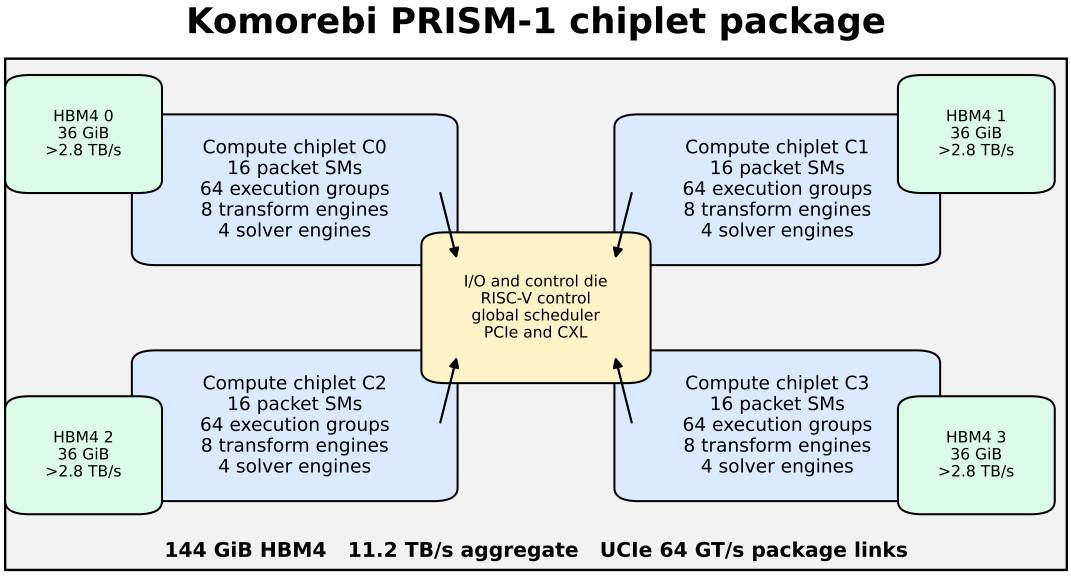
PROJECTOR-RESOLVED SEED CALCULUS

Komorebi PRISM-1

A Packet-Programmable Mixed-Representation Processor

CUDA-like packet execution, PMR–GEMM, transform atlases, structured operators, edge recovery, and certified reconstruction



Komorebi P.

Final updated architecture, programming model, CUDA compatibility path, compiler contract, verification plan, and executable model

Abstract

Komorebi PRISM-1 is a programmable chiplet accelerator for computations whose useful form is known before execution. Its native objects are packet graphs, transform plans, block operators, low-rank factors, Kronecker products, sparse factor chains, refinement trees, small algebraic systems, and reconstruction certificates. A dense array is one possible representation. The architecture never requires one to exist.

The final reference design contains four compute chiplets, 64 Packet Streaming Multiprocessors, 256 packet execution groups, configurable transform engines, resident small solvers, exact arithmetic support, 192 MiB of on-package SRAM, and four HBM4 stacks. Each Packet Streaming Multiprocessor schedules packet warps across four execution groups and a shared scratchpad. The execution model is CUDA-like in its hierarchy and programmability, but it keeps atlas identity, parent-child relations, output contracts, precision weights, and certificate state visible to hardware.

The compiler begins with Packet IR and chooses among explicit dense, block, block-sparse, low-rank, block-diagonal-plus-low-rank, Kronecker, sparse-chain, transform-domain, and packet-graph plans. PMR–GEMM supplies the recursive mixed-representation planner. The same Packet IR can lower to a CUDA compatibility backend or to PRISM. Earlier CUDA implementations already establish the software path through grouped matrix kernels, CUDA Graph replay, asynchronous transfers, FFT and sparse libraries, compressed products, Woodbury solves, and packetwise mixed precision.

The paper specifies the package, Packet Streaming Multiprocessor, memory hierarchy, network, command format, compiler, output contracts, certificate protocol, performance accounting, FPGA path, and verification plan. Existing CUDA measurements are kept separate from PRISM architecture estimates. Representation gain, kernel gain, architecture gain, and amortization are reported as different quantities. The design remains an architecture specification. It is not fabricated silicon and it does not claim measured PRISM frequency, power, or throughput.

Status and scope

This is the final updated architecture draft. The packet types, Packet IR, output contracts, command groups, Packet Streaming Multiprocessor boundary, memory hierarchy, compiler routes, error-budget rules, CUDA compatibility path, and verification ladder are specified. Circuit timing, SRAM banking, NoC virtual channels, floorplanning, hardened interface IP, and process-node choices remain implementation work.

The package baseline uses public interfaces that are available or specified at the time of this draft. UCIe 3.0 includes 48 and 64 GT/s die-to-die classes. PCIe 7.0 specifies 128 GT/s. Micron lists 36 GB HBM4 in high-volume production with bandwidth above 2.8 TB/s per stack. The management cores use the ratified RISC-V vector programming model. These facts establish a plausible component baseline. They do not validate the PRISM microarchitecture.

PRISM-1 does not preserve a dense matrix. It preserves the mathematical output contract. A packet program may remain a factor graph, transform plan, sparse chain, low-rank correction, or executable operator from beginning to end.

Executive summary

The design is built around seven practical decisions.

1. Keep representation visible from the calculus front end to the command queue.
2. Treat a dense array as an optional leaf or requested output, not as mandatory machine state.
3. Use one Packet IR for the CUDA backend and the PRISM backend.
4. Schedule packet warps on programmable Packet Streaming Multiprocessors.
5. Keep factors, transform plans, and packet graphs resident whenever later stages can consume them directly.
6. Keep exact archives separate from compact predictive states.
7. Make residuals and error bounds part of the commit path.

The software basis already exists. On the recorded RTX 3070 Ti Laptop GPU, an equal-block packet operator was measured at 0.046032 ms and ran 14.8919× faster than its matched device-resident dense GPU row. A BDLR packet path with CUDA Graph replay was measured at 0.079872 ms and ran 10.9355× faster than the matched dense GPU row. Graph replay improved the same packet representation by about 1.218×. A compressed BDLR product took 0.1582 ms, while materializing the output took 0.5073 ms. These measurements show that the calculus can already execute on CUDA without preserving a dense matrix.

PRISM addresses the work left after the representation has already improved. Its likely advantages come from resident factor reuse, packet gather and scatter, transform-to-solver handoff, grouped small-block dispatch, local refinement, deterministic graph replay, and certificate accumulation. The architecture model estimates gains of 2.96× for the resident edge-recovery chain, 1.64× for a fused multi-atlas path, and 1.42× for a shape-batched Bayesian portfolio. Those are architecture estimates. They are not silicon measurements.

The earlier 37.8× dense-route comparison is retained only as a total route model. It combines a better representation with a different architecture. It is not reported as a PRISM-over-CUDA architecture gain.

Contents

Abstract i

Status and scope ii

Executive summary iii

1 Design brief 1

1.1 The design problem . . . 1
1.2 Required behavior . . . 1
1.3 Design rules . . . 1
1.4 What this design does not promise . . . 2

2 Why this chip exists 3

2.1 The representation is part of the computation . . . 3
2.2 The target workload . . . 3
2.3 What the chip is not . . . 4
2.4 No dense preservation requirement . . . 4
2.5 CUDA-like execution hierarchy . . . 5
2.6 The CUDA implementation is the software foundation . . . 5

3 Mixed-representation execution 6

3.1 Output contracts . . . 6
3.2 Compatible middle bases . . . 7
3.3 Sparse chains and transform plans . . . 7
3.4 PMR–GEMM . . . 7
3.5 Hybrid fallback . . . 8

4 Reference package 8

4.1 Chiplet organization . . . 8
4.2 Reference targets . . . 9
4.3 Compute-chiplet layout . . . 10
4.4 Bandwidth balance . . . 10
4.5 Scale-down configurations . . . 11
4.6 Modeled arithmetic peaks . . . 12

5 Packet Streaming Multiprocessor 12

5.1 Multiprocessor anatomy . . . 12

5.2 Issue and completion path . . . 12
5.3 Packet scratchpad . . . 13
5.4 Execution groups . . . 13
5.5 Representation router . . . 14
5.6 Projector DMA . . . 14
5.7 Certificate and commit path . . . 14

6 Descriptors as the hardware language **15**
6.1 Packet descriptor entry . . . 15
6.2 Transfer edge descriptor . . . 15
6.3 Atlas descriptor . . . 16

7 Memory hierarchy **16**
7.1 Exact archive and active representations . . . 16
7.2 Representation-first storage . . . 16

8 Packet network and scheduling **17**
8.1 PSM mesh . . . 17
8.2 Packet warps and dependency tokens . . . 18
8.3 Local refinement . . . 18

9 Transform fabric **19**
9.1 One configurable engine family . . . 19
9.2 Exact analysis and synthesis . . . 20

10 Transfer engines **20**
10.1 Block diagonal and block sparse . . . 20
10.2 BDLR dataflow . . . 20
10.3 Where BDLR remains useful . . . 21
10.4 Compressed composition . . . 23
10.5 Woodbury solve . . . 23

11 Mixed precision and certificates **23**
11.1 Per-edge precision . . . 23
11.2 Precision allocation . . . 23

12 Unknown-edge and Gibbs reconstruction **24**
12.1 Why the chip includes a small-solver path . . . 24
12.2 Resident Hankel and Prony operations . . . 25

12.3 Small-solver organization . . . 25
12.4 Why native edge synthesis matters . . . 26

13 PR-LASM execution **26**
13.1 Exact archive and compact state . . . 26
13.2 Resident state layout . . . 26
13.3 Typed Bayesian programs . . . 27
13.4 Memory and planning . . . 27

14 Exact arithmetic mode **27**
14.1 Residue and modular packets . . . 27
14.2 Why this belongs on the chip . . . 27

15 Packet Seed Command interface **28**
15.1 Command groups . . . 28
15.2 Representative commands . . . 28
15.3 Completion record . . . 29
15.4 Protection and replay . . . 29

16 Compiler **29**
16.1 Routing policy . . . 29
16.2 Shared Packet IR . . . 29
16.3 Compiler flow . . . 30
16.4 Legality pass . . . 30
16.5 Representation pass . . . 31
16.6 PMR–GEMM planning . . . 31
16.7 Placement and fusion . . . 31
16.8 CUDA compatibility backend . . . 31
16.9 PRISM backend . . . 32
16.10 Worked BDLR compilation . . . 32
16.11 Certificate plan . . . 32

17 End-to-end execution traces **32**
17.1 Block diagonal plus low rank . . . 32
17.2 Unknown-edge reconstruction . . . 33
17.3 One PR-LASM step . . . 33
17.4 Why these traces matter . . . 33

18 Compositional correctness **33**

19 Performance accounting **34**
19.1 Four different sources of gain . . . 34
19.2 Measured CUDA foundation . . . 34
19.3 Representation reduction . . . 35
19.4 PRISM architecture model . . . 36
19.5 Total route model and architecture-only models . . . 36
19.6 Sensitivity and crossover . . . 37
19.7 Negative controls . . . 38

20 Power, area, and thermal design **39**
20.1 Power budget . . . 39
20.2 Area allocation . . . 40
20.3 Thermal behavior . . . 40

21 Reliability, protection, and recovery **40**
21.1 Protected state . . . 40
21.2 Failure containment . . . 41
21.3 Determinism . . . 41
21.4 Security boundary . . . 41

22 Programming model **41**
22.1 Four software layers . . . 41
22.2 Host control . . . 41
22.3 Device libraries . . . 41

23 Verification **42**
23.1 Verification ladder . . . 42
23.2 Behavioral tests . . . 42
23.3 RTL boundary . . . 43
23.4 What must be verified before tapeout . . . 44

24 Implementation path **44**
24.1 Stage 1: software and CUDA foundation . . . 44
24.2 Stage 2: trace-driven backend comparison . . . 45
24.3 Stage 3: FPGA Packet Streaming Multiprocessor . . . 45
24.4 Stage 4: one compute chiplet . . . 45
24.5 Stage 5: full package . . . 45
24.6 Development risks . . . 46
24.7 Architecture freeze checklist . . . 46

25 What would count as success **47**

26 Final architecture statement **47**

A Reference data sheet **47**

B Operation-to-engine map **49**

C Executable model and source layout **49**

D RTL module boundaries **49**

E Implementation baseline **50**

List of Figures

1.1 One packet command from archive to commit. The provisional state prevents an unchecked result from becoming visible to downstream work. . . . 2

1.2 The compiler is expected to route weakly structured work away from the packet path. . . 2

2.1 The seed-calculus operations become a hardware pipeline without losing their mathematical type. . . . 4

2.2 One operator may have several executable representations. The explicit array is one optional leaf. . . . 5

2.3 The CUDA and PRISM backends execute the same Packet IR and preserve the same output contracts. . . . 6

3.1 Materialization is required only when the requested output is the explicit full array. . . . 7

3.2 PMR–GEMM chooses the least-cost certified plan that satisfies the requested output contract. 8

4.1 The reference PRISM-1 package. The center die carries host I/O, global scheduling, RISC-V control, security, and telemetry. . . . 9

4.2 Reference compute-chiplet floorplan. The drawing is functional. Bank count, channel width, and physical dimensions remain open. . . . 10

4.3 Reference bandwidth levels. Local SRAM bandwidth is a circuit-level target and is not assigned a final number in this draft. . . . 11

5.1 One Packet Streaming Multiprocessor. A packet warp may use several execution forms without converting the operator into one explicit array. . . . 12

5.2 Command states from queue entry to commit or local replay. . . . 13

5.3 Execution-group resources inside the multiprocessor. The figure shows matrix, vector, memory, nonlinear, and certificate paths. . . . 14

6.1 One packet descriptor entry. The exact bit allocation may change during RTL design, but the semantic fields are fixed. . . . 15

6.2 A transfer edge descriptor keeps the operator class and certificate weight attached to the edge. 15

7.1 Representation-first memory hierarchy. The explicit full array is created only under a matching output contract. . . . 16

7.2 Stored BDLR representation for the model case $n = 8192$, $b = 64$, and $r = 8$. . . . 17

8.1 One chiplet mesh. Each displayed router node represents one Packet Streaming Multiprocessor rather than one execution group. . . . 18

8.2 A parent packet is replaced by a local child graph while remote descriptors remain unchanged. 19

9.1 The transform fabric supports several exact atlases without storing separate fixed data paths for each one. . . . 19

10.1 The BDLR path keeps the low-rank representation compressed. . . . 21

10.2 Modeled BDLR speedup across block width and correction rank. The comparison includes arithmetic, data movement, and graph overhead from the reference model. . . . 22

10.3 Dense-to-factor storage reduction for the same parameter sweep. . . . 22

11.1 More sensitive packet edges receive tighter precision targets. . . . 24

12.1 The self-calibrating edge-recovery graph from high-frequency packet to native singular synthesis. . . . 24
12.2 Illustrative resident pipeline for unknown-edge recovery. Transform and solver stages overlap where data dependencies permit. . . . 25
13.1 Exact archive, compact states, mechanism portfolio, and transition graph in one resident PR-LASM step. . . . 26
13.2 Mechanism families share resident state matrices and small-solver batches. . . . 27
15.1 Packet Seed Command groups. . . . 28
16.1 Reference routing logic. The final compiler will use calibrated costs, but the reject path remains part of the specification. . . . 29
16.2 Shared Packet IR makes CUDA the software foundation and matched baseline for PRISM. 30
16.3 The packet compiler preserves the output contract while changing representation, placement, precision, and schedule. . . . 30
16.4 The mixed-representation planner selects a certified plan under the requested output contract. 31
16.5 One BDLR operator through the compiler. The dense matrix is never formed. . . . 32
19.1 Representation, packet-CUDA, graph-replay, and output-contract gains answer different questions. . . . 34
19.2 Measured CUDA packet timings from the earlier structured-operator study. . . . 35
19.3 Reference PRISM roofline with representative packet workloads. . . . 36
19.4 Total route sensitivity across block width and correction rank. . . . 37
19.5 Representation-storage reduction across block width and correction rank. . . . 38
19.6 PRISM should accelerate strong reusable structure and reject weak or one-off structure. . 39
20.1 Reference power allocation. . . . 39
20.2 Target compute-chiplet area allocation. . . . 40
23.1 Verification ladder from mathematical contracts to measured silicon. . . . 42
24.1 The software and CUDA stages precede FPGA and silicon work. . . . 44
24.2 Recommended FPGA overlay for measuring the architectural features that do not reduce to an ordinary CUDA kernel. . . . 45
24.3 Development risk map. Values are planning judgments rather than measured frequencies. 46
B.1 The main calculus operations and their primary hardware engines. . . . 49

1 Design brief

1.1 The design problem

Many computations begin with structure that disappears before hardware sees the job. A block operator becomes one dense matrix. A Fourier or wavelet atlas becomes an unnamed array. A parent packet and its children become unrelated buffers. A low-rank correction is materialized even when every later stage could have consumed its factors.

PRISM-1 keeps those relations explicit. The architecture is useful only when that choice saves arithmetic, storage, data movement, or scheduling work. The compiler is allowed to decline the packet path.

1.2 Required behavior

Requirement	Meaning
Exact packet synthesis	A declared complete packet family must reconstruct the original object within the stated numerical format
Typed transfer	Every operator edge retains its source packet, destination packet, shape, rank, precision, and sensitivity
Local refinement	Replacing one parent by a child graph must not force unrelated tasks to be regenerated
Representation preservation	Block, sparse, low-rank, Kronecker, factor-chain, transform, and packet-graph forms remain executable without an explicit full array
Provisional execution	Results that require a certificate remain hidden from downstream work until the check passes
Deterministic replay	Selected graph regions can use fixed reductions and stable descriptors for reproducible reruns
Graceful rejection	Work with weak structure is routed to the CUDA backend, a general-purpose path, or a generic leaf

1.3 Design rules

1. The mathematical object and its storage representation are not assumed to be the same thing.
2. A packet descriptor is part of the architectural state, not debugging metadata.
3. An application result, a factor graph, a transform plan, and an explicit array are different output contracts.
4. A numerical result is not committed merely because the kernel finished.
5. A failed certificate should identify the local graph region that needs replay or refinement.
6. The first prototype should preserve the final command and descriptor formats even if it implements fewer engines.

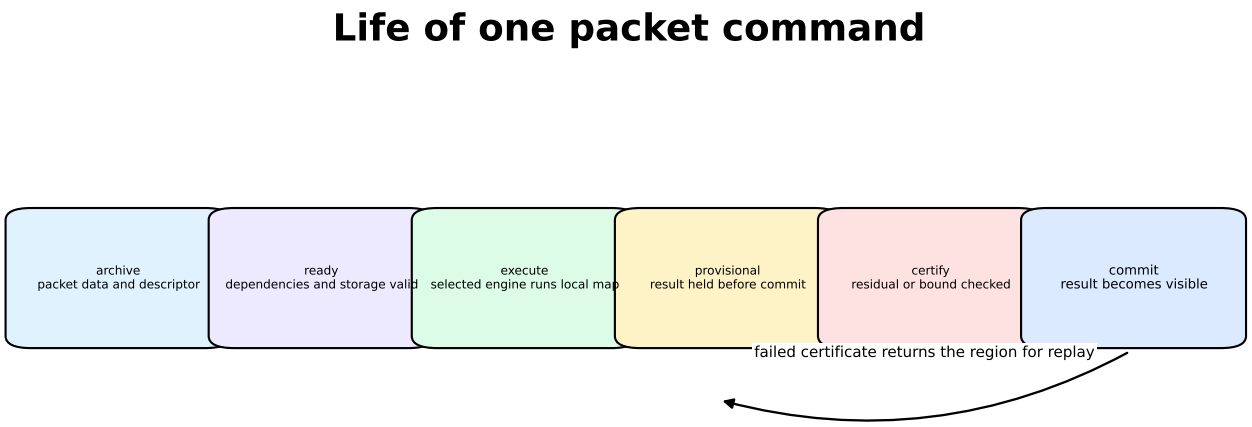


Figure 1.1: One packet command from archive to commit. The provisional state prevents an unchecked result from becoming visible to downstream work.

