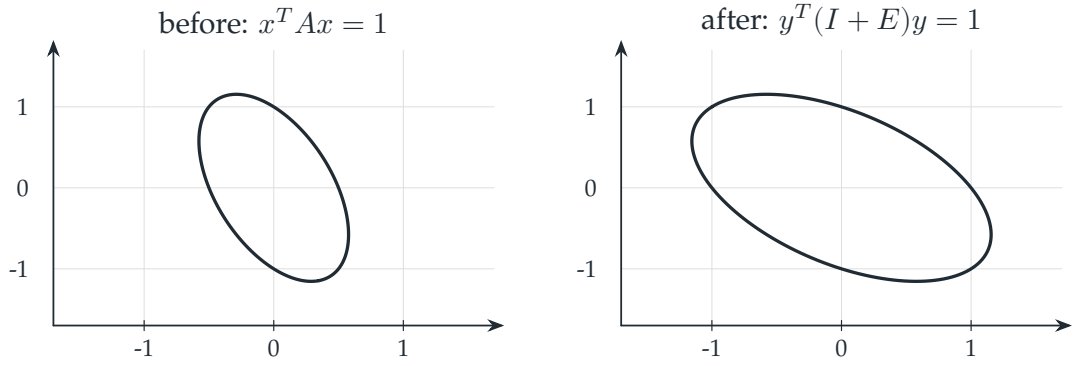
V18 / BLOCK JACOBI AS A GEOMETRIC CHANGE

Measure coupling after normalization

Let

$$A = \begin{pmatrix} 4 & 1 \\ 1 & 1 \end{pmatrix}, \quad D = \operatorname{diag}(4, 1), \quad B = D^{-1/2} = \operatorname{diag}(1/2, 1).$$

Then $BAB = I + E = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}$. Its exact eigenvalues are $1/2$ and $3/2$, giving condition number three.



Exact normalized eigenvalues: $1/2, 3/2$. Hence $\kappa_2 = 3$.

The unit circles of the energy norms

The left ellipse is $x^T Ax = 1$. Writing $x = By$ gives the right ellipse $y^T(BAB)y = 1$. The drawing samples these explicitly defined curves; the norm and eigenvalue equalities are checked symbolically.

Here $\|E\|_2 = 1/2$. The manuscript's general inequality

$$\kappa_2(D^{-1/2}AD^{-1/2}) \le \frac{1+\eta}{1-\eta}, \qquad \|E\|_2 \le \eta < 1,$$

is attained. The raw off-diagonal entry is one; its meaningful size is one-half only after normalization by the diagonal energy scales.

Keep the convergence input separate

The condition bound can be inserted into a conjugate-gradient convergence estimate under that method's positive-definite assumptions. The CG estimate is one of the separately tracked classical inputs. The algebraic normalization theorem does not certify every stopping rule, floating-point iteration, or arbitrary preconditioner.

> **Read this as:** A small raw coupling is not scale-invariant. Normalize against the diagonal blocks before deciding whether the coupling is weak enough for a condition bound.

V19 / DISCOVERY AND PERTURBATION CERTIFICATES

Twelve small blocks, one coherent error

Split $\mathbb{Q}^4$ into scalar packets and let

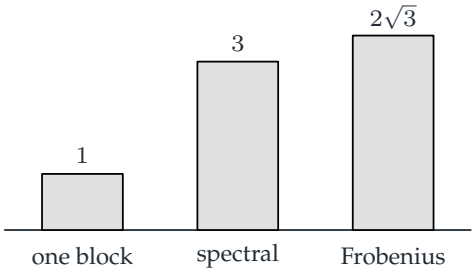
$$A = I + E, \qquad E = \frac{1}{100}(\mathbf{1}\mathbf{1}^T - I).$$

Dropping every off-diagonal block leaves I. Each discarded entry is only $1/100$, but the total error is not bounded by that single-entry size.

$100E$: twelve omitted entries

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

error norms, in units of $1/100$



$\|E\|_F^2 = 12/10^4$, while $E\mathbf{1} = (3/100)\mathbf{1}$.

Two norms answer different questions

There are twelve omitted entries, so $\|E\|_F^2 = 12/10^4$. The all-one vector is an eigenvector with eigenvalue $3/100$, while the three orthogonal directions have eigenvalue $-1/100$. Hence $\|E\|_2 = 3/100$.

For orthogonal coordinate blocks, the squared Frobenius norms add exactly. Operator norms instead require a suitable bound or further spectral information. The maximum omitted block norm is generally insufficient.

Input sensitivity is additional information

The packet perturbation estimate is

$$\|(\widehat{A} - A)x\| \le \sum_e \|E_e\| \, \|P_{\alpha(e)}x\|.$$

It can be much tighter for an input that scarcely occupies an affected source packet. To obtain a uniform operator bound, the input dependence must be bounded uniformly over unit vectors. A certificate for one input is not automatically a certificate for every right-hand side.

These are a posteriori checks for supplied projectors. They do not discover the best packet system or prove that a thresholding heuristic is globally optimal.

