

DLR: Efficient Defect-Tolerant Routing at Wafer Scale

Yi-Chieh Wang
University of Edinburgh
Edinburgh, United Kingdom
mike.wang@ed.ac.uk

Congjie He
University of Edinburgh
Edinburgh, United Kingdom
congjie.he@ed.ac.uk

Haocheng Xiao
University of Edinburgh
Edinburgh, United Kingdom
haocheng.xiao@ed.ac.uk

Boris Grot
University of Edinburgh
Edinburgh, United Kingdom
Boris.Grot@ed.ac.uk

Luo Mai
University of Edinburgh
Edinburgh, United Kingdom
luo.mai@ed.ac.uk

Abstract

Wafer-scale integration enables unprecedented compute density on a single chip, affording nearly a million processing elements (PEs) in a mesh-connected topology. However, wafer-scale meshes must tolerate inevitable manufacturing defects. Existing industrial practice for wafer-scale relies on holding healthy PEs back. Repair mechanisms either substitute spares through redundant links or disable entire rows and columns, costing tens of thousands of PEs in production. Prior fault-tolerant (FT) routing schemes fall short at wafer-scale by requiring prohibitively large per-router state or many sets of virtual channels, or require disabling healthy PEs to enforce regular fault shapes.

We observe that under realistic defect densities, although the chip contains nearly a million PEs, only several dozen are expected to be faulty, leaving the mesh almost entirely intact. This suggests that fault tolerance should not be imposed uniformly across the whole wafer, but instead only where faults actually disrupt regularity. Motivated by this insight, we present Defect-driven Locality-preserving Routing (DLR), a fault-tolerant routing scheme that decomposes the defective wafer mesh into regular, defect-free submeshes and applies standard low-cost routing within each submesh. Routing across submeshes (i.e., across fault boundaries) is done over a virtual spanning tree. This hierarchical structure preserves deadlock freedom with only two sets of virtual channels (VCs), just one more than the fault-free baseline. DLR achieves 100% utilization of healthy PEs, low per-router storage and hardware overhead, competitive path lengths, and low offline algorithmic complexity; none of the prior techniques manage to achieve all of these requirements.

CCS Concepts: • CCS Concepts pending;

Keywords: wafer-scale computing, fault-tolerant routing, network-on-chip, deadlock freedom, manufacturing defects

ACM Reference Format:

Yi-Chieh Wang, Congjie He, Haocheng Xiao, Boris Grot, and Luo Mai. 2027. DLR: Efficient Defect-Tolerant Routing at Wafer Scale.

In *Proceedings of the 32nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '27)*, April 11–15, 2027, Heraklion, Greece. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/XXXXXXX.XXXXXX>

1 Introduction

Wafer-scale integration, whereby an entire wafer is fabricated and deployed as a single chip, enables extreme compute density of scale that can accommodate an entire modern AI workload. For instance, the Cerebras WSE-3 [17, 33], the state-of-the-art commercial wafer-scale chip, integrates nearly a million processing elements (PEs) on 46,225 mm² of silicon connected by a 2D mesh network. This scale of integration eliminates the off-chip memory and inter-chip communication bottleneck that hinders conventional multi-chip systems, enabling ultra-low latency for workloads such as large language model (LLM) inference [12].

A critical and unavoidable challenge for wafer-scale systems is manufacturing defects. Every fabricated wafer contains defects whose locations vary from wafer to wafer (§2.2). At die scale, a defect costs at most the die it lands on: that die is binned to a lower grade or discarded. At wafer scale, every defect lands in the one product, and tolerating defects on-chip is the only cost-effective option. Cerebras' patent describes repairing defects by substituting spare PEs or by disabling the entire rows and columns that contain them [18]; in production, approximately 7% of PEs are thus disabled for defect tolerance – a loss of 70,000 PEs on the WSE-3 [33].

An alternative to such brute-force and wasteful solutions is fault-tolerant (FT) routing, which would allow *all* of the functional PEs to be used by routing around faulty ones. While conceptually simple, FT routing at wafer scale is challenging because it must jointly satisfy *memory efficiency*, *scalability in offline route computation*, *defect tolerance generality across both random and spatially correlated defect patterns*, and *microarchitectural efficiency* (§2.4). These requirements are in tension: richer forwarding tables improve route quality but strain the limited per-PE SRAM, while reducing hardware overhead limits the deadlock-avoidance mechanisms available. We find that no existing FT routing scheme meets all four requirements at meshes containing a million PEs.