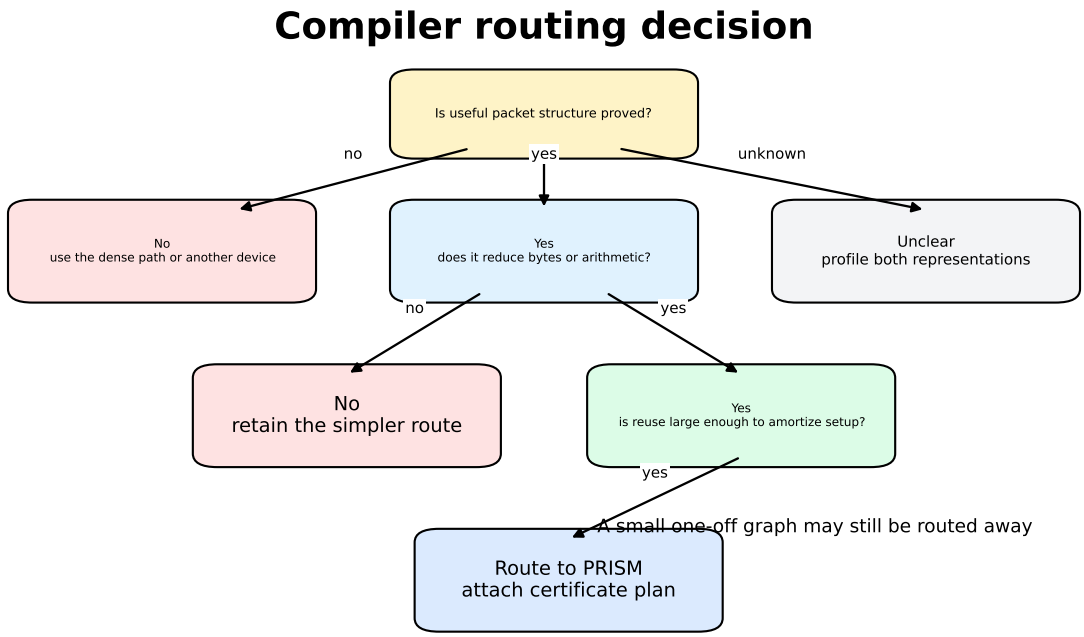


Figure 1.2: The compiler is expected to route weakly structured work away from the packet path.

1.4 What this design does not promise

PRISM-1 is not a universal replacement for a GPU or CPU. It does not claim a better asymptotic exponent for generic dense linear algebra. It does not claim that every operator has a useful structured representation.

It does not treat modeled roofline numbers as measured performance. Its case rests on workloads where structure is real, reusable, and large enough to amortize descriptor, planning, and scheduling cost.

2 Why this chip exists

2.1 The representation is part of the computation

A conventional accelerator usually receives a tensor after the representation has already been chosen. It may execute that tensor well and still miss the simpler object underneath it. The operator may be block diagonal, block sparse, low rank, generated by a transform, or recoverable from a much smaller packet graph.

Projector-resolved computation starts before that flattening step. Let

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

For an operator $T : X \to X$, the native objects are the transfer blocks

$$T_{\beta\alpha} = P_\beta T P_\alpha.$$

The complete operator is reconstructed by

$$\boxed{T = \sum_{\alpha,\beta} T_{\beta\alpha}.}$$

The zero pattern, block shapes, rank structure, parent-child relations, and packet sensitivities are part of the problem. PRISM-1 carries them into the execution model instead of asking software to rediscover them after every kernel boundary.

2.2 The target workload

The design focuses on operations that recur throughout the calculus:

- residue and phase projectors
- polyphase extraction and interleaving
- Fourier, cosine, Hadamard, and wavelet atlases
- packet-tree refinement and coarsening
- block-diagonal and block-sparse transfers
- block diagonal plus low rank operators
- exact compressed products and Woodbury solves
- small Hankel, QR, SVD, root, and Vandermonde problems
- nonlinear packet interactions and memory kernels
- conjugate Bayesian regression and posterior mixtures
- exact modular, CRT, and valuation operations

- residual and error-bound certificates

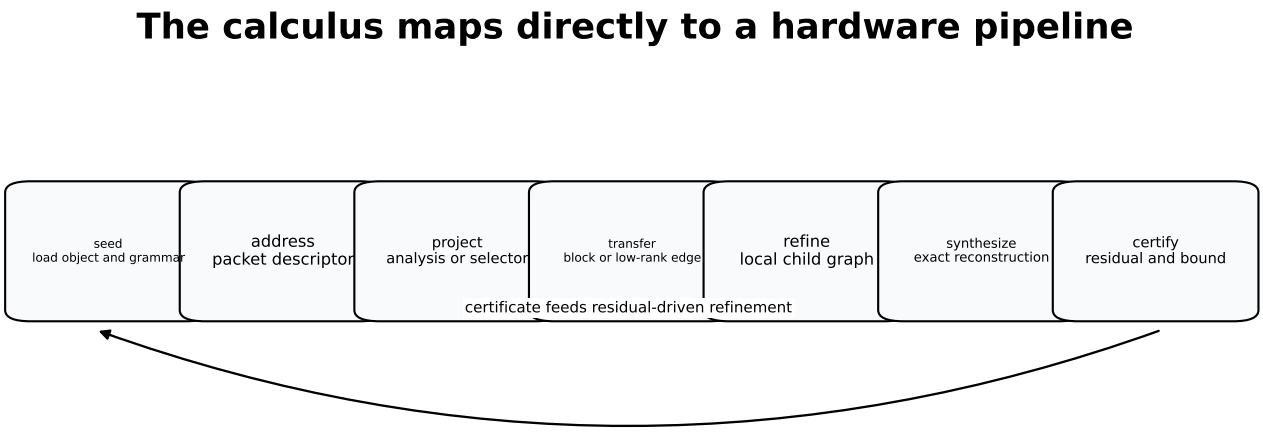


Figure 2.1: The seed-calculus operations become a hardware pipeline without losing their mathematical type.

2.3 What the chip is not

PRISM-1 should not be used simply because a packet description exists. A generic dense matrix with no reusable structure may run better through a conventional library path. A tiny one-off transform may not amortize descriptor setup. A graph that changes completely at every step may not benefit from replay. The compiler should reject or reroute those cases.

2.4 No dense preservation requirement

The architecture does not begin with a dense matrix and try to compress it after the fact. It begins with a typed operator. That operator may be represented as

$$\boxed{\begin{aligned}\mathfrak{R} = \{&\text{explicit array},\ \text{block},\ \text{block sparse},\ \text{low rank},\ \text{BDLR},\\ &\text{Kronecker},\ \text{sparse chain},\ \text{transform plan},\ \text{packet graph}\}.\end{aligned}}$$

A dense representation is useful when no better form exists or when a downstream user explicitly asks for every entry. It is not a state that the chip must preserve.

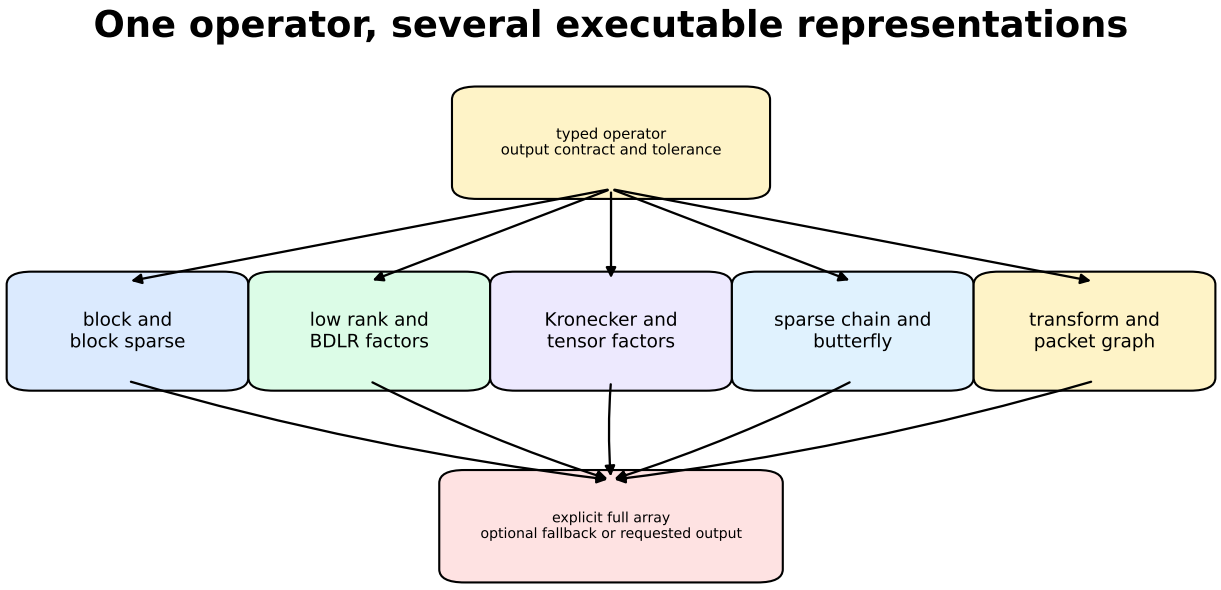


Figure 2.2: One operator may have several executable representations. The explicit array is one optional leaf.

2.5 CUDA-like execution hierarchy

CUDA made heterogeneous arithmetic programmable through a hierarchy of lanes, warps, blocks, grids, and memories. PRISM uses an analogous execution hierarchy while retaining the mathematical structure that CUDA kernels normally receive only through software conventions:

$$\boxed{\text{lane} \longrightarrow \text{packet warp} \longrightarrow \text{execution group} \longrightarrow \text{Packet Streaming Multiprocessor} \longrightarrow \text{packet graph}.}$$

A packet warp may invoke a matrix group, vector lanes, a transform engine, a small solver, exact arithmetic, or a memory operator. The representation router chooses the form declared by Packet IR. Several forms may appear in one graph.

2.6 The CUDA implementation is the software foundation

The packet algorithms already run on CUDA. Equal blocks are stacked and sent to grouped or batched matrix kernels. BDLR application uses a block batch followed by two low-rank contractions. Repeated graphs use CUDA Graph replay. Fourier and sparse stages use established libraries. The runtime keeps factors resident when the output contract allows it.

This matters for the chip design. PRISM is not a replacement for the CUDA algorithms. It is a hardware target for the same Packet IR. The CUDA backend remains the development platform, portability path, and matched comparison.

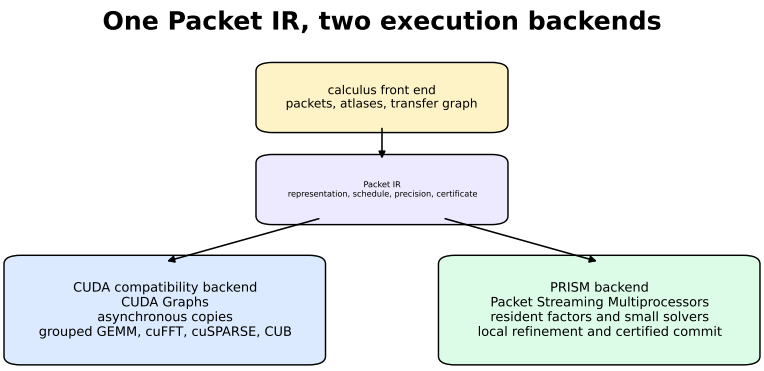


Figure 2.3: The CUDA and PRISM backends execute the same Packet IR and preserve the same output contracts.

3 Mixed-representation execution

3.1 Output contracts

A computation is not defined only by the numerical entries of a hypothetical matrix. The caller also specifies what must be returned. PRISM uses five principal contracts:

1. apply an operator to one or more vectors
2. return a compressed operator
3. return a factor or packet graph
4. return a transform plan
5. return every entry of an explicit array

Only the fifth contract forces materialization.

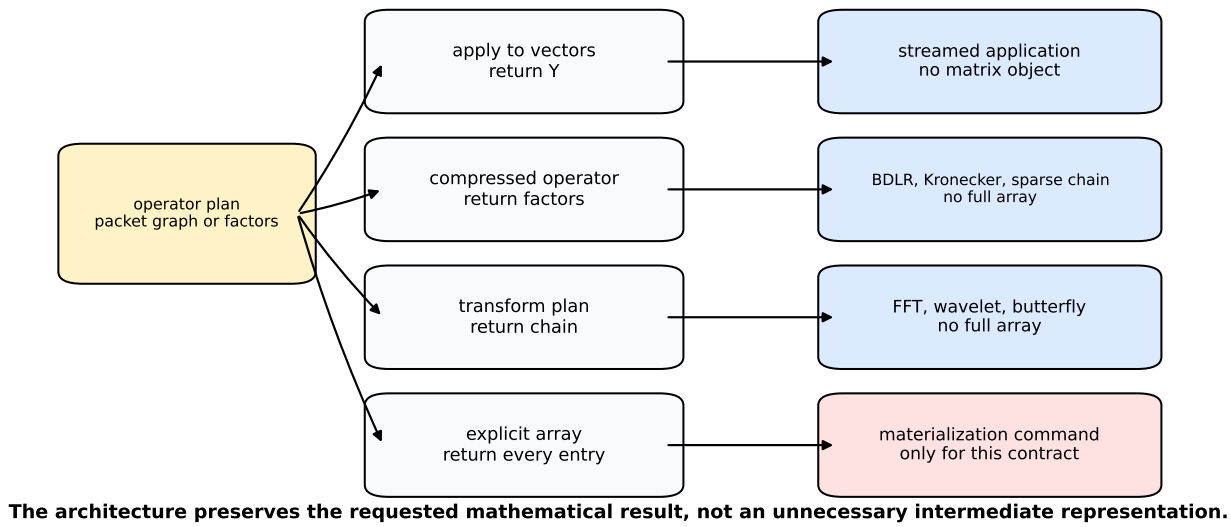


Figure 3.1: Materialization is required only when the requested output is the explicit full array.

3.2 Compatible middle bases

The intermediate basis is part of the optimization. For any invertible S,

$$AB = (AS)(S^{-1}B).$$

Suppose

$$A = LD_AS^{-1}, \qquad B = SD_BR.$$

Then

$$\boxed{AB = L(D_AD_B)R.}$$

When L, S, and R have fast application and D_A, D_B are block diagonal, the product inherits the blockwise cost plus the transform costs. Fourier diagonalization of circulants, simultaneous diagonalization, symmetry sectors, and tensor factorizations are instances of the same rule.

3.3 Sparse chains and transform plans

A matrix may be dense as an array and cheap as an operator. If

$$A = F_LF_{L-1}\cdots F_1$$

and each factor has $O(n)$ nonzero entries, then applying A costs $O(nL)$. If $L = O(\log n)$, the cost is $O(n \log n)$. FFT and butterfly operators fit this pattern. PRISM stores the factor chain and executes it. It does not store the dense array represented by the chain.

3.4 PMR–GEMM

Packet Mixed-Representation GEMM is the compiler planner for operator products. It accepts the operator pair, tolerance, output contract, candidate transforms, and packet tree. At each node it may use a generic

leaf, a low-rank contraction, a Kronecker identity, a sparse factor chain, a transform-domain rule, or a recursive split.

For a fixed candidate tree, each choice o has a local cost $c(o)$, a local error $e(o)$, and zero or more children. The planner evaluates

$$J(v) = \min_{o \in \mathcal{O}_v} \left[c(o) + \lambda e(o)^2 + \sum_{u \in \mathrm{child}(o)} J(u) \right].$$

Bottom-up evaluation is globally optimal on the fixed tree because every feasible plan consists of one local choice followed by feasible child plans.

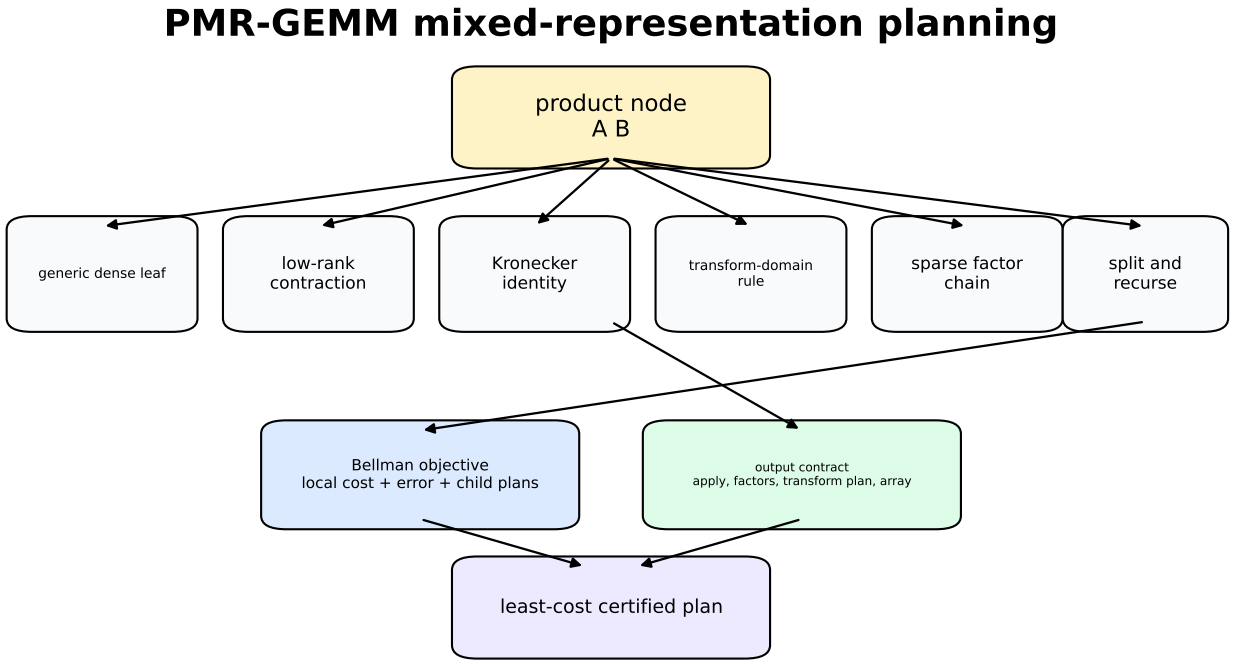


Figure 3.2: PMR–GEMM chooses the least-cost certified plan that satisfies the requested output contract.

3.5 Hybrid fallback

The planner always includes a generic leaf. This protects the worst case. Random dense matrices with no useful local rank, transform, tensor, or packet structure return to the generic path. The architecture is strongest when the mixed plan eliminates arithmetic or movement. It does not claim that naming a packet makes an operator cheaper.

4 Reference package

4.1 Chiplet organization

The reference package contains four compute chiplets, one I/O and control die, and four HBM4 stacks. Each compute chiplet contains 16 Packet Streaming Multiprocessors. Every multiprocessor contains four packet execution groups, giving 64 execution groups per chiplet and 256 across the package. Each chiplet

also contains eight transform engines, four small-solver engines, one exact arithmetic cluster, and 16 MiB of shared SRAM.

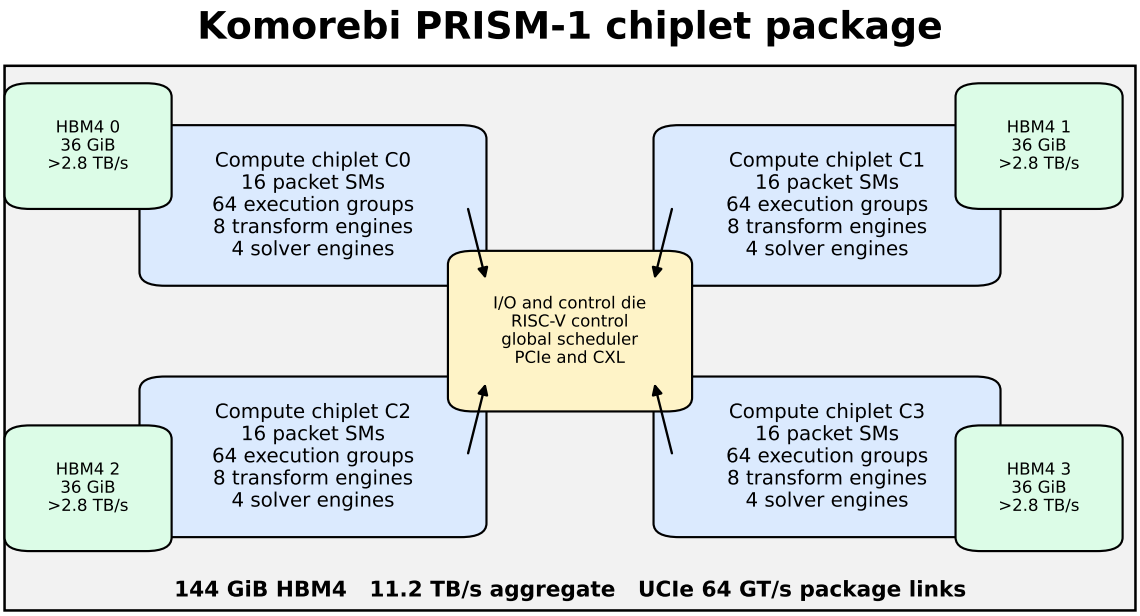


Figure 4.1: The reference PRISM-1 package. The center die carries host I/O, global scheduling, RISC-V control, security, and telemetry.