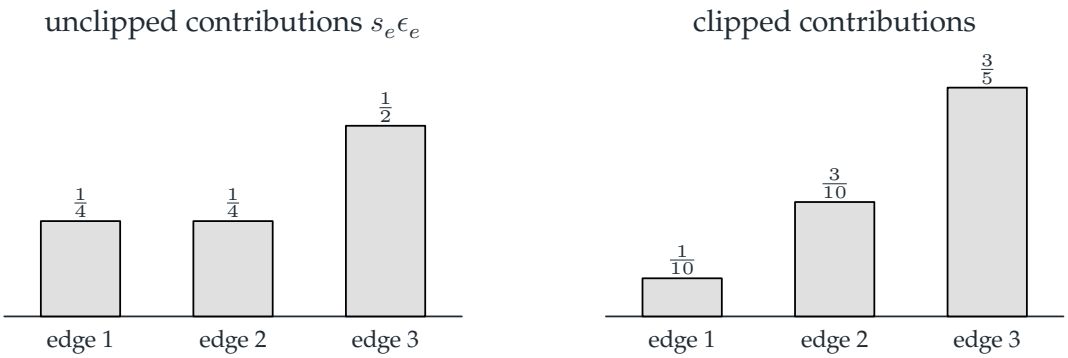
> **Read this as:** A proposed compression needs a residual certificate in the requested norm and input scope. Small individual blocks can align into a larger collective error.

V20 / CONTINUOUS AND CLIPPED PRECISION

Spend error where it is cheapest

Take sensitivities $s = (1, 2, 4)$, cost coefficients $c = (1, 1, 2)$, and budget $\tau = 1$. Minimizing $\sum_e c_e \log(1/\epsilon_e)$ subject to $\sum_e s_e \epsilon_e = 1$ gives

$$\epsilon_e = \frac{c_e}{4 s_e}, \qquad \epsilon = (1/4, 1/8, 1/8).$$



Both allocations spend exactly one unit of error budget.

The budget is weighted

The three contributions to the error allowance are $(1/4, 1/4, 1/2)$. Equal cost coefficients give equal weighted contributions, not equal tolerances. The third edge has twice the cost coefficient of the second, which offsets its doubled sensitivity in this example.

Now impose upper tolerances $m = (1/10, 1/2, 1/2)$. The first edge must stop at its tighter cap. With multiplier $\lambda = 10/3$, the clipped solution is

$$\epsilon_e = \min\{m_e, c_e/(\lambda s_e)\}, \qquad \epsilon = (1/10, 3/20, 3/20).$$

Its weighted contributions are $(1/10, 3/10, 3/5)$ and still sum to one. The checker verifies the active cap, the multiplier equations, and the budget exactly.

From tolerance to a real data type

The continuous objective is a model. Actual kernels offer discrete precisions and may use different accumulation types. A tolerance below one does not identify a particular hardware format. The finite option program on the next page is the correct next step once admissible costs and error bounds have been supplied.

> **Read this as:** Precision allocation is an optimization over certified sensitivities and costs. It does not infer error bounds from a data-type name alone.

V21 / THREE EDGES, TWENTY-SEVEN ASSIGNMENTS

An exact budget is not a greedy choice

Each edge has three allowed $(\text{integer weight}, \text{cost})$ options:

$$1 : (0, 7), (2, 3), (4, 1),$$
$$2 : (0, 6), (1, 4), (3, 1),$$
$$3 : (0, 9), (2, 5), (5, 1).$$

There are exactly $3^3 = 27$ full assignments. The table stores the minimum cost at each *exact* weight; columns zero through eight are displayed.

	0	1	2	3	4	5	6	7	8
edges 0	0	∞	∞	∞	∞	∞	∞	∞	∞
edges 1	7	∞	3	∞	1	∞	∞	∞	∞
edges 2	13	11	9	7	7	4	∞	2	∞
edges 3	22	20	18	16	14	12	12	9	8

Columns are exact weights; the budget optimum is the minimum across allowed columns.

Read the final row correctly

At budget four, minimize across columns zero through four. The optimum costs 14: choose weights $(2, 0, 2)$ with costs $(3, 6, 5)$. The zero-weight choice on edge two is its first, more expensive option. A locally cheap choice can consume too much of the shared budget.

The checker compares the recurrence with complete enumeration for every budget from zero through twelve. Infeasible states remain $+\infty$; they are not zero-cost states.

Quantization must be conservative

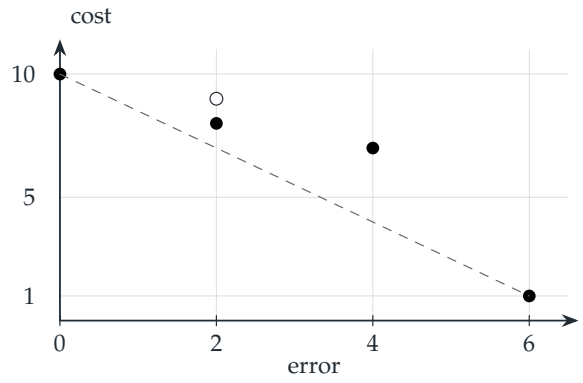
Suppose real errors are $3/50$ and $2/25$, and one integer unit represents $1/10$. Rounding down gives two zero weights, even though the true sum is $7/50 > 1/10$. Rounding up gives a certified upper envelope instead. The budget method is exact for its integers; it cannot repair unsafe integers supplied by a preprocessing step.