Existing FT routing approaches fall into two categories (§2.4), neither of which meets the wafer-scale requirements. *Table-based* approaches such as Autonet [29], segment-based routing [20], and Schley [28] precompute per-PE forwarding tables that guarantee 100% routability, but the storage required for these routing tables scales with the number of PEs (N) for Autonet, segment-based, and Schley with a small number of virtual-channel (VC) sets. On a wafer with a million PEs, the per-PE storage for the routing table exceeds the SRAM budget of the entire PE. Schley can reduce storage to $O(N^{2/V})$ by increasing the number of VC sets V , but this proportionally increases the associated buffer area (§5.1). *Tableless* approaches such as fault-ring routing [3] avoid per-PE tables but enforce topological regularity around faults, which forces disabling of healthy PEs to satisfy this constraint. Under clustered or linear defect patterns, the number of sacrificed healthy PEs can grow as $O(F^2)$, where F is the number of faulty PEs or links. In short, no FT routing algorithm exists today that can achieve low hardware and memory overhead while also preserving all healthy PEs.

We observe that at reported defect densities for advanced process nodes (~ 0.1 defects/cm² [7]), merely several dozen PEs would be defective, on average, out of the entire million-PE mesh (e.g., ~ 46 PEs on a wafer-scale chip). Thus, the mesh is nearly intact. Meanwhile, existing FT approaches are designed to contend with a large fraction of faulty PEs, thereby imposing a significant overhead by targeting the worst case. Our insight is to leverage the fact that the mesh is mostly intact to dramatically reduce the cost and complexity of FT routing.

Building on this insight, we propose **Defect-driven Locality-preserving Routing (DLR)**, an FT routing algorithm designed for wafer-scale meshes. DLR partitions the defect-containing mesh into a set of regular (rectangular-shaped) defect-free *submeshes*. Faulty PEs and links define the partition boundaries. Since each submesh is defect-free, existing low-cost routing (e.g., XY or turn-based) can be used inside the submesh to achieve both deadlock freedom and minimal path length. Traffic crossing submesh boundaries is routed over a spanning tree of the submesh adjacency graph using a monotonic ordering that guarantees deadlock freedom. The two routing modes – intra- and inter-submesh – coexist deadlock-free using only two sets of VCs, which is just one set of VCs more than the ideal fault-free baseline mesh¹. The relatively small number of faulty PEs and links on a wafer-scale chip naturally result in a small number of submeshes, which keeps the size of the spanning tree small and the associated hardware cost for forwarding tables low. On an intact wafer, DLR requires zero forwarding state; each defect adds merely $O(\log N)$ bits to every PE's forwarding table.

To summarize, this paper makes several contributions:

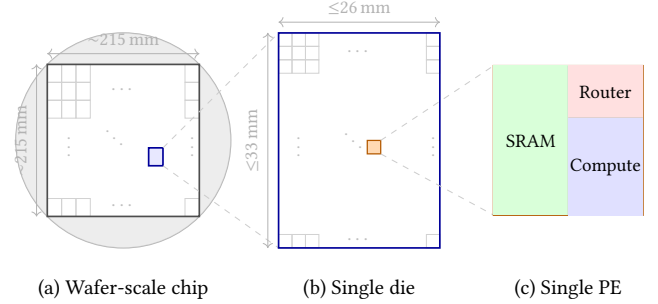


Figure 1. Wafer-scale chip structure. (a) A 12-inch (300 mm) wafer with inscribed square (~ 215 mm side); the square is diced into identical dies. (b) Zoom-in of a single die, all PEs connected by a 2D mesh. (c) A single PE (details in §3).