4.2 Reference targets

Table 4.1: Reference architecture targets. These values belong to the model, not measured silicon.

Property	Reference target
Compute chiplets	4
Packet Streaming Multiprocessors	64
Packet execution groups	256
Execution group array	16×16 matrix array plus vector lanes
Clock	1.5 GHz
PSM scratchpad	2 MiB across four independently banked regions
Shared SRAM	16 MiB per compute chiplet
Total local and shared SRAM	192 MiB
Transform engines	32
Small-solver engines	16
Exact arithmetic clusters	4
HBM4	4 stacks, 144 GiB total
Aggregate HBM bandwidth	greater than 11.2 TB/s
Package fabric	UCIe 3.0 class, up to 64 GT/s
Host interface	PCIe 7.0 x16 class, optional CXL
Reference power envelope	500 W

4.3 Compute-chiplet layout

The reference chiplet places a 4×4 mesh of Packet Streaming Multiprocessors at its center. Every multiprocessor contains four execution groups, giving 64 matrix and vector groups per chiplet. Transform engines sit along one edge, solver engines along another, shared SRAM banks line the data path, and the UCIe router occupies a package-facing edge. This is a planning floorplan rather than a placed design. Its purpose is to make communication and locality assumptions visible early.

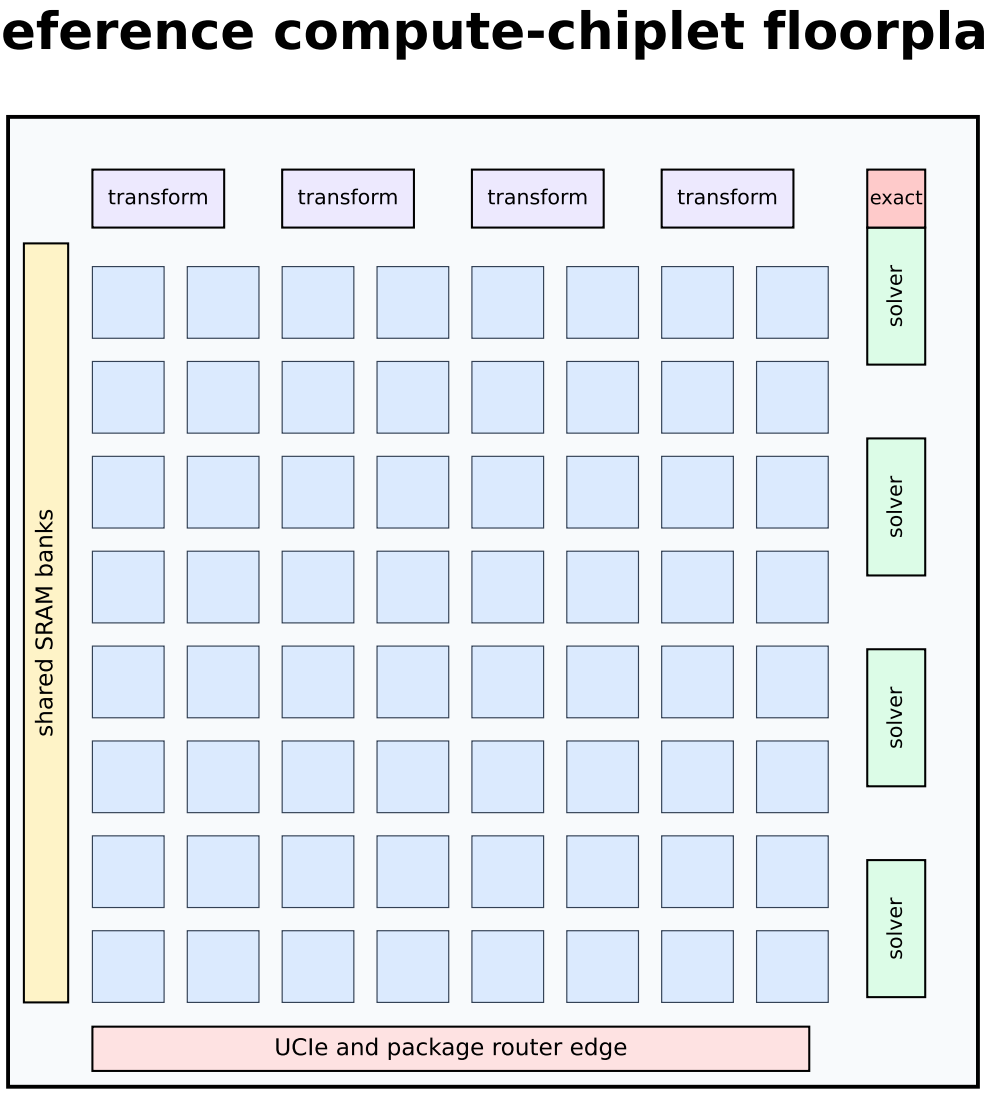


Figure 4.2: Reference compute-chiplet floorplan. The drawing is functional. Bank count, channel width, and physical dimensions remain open.

4.4 Bandwidth balance

The host link is intended for graph submission, coarse data movement, and result return. It is not the steady-state path for every packet. Reusable graphs, factors, and active atlases should remain in HBM or on-package SRAM. The modeled package fabric is wider than aggregate HBM bandwidth so inter-chiplet

traffic does not become an immediate architectural bottleneck in the reference configuration.

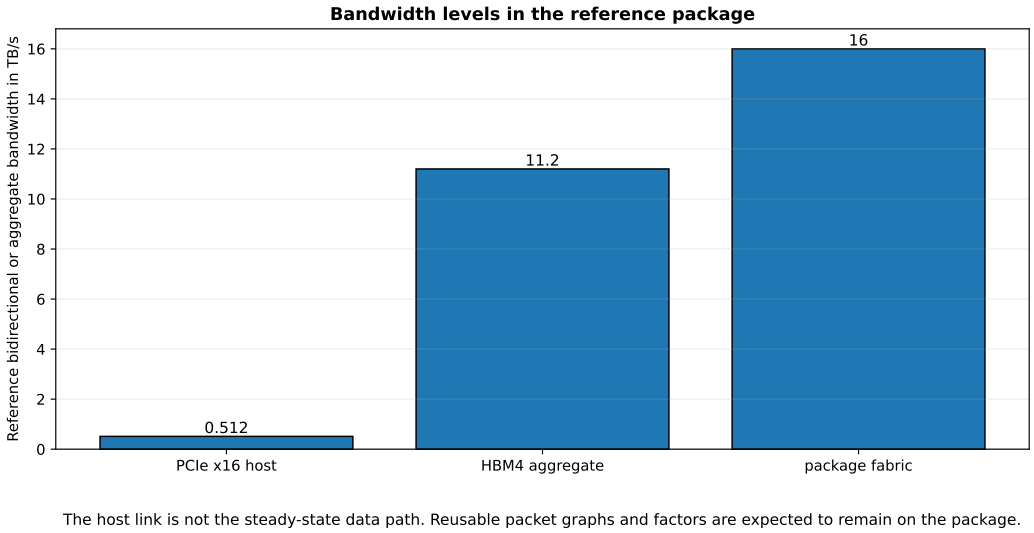


Figure 4.3: Reference bandwidth levels. Local SRAM bandwidth is a circuit-level target and is not assigned a final number in this draft.

4.5 Scale-down configurations

The same Packet IR, commands, and descriptors support smaller devices.

Table 4.2: Suggested development configurations.

Configuration	PSMs	Execution groups	Transform	Solver	Intended use
FPGA overlay	1	4	1	1	Command, DMA, representation routing, replay, and certificate validation
Shuttle chiplet	4 to 8	16 to 32	2 to 4	1 to 2	Measured locality, solver occupancy, and Packet IR conformance
Reference chiplet	16	64	8	4	Full chiplet mesh and resident solver experiments
Full package	64	256	32	16	Data-center scale structured workloads

4.6 Modeled arithmetic peaks

With 256 execution groups and one 16×16 matrix array per group, the model yields

$$\begin{aligned} \text{FP16 or BF16} &\approx 196.6 \text{ TFLOP/s}, \\ \text{FP32} &\approx 98.3 \text{ TFLOP/s}, \\ \text{FP64} &\approx 24.6 \text{ TFLOP/s}. \end{aligned}$$

These are model peaks. The design is not organized around a claim of superior generic dense throughput. Its purpose is to execute fewer operations and move fewer bytes when the selected representation proves that the explicit array is unnecessary.

5 Packet Streaming Multiprocessor

5.1 Multiprocessor anatomy

The Packet Streaming Multiprocessor is the independently scheduled unit of PRISM. It accepts packet warps, reads their representation and output contracts, and routes work across four execution groups, the packet scratchpad, asynchronous DMA, transform and solver interfaces, and the certificate path.

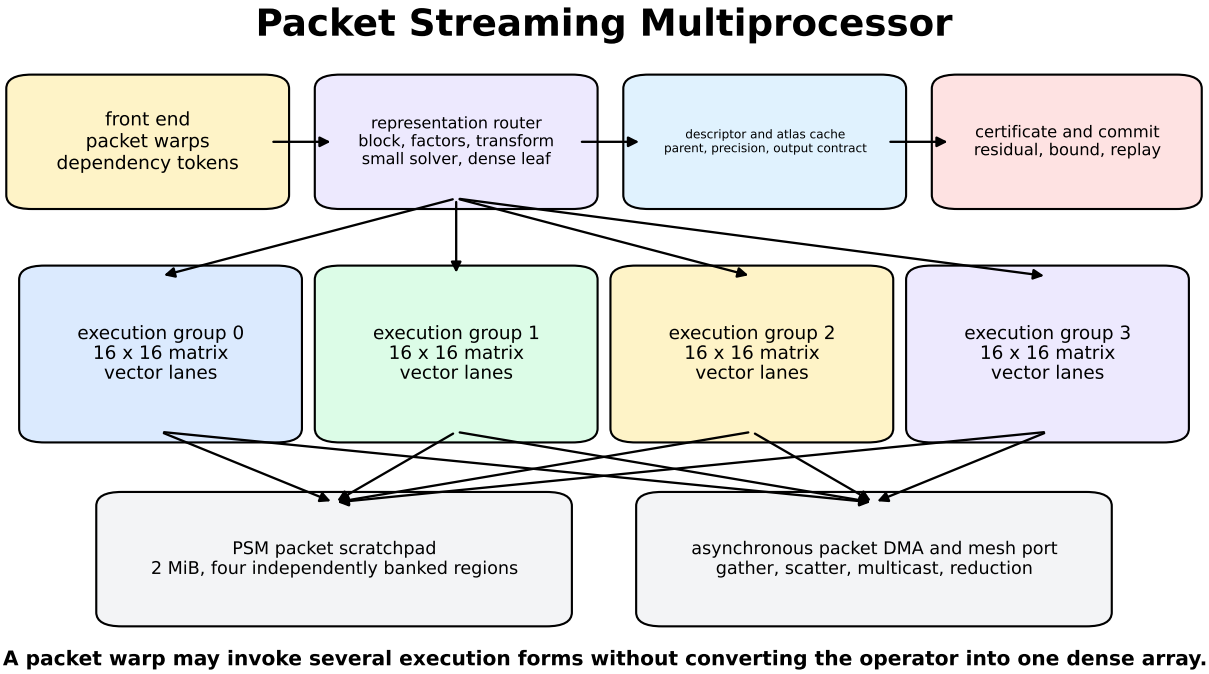


Figure 5.1: One Packet Streaming Multiprocessor. A packet warp may use several execution forms without converting the operator into one explicit array.

5.2 Issue and completion path

A packet warp becomes ready when its dependency token is satisfied and the packet directory confirms that the source representation is resident or in flight. The front end selects the representation route, reserves

destination storage, and records the required certificate. Kernel completion produces a provisional result. The result becomes visible only after the check passes.

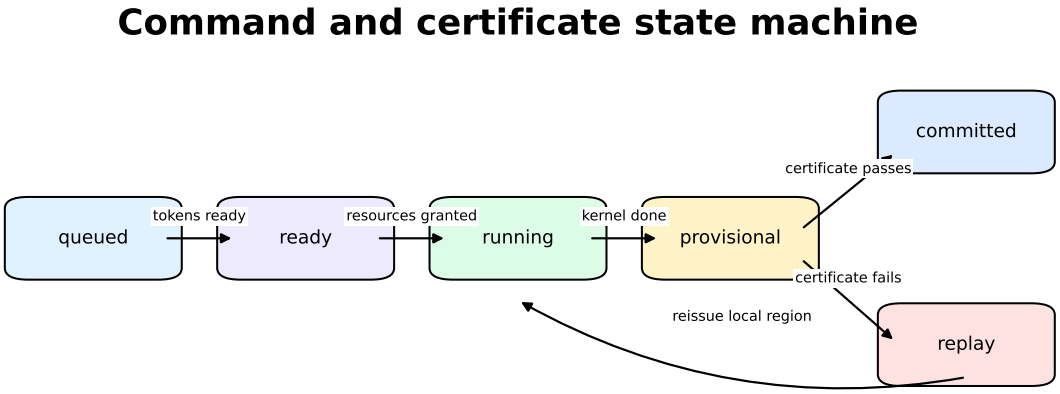


Figure 5.2: Command states from queue entry to commit or local replay.

Completion and commitment remain separate. A kernel may finish while its residual exceeds the declared budget. The distinction matters for mixed precision, iterative solves, approximate transforms, recovered edge packets, and low-rank recompression.

5.3 Packet scratchpad

Each multiprocessor has 2 MiB of scratchpad divided into four independently banked regions. An execution group may use one bank privately or several groups may share a packet region. Separate banks hold active data, factors, descriptors, tokens, solver panels, and certificate state. Double buffering supports transforms and repeated grouped matrix batches.

5.4 Execution groups

Each of the four execution groups contains a 16×16 matrix array and programmable vector lanes. Equal-shape blocks can be grouped across execution groups. The same lanes handle low-rank contractions, Kronecker mode products, sparse-chain factors, selectors, reductions, index generation, and generic leaves.

Complex arithmetic uses paired real lanes. FP64 is available for certificates, small solvers, rank decisions, phase recovery, and sensitive reductions. Integer and fixed-point modes support selectors, image packets, modular arithmetic, and exact accumulators.

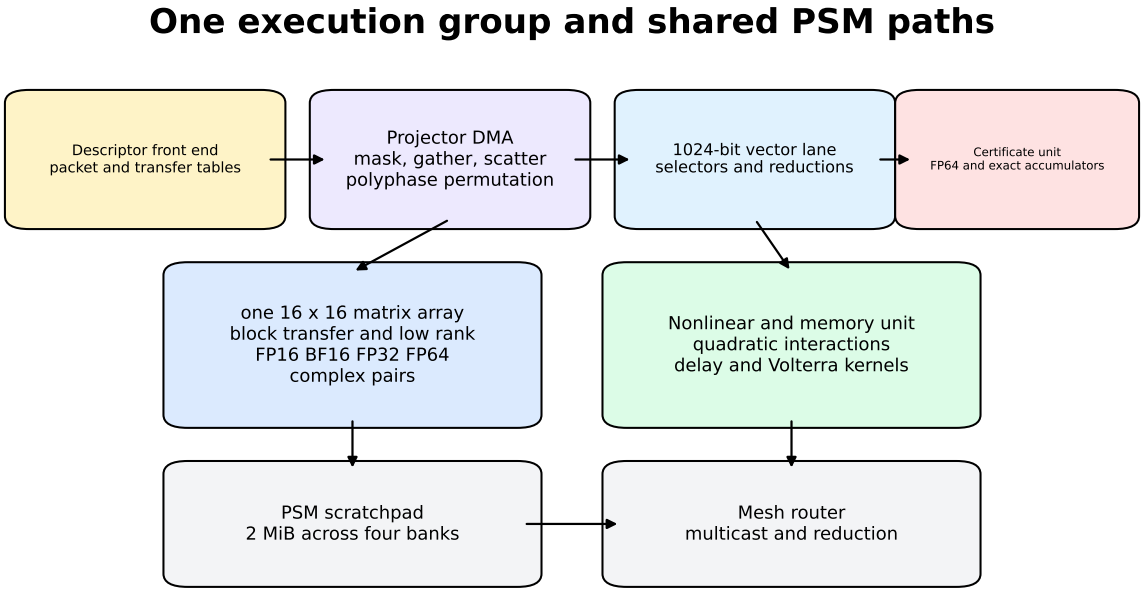


Figure 5.3: Execution-group resources inside the multiprocessor. The figure shows matrix, vector, memory, nonlinear, and certificate paths.

5.5 Representation router

The router reads the operator form from Packet IR. It dispatches block work to grouped matrix lanes, factor work to contraction lanes, transform plans to the transform fabric, small algebraic systems to resident solvers, and weakly structured leaves to the generic path. One packet warp may contain several representations.

5.6 Projector DMA

The asynchronous packet DMA accepts selectors represented as intervals, strided slices, residue classes, polyphase channels, masks, index lists, transform bands, packet-tree nodes, and compact basis factors. A selector is applied through addressing and compact arithmetic. No dense projector matrix is required.

5.7 Certificate and commit path

The certificate unit accumulates residuals, packet norms, trace quantities, energy partitions, branch margins, and mixed-precision bounds. Deterministic mode uses a fixed reduction tree and wide accumulation. The commit controller releases a provisional result only after the declared acceptance rule passes.

6 Descriptors as the hardware language

6.1 Packet descriptor entry

A packet descriptor identifies the packet, its parent, its atlas, its selector, its shape, its location, its precision, and its error budget.

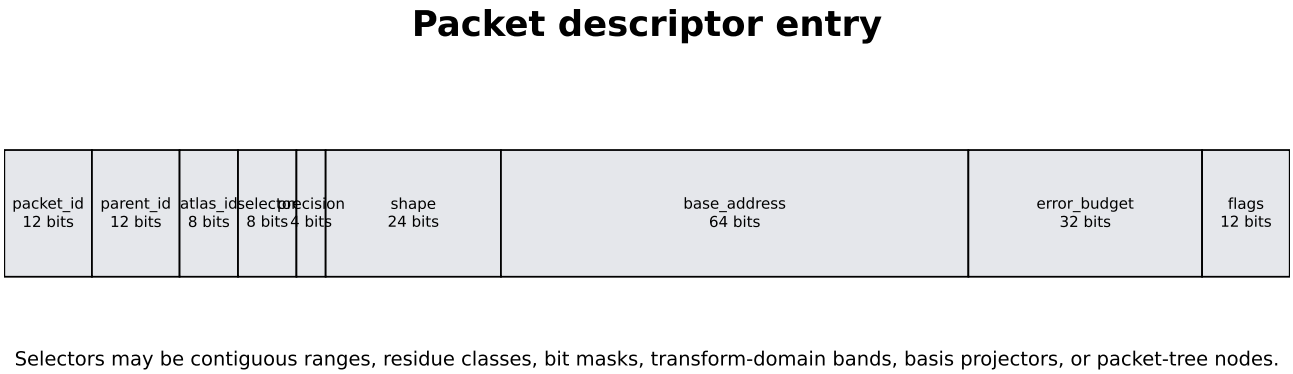


Figure 6.1: One packet descriptor entry. The exact bit allocation may change during RTL design, but the semantic fields are fixed.

Definition 6.1 (Packet descriptor)**.** *A packet descriptor is a typed record*

$$\mathsf{P} = (i, p, a, \sigma, \mathbf{n}, \ell, \pi, \tau),$$

where i is the packet ID, p is the parent, a is the atlas, σ is the selector type, $\mathbf{n}$ is the shape, ℓ is the storage location, π is the precision tag, and τ is the allowed local error contribution.

6.2 Transfer edge descriptor

The transfer descriptor is the hardware form of $P_\beta T P_\alpha$. It identifies the source and destination packets, operator class, shape, rank, precision, sensitivity, payload, and dependency token.