> **Read this as:** The table optimizes the declared option set exactly. Accuracy depends separately on the validity and conservativeness of the option weights.

V22 / FIVE PRUNINGS, ONE NONCONVEX FRONTIER

A penalty can miss the budget optimum

A root r has children a, b, each with two terminal children. Keeping r costs one with error six; splitting it adds overhead one. Keeping either a or b costs three with error two. Splitting a costs four with zero error, and splitting b costs five with zero error.



Budget $B = 2$: cost 8

Penalty optimum: root or full split

$8 + 2\lambda \le 1 + 6\lambda$
and $8 + 2\lambda \le 10$
cannot both hold.

The five complete prunings have $(\text{cost}, \text{error})$ pairs

$$(1, 6),\ (7, 4),\ (9, 2),\ (8, 2),\ (10, 0).$$

At budget two the unique cheapest pruning is $\{a_0, a_1, b\}$, costing eight. It is Pareto optimal, but never minimizes cost $+\lambda$ err for any $\lambda \ge 0$.

See the incompatible inequalities

To beat the root, that pruning would require $8 + 2\lambda \le 1 + 6\lambda$, hence $\lambda \ge 7/4$. To beat the full split, it would require $8 + 2\lambda \le 10$, hence $\lambda \le 1$. No multiplier meets both conditions.

The exact budget recurrence retains the point because it enumerates admissible weight allocations. A sweep of Lagrangian parameters sees only supported frontier points. The two compiler formulations are therefore genuinely different, even on this tiny tree.

> **Read this as:** A penalty parameter is not a universal substitute for an explicit budget. Discrete, nonconvex frontiers can contain useful optima that no penalty selects.

V23 / LOCAL DEPENDENCIES AND STRICT MARGINS

Recompile the changed part, not everything

Use the same tree with $\lambda = 2$. Increase only the leaf cost at a_0 from two to three. The exact subtree values change at a_0, its parent a, and the root r. Every value in the b subtree remains untouched.

$a0$	$a1$	a	$b0$	$b1$	b	r
2	2	4	2	3	5	10
3	2	5	2	3	5	11

Top: old values. Bottom: new values. Only a0, a, and r change.

Why the update is local

A dynamic-programming value depends only on the local keep/split data and its children's values. A changed leaf can therefore affect its ancestors but not a disjoint sibling subtree. Recomputing the dependency closure yields the same global optimum as a fresh traversal. In this example, the root value increases from ten to eleven.

Strict gaps protect decisions

Before the update, the root keep score is 13 and its split score is 10, giving gap three. If each score changes by less than $3/2$, their strict ordering is retained. Allowing exactly $3/2$ in both directions can produce a tie, so the strict inequality in the stability statement matters.

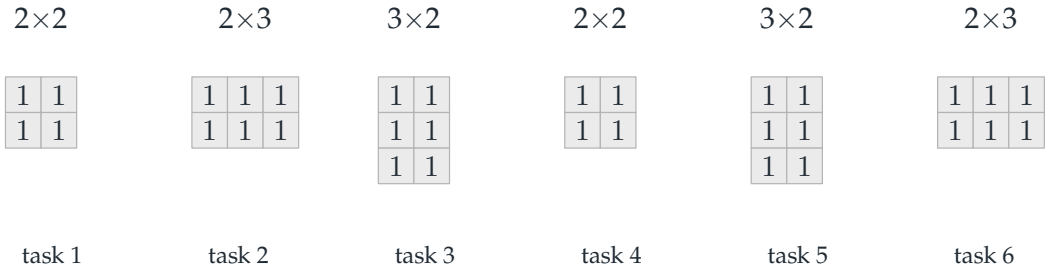
The bound applies to the completed scores after child subproblems have been evaluated. Bounds on raw leaf perturbations must first be propagated to those scores. Shared caches, overlapping subproblems, or global rank constraints can introduce dependencies outside a tree and invalidate this simple locality argument.

> **Read this as:** Local recompilation is exact because the model has local dependencies. The locality is a mathematical hypothesis, not an automatic property of every compiler optimization.

V24 / A LAUNCH MODEL IS NOT A BANDWIDTH MODEL

Batch shapes, account for traffic

Six independent blocks have the displayed dimensions. When right-hand-side count, scalar type, accumulation type, and kernel contract also agree within each class, they can be grouped into three batches. That reduces the modeled launch count from six to three without changing their arithmetic.



Three shape classes, six independent tasks.

Grouping does not remove arithmetic, data transfer, or shared-output dependencies.