- We design Defect-driven Locality-preserving Routing (DLR), a novel FT routing scheme that exploits the fact that nearly all of the mesh is intact. DLR partitions the mesh into fault-free rectangular submeshes, uses the baseline low-cost routing algorithm (e.g., XY) inside each submesh, and routes across submeshes over a spanning tree. By construction, DLR guarantees deadlock freedom using only 2 VC sets and maximal routability: every connected pair of healthy PEs remains routable.
- DLR scales to wafer-scale meshes. Its per-PE forwarding state grows with the defect count F rather than the mesh size N : $O(F \log N)$, or 99 B per PE on a million-PE mesh ($1,024 \times 1,024$) at 0.1 defects/cm² (989 B even at $10\times$ that density), whereas Autonet requires 1.9 MB and segment-based 384 KB – both exceeding the tens-of-KB per-PE SRAM capacity of wafer-scale chips. Offline route computation takes $O(N + F^2 \log F)$ time: 0.077 s for a million-PE mesh (under 1 s even at $10\times$ defect density). At that scale, Autonet and segment-based simply cannot complete route computation.
- We evaluate DLR against fault-ring and Schley across six defect patterns and three defect densities. DLR sacrifices no healthy PEs under any pattern, whereas the comparably lightweight fault-ring forfeits up to 23% under clustered defects. On most patterns characteristic of wafer-scale workloads, DLR achieves path lengths and latency competitive with Schley (VC=8) while using $4\times$ fewer VC sets.

2 Background

2.1 Characteristics of wafer-scale systems

A wafer-scale system fabricates a single processor spanning an entire wafer (illustrated by Figure 1). A lithographic stepper exposes a common reticle at each die position; adjacent reticle fields are then *stitched* together so that inter-die links

¹We assume that one set of VCs is necessary to enforce deadlock freedom at the protocol level; e.g., by separating requests and replies.

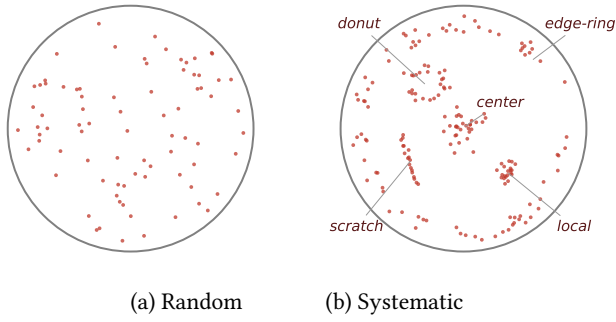


Figure 2. Wafer defect patterns [36]: (a) random contamination produces spatially uniform defects; (b) systematic process variations produce structured patterns such as center clusters, edge rings, scratches, and localized hotspots.

form a seamless on-chip network [16, 17, 33]. Each die contains on the order of ten thousand processing elements (PEs), yielding around one million PEs across the full wafer. Each PE integrates a compute unit and a router, but only a small amount (tens of KB) of local SRAM due to the tight area budget. There is no shared memory or global cache hierarchy. Instead, PEs communicate through explicit message passing over the on-chip network. As a result, routing quality becomes a first-order determinant of system performance.

The network connecting PEs across the wafer is typically organized as a 2D mesh, as evidenced in today’s commercial wafer-scale systems, namely the Cerebras WSE-2 and WSE-3 [16, 17, 33]. The mesh topology naturally addresses the constraints of wafer-scale systems, which include low design complexity, regularity (great for developers and hardware designers), hardware frugality both at the router and wire levels, and low per-hop latency.

2.2 Defect tolerance is key for wafer-scale routing

In a single-die chip, defective dies can be discarded or yield-binned. This is not an option for monolithic wafer-scale, since the entire wafer *is* the “chip”; this means that defects must be tolerated *within* the system. Semiconductor defects are commonly classified into two categories (Figure 2) [34]. *Random* defects arise from particle contamination and vary from wafer to wafer, producing spatially uniform failures. *Systematic* defects originate from process or equipment issues and exhibit spatially correlated patterns such as center clusters, edge rings, scratches, and localized hotspots [34].

The expected number of defects scales with the chip area times the defect density ($D_0 \times A_{\text{chip}}$) [30]. At $D_0 = 0.1$ defects/cm², representative of advanced process nodes [7], a wafer-scale chip with active area ≈ 465 cm² contains ~ 46 defects on average. Depending on the physical layout, multiple defects may map to the same PE or link, so the number of faulty mesh components may be lower. Even in the pessimistic case where each defect disables a distinct PE, 46 faulty PEs in a

$1,024 \times 1,024$ mesh account for only 0.004% of the fabric. The huge implication is that the mesh is almost entirely intact.