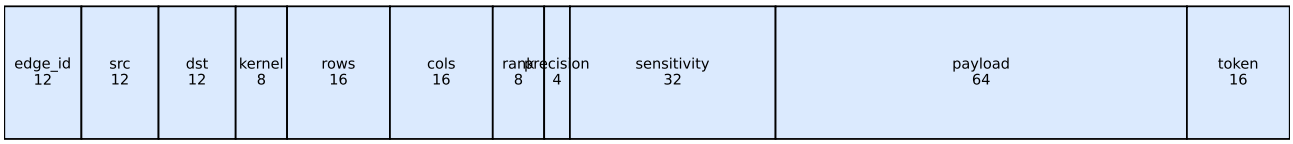


Figure 6.2: A transfer edge descriptor keeps the operator class and certificate weight attached to the edge.

6.3 Atlas descriptor

An atlas descriptor records an analysis graph, a synthesis graph, normalization data, and packet selectors. A unitary transform uses its adjoint for synthesis. A redundant exact frame records the canonical dual or another declared synthesis map satisfying

$$\mathcal{S}\mathcal{A} = I.$$

7 Memory hierarchy

7.1 Exact archive and active representations

The exact archive, cold factors, large graphs, and model history remain in HBM. Completeness does not imply a dense array. An exact archive may be a transform tuple, factor graph, sparse chain, packet tree, geometric edge list, or explicit sample field. Shared SRAM stores active packet graphs and reusable representations. PSM scratchpads hold local blocks, factor panels, transform stages, solver panels, and residuals.

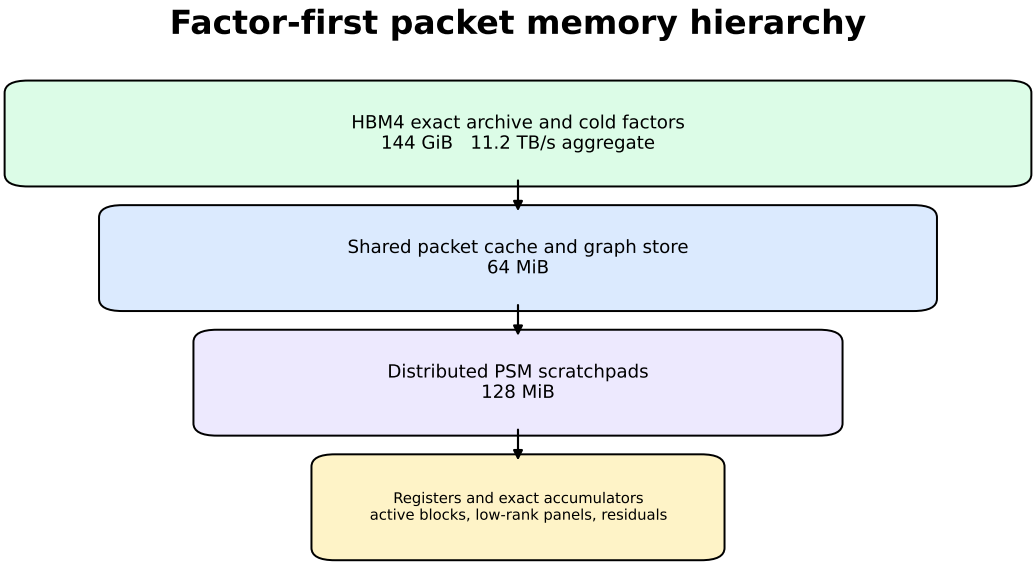


Figure 7.1: Representation-first memory hierarchy. The explicit full array is created only under a matching output contract.

7.2 Representation-first storage

For an equal-block BDLR operator

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

with dimension n, block width b, and rank r, the stored scalar count is

$$\boxed{n(b + 2r)}$$

in place of n^2.

Proposition 7.1 (BDLR representation reduction)**.** *If $b + 2r < n$, BDLR storage reduces the scalar count by*

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

The same factor appears in the ideal apply count for a fixed number of right-hand sides.

Proof. The block-diagonal part stores $(n/b)b^2 = nb$ scalars. The two rank factors store $2nr$. The total is $n(b + 2r)$. A full array stores n^2 scalars. □

The same rule extends beyond BDLR. A Kronecker operator stores its mode factors. A butterfly stores its sparse stages. A transform-domain operator stores the plan and diagonal or block payload. A packet graph stores packet descriptors and active edge payloads. The active representation is the operator until a different output contract is requested.

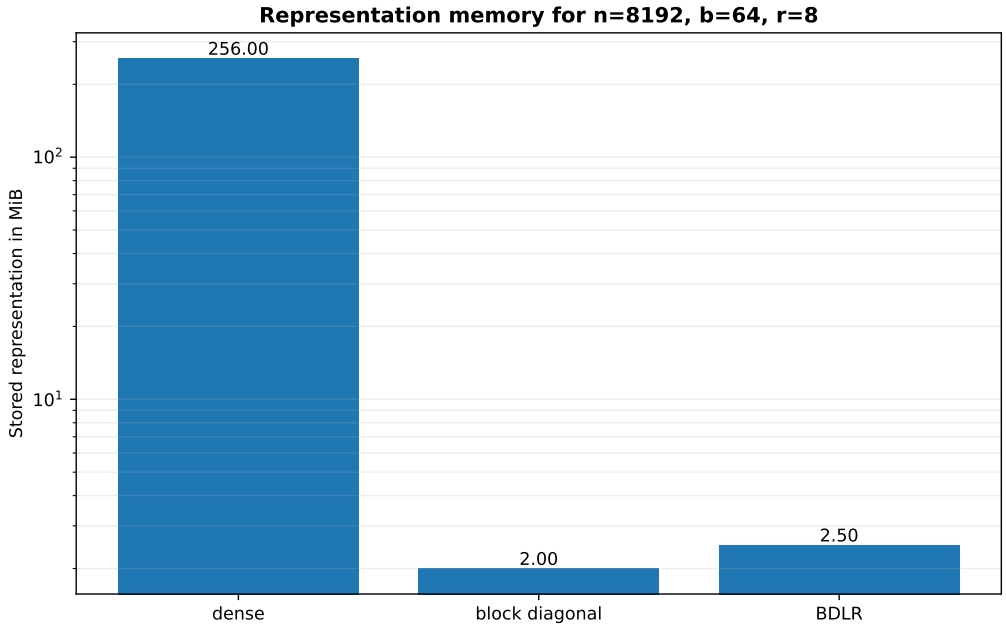


Figure 7.2: Stored BDLR representation for the model case $n = 8192$, $b = 64$, and $r = 8$.

8 Packet network and scheduling

8.1 PSM mesh

Each compute chiplet uses a 4×4 mesh of Packet Streaming Multiprocessors. Routers support unicast, multicast, and tree reduction. Multicast distributes low-rank factors, Kronecker factors, and transform data. Reduction combines destination contributions, small contractions, posterior scores, and residuals.

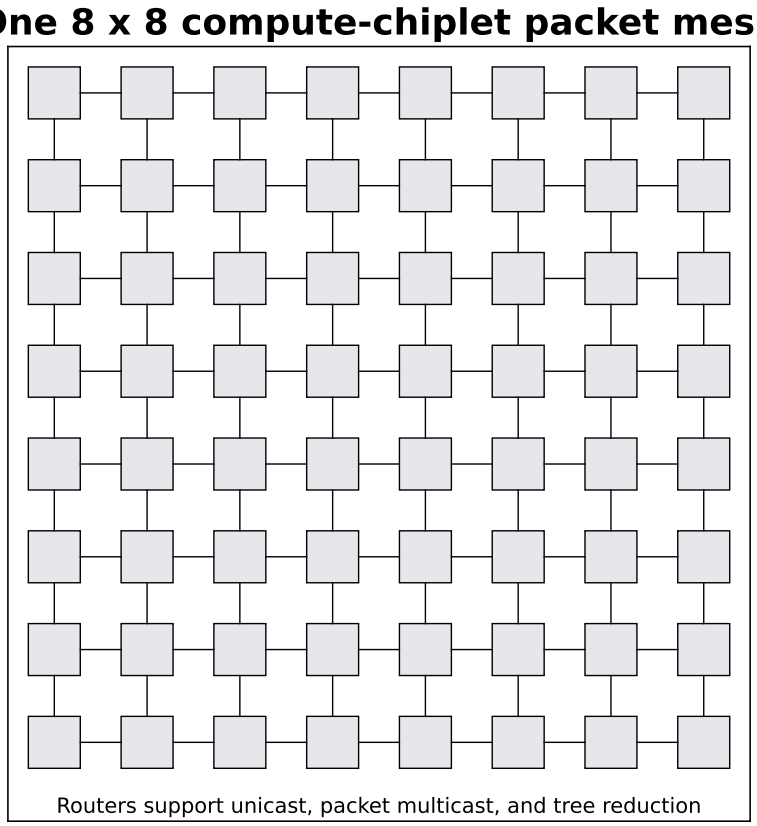


Figure 8.1: One chiplet mesh. Each displayed router node represents one Packet Streaming Multiprocessor rather than one execution group.

8.2 Packet warps and dependency tokens

Packet warps group operations that share a representation, shape class, packet relation, or certificate boundary. A warp becomes ready when its dependency tokens are satisfied. Completion tokens wake downstream work. Replayable regions keep descriptors and placement until a certificate mismatch or local refinement invalidates the region.

8.3 Local refinement

Suppose a parent packet task v is replaced by an acyclic child module $H = (W, F)$. Predecessors of v connect to child entries. Child exits connect to successors of v.

Theorem 8.1 (Topological splice)**.** *Let $\prec$ be a topological order of the original graph and let $\prec_H$ be a topological order of the child module. Replacing v in $\prec$ by $\prec_H$ gives a topological order of the refined graph. Every unaffected pair of context tasks keeps its relative order.*

Proof. Every context edge is unchanged. Every child edge is respected by $\prec_H$. Every predecessor of v appeared before v and therefore before the inserted child order. Every successor appeared after v and therefore after the inserted child order. Removing the inserted children restores the original context order. □

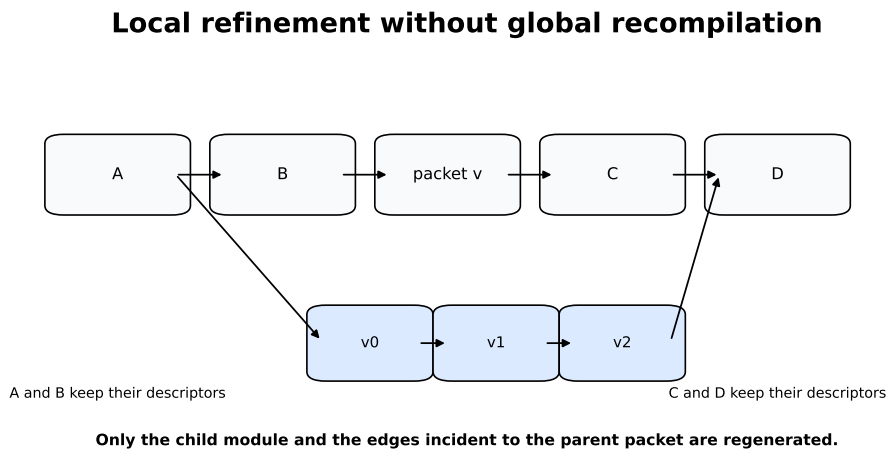


Figure 8.2: A parent packet is replaced by a local child graph while remote descriptors remain unchanged.

9 Transform fabric

9.1 One configurable engine family

The transform engines share butterfly, permutation, lifting, and selector hardware. They support FFT, DCT, FWHT, Haar, biorthogonal lifting, and polyphase transforms.

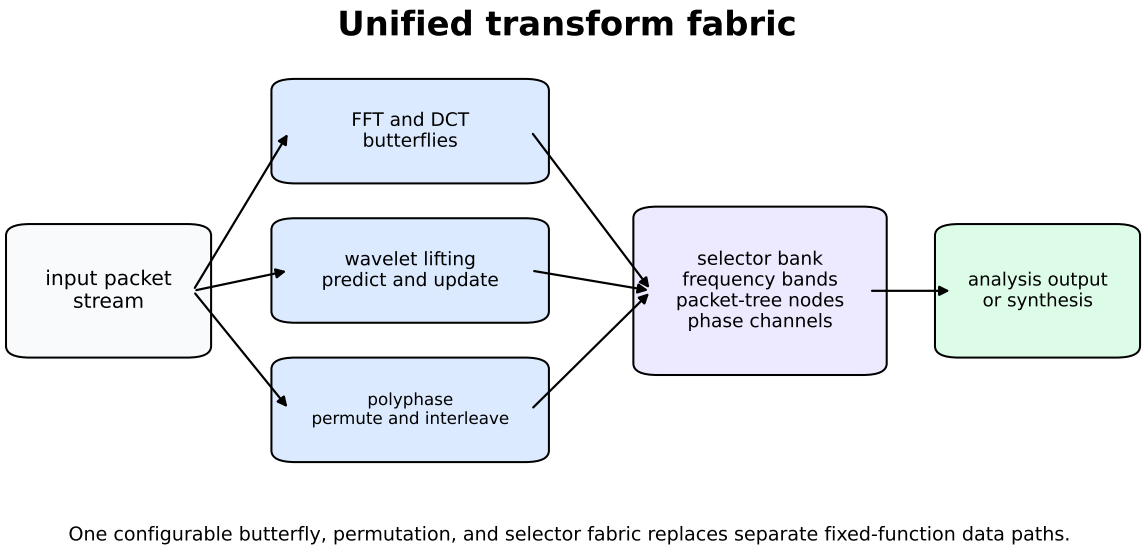


Figure 9.1: The transform fabric supports several exact atlases without storing separate fixed data paths for each one.

9.2 Exact analysis and synthesis

Let U be unitary and let E_α be coordinate selectors. The pulled-back projectors are

$$P_\alpha = U^* E_\alpha U.$$

The hardware analysis applies U and selects the coordinates. Synthesis scatters the selected coordinates and applies U^*.

Theorem 9.1 (Transform-atlas hardware contract)**.** *If the selector family is complete, the analysis and synthesis command graphs satisfy*

$$\boxed{\mathcal{S}\mathcal{A} = I.}$$

If the selector family is incomplete, the certificate must report the omitted-coordinate residual.

Proof. For a complete selector family, $\sum_\alpha E_\alpha = I$. Hence

$$\mathcal{S}\mathcal{A}x = U^* \left(\sum_\alpha E_\alpha \right) Ux = U^* U x = x.$$

The incomplete case replaces the identity by the declared selector sum. □

10 Transfer engines

10.1 Block diagonal and block sparse

Block-diagonal operators run as independent shape batches. Block-sparse operators are grouped by edge shape and destination. Destination packet accumulation can be local, multicast, or reduced through the mesh.

10.2 BDLR dataflow

For

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

the apply is

$$AX = DX + U(V^* X).$$

The diagonal blocks remain distributed. The $r \times q$ contraction is reduced and broadcast.

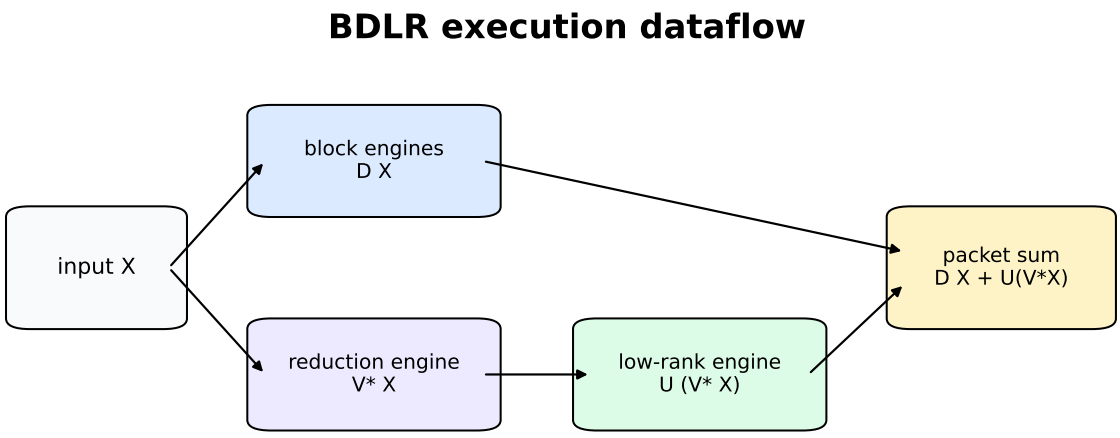


Figure 10.1: The BDLR path keeps the low-rank representation compressed.

Theorem 10.1 (BDLR apply count)**.** *For q right-hand sides and equal packet width b, the modeled arithmetic count is*

$$\boxed{2qn(b+2r).}$$

A dense apply requires $2qn^2$ operations.

Proof. The block-diagonal product costs $2qnb$. The contraction V^*X costs $2nrq$. Multiplication by U costs another $2nrq$. Adding the terms does not change the leading expression. □

10.3 Where BDLR remains useful

The low-rank path is not uniformly favorable. Its benefit depends on block width, correction rank, right-hand-side count, and factor reuse. The reference model sweeps a range of block sizes and ranks while holding $n = 8192$ and $q = 32$ fixed.

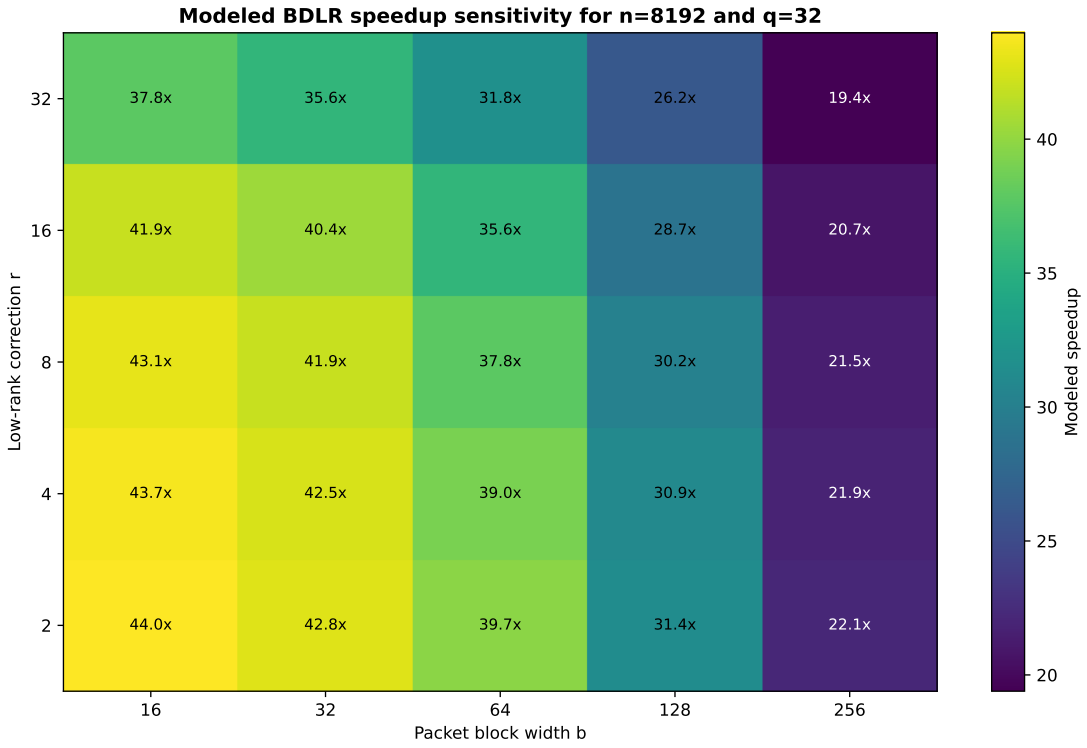


Figure 10.2: Modeled BDLR speedup across block width and correction rank. The comparison includes arithmetic, data movement, and graph overhead from the reference model.