Independence must be real

The batching statement assumes compatible backend support and independent block tasks. If several blocks accumulate into one target, the reduction or accumulation policy is additional work. Equal matrix dimensions alone do not remove that dependency. Different accumulation types also define different shape classes under the paper's convention.

Count the data separately

For three diagonal 2×2 blocks, eight-byte scalars, and q right-hand sides, the stated one-pass model moves

$$B_{\text{stream}} = 8 \cdot 3(4 + 2q + 4q) = 96 + 144q \text{ bytes.}$$

It gives 240 bytes at $q = 1$ and 672 bytes at $q = 4$. It deliberately assumes no reuse across edges. A cache-aware implementation may have another traffic model, but it must state one.

Given operation capacity P and bandwidth W, $T \geq \max(F/P, B/W)$ is a lower bound, not an achieved runtime. Launch overhead, imperfect overlap, and utilization can make the actual time much larger.

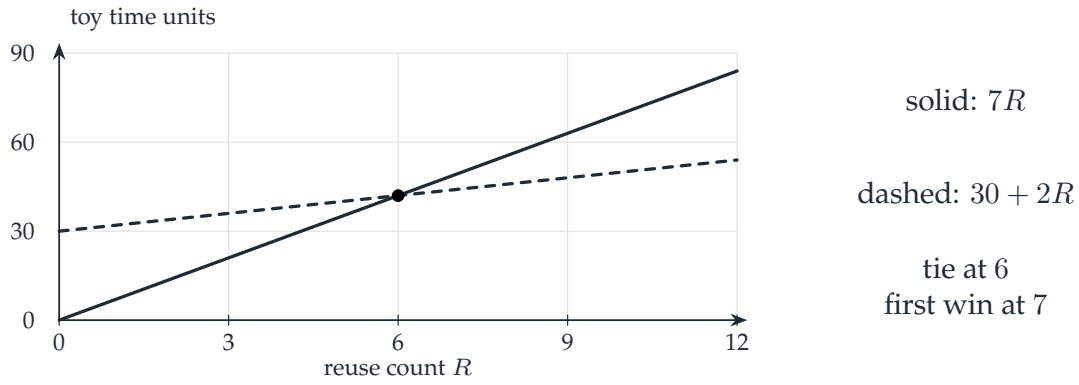
> **Read this as:** Grouping can remove launches while leaving the same products and data movement. Arithmetic, traffic, and scheduling are related constraints, not one interchangeable cost.

V25 / SETUP AND REUSE IN ONE LINEAR MODEL

The first win comes after the tie

Suppose the baseline costs seven units per use. A candidate costs 30 units to set up and two per use. These are synthetic constants chosen to expose the exact threshold:

$$T_b(R) = 7R, \qquad T_c(R) = 30 + 2R.$$



At $R = 6$, both cost 42. The candidate becomes strictly faster at $R = 7$, where it costs 44 instead of 49. The minimum profitable integer reuse count is

$$\left\lfloor \frac{30}{7-2} \right\rfloor + 1 = 7.$$

Replacing the floor-plus-one rule with a ceiling would be wrong when the ratio is already an integer and strict improvement is required.

Which costs belong to setup?

Discovery, compilation, host/device transfer, graph capture, and factor preparation can have different reuse lifetimes. If a matrix changes but its shape does not, a captured schedule might remain reusable while its numerical factors do not. A single setup number is meaningful only after those lifetimes are fixed.

Materialization has the opposite tradeoff: pay once for an explicit representation, then use a potentially faster dense downstream operation. It pays off only when the explicit per-use cost is lower and the reuse count exceeds the corresponding conversion threshold.

> **Read this as:** A faster steady-state kernel can lose end-to-end. State the reuse regime and count the tie separately from the first strict win.

V26 / A BENCHMARK ROW IS A CONTRACT

Same output before comparing times

Consider four synthetic timing rows for a product. Row B returns factors, while row A returns an explicit matrix. A ratio $12/2 = 6$ does not compare two implementations of the same returned object. Adding the decoding cost changes the candidate to row C.

row	output	scope	time
A	explicit matrix	materialization included	12
B	low-rank factors	factor product only	2
C	explicit matrix	factor product + decode	14
D	explicit matrix	different input precision	1

Synthetic times: A versus C is contract-aligned. A versus B is not a sixfold speedup.

What a fair match retains

The input values, dimensions, layout, source precision, correctness reference, returned output, and timing scope must agree. Row D changes the input precision; that can be a useful accuracy tradeoff, but it is not the same case merely because the output has the same shape.

If a downstream task consumes factors directly, row B has a valid compressed-output contract. It should be compared with another method that supplies that downstream result, not forced to materialize an unused array. Fairness does not mean every algorithm must use the same internal representation.

How the original measurements are treated here

The preserved manuscript contains its historical CPU/GPU corpus. Replaying its figure generator redraws those saved observations. It does not rerun the recorded device, establish present hardware performance, or transform logical avoided work into physically executed floating-point operations. No new GPU timing is added by this guide.