2.3 Novel requirements for wafer-scale routing

On a defect-free 2D mesh, dimension-order routing (e.g., XY) provides minimal-path, deadlock-free forwarding with $O(1)$ per-hop decision complexity. Defects break these guarantees: detours around faulty PEs or links violate the prescribed dimension order, which can in turn create circular channel dependencies and lead to deadlock. FT routing algorithms restore deadlock-free connectivity on the resulting irregular mesh. At wafer scale, however, these algorithms must satisfy four novel requirements that concern efficiency, scalability, and generability:

1. **Memory Efficiency: Per-PE forwarding state must fit in limited SRAM.** Each PE has only tens of kilobytes of local SRAM. The forwarding state for fault tolerance must occupy only a small fraction.
2. **Architecture Efficiency: Per-PE hardware overhead must not grow with mesh size.** Since each PE is area-constrained, any additional hardware for fault tolerance must remain bounded as the mesh scales.
3. **Routing Computation Scalability: Forwarding state computation must scale to a million or more PEs.** The algorithm that computes (offline) per-PE forwarding state must complete in practical time at wafer scale.
4. **Defect Tolerance Generality: Must handle both random and structured defect patterns.** As discussed in §2.2, manufacturing produces both spatially uniform and clustered defects; the algorithm must not assume a particular defect distribution.

2.4 Why existing NoC routing methods fail?

None of existing FT routing algorithms (illustrated in Figure 3) can fully meet the above requirements:

Table-based FT Routing. Table-based approaches precompute a per-PE forwarding table offline. At each hop, the router looks up the destination in its local table to determine the output port. This guarantees $O(1)$ per-hop routing latency but requires storing the table at every PE.

Autonet [29] constructs a spanning tree over all healthy PEs, then classifies every link as *up* (toward the root) or *down* (away from the root). Packets may travel any number of up links followed by any number of down links, but never up after down—this acyclicity guarantees deadlock freedom (Figure 3a). Each PE stores a forwarding table with one entry per destination, yielding $O(N)$ forwarding state per PE. Building these tables requires computing a forwarding entry for each of N PEs at each of N destinations, resulting in $O(N^2)$ offline route computation time. At wafer scale, the $O(N)$ per-PE storage exceeds the SRAM budget, and the tree structure concentrates traffic on links near the root.

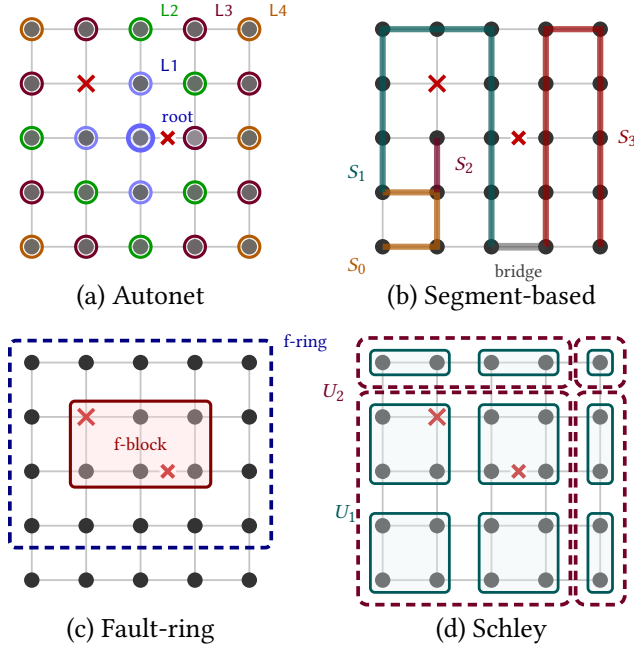


Figure 3. Fault-tolerant routing approaches on a 5×5 mesh with one router defect (X) and one channel defect. (a) Autonet: BFS spanning tree from root; nodes colored by tree depth (L1–L4). (b) Segment-based: segments S_0 – S_3 with a subnet bridge. (c) Fault-ring: rectangular fault block (red) enclosed by a fault ring (dashed). (d) Schley: level-1 units (U_1 , solid) grouped into level-2 units (U_2 , dashed).