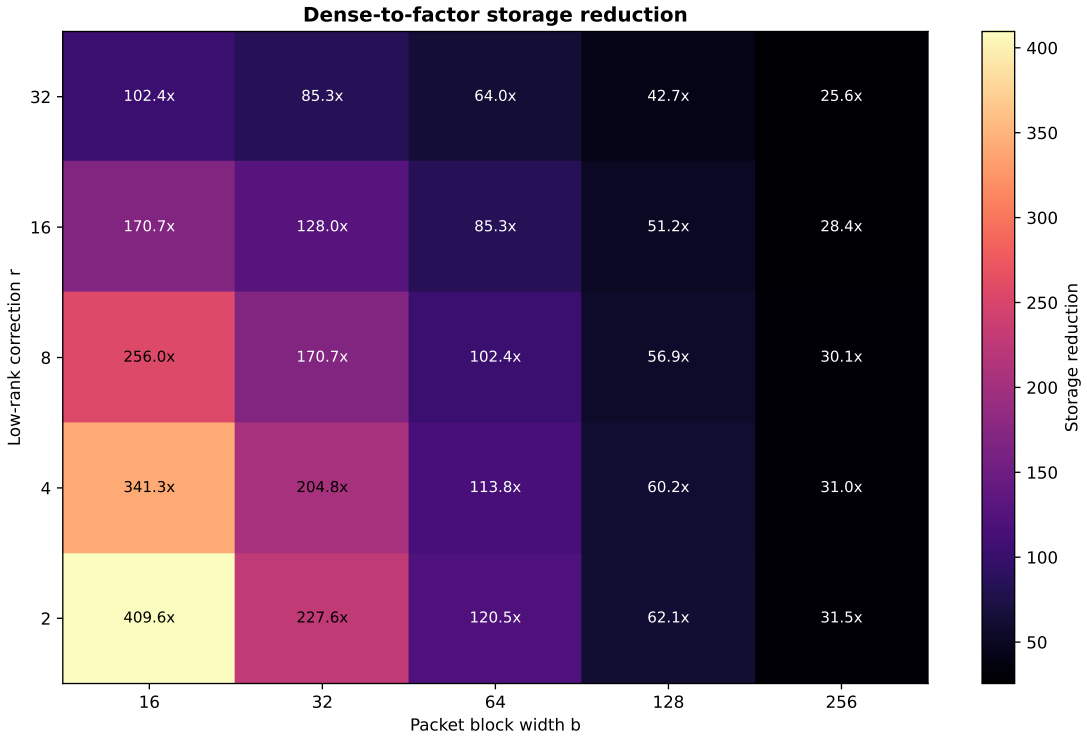


Figure 10.3: Dense-to-factor storage reduction for the same parameter sweep.

The compiler should not infer a gain from the name of the representation. It should evaluate the actual

block width, rank, reuse, and output contract. A high-rank correction or a request for immediate dense materialization may remove most of the benefit.

10.4 Compressed composition

When downstream stages accept BDLR factors, composition returns new block factors and low-rank factors. Materialization is a separate output contract. The compiler cannot compare a compressed product with a dense explicit matrix without accounting for that difference.

10.5 Woodbury solve

The chip uses packet solves with D_α and one small dense correction:

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

The solver engines handle the rank-sized middle system.

11 Mixed precision and certificates

11.1 Per-edge precision

A global dtype ignores the fact that packet transfers can have very different sensitivities. Each edge carries its precision and sensitivity.

Theorem 11.1 (Packet error bound)**.** *Let*

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

If edge e starts in packet $\alpha(e)$, then

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

Proof. Exact packet reconstruction gives

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

The triangle inequality gives the bound. □

11.2 Precision allocation

Under a logarithmic cost model and total error budget τ, the continuous allocation is

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

where s_e is sensitivity and c_e is cost. The compiler maps the result to supported formats.

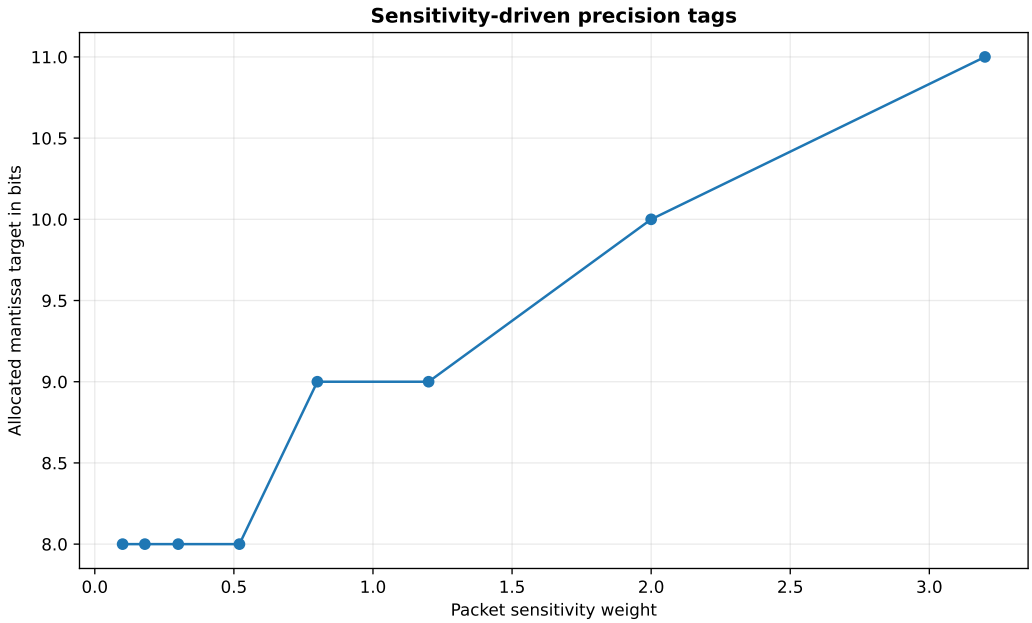


Figure 11.1: More sensitive packet edges receive tighter precision targets.

12 Unknown-edge and Gibbs reconstruction

12.1 Why the chip includes a small-solver path

The self-calibrating reconstruction uses large transforms and very small algebraic systems in one graph. A general GPU often pays high launch and movement cost around the small systems. PRISM keeps the full chain resident.

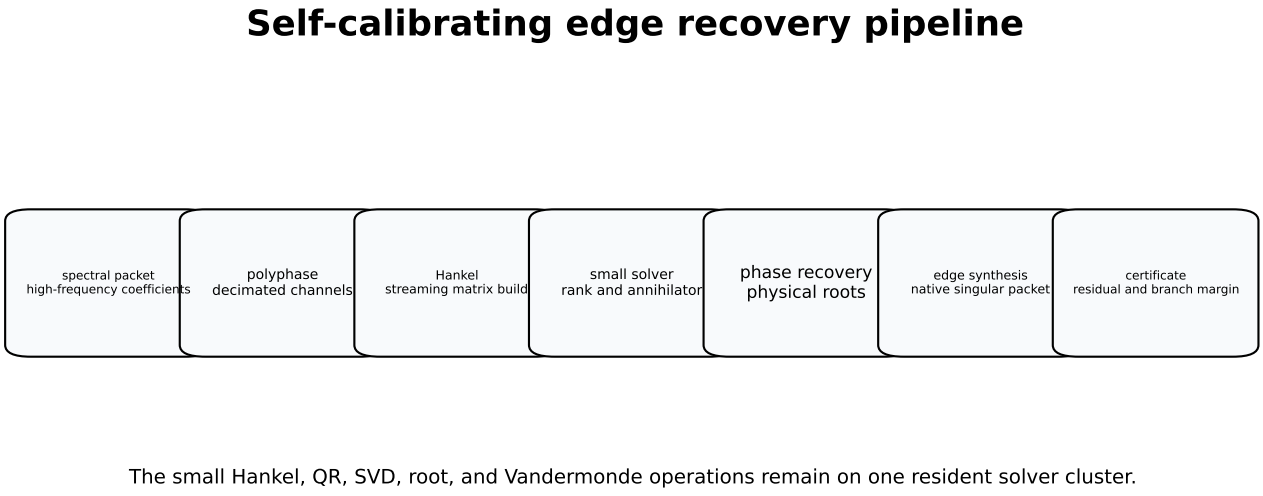


Figure 12.1: The self-calibrating edge-recovery graph from high-frequency packet to native singular synthesis.

12.2 Resident Hankel and Prony operations

The edge-recovery engines perform:

1. Fourier packet extraction
2. decimated polyphase grouping
3. streaming Hankel construction
4. rank estimation
5. annihilator solve
6. polynomial root recovery
7. physical phase selection
8. Vandermonde amplitude solve
9. edge packet synthesis
10. residual and branch-margin checks

12.3 Small-solver organization

The solver cluster is built for matrices that are too small to fill the PSM mesh but too frequent to send back to the host. A local panel store feeds QR, Cholesky, Jacobi-style SVD, companion-matrix roots, and Vandermonde solves. FP64 is the default format for rank decisions, phase recovery, and branch-margin checks.

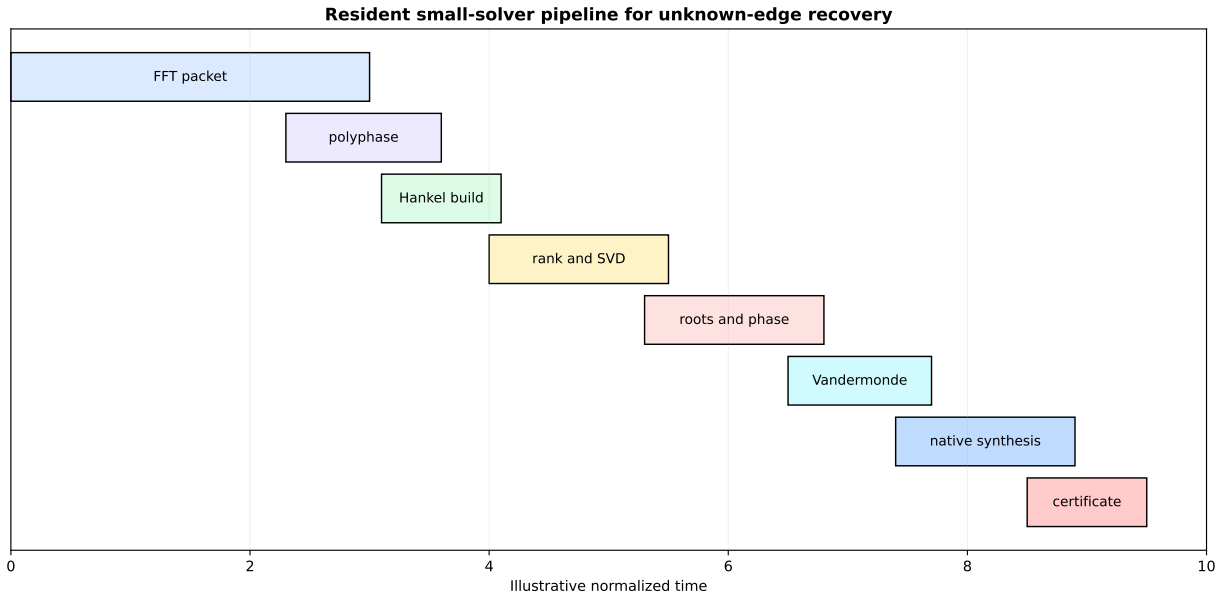


Figure 12.2: Illustrative resident pipeline for unknown-edge recovery. Transform and solver stages overlap where data dependencies permit.

The solver cluster is also used by Bayesian regression batches, Woodbury corrections, packet Krylov restarts, and small exactness checks. It is not intended to compete with a large dense eigensolver.

12.4 Why native edge synthesis matters

After edge recovery, the discontinuity is synthesized as an edge packet. The smooth residual is reconstructed spectrally. The chip does not force the recovered singular packet back through a truncated global Fourier sum.

13 PR-LASM execution

13.1 Exact archive and compact state

The exact observation archive is kept in complete spatial, Fourier, Haar, polyphase, and categorical atlases. Macro, meso, and micro states are separate statistical packet types with dimensions 48, 96, and 192 in the reference model.

13.2 Resident state layout

The exact archive and compact predictive states remain separate in memory and in the descriptor type system. This prevents a compact state vector from being mistaken for an invertible representation of the signal.

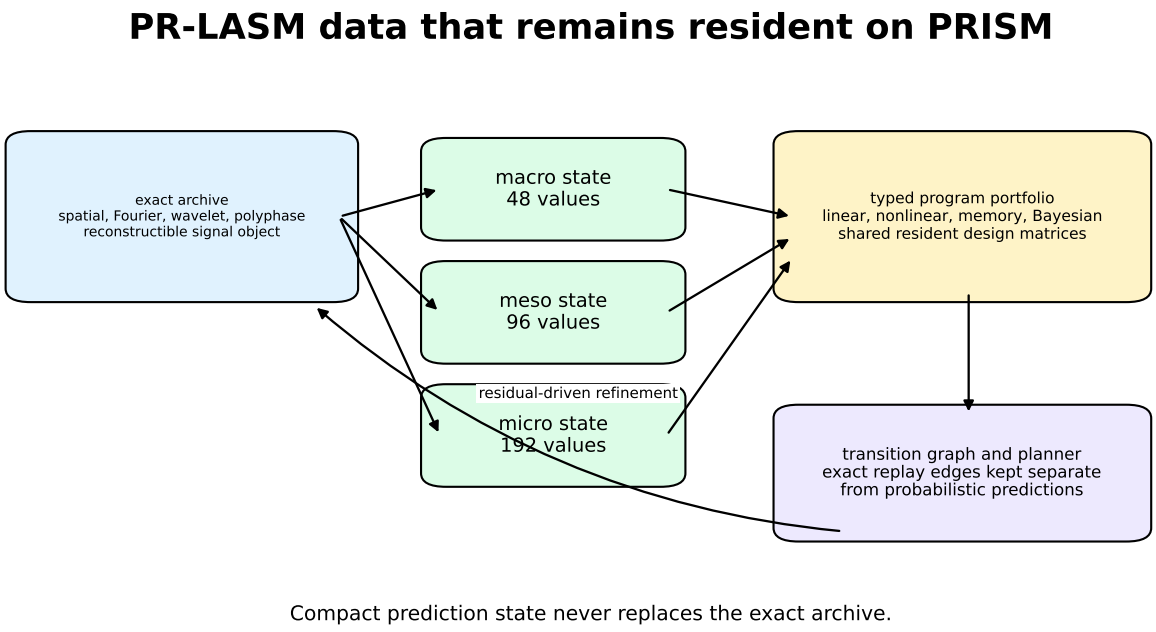


Figure 13.1: Exact archive, compact states, mechanism portfolio, and transition graph in one resident PR-LASM step.

13.3 Typed Bayesian programs

The mechanism portfolio shares state matrices while allowing different operators, atlases, resolutions, and memory laws. The solver engines batch models by feature shape.

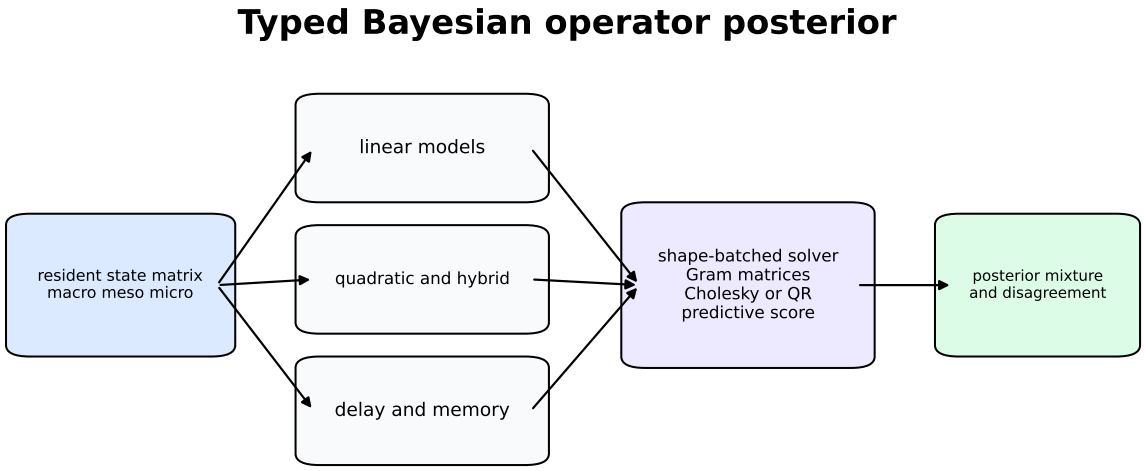


Figure 13.2: Mechanism families share resident state matrices and small-solver batches.

13.4 Memory and planning

Delay, power-law, exponential, and Volterra memory packets run in the nonlinear and memory units. The graph scheduler stores exact deterministic transitions separately from probabilistic predictions. Replay stops when a state or certificate mismatch is detected.

14 Exact arithmetic mode

14.1 Residue and modular packets

One exact arithmetic cluster per compute chiplet supports 64-bit modular addition, multiplication, valuation, residue selection, and CRT merge. Longer values use limb vectors in PSM scratchpad.

14.2 Why this belongs on the chip

The arithmetic calculus and the signal calculus share the same structural operations:

$$\text{residue packet} \to \text{transfer} \to \text{refinement} \to \text{weighted reconstruction}.$$

The exact arithmetic path is smaller than the floating-point path, but it allows the same descriptor and graph system to execute p-adic, recurrence, and finite-field packet workloads.

15 Packet Seed Command interface

15.1 Command groups

The first implementation uses memory-resident 256-bit commands. RISC-V control cores fill a queue and ring a doorbell. The command groups are shown below.

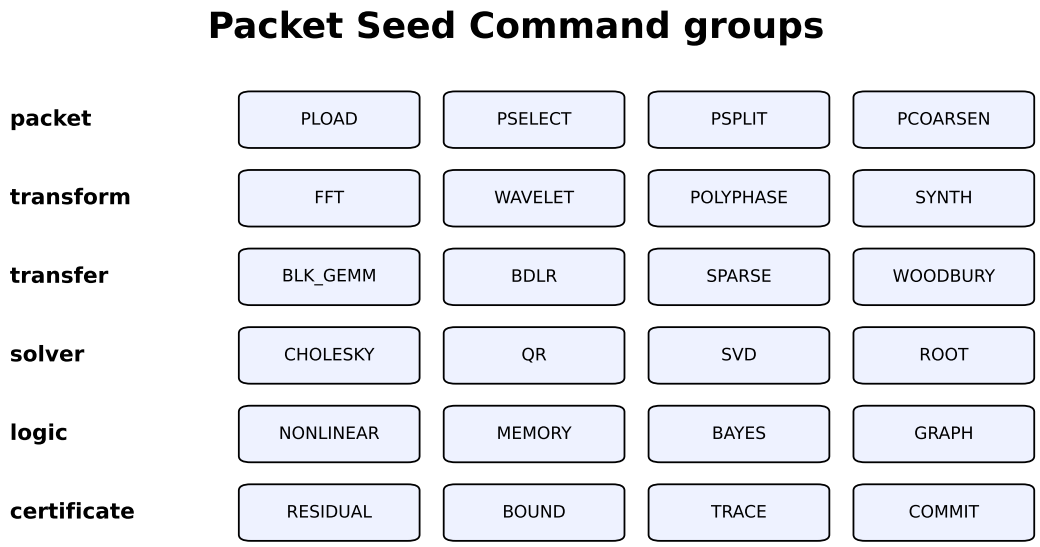


Figure 15.1: Packet Seed Command groups.

15.2 Representative commands

Command	Meaning
`PSELECT`	Apply a declared packet selector without materializing a projector matrix
`PSPLIT`	Replace a parent packet by declared children
`PCOARSEN`	Reconstruct a parent from a complete child family
`POLYPHASE`	Extract, permute, or interleave phase channels
`WAVELET_LIFT`	Execute predict, update, and scale steps
`BLK_GEMM`	Execute a batch of equal-shape transfer blocks
`BDLR_APPLY`	Apply block diagonal plus low rank without materialization
`HANKEL_BUILD`	Stream moments into a small Hankel matrix
`ROOT_PHASE`	Recover physical roots from decimated channels
`BAYES_FIT`	Fit one shape-class batch of conjugate regressions
`GRAPH_REFINE`	Replace one parent task by a local child graph
`RESIDUAL`	Form and store a reconstruction residual
`COMMIT`	Mark a result valid after required certificates pass

15.3 Completion record

Every completed command produces a compact record containing the completion token, status, cycle count, bytes moved, exception code, residual summary, and certificate pointer. Downstream dependencies consume the token only after the commit state is reached.

15.4 Protection and replay

Descriptor tables, command queues, and certificate records require ECC or parity protection. A command that encounters an uncorrectable descriptor error is aborted before execution. A certificate failure invalidates only the declared replay region unless the error is marked package-wide.

16 Compiler

16.1 Routing policy

The compiler first asks whether the structure changes the cost of the computation. A named packet decomposition is not sufficient. The route decision uses representation size, expected reuse, graph size, materialization requirements, and measured or modeled engine cost.