Exact representation gains, device gains, kernel gains, and amortization gains can be isolated only through appropriately matched rows. A total ratio that mixes them is still an observation, but not a decomposition theorem.

> **Read this as:** A timing number without its case and scope is incomplete data. Preserve the mathematical output contract before claiming a speedup.

V27 / FACTORS, APPLICATION, AND AN INDEPENDENT REFERENCE

Close the loop on the running example

Return to the two operators from V9. The exact product representation has diagonal part $D_A D_B$ and two-column correction factors U_C, V_C. Apply it to the original $x = (1, -1, 2, 0, -2, 3)^T$.

$$V_C^T x = (2, 13)^T, \qquad (AB)x = D_A D_B x + U_C(V_C^T x).$$

$V_C^T x$	$D_A D_B x$	$U_C(V_C^T x)$	$(AB)x$
2	1	25	26
13	−5	21	16
	18	25	43
	6	21	27
	−12	25	13
	44	21	65

Two intermediate scalars and six output coordinates, with an exact dense reference.

A complete reader calculation

The local contribution is $(1, -5, 18, 6, -12, 44)^T$. The correction is $(25, 21, 25, 21, 25, 21)^T$. Their sum is

$$(26, 16, 43, 27, 13, 65)^T.$$

The checker forms the dense matrices A and B independently, multiplies them, applies the product to x, and compares all six components. The dense array is a validation reference, not a requirement of the factored application route.

Choose what happens next

For another input, reuse the factors. For a required explicit matrix, pay materialization. For a long product chain, track rank growth and decide whether recompression is useful. If approximation is introduced, transport its error through the later operators rather than reporting only a local singular tail.

This small example does not predict a speedup: at dimension six, implementation overhead can dominate. Its purpose is to make the semantics auditable. The same identity can then be used inside a larger, separately benchmarked workload.

> **Read this as:** The full route is: represent the exact operator, retain every interaction, return the requested object, and validate against an independent reference. Performance is a further experiment.

V28 / FORMAL, EXACT, NUMERICAL, AND EMPIRICAL EVIDENCE

What this edition certifies

The preserved v0.6.0 release records 56 of 56 manuscript-proved finite items, 392 named theorem declarations in 40 modules, and 66 separately counted construction audits. Of the theorem declarations, 19 are reused foundation results. These are inherited records, not a new Lean run for this editorial edition.

Evidence	Its precise use here
Saved Lean records	Preserve the exact hypotheses and declared finite formalization boundary.
Fresh portable checks	Reproduce the guide's displayed matrices, identities, integer models, and finite optima. They are not extracted Lean proofs.
Vector drawings	Display explicit arrays, schedules, and sampled geometric formulas. Pixel accuracy is not algebraic certification.
Separate software checks	The release verification report records actual runs, tolerances, failures, and unavailable gates; these are separate from this guide's exact arithmetic.
Historical GPU corpus	Remains a record of the supplied original cases and environment. It is not a new measurement.

The recompression distinction

The sixth block derives best-rank and singular-tail conclusions from ordered thin-SVD equations, rather than assuming an optimality certificate. It does not independently establish general SVD existence, certify a floating-point QR/SVD algorithm, or identify every result with a canonical singular-values API. Those are separate statements.

The four classical entries remain visible

The manuscript separately tracks Eckart–Young–Mirsky, the conjugate-gradient energy estimate, the floating-point product model, and simultaneous spectral decomposition. The strengthened ordered-SVD proof does not silently promote all associated existence and numerical-algorithm claims.

Limits of exact compilation

The optimizer needs a finite candidate tree with additive data. Nonlocal cache effects, overlapping decompositions, global rank coupling, and unrestricted discovery can destroy its optimal substructure. No new generic dense-matrix exponent or universal accelerator claim follows from this guide.

> **Read this as:** This is a reader-facing reconstruction of the paper's finite concepts, with reproducible examples and preserved source evidence. It changes exposition, not the formal theorem contract.

V29 / THE INNER PRODUCT IS PART OF THE CONTRACT

A complex range needs an adjoint

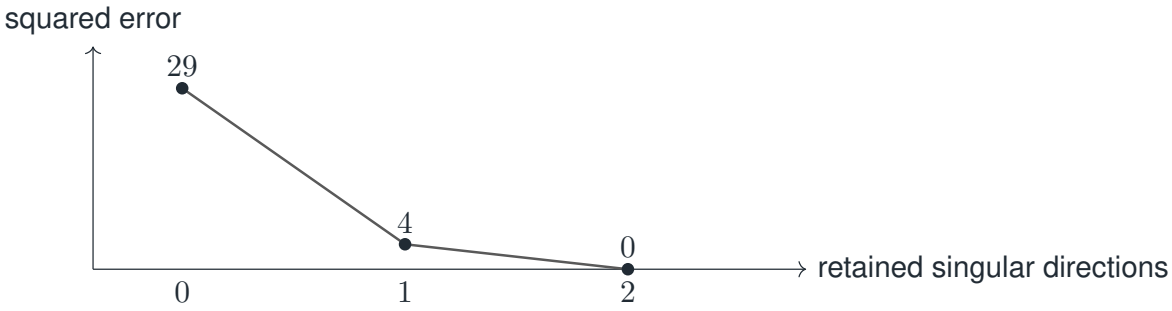
The earlier compact example is real. Over $\mathbb{C}$, an isometry means $Q^*Q = I$, with conjugation. Take the exact Gaussian-rational factors

$$Q_L = \begin{pmatrix} 3/5 & 0 \\ 4\mathrm{i}/5 & 0 \\ 0 & 1 \end{pmatrix}, \qquad Q_R = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{pmatrix}, \qquad G = \mathrm{diag}(5, 2).$$