Segment-based routing [20] partitions the network into groups of nodes called *segments* and places one bidirectional turn restriction per segment to break cyclic dependencies (Figure 3b). Segments are independent, allowing turn restrictions for deadlock avoidance to be chosen locally within each segment. Like Autonet, the per-PE forwarding state is $O(N)$ and route computation time is $O(N^2)$. Segment-based routing offers better path diversity than Autonet, but the segmentation quality depends on heuristic choices (starting routing, search order). A packet travels along nodes within a segment or crosses to an adjacent segment, with turn restrictions applied at segment transitions to prevent deadlock. Poor segmentation quality can force packets through many segment crossings, resulting in long detours.

Schley [28] groups PEs into hierarchical units at multiple levels (Figure 3d). With h hierarchy levels and $2h$ *virtual channel* (VC) sets – each set an independent class of logical channels (with its own buffers) sharing the physical link – the per-PE routing table reduces to $O(N^{1/h})$ and route computation time to $O(N^{1+1/h} \log N)$. More levels reduce the size of the routing table: at VC=2 ($h=1$), Schley reduces to flat up*/down* with $O(N)$ table entries per PE; at VC=4 ($h=2$), $O(N^{1/2})$; at VC=8 ($h=4$), $O(N^{1/4})$. However, this trades one form of storage for another: each additional VC set brings its

own per-port flit buffers (one per VC in the set), so the buffer area grows linearly with the number of VC sets. Schley thus reduces routing table size at the cost of increased buffer area.

Tableless FT Routing. Tableless approaches store no per-PE forwarding table. Instead, they make routing decisions online based on local fault geometry, eliminating the storage bottleneck. For instance, *Fault-ring routing* [3] constructs *rings* around rectangular fault regions and detours packets along these rings (Figure 3c). The per-PE forwarding state is $O(1)$ – each router only needs to know whether it borders a *fault block* – and the ring construction is $O(F^2)$. This makes fault-ring highly scalable in both storage and computation. However, fault-ring requires defects to be enclosed in *rectangular* fault blocks, which may disable additional healthy PEs to maintain the rectangular shape [3]. The number of disabled healthy PEs thus depends on the defect pattern.

Takeaway. Table-based approaches (Autonet, segment-based, Schley) guarantee 100% routability among healthy PEs, but their per-PE forwarding state and computation cost grow with mesh size, making them infeasible at wafer scale. Tableless routing (fault-ring) is scalable, but trades PE utilization for the rectangular fault block requirement and is therefore sensitive to defect pattern structure. In this work, we propose a scalable routing algorithm for wafer-scale mesh systems that achieves 100% routability among healthy PEs with affordable per-PE memory overhead.

3 System and Defect Model

We model a wafer-scale system as a 2D mesh $M = (V, E)$ of $K \times K$ *processing elements* (PEs), where K is the mesh side length and $N = K^2 = |V|$ is the total PE count. Each PE at coordinate (x, y) consists of two components:

- **Compute unit:** A core with local SRAM, capable of executing arithmetic operations and hosting program state (e.g., weight shards, activations, KV-cache entries).
- **Router:** A crossbar switch co-located with the compute unit. Each router has up to four directional ports (N, S, E, W) plus one local port connecting to its compute unit. The router forwards packets between its local compute unit and neighboring routers. The router typically occupies 10–20% of the PE’s area [9, 13, 23].

A *channel* is a unidirectional link between two adjacent routers. A bidirectional connection consists of two opposing channels. Each channel may support multiple *virtual channels* (VCs) for deadlock avoidance and traffic isolation.

Defect model. We model three types of permanent manufacturing defects:

- **Compute-unit defect:** The compute unit at PE v is non-functional. The co-located router remains operational – v cannot execute computation but can still forward traffic for other PEs.
- **Router defect:** The router at PE v is non-functional. Both computation and forwarding are disabled – v becomes a *faulty node* that blocks all traffic through its location. All channels incident to v are effectively dead. This is the most severe defect type.
- **Channel defect:** A unidirectional channel between two adjacent routers is non-functional. Both endpoint routers remain operational, but packets cannot traverse the link in the failed direction.