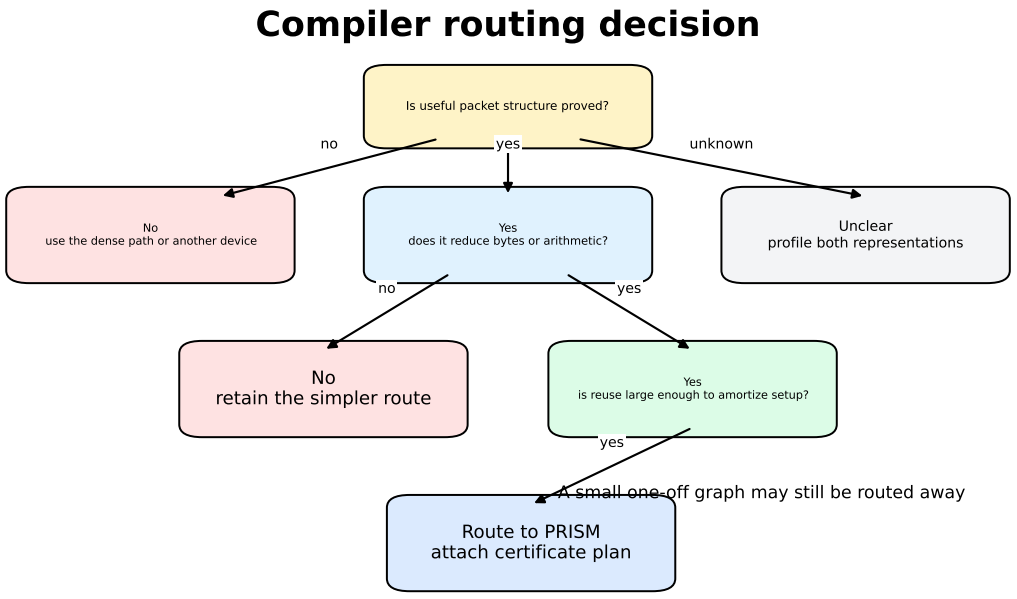


Figure 16.1: Reference routing logic. The final compiler will use calibrated costs, but the reject path remains part of the specification.

16.2 Shared Packet IR

The CUDA and PRISM backends consume the same typed graph. Packet IR records representation form, packet and atlas identity, output contract, precision, reuse, dependencies, and certificate requirements.

Backend selection does not change the mathematical result.

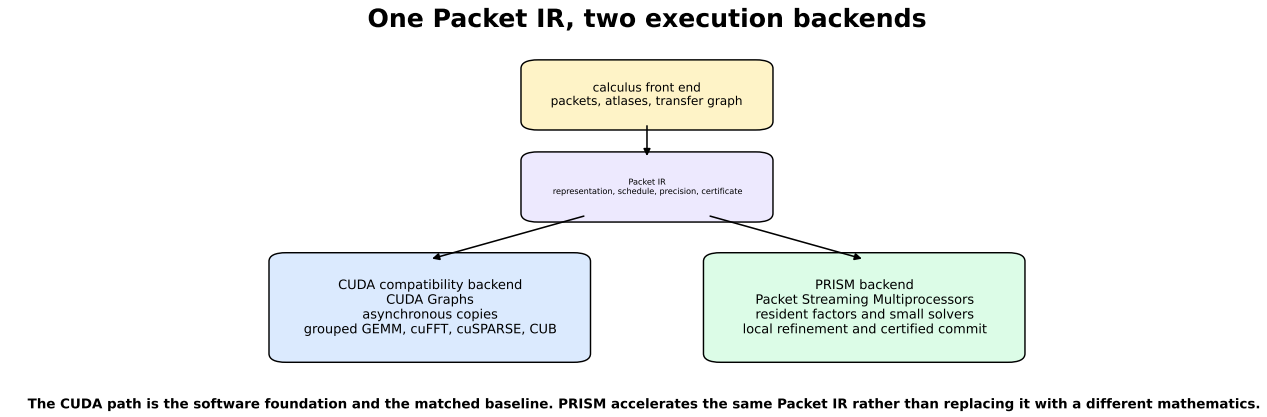


Figure 16.2: Shared Packet IR makes CUDA the software foundation and matched baseline for PRISM.

16.3 Compiler flow

The compiler begins with a typed graph rather than an unstructured kernel list.

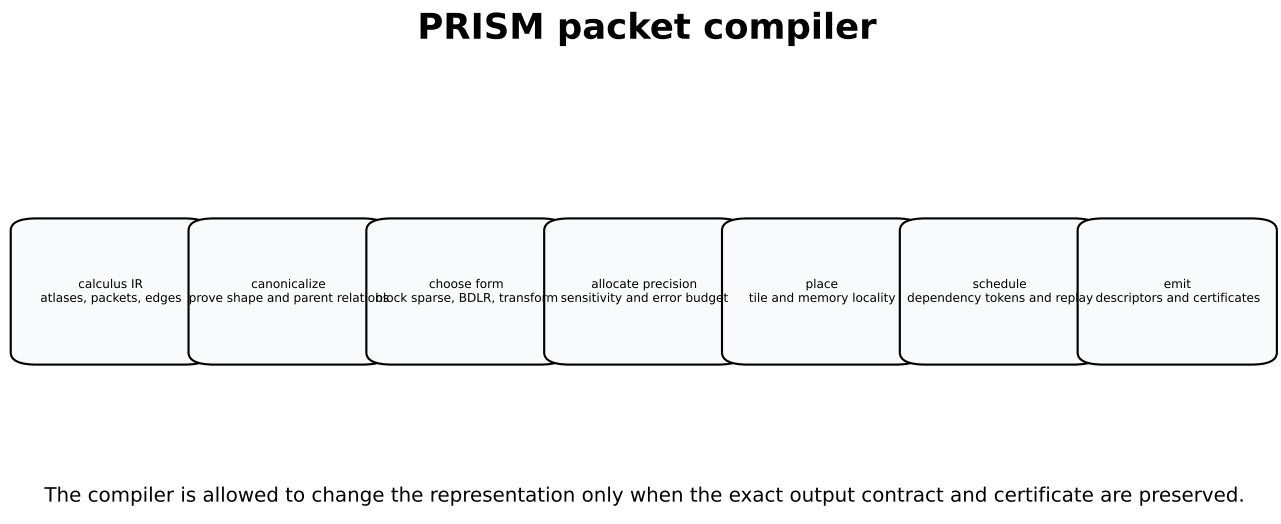


Figure 16.3: The packet compiler preserves the output contract while changing representation, placement, precision, and schedule.

16.4 Legality pass

Before optimization, the compiler checks packet completeness, shape compatibility, parent-child identities, analysis-synthesis pairing, output contracts, and certificate requirements.

16.5 Representation pass

The operator is classified as explicit dense, block diagonal, block sparse, low rank, BDLR, Kronecker, sparse factor chain, transform generated, packet graph, small solver, or nonlinear packet interaction. A form is chosen only when it reduces the declared cost or is required by the output contract.

16.6 PMR–GEMM planning

The representation pass invokes the Bellman planner on the candidate packet tree. The planner considers generic, block, low-rank, Kronecker, sparse-chain, transform, and split choices. The explicit array contract filters out plans that do not return every entry. Apply, factor, and transform contracts leave compressed plans available.

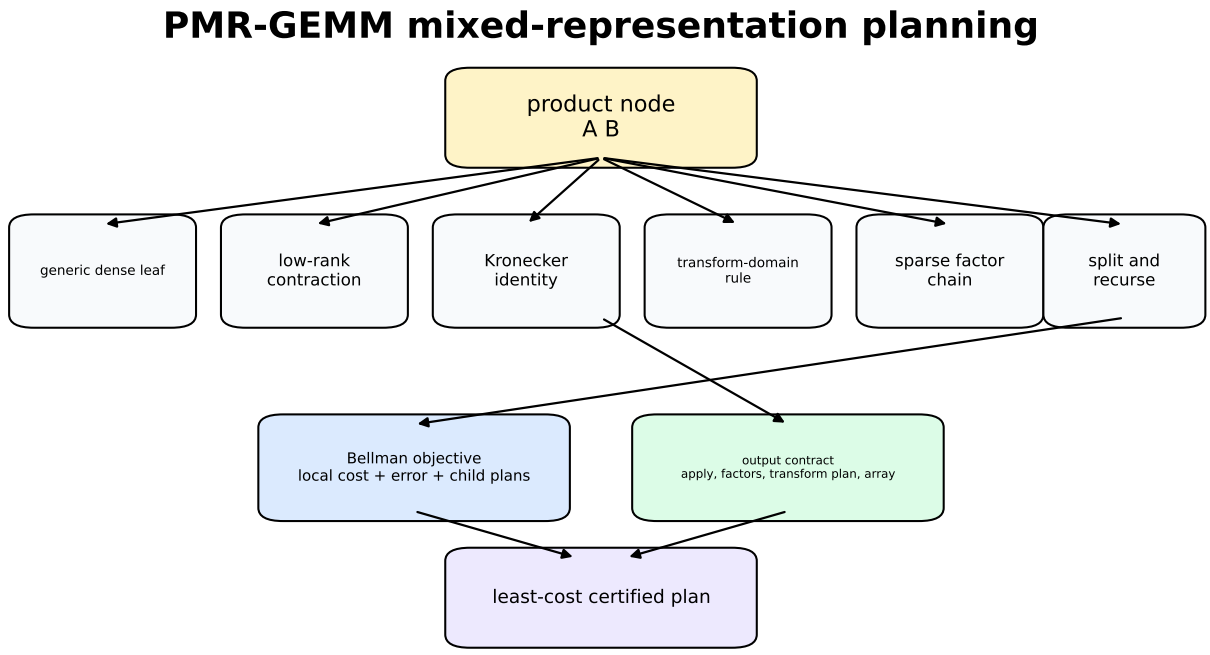


Figure 16.4: The mixed-representation planner selects a certified plan under the requested output contract.

16.7 Placement and fusion

The compiler keeps together parent and child packets, diagonal blocks with corresponding factor rows, transform stages sharing one coefficient stream, and small solver chains. Analysis, packet selection, feature reduction, and synthesis may be fused when the exact atlas contract is unchanged.

16.8 CUDA compatibility backend

The CUDA backend maps Packet IR to grouped GEMM, cuBLAS and cuBLASLt, cuFFT, cuSPARSE, CUB reductions, asynchronous copies, stream-ordered allocation, and CUDA Graph replay. It remains the first implementation target and the reference for same-representation comparisons.

16.9 PRISM backend

The PRISM backend replaces launch-heavy graph regions with resident packet warps, PSM scratchpads, representation routing, transform-to-solver handoff, local refinement, and certificate-gated commit. The backend does not change the representation chosen by the compiler unless the cost model requests a new plan.

16.10 Worked BDLR compilation

A BDLR operator enters the compiler as the typed identity $A = D+UV^*$. No explicit matrix is constructed. The legality pass checks dimensions and the requested output form. The representation pass keeps the factors. Placement co-locates each diagonal block with the corresponding rows of U and V. Precision is assigned per transfer contribution. The schedule computes DX, reduces V^*X, broadcasts the small contraction, and forms $U(V^*X)$ locally.

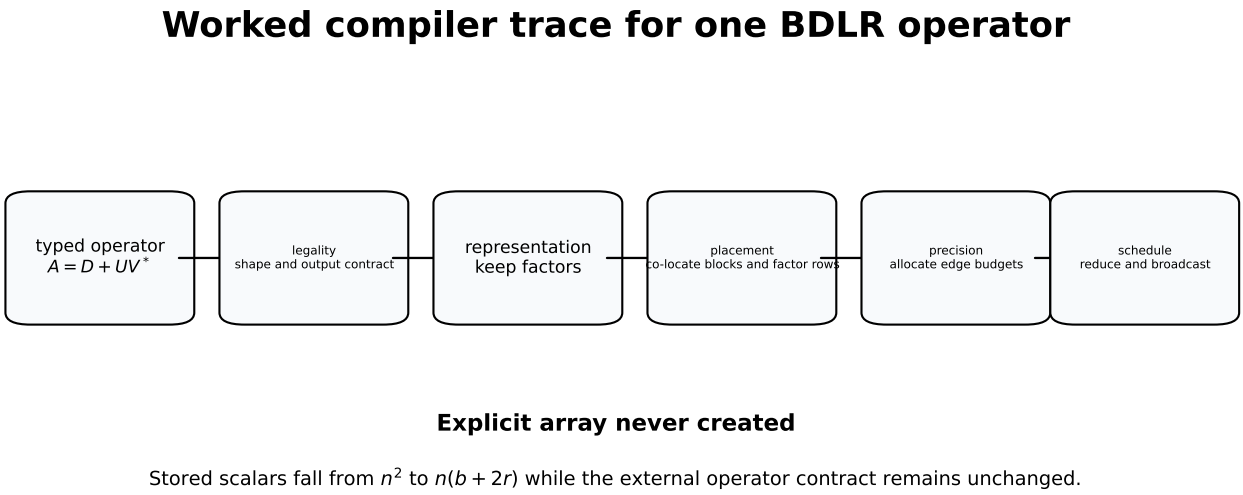


Figure 16.5: One BDLR operator through the compiler. The dense matrix is never formed.

16.11 Certificate plan

The compiler decides where direct residuals are affordable and where proven upper bounds are used. A graph may contain several certificate boundaries rather than one final check.

17 End-to-end execution traces

17.1 Block diagonal plus low rank

The host submits packet descriptors for the diagonal blocks and factor rows. The graph contains one block batch, one reduction for V^*X, one factor broadcast, one local update, and one certificate. The dense

matrix is absent from the graph and from memory.

The external contract is still ordinary matrix application. A direct residual may compare the structured output with a dense reference during bring-up. Production runs can use a proven bound or a spot-check policy when a full dense reference would defeat the representation saving.

17.2 Unknown-edge reconstruction

The transform engine extracts the chosen Fourier packet. The projector DMA groups decimated phase channels. The solver cluster builds the Hankel matrix, estimates rank, finds the annihilator, recovers roots, resolves physical phases, and solves amplitudes. The recovered edge packet is synthesized directly. The smooth residual follows the spectral path. A final residual and branch-margin record are written before commit.

17.3 One PR-LASM step

The exact archive remains in HBM. Macro, meso, and micro states remain in SRAM. Mechanism families reuse the same state matrices. The small solvers update their posterior records. The planner reads exact transition edges separately from probabilistic predictions. A large residual can request local packet refinement without invalidating the rest of the graph.

17.4 Why these traces matter

The three examples use different engines but share one architectural pattern. The representation remains typed, local work is scheduled by dependency tokens, intermediate factors remain resident, and the result is not committed until the declared check passes.

18 Compositional correctness

Theorem 18.1 (Hardware reconstruction theorem)**.** *Let a finite calculus graph have exact packet analysis and synthesis, exact transfer kernels, and an acyclic dependency graph. Suppose every hardware command implements its declared local map and every packet sum uses the declared complete family. Then the committed hardware output equals the mathematical graph output.*

Proof. Order the graph topologically. Each source command loads the declared seed or packet. Assume every predecessor of a node has the correct value. The command implements the node's declared local map, so the node output is correct. Packet accumulation is exact because the declared child or source family is complete. Induction reaches every sink. The final synthesis therefore equals the mathematical graph output. The commit command changes status, not value. □

Theorem 18.2 (Certified commit)**.** *Suppose a command result is provisional until a certificate $c(y)$ is evaluated. If the commit controller releases y only when $c(y)$ satisfies the declared acceptance rule, then every visible result has passed that rule. A failed result cannot satisfy a downstream completion token.*

Proof. The provisional buffer is not addressable through the committed packet directory. Passing the rule changes the directory state and releases the completion token. Failure leaves the committed state

unchanged and redirects the declared graph region to replay or abort. Downstream commands therefore cannot observe an unchecked value through the normal dependency path. □

Corollary 18.3 (Local refinement correctness)**.** *If a parent packet is replaced by a child graph whose synthesis equals the parent interface, the refined hardware graph has the same external mathematical output.*

Proof. The topological-splice theorem gives a legal schedule. Exact child synthesis gives the same value at the parent interface. Every downstream command therefore receives the same input. □

19 Performance accounting

19.1 Four different sources of gain

A fair result must say what changed. The total route may improve because the mathematics removed work, because the same representation ran better on another device, because a kernel was fused or graph-replayed, or because one-time setup was amortized. These are different quantities:

$$S_{\text{total}} = S_{\text{representation}} S_{\text{architecture}} S_{\text{kernel}} S_{\text{amortization}}.$$

This is a bookkeeping identity only when every factor comes from matched rows with the same output contract and timing scope. Otherwise the factors are reported separately.

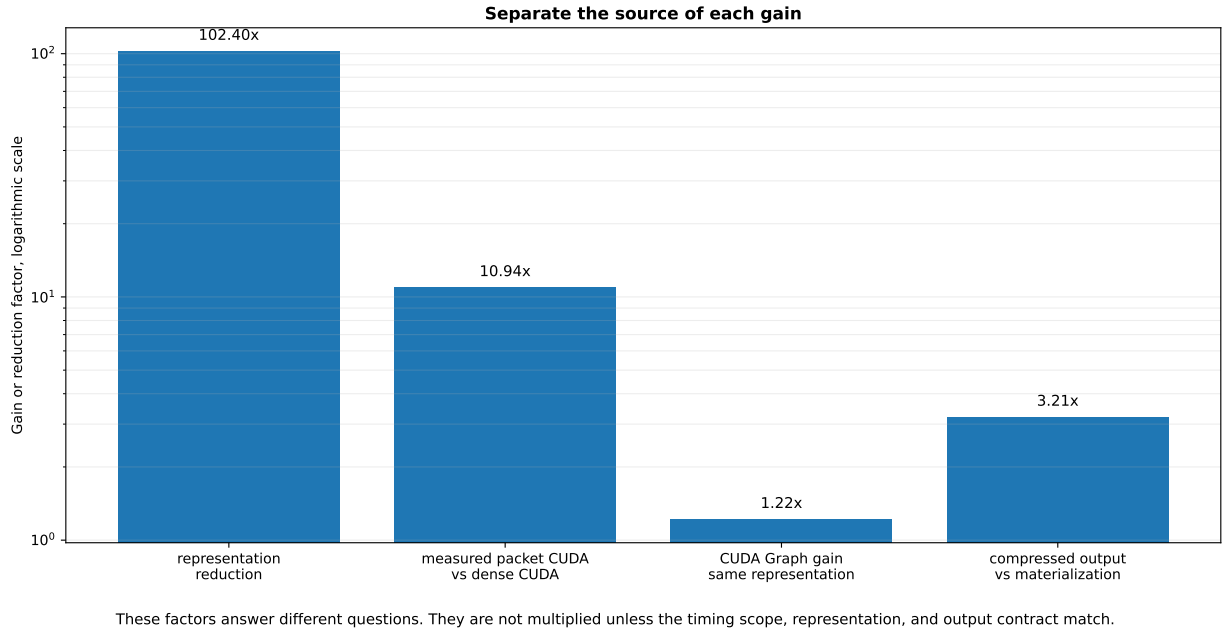


Figure 19.1: Representation, packet-CUDA, graph-replay, and output-contract gains answer different questions.

19.2 Measured CUDA foundation

The prior packet-GPU study contains 154 matched CPU and GPU rows. The recorded GPU is an RTX 3070 Ti Laptop GPU. For $n = 8192$, block width 64, and 32 right-hand sides, the equal-block packet path took 0.046032 ms and was 14.8919× faster than the matched device-resident dense GPU row. For BDLR

with rank eight, the standard packet path took 0.097280 ms and CUDA Graph replay took 0.079872 ms. The graph row was 10.9355× faster than the matched dense GPU row. The graph improvement over the standard packet path was about 1.218×.

A compressed BDLR product at $n = 1024$ took 0.1582 ms. Materializing the explicit output took 0.5073 ms. Those rows have different output contracts. The comparison shows the cost of requesting every entry. It is not a claim that one result is a faster implementation of the other.