Their product is a genuinely complex rectangular operator:

$$A = Q_L G Q_R^* = \begin{pmatrix} 3 & 0 \\ 4\mathrm{i} & 0 \\ 0 & -2\mathrm{i} \end{pmatrix}, \qquad A^*A = \mathrm{diag}(25, 4).$$

$A^*A = \mathrm{diag}(25, 4)$



$Q_L^*Q_L = I$ $\qquad Q_L^TQ_L = \mathrm{diag}(-7/25, 1)$

The small calculation exposes a wrong convention

The first column of Q_L has Hermitian squared length $9/25 + 16/25 = 1$. Its bilinear square without conjugation is $9/25 - 16/25 = -7/25$. Thus $Q_L^TQ_L$ is not the identity. The portable negative control replaces the adjoint by transpose and rejects the resulting isometry claim.

Keeping the first diagonal entry of G gives A_1 with its second column zero. The residual has squared Frobenius norm four and Gram matrix $\mathrm{diag}(0, 4)$, hence spectral norm two. Both errors agree with the discarded core singular value. Every displayed arithmetic operation occurs in $\mathbb{Q}(\mathrm{i})$.

What the finite check establishes

It checks both isometries, the reconstructed ambient matrix, its Gram matrix, and the explicit rank-one residual. Global best-rank optimality still comes from the manuscript's compression and lower-bound proof; this two-column calculation is a worked instance of that argument. No existence theorem or numerical SVD routine is executed.

> **Read this as:** Conjugation preserves the complex Euclidean norm. A transpose is correct in the vectorization identity on V7, but an adjoint is required for an isometric range lift.

V30 / A FINITE INVERSE WITH A CANCELLED ROUTE

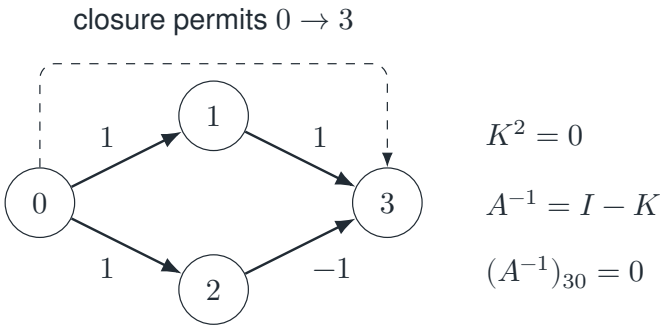
Inverse reachability is still an inclusion

Reuse the signed diamond from V14, now as a complete operator:

$$A = I + K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix}.$$

The two routes from coordinate zero to coordinate three cancel, so $K^2 = 0$. Consequently

$$A^{-1} = I - K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 1 \end{pmatrix}.$$



What the closure permits

The directed transitive closure includes $0 \to 3$, because it records existence of a route rather than the signed sum of its coefficients. Nevertheless $(A^{-1})_{30} = 0$. Reachability is therefore an upper envelope for inverse support, just as it is for product support.

The portable checker computes the inverse independently by rational row elimination, compares it with $I - K$, constructs Boolean reachability, and checks every inverse entry outside that closure. It separately rejects the assertion that every reachable transfer is nonzero.

Why the general proof is broader

This example happens to have a terminating Neumann polynomial. The general finite-dimensional theorem does not require nilpotence or a small norm. For a fixed source coordinate, its reachable-coordinate subspace is invariant under A. If A is invertible, the restricted map is injective. Equal finite domain and codomain dimensions make it surjective, so the inverse preserves the same subspace.

The finite-dimensional and invertibility hypotheses matter. In an infinite-dimensional space, an injective restriction can fail to be onto. A singular matrix does not have an inverse whose support can be tested at all.

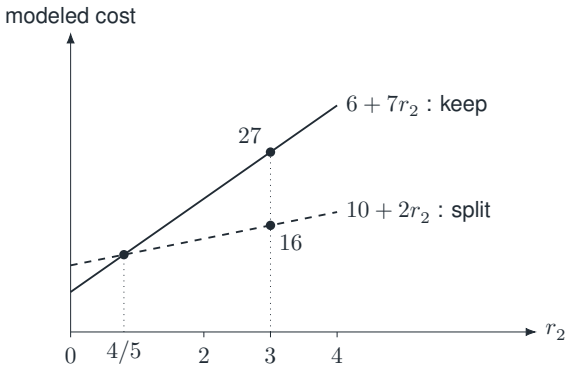
> **Read this as:** The inverse can never transmit outside the reachable subspace under the theorem's hypotheses. Inside that subspace, coefficient cancellation can still remove a structurally permitted interaction.