Given a defect map $\mathcal{F} = (\mathcal{F}_{\text{compute}}, \mathcal{F}_{\text{router}}, \mathcal{F}_{\text{channel}})$, the set of *routing-available* PEs is $V_R = V \setminus \mathcal{F}_{\text{router}}$, which includes PEs with compute-unit defects that can still forward traffic. The set of *healthy mappable* PEs is $V_H = V_R \setminus \mathcal{F}_{\text{compute}}$. DLR takes the defect map as given: wafer-scale systems such as the WSE already detect and provide it for their repair mechanisms [18]. All routes are computed offline from this map; runtime routing is a pure table lookup. A defect that appears after deployment is handled by a reboot and re-detection.

Key observation. Only router defects and channel defects affect routing topology. Compute-unit defects reduce compute capacity but do not block traffic. The defect count relevant to DLR’s partitioning is therefore $F = |\mathcal{F}_{\text{router}}| + |\mathcal{F}_{\text{channel}}|$, which may be significantly smaller than the total defect count.

4 DLR Design and Implementation

4.1 Design intuitions and overview

At realistic defect densities ($\sim 0.1/\text{cm}^2$), a wafer-scale mesh of over a million PEs contains only tens of router defects (Section 2.2), a miniscule fraction of the entire mesh. The sparsity of the defects can be exploited to partition the mesh into a small number of regular (rectangular) regions that are entirely *defect-free*; we refer to these regions as *defect-free submeshes*. Within each defect-free submesh, the mesh is structurally intact: any deadlock-free routing algorithm (e.g., XY, west-first, negative-first) works optimally without modification. Only traffic crossing a submesh boundary requires fault-tolerant routing.

DLR exploits the defect-free submeshes to create a scalable fault-tolerant deadlock-free routing scheme. The key insight is that the state-of-the-art routing can be used inside the defect-free submeshes, while only the submeshes themselves must be connected with fault-aware routing.

Further, DLR uses *up*/down** routing [29] to provide fault-aware deadlock-free routing across submeshes. Specifically, a spanning tree over the submesh adjacency graph designates one submesh as the root and defines a topological order: the

root receives the highest order, and order decreases outward (§4.3). This order classifies every inter-submesh link as *up* (toward higher-order submeshes) or *down* (toward lower-order submeshes). All traffic starts on **VC set 0**:

- **Intra-submesh traffic** stays entirely on VC set 0, using any deadlock-free routing (e.g., XY) on the intact rectangular submesh. This ensures optimal (i.e., minimal) paths at zero overhead.
- **Inter-submesh traffic** follows a path through the submesh adjacency graph respecting the topological order: ascending first, then descending. While ascending (switching to higher-order submeshes), the packet remains on **VC set 0**. On the first descending link (switching to a lower-order submesh), the packet switches to **VC set 1** and remains there for all subsequent hops – including any intra-submesh legs in the destination submesh. No VC set 1 $\rightarrow$ VC set 0 transition is permitted to ensure deadlock freedom.

Since the adjacency graph has only $O(F)$ nodes (tens, not hundreds of thousands), inter-submesh paths are short. DLR achieves three essential properties:

1. **100% routability**, provided the routing-available PEs form a connected graph.
2. **Deadlock freedom** with only 2 VC sets – just one VC set more than a vanilla mesh.
3. **Minimal intra-submesh paths**: source–destination pairs within the same defect-free submesh achieve minimal hop count, identical to a defect-free mesh. Only pairs crossing submesh boundaries might incur additional hops (§5).

DLR’s routing state is computed offline in three stages (Figure 4): (1) *Submesh Partitioning* (§4.2) identifies maximal defect-free submeshes; (2) *Topological Ordering* (§4.3) builds a spanning tree over the submesh adjacency graph and derives a topological order that classifies every inter-submesh link as up or down; (3) *Route Computation* (§4.4) computes and loads (at boot time) per-PE routing state (submesh ID + forwarding table + VC assignment) and LUT into each PE.