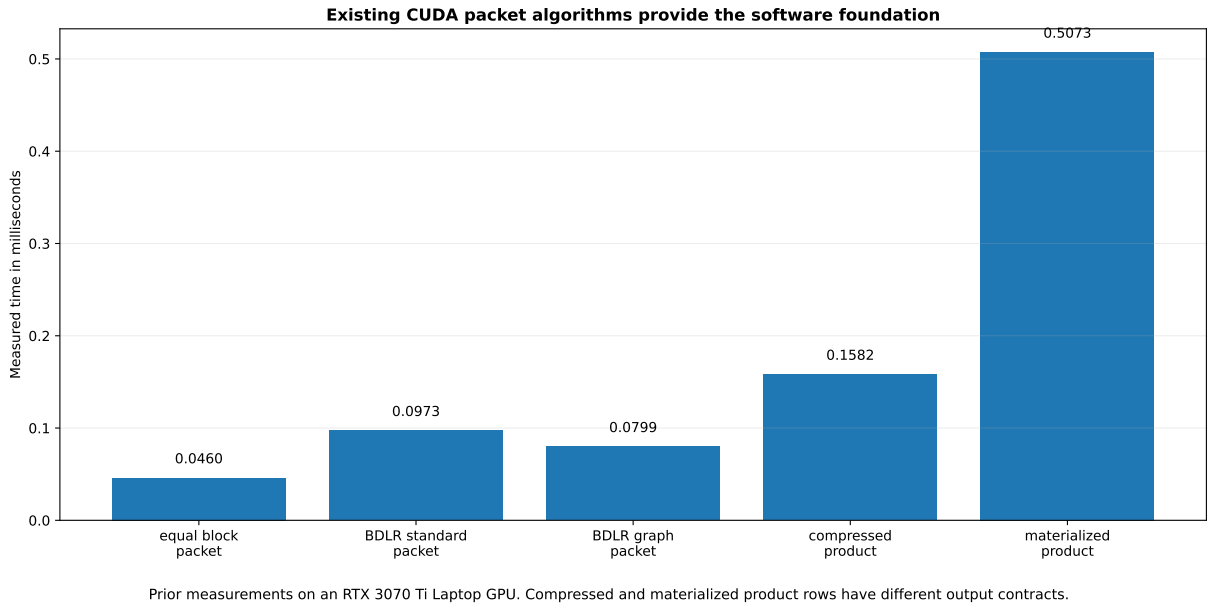


Figure 19.2: Measured CUDA packet timings from the earlier structured-operator study.

19.3 Representation reduction

For

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

with dimension n, block width b, rank r, and q right-hand sides, a factor-first application uses

$$2qn(b + 2r)$$

modeled operations and stores

$$n(b + 2r)$$

scalars. The corresponding dense counts are $2qn^2$ and n^2. For

$$n = 8192, \qquad b = 64, \qquad r = 8, \qquad q = 32,$$

the arithmetic and storage reduction is

$$\boxed{\frac{n}{b + 2r} = 102.4.}$$

This reduction belongs to the representation. A CUDA backend and a PRISM backend can both use it.

19.4 PRISM architecture model

The architecture model uses

$$T = \max\left(\frac{F}{\eta_c P}, \frac{B}{\eta_b W}\right) + T_{\text{graph}},$$

where F is arithmetic, B is data movement, P is peak arithmetic, W is bandwidth, and η_c, η_b are efficiency factors. The model is used to decide which hardware paths are worth building. It is not a silicon measurement.

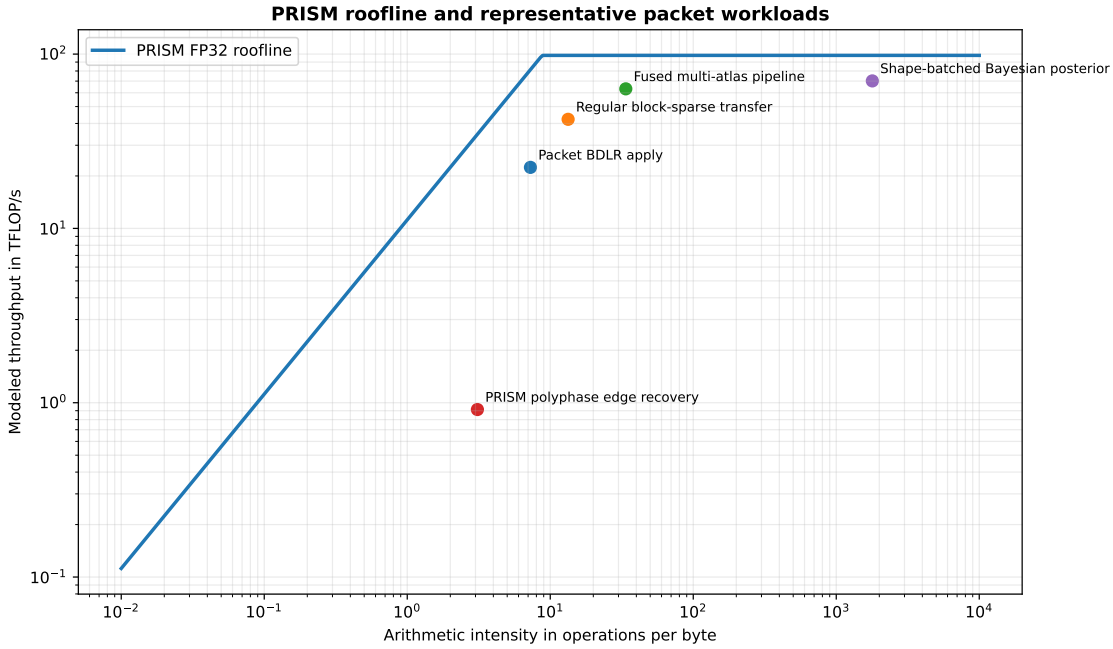


Figure 19.3: Reference PRISM roofline with representative packet workloads.

19.5 Total route model and architecture-only models

The earlier dense-route comparison predicts 70.79 microseconds for a dense application and 1.87 microseconds for the PRISM BDLR route. The ratio is about 37.8. This is a total route model. It includes representation reduction and therefore is not labeled PRISM-over-CUDA architecture gain.

The more relevant architecture models begin with the same structured pipeline on both sides. They estimate:

Workload	Generic structured path	PRISM path
Fused multi-atlas analysis	14.73 μs	8.96 μs
Polyphase edge recovery	4.00 μs	1.35 μs
Bayesian model portfolio	427.93 μs	300.52 μs

The corresponding modeled gains are 1.64×, 2.96×, and 1.42×. These are the present estimates of what residency, fusion, local solvers, and packet scheduling may add after a structured representation is already in use.

19.6 Sensitivity and crossover

No single row should define the architecture. BDLR gains depend on block width and correction rank. Transform gains depend on size, reuse, and fusion. Sparse graphs depend on regularity. Small solvers depend on batch size and handoff cost.

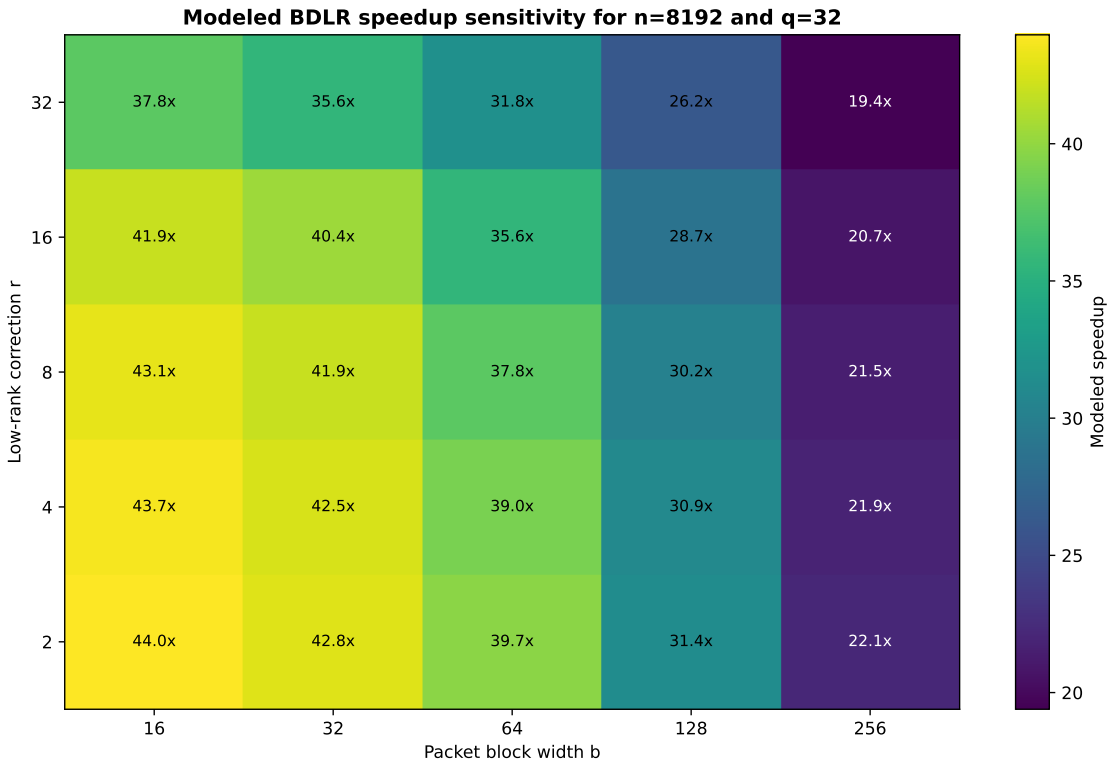


Figure 19.4: Total route sensitivity across block width and correction rank.

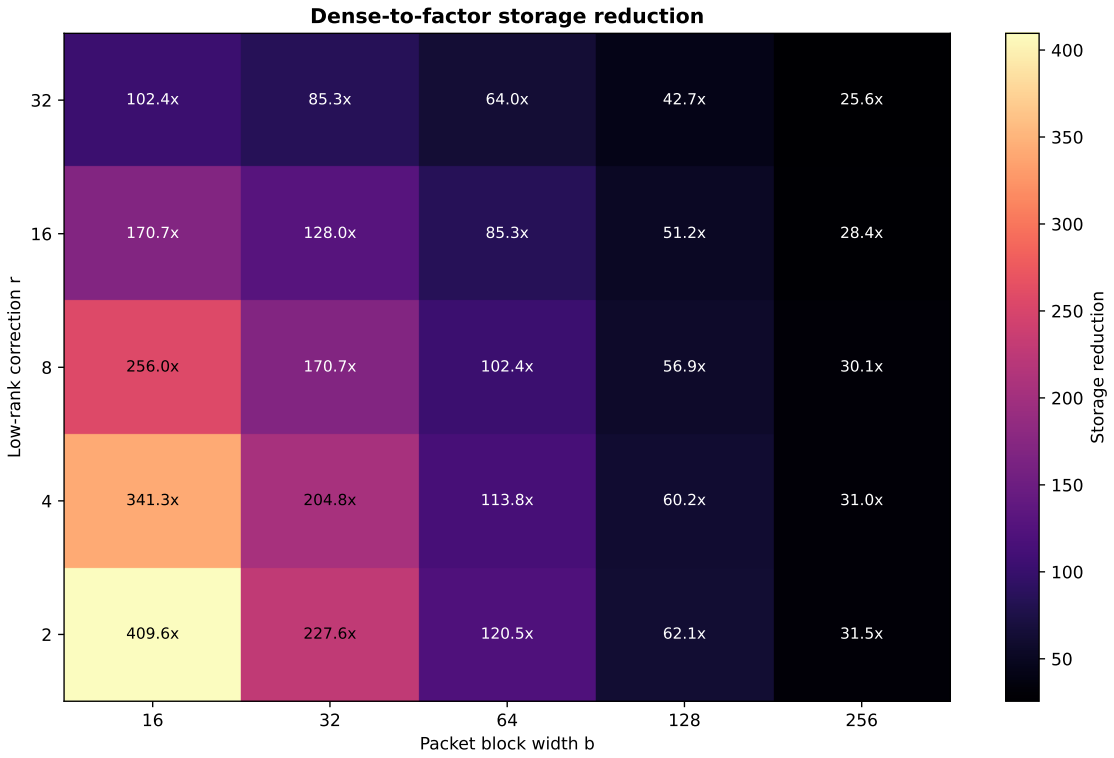


Figure 19.5: Representation-storage reduction across block width and correction rank.

19.7 Negative controls

The earlier CUDA study retained cases where structure lost. Recursive HODLR was slower than dense GPU execution in the tested case. A GPU Walsh-Hadamard transform lost to the CPU at the smaller tested length. A two-stream dense pipeline lost to the serial path. Iterative refinement lost to a direct FP64 solve. The PRISM compiler must preserve these negative controls and route similar work away from specialized paths.

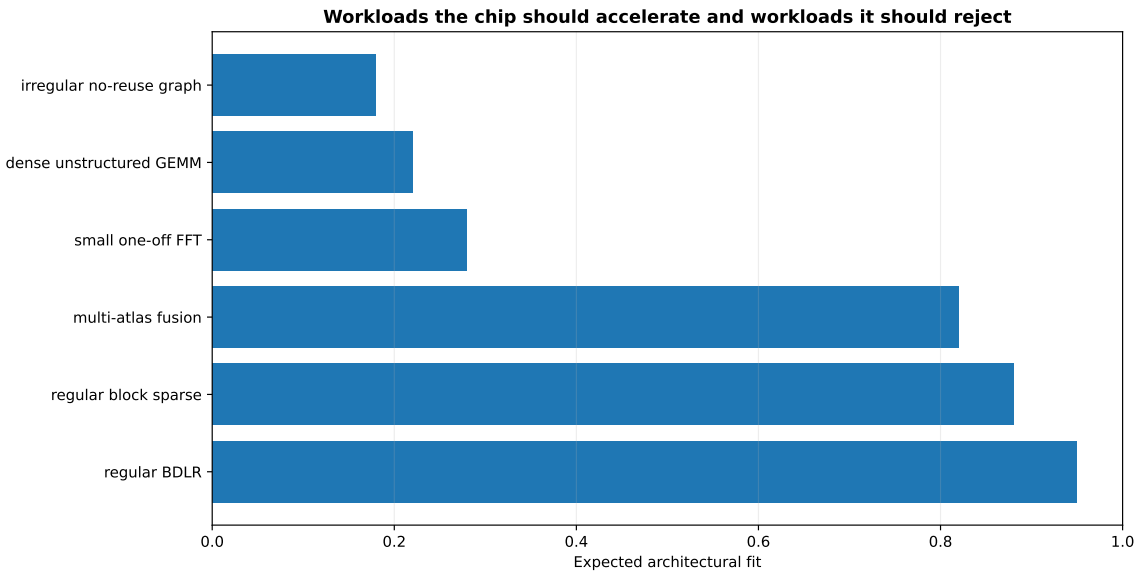


Figure 19.6: PRISM should accelerate strong reusable structure and reject weak or one-off structure.

20 Power, area, and thermal design

20.1 Power budget

The reference package targets a 500 W envelope. The model allocates power to execution groups, SRAM, the packet network, HBM PHYs, transform and solver blocks, control, and margin.

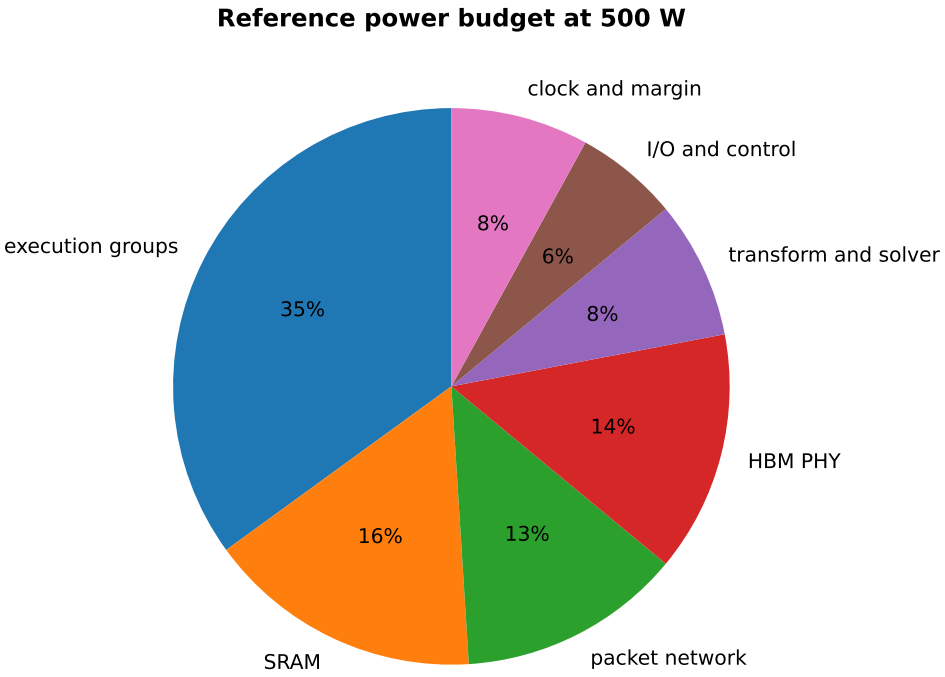


Figure 20.1: Reference power allocation.

20.2 Area allocation

Distributed SRAM is the largest target area because the architecture depends on resident packets and factors. Matrix and vector lanes occupy the next largest share.

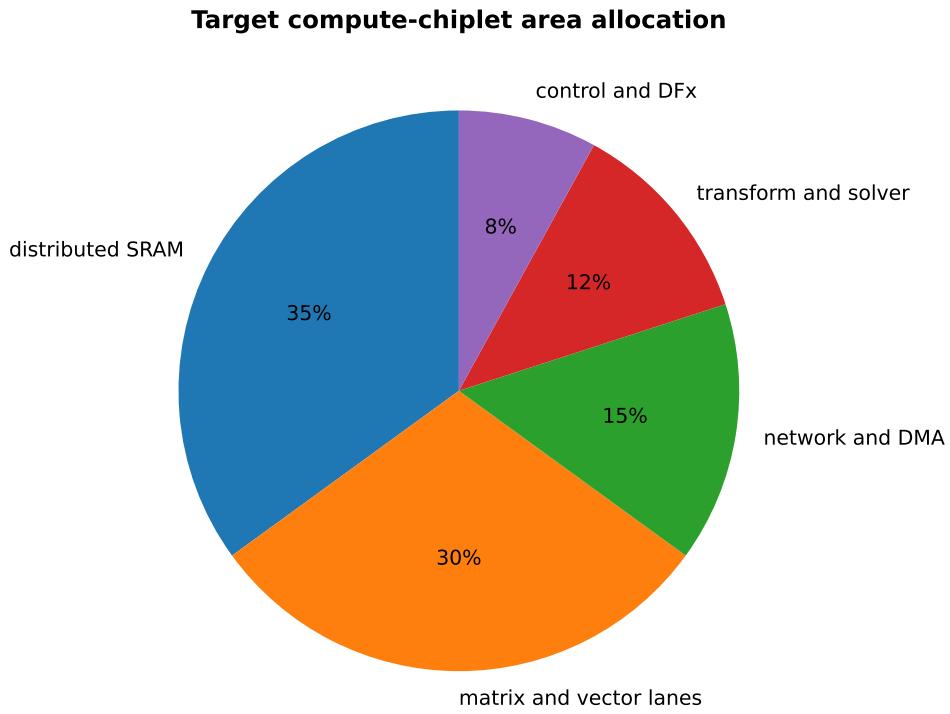


Figure 20.2: Target compute-chiplet area allocation.

20.3 Thermal behavior

The scheduler can spread long transform and matrix batches across chiplets. UCIe priority sideband and fast-throttle features are natural matches for package-wide thermal events. Local descriptor replay can be paused without discarding the packet graph.

21 Reliability, protection, and recovery

21.1 Protected state

Packet descriptors, transfer descriptors, dependency tokens, and certificate records control meaning as well as data movement. They require stronger protection than ordinary cache metadata. The reference plan uses ECC for SRAM-resident tables, parity on command queues, sequence numbers on replay regions, and end-to-end tags on HBM packet records.

21.2 Failure containment

An execution-group or PSM error should not force a package-wide reset unless the packet directory or global scheduler has lost coherence. Local kernel faults invalidate the destination packet and its descendants. A link or memory fault invalidates every packet whose provenance crosses the failed resource. The compiler emits provenance information so the runtime can identify the affected region.

21.3 Determinism

Deterministic mode fixes the reduction tree, solver ordering, and replay schedule. It is slower than the highest-throughput mode. Its purpose is bring-up, regression testing, exact replay, and certificate comparison.

21.4 Security boundary

The architecture does not introduce a new trust model. Host isolation, command validation, memory protection, firmware signing, and secure update remain required. Packet descriptors make the dataflow easier to audit, but they do not replace platform security.