V31 / SHARED COMPILATION HAS ITS OWN OBJECTIVE

One tree can serve a weighted family

Use a tree with one root and two terminal children. There are exactly two admissible prunings: keep the root, or split into both children. All declared errors and the split overhead are zero. Their additive total costs are:

Cost component	Keep	Split
Setup paid once	4	2
Operator 1, per use	2	8
Operator 2, per use	7	2



With reuse weights $(r_1, r_2) = (1, 3)$, the aggregated objectives are

$$J_{\text{keep}} = 4 + 2 + 3 \cdot 7 = 27, \qquad J_{\text{split}} = 2 + 8 + 3 \cdot 2 = 16.$$

The exact shared optimum is split. Operator 1 alone prefers keep on its per-use costs, while operator 2 prefers split. Choosing independently and trying to combine those decisions would not solve the declared one-pruning problem.

A hand-computable decision threshold

Fix $r_1 = 1$ and vary $r_2 = r \geq 0$. Keeping costs $6 + 7r$ and splitting costs $10 + 2r$. Splitting is strictly cheaper when $r > 4/5$; its first integer reuse win is $r = 1$. The checker evaluates both admissible choices for integers zero through four and checks the displayed totals at three.

Where the common-tree proof applies

The family corollary aggregates local keep costs and errors, then uses the same exact subtree recurrence. In this example, each row denotes the sum of the selected local leaf terms; it does not conceal a nonlocal discount. A setup shared by distant unrelated nodes, overlapping candidates, or a global resource constraint would need an enlarged state or a different model.

> **Read this as:** Reuse weights change the objective before compilation. Shared optimality is exact for the supplied common tree and additive family costs, rather than for an unspecified set of future workloads.

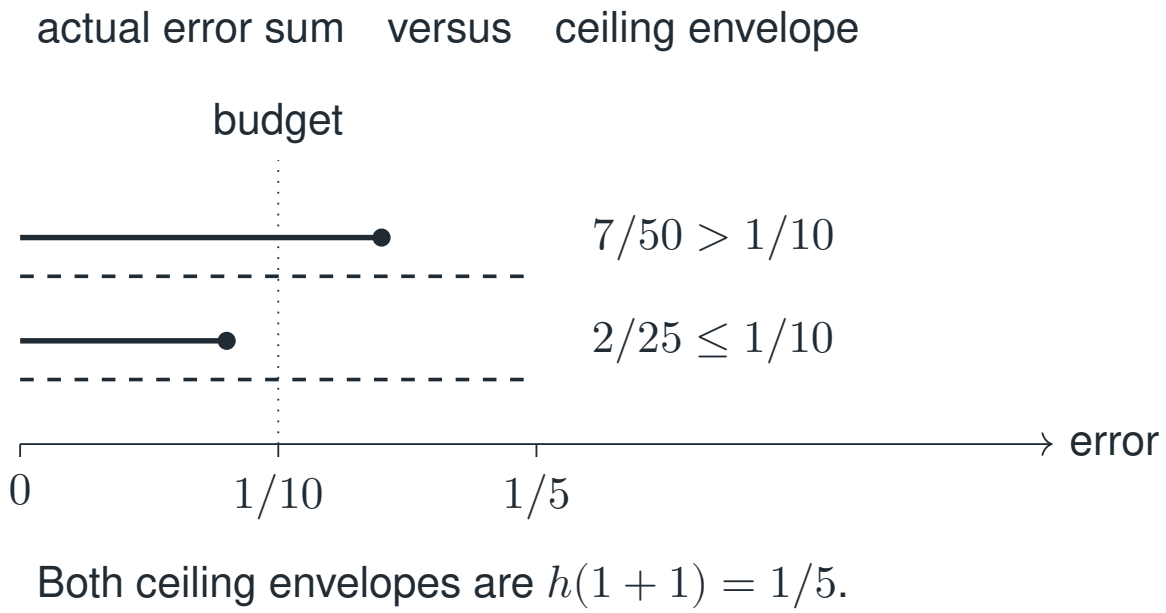
V32 / QUANTIZATION AND THE SIZE OF THE BUDGET TABLE

Safe integers can still be conservative

Let one integer unit represent $h = 1/10$ of certified error. For an individual nonnegative error e, choosing

$$w = \lceil e/h \rceil \qquad \text{ensures} \qquad e \le hw.$$

Thus integer feasibility $\sum w \le B$ implies the real bound $\sum e \le hB$. This implication is checked before applying the exact option or tree recurrence.



A false certificate and a conservative rejection

For errors $(3/50, 2/25)$, rounding down assigns weights $(0, 0)$ although their real sum is $7/50 > 1/10$. Rounding up gives $(1, 1)$ and correctly excludes the assignment from budget one.