22 Programming model

22.1 Four software layers

The software stack has four levels:

1. a high-level calculus graph
2. shared Packet IR
3. CUDA or PRISM backend lowering
4. Packet Seed Commands, descriptor tables, and the runtime queue

The release includes JSON examples for BDLR application, PMR–GEMM planning, self-calibrating edge recovery, and a PR-LASM step.

22.2 Host control

RISC-V management cores run the command runtime, graph loader, memory manager, telemetry, and error handling. The CUDA backend uses the same Packet IR before final lowering. Ordinary vector code handles control-side data preparation. The accelerator command queue is the stable interface to the packet engines.

22.3 Device libraries

A practical library should expose:

- packet and atlas construction
- transform analysis and synthesis
- transfer graph creation
- BDLR and block-sparse operators
- edge-recovery graphs
- Bayesian program batches
- certificate policies
- graph replay and local refinement

23 Verification

23.1 Verification ladder

The architecture is specified so each development stage has a clear target. Mathematical contracts come first. The behavioral model checks those contracts. Bit-accurate kernels then define rounding and exception behavior. RTL simulation and formal properties check control. An FPGA overlay measures real data movement before a chiplet is attempted.

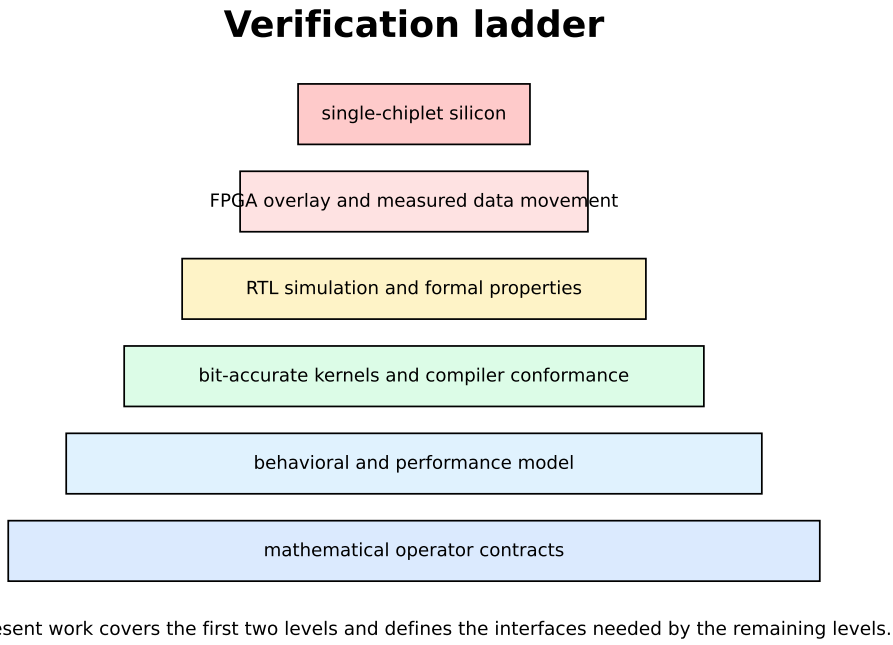


Figure 23.1: Verification ladder from mathematical contracts to measured silicon.

23.2 Behavioral tests

The executable model currently verifies:

- complete residue-packet reconstruction

- exact polyphase round trip
- exact Haar round trip
- BDLR factor execution against dense materialization
- local DAG substitution
- derived package totals
- descriptor contracts
- precision-budget allocation
- certificate-gated commit
- local refinement descriptor bounds
- compiler routing decisions
- BDLR sensitivity tables
- package bandwidth hierarchy
- Packet Streaming Multiprocessor hierarchy
- PMR–GEMM Bellman planning
- output-contract legality without mandatory dense state
- measured CUDA baseline registry
- typed gain decomposition

Certificate 23.1 (Current executable result)**.** *The behavioral and architecture model passes*

$$\boxed{18/18 \textit{ self-tests}}$$

and the independent pytest suite passes

$$\boxed{18/18 \textit{ tests.}}$$

The largest semantic residual is the BDLR comparison residual

$$1.42 \times 10^{-14},$$

which is consistent with floating-point roundoff.

23.3 RTL boundary

The release contains a SystemVerilog architecture skeleton for descriptors, command buffering, dependency tokens, the packet directory, representation routing, PSM scheduling, execution-group control, the residual path, certificate-gated commit, and the top-level queue interface. No HDL compiler or synthesis tool was available in the build environment. The RTL skeleton is therefore not described as verified or synthesizable silicon.

23.4 What must be verified before tapeout

A serious implementation requires:

- bit-accurate kernel models
- formal checks of command ordering and descriptor protection
- CDC and reset verification
- UCIe, HBM, PCIe, and optional CXL compliance IP integration
- SRAM and NoC timing closure
- thermal and power-integrity analysis
- fault injection for replay and certificate failure
- compiler and runtime conformance suites

24 Implementation path

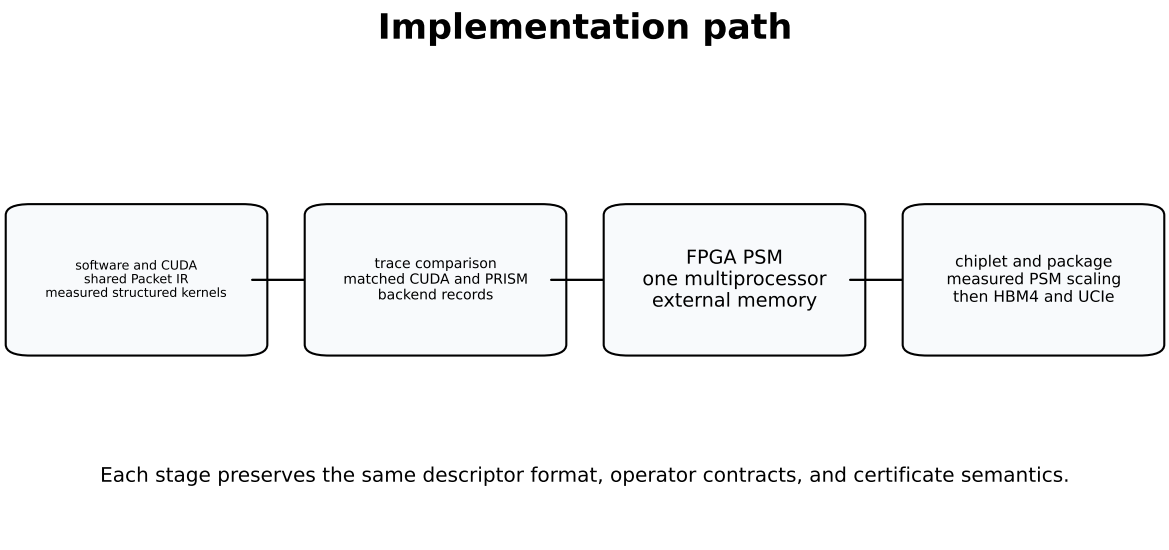


Figure 24.1: The software and CUDA stages precede FPGA and silicon work.

24.1 Stage 1: software and CUDA foundation

The calculus already has CPU references, CUDA packet kernels, grouped block execution, compressed BDLR products, Woodbury solves, CUDA Graph replay, mixed-precision certificates, transform paths, and measured structured-operator baselines. This stage supplies the semantics, Packet IR, workload traces, and matched comparison.

24.2 Stage 2: trace-driven backend comparison

The next software stage lowers the same Packet IR to CUDA and to the PRISM simulator. It records representation, bytes, arithmetic, launch count, solver size, reuse, residual, and output contract. This stage identifies the graph regions that remain bottlenecked after CUDA has already used the better algorithms.

24.3 Stage 3: FPGA Packet Streaming Multiprocessor

The FPGA overlay should implement one Packet Streaming Multiprocessor, packet DMA, one transform path, one matrix and factor path, one small solver, the packet directory, dependency tokens, residual accumulation, and certificate-gated commit. The descriptor and Packet IR formats should match the later ASIC.

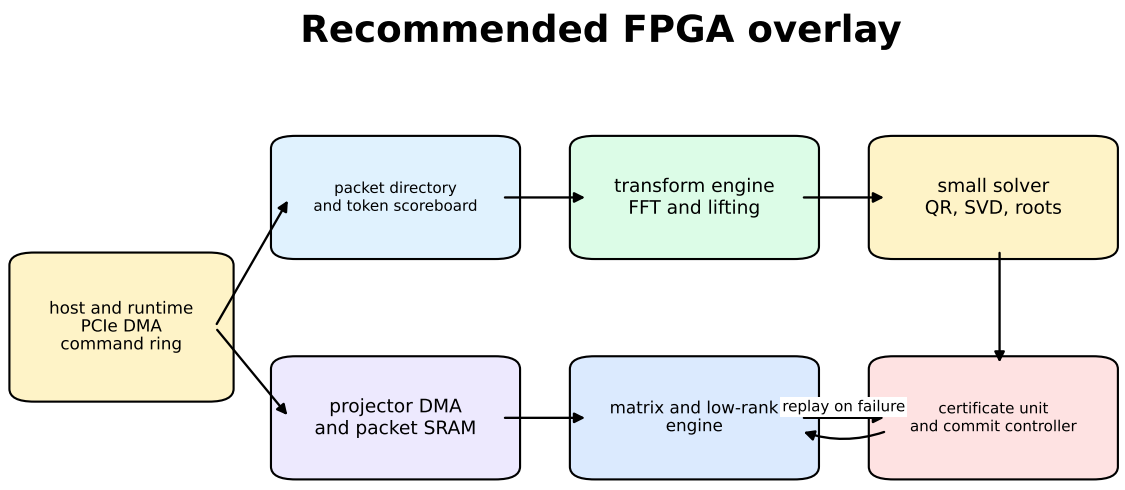


Figure 24.2: Recommended FPGA overlay for measuring the architectural features that do not reduce to an ordinary CUDA kernel.

24.4 Stage 4: one compute chiplet

A shuttle-scale device can use four to eight Packet Streaming Multiprocessors, one transform cluster, one small-solver cluster, external memory, and a simple host link. This stage should measure factor residency, scratchpad bandwidth, packet-warp occupancy, transform-to-solver handoff, and local refinement.

24.5 Stage 5: full package

The full package adds four compute chiplets, HBM4, UCIe, host I/O, and package-wide management. It is justified only after the CUDA and FPGA stages show a stable same-representation advantage on real workloads.

24.6 Development risks

The highest risks are package power, HBM integration, small-solver occupancy, network congestion, compiler misclassification, descriptor overhead, and false certificate acceptance. Early CUDA and FPGA measurement is the main mitigation.

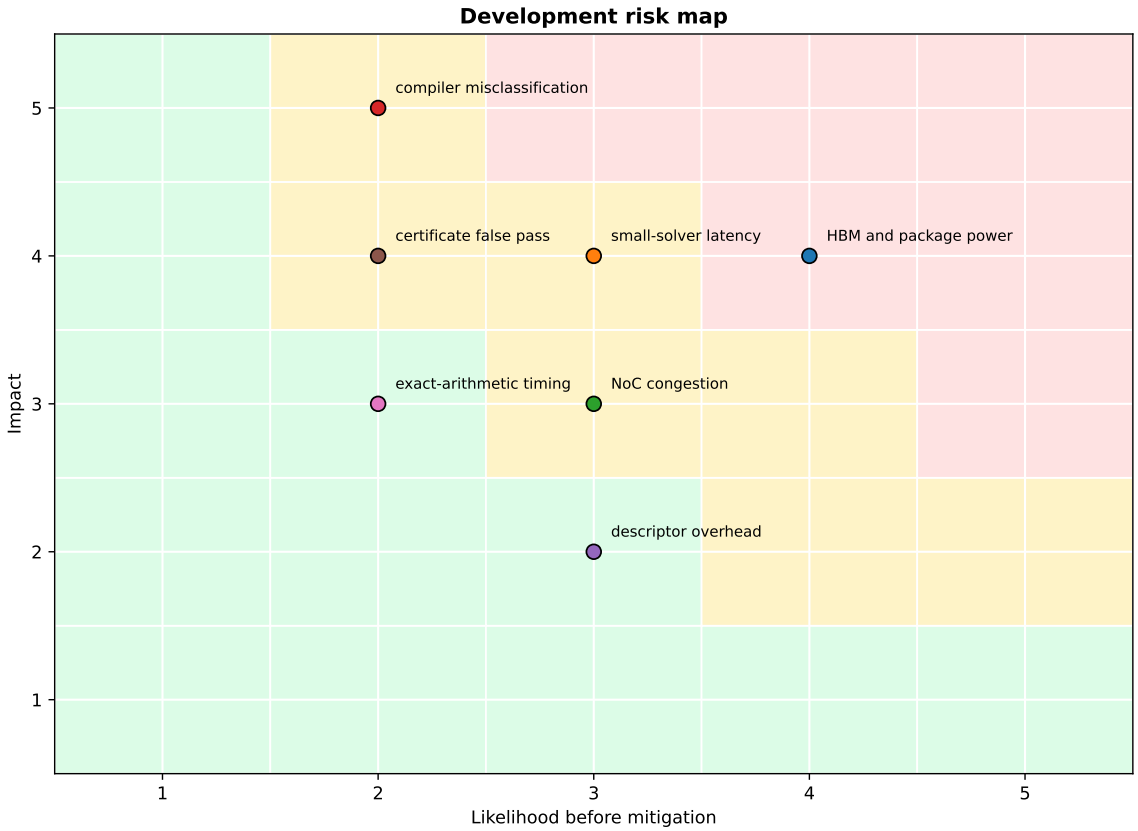


Figure 24.3: Development risk map. Values are planning judgments rather than measured frequencies.

24.7 Architecture freeze checklist

The following items should be frozen before production RTL begins.

1. Packet, transfer, atlas, representation, and certificate descriptor semantics
2. Packet IR representation forms and output contracts
3. Packet-warp and Packet Streaming Multiprocessor scheduling rules
4. command and completion-record formats
5. commit and replay rules
6. supported numerical formats and rounding behavior
7. minimum transform, factor, and solver operation set
8. CUDA and PRISM backend conformance tests
9. error and exception reporting

SRAM bank count, issue width, NoC virtual channels, exact root-solver implementation, and final DVFS domains may remain implementation choices until synthesis and floorplanning begin.

25 What would count as success

The chip should be judged against claims that can be measured:

1. structured operations move fewer bytes than an explicit-array route
2. BDLR, PMR–GEMM, and regular block-sparse graphs sustain useful Packet Streaming Multiprocessor occupancy
3. transform analysis and synthesis remain exact within the declared numerical format
4. mixed-precision results satisfy their packet bounds
5. local refinement changes only the local descriptor region
6. edge recovery keeps the small algebraic chain resident
7. Bayesian portfolios reuse state matrices rather than reloading them per model
8. negative controls are routed to CUDA, a general device, or a generic leaf when PRISM is not the better path

A successful PRISM chip would not prove the calculus. The mathematics already supplies the contracts. The hardware question is whether keeping those contracts visible saves enough scheduling, movement, and handoff cost beyond the already optimized CUDA implementation to justify a dedicated machine.

26 Final architecture statement

The final design follows one principle:

> Do not materialize, move, or schedule structure that the calculus has already declared inactive.

The machine executes

$$\begin{aligned} \text{seed object} &\longrightarrow \text{Packet IR} \\ &\longrightarrow \text{mixed-representation plan} \\ &\longrightarrow \text{packet warps on CUDA or PRISM} \\ &\longrightarrow \text{local refinement and synthesis} \\ &\longrightarrow \text{certificate and commit.} \end{aligned}$$

PRISM-1 is a CUDA-like packet-programmable processor, not a dense-matrix preservation machine. Its hardware state may be a factor graph, transform chain, packet tree, sparse operator, explicit array, or mixture of those forms. The requested output contract determines what must be returned.

The CUDA backend proves that the calculus can run before silicon. The FPGA and ASIC question is narrower and testable: whether resident packet scheduling, factor reuse, transform-to-solver handoff, local refinement, and certificate-gated execution add enough value beyond the packet CUDA path.

A Reference data sheet

Item	PRISM-1 reference target
Compute chiplets	4
Packet Streaming Multiprocessors	64 total, 16 per compute chiplet
Packet execution groups	256 total, 4 per multiprocessor
Execution group	16×16 matrix array plus programmable vector lanes
Clock	1.5 GHz model target
Packet scratchpad	2 MiB per multiprocessor across four banked regions
Shared SRAM	16 MiB per compute chiplet
Total local and shared SRAM	192 MiB
FP16 and BF16 model peak	196.6 TFLOP/s
FP32 model peak	98.3 TFLOP/s
FP64 model peak	24.6 TFLOP/s
Transform engines	32
Small-solver engines	16
Exact arithmetic clusters	4
HBM4	4 stacks
HBM capacity	144 GiB
HBM bandwidth	greater than 11.2 TB/s aggregate
Package links	UCIe 3.0 class at up to 64 GT/s
Host link	PCIe 7.0 x16 class, optional CXL
Power target	500 W
Representation forms	explicit array, block, block sparse, low rank, BDLR, Kronecker, sparse factor chain, transform plan, packet graph
Output contracts	apply, compressed operator, factor graph, transform plan, explicit array
Software backends	CUDA compatibility backend and PRISM backend from shared Packet IR

B Operation-to-engine map

Calculus operations and hardware engines

polyphase	projector DMA
Fourier	transform
wavelet	transform
BDLR	matrix + low rank
block sparse	matrix + router
Bayesian	solver
Prony	solver + transform
Volterra	memory unit
exact modular	exact arithmetic

Figure B.1: The main calculus operations and their primary hardware engines.

C Executable model and source layout

The source package contains the executable architecture model, independent tests, compiler examples, documentation, generated figures, machine-readable results, and an RTL module skeleton. Raw source listings are kept in the release rather than printed across the final paper.

Path	Role
`prism_chip_model.py`	Behavioral operations, roofline model, sensitivity sweeps, figures, and self-tests
`tests/`	Independent contract tests
`compiler_ir/`	BDLR, PMR–GEMM, unknown-edge, and PR-LASM graph examples
`rtl_skeleton/`	Descriptor, queue, scoreboard, packet directory, representation router, PSM scheduler, execution group, residual, commit, and top-level module boundaries
`docs/`	Data sheet, command specification, compiler notes, and microarchitecture notes
`results/`	Machine-readable architecture results

D RTL module boundaries

Module	Responsibility
`prism_descriptor_pkg`	Command opcodes and packed descriptor types
`prism_command_fifo`	Command-ring buffering and flow control
`prism_token_scoreboard`	Dependency-token readiness and completion tracking
`prism_packet_directory`	Packet validity, provisional state, commit state, and storage location
`prism_residual_accumulator`	Wide residual accumulation
`prism_commit_controller`	Certificate comparison, commit, replay, and abort decisions
`prism_representation_router`	Dispatch among block, factor, transform, solver, packet-graph, and generic leaves
`prism_psm_scheduler`	Packet-warp issue, dependency tracking, replay-region control, and execution-group assignment
`prism_execution_group`	Matrix, vector, selector, reduction, and provisional-result boundary for one execution group
`prism_tile_shell`	Compatibility shell for one execution group during staged RTL development
`prism_top`	Queue, scoreboard, packet directory, and completion interface shell

The module set remains a boundary specification rather than production RTL. A production implementation still requires bit-accurate datapaths, interface IP, reset and clock-domain design, verification properties, synthesis, timing closure, and physical design.

E Implementation baseline

Internal algorithmic basis

The architecture integrates three existing project components.

- Dense Transfer Algebra supplies PMR–GEMM, compatible middle bases, Kronecker rules, sparse factor chains, and the generic fallback.
- Projector-Resolved Seed Calculus VIII supplies exact block, BDLR, compressed-product, Woodbury, packet-Krylov, mixed-precision, amortization, and CUDA Graph results.
- The CUDA backend supplies the working software route through grouped matrix kernels, FFT and sparse libraries, asynchronous transfers, graph replay, and packetwise certification.

These algorithms are architecture inputs. PRISM does not claim them as consequences of the chip.

The package assumptions use current public baselines from the UCIe Consortium, PCI-SIG, Micron, and RISC-V International. The relevant values are the 48 and 64 GT/s classes in UCIe 3.0, the 128 GT/s PCIe 7.0 host-link class, Micron's 2048-bit HBM4 interface with bandwidth above 2.8 TB/s per stack, and the ratified RISC-V Vector 1.0 programming model. These sources establish a plausible interconnect, memory, and control baseline. They do not validate the PRISM microarchitecture.