Now change the errors to $(3/50, 1/50)$. Their sum is $2/25 \le 1/10$, but separate ceilings still give $(1, 1)$. The integer model rejects a genuinely real-feasible assignment. Conservative rounding protects soundness; it need not preserve every optimum of the unquantized problem.

Why the complexity is pseudo polynomial

For a declared integer budget B, a table has $B + 1$ possible exact-weight states. A direct merge checks at most $(B + 1)^2$ state pairs. At $B = 255$, this means 256 states and 65,536 candidate pairs at a merge, even though the budget fits in eight binary digits.

With a bounded number of children, the paper's direct tree implementation therefore has quadratic budget dependence at each node. Polynomial dependence on the numerical value of B does not mean polynomial dependence on the binary input length. These are counts of declared states and pairs, not measured runtimes or claims that every state is reachable.

> **Read this as:** The dynamic program is exact for its integer data. Conservative conversion and the numerical size of the budget are separate obligations that determine its accuracy and practical scale.

References

[1] Sivaram Ambikasaran and Eric Darve, *An o n log n fast direct solver for partial hierarchically semi separable matrices*, Journal of Scientific Computing **57** (2013), no. 3, 477–501.

[2] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz, *Minimizing communication in numerical linear algebra*, SIAM Journal on Matrix Analysis and Applications **32** (2011), no. 3, 866–901.

[3] Richard Bellman, *Dynamic programming*, Princeton University Press, Princeton, NJ, 1957.

[4] James R. Bunch and Christopher P. Nielsen, *Updating the singular value decomposition*, Numerische Mathematik **31** (1978), 111–129.

[5] Erin Carson and Nicholas J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, SIAM Journal on Scientific Computing **40** (2018), no. 2, A817–A847.

[6] Ronald R. Coifman and Mladen Victor Wickerhauser, *Entropy based algorithms for best basis selection*, IEEE Transactions on Information Theory **38** (1992), no. 2, 713–718.

[7] James W. Cooley and John W. Tukey, *An algorithm for the machine calculation of complex fourier series*, Mathematics of Computation **19** (1965), no. 90, 297–301.

[8] Chandler Davis and William M. Kahan, *The rotation of eigenvectors by a perturbation. iii*, SIAM Journal on Numerical Analysis **7** (1970), no. 1, 1–46.

[9] Timothy A. Davis, *Direct methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2006.

[10] James Demmel, Ioana Dumitriu, and Olga Holtz, *Fast linear algebra is stable*, Numerische Mathematik **108** (2007), no. 1, 59–91.

[11] Carl Eckart and Gale Young, *The approximation of one matrix by another of lower rank*, Psychometrika **1** (1936), no. 3, 211–218.

[12] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran, *Cache oblivious algorithms*, Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1999, pp. 285–297.

[13] Gene H. Golub and William Kahan, *Calculating the singular values and pseudo inverse of a matrix*, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis **2** (1965), no. 2, 205–224.

[14] Gene H. Golub and Charles F. Van Loan, *Matrix computations*, fourth ed., Johns Hopkins University Press, Baltimore, MD, 2013.

[15] William W. Hager, *Updating the inverse of a matrix*, SIAM Review **31** (1989), no. 2, 221–239.

[16] Nathan Halko, Per Gunnar Martinsson, and Joel A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review **53** (2011), no. 2, 217–288.

[17] Nicholas J. Higham, *Accuracy and stability of numerical algorithms*, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 2002.

[18] Nicholas J. Higham and Theo Mary, *Mixed precision algorithms in numerical linear algebra*, 2021, MIMS EPrint 2021.20, Sections 3–4; classical real dot-product forward-error model.

[19] Jia Wei Hong and H. T. Kung, *I o complexity: The red blue pebble game*, Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1981, pp. 326–333.

[20] IEEE Standards Association, *Ieee standard for floating point arithmetic*, IEEE Std 754 2019, 2019.

[21] Per Gunnar Martinsson and Joel A. Tropp, *Randomized numerical linear algebra: Foundations and algorithms*, Acta Numerica **29** (2020), 403–572.

[22] Leon Mirsky, *Symmetric gauge functions and unitarily invariant norms*, The Quarterly Journal of Mathematics **11** (1960), 50–59.

[23] NVIDIA Corporation, *Cuda c++ programming guide*, NVIDIA Corporation, Archive version 12.8.1, Section 3.2.8.7 (CUDA Graphs), accessed 6 October 2026.

[24] Ivan V. Oseledets, *Tensor train decomposition*, SIAM Journal on Scientific Computing **33** (2011), no. 5, 2295–2317.

[25] Yousef Saad, *Iterative methods for sparse linear systems*, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[26] Volker Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik **13** (1969), 354–356.

[27] Samuel Williams, Andrew Waterman, and David Patterson, *Roofline: An insightful visual performance model for multicore architectures*, Communications of the ACM **52** (2009), no. 4, 65–76.