4.2 Submesh Partitioning

The first stage partitions the defective mesh into a set of rectangular defect-free submeshes that collectively cover every routing-available PE.

Defect-driven partitioning algorithm. Standard deadlock-free routing algorithms (XY, west-first, negative-first, odd-even, etc.) require a rectangular mesh topology. Non-rectangular regions break the monotonic channel ordering that these algorithms rely on for acyclic channel dependency graphs. Rectangularity is therefore the minimum structural requirement for zero-overhead intra-submesh routing.

DLR partitions by *cutting at defect locations* rather than at arbitrary midpoints (as in standard recursive coordinate

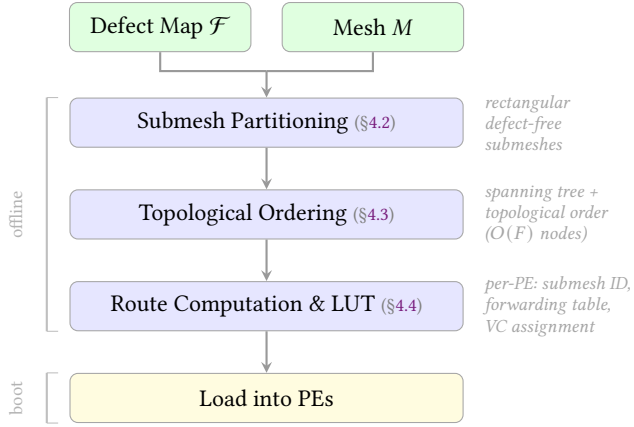


Figure 4. DLR pipeline.

bisection). Given a rectangular region containing defects, the algorithm selects a router defect and cuts the region into up to four sub-rectangles around it, as illustrated in Figure 5b. The cut separates the defect’s column and row from the rest of the region, producing a left column, a bottom block, a right block, and a top cell — each recursively partitioned if it contains further defects. Channel defects split the containing region into two halves along the broken link (Figure 5c). By cutting exactly at defect locations, the algorithm avoids splitting defect-free regions and produces fewer, larger submeshes.

Complexity. Each defect triggers one split into at most 4 sub-rectangles, so the algorithm produces $O(F)$ submeshes. Each recursive call filters the defect list for faults within the current region in $O(F)$ time, yielding $O(F^2)$ total time — negligible since $F \ll N$. (This is reducible to $O(F \log F)$ by pre-sorting the defect list, but at $F \approx 46$ the difference is immaterial.)

4.3 Topological Ordering

The second stage assigns each submesh a numeric ID that determines the routing direction for inter-submesh traffic.

Inter-submesh links. Two submeshes are *adjacent* if a PE on the boundary of one has a healthy channel to a PE on the boundary of the other. These boundary PEs and the channels between them are the *inter-submesh links* — the physical connections through which traffic crosses from one defect-free submesh to another. For example, in Figure 5d, S_5 and S_3 are adjacent because PE (2, 2) in S_5 has a direct channel to PE (2, 1) in S_3 (highlighted in blue). Let $\mathcal{S}$ denote the set of all submeshes produced by the partitioning step; there are at most $4F$ submeshes (each router defect produces up to 4, each channel defect up to 2).

Topological order. DLR constructs a spanning tree over the submeshes and assigns order in *reverse level order*: one

submesh is designated as the root and receives the highest order; its neighbors receive the next-highest; and so on outward, with leaves receiving the lowest. This numbering induces a total order on submeshes: every inter-submesh link is classified as *up* (toward a higher-order submesh) or *down* (toward a lower-order submesh). The routing algorithm (§4.4) uses this classification to ensure deadlock-freedom.

The spanning tree guarantees that for any two submeshes, a valid path exists that first ascends (follows increasing order) and then descends (follows decreasing order) — this is the path through their lowest common ancestor in the tree.

Root selection. The topological order and deadlock-freedom guarantees hold for any choice of root. In practice, we select the submesh containing the mesh center coordinate, which tends to reduce tree depth and thus inter-submesh path lengths. This choice is a heuristic, not a requirement — depending on the defect distribution, other submeshes may be equally good candidates. Figure 5d illustrates the resulting topological order with S_5 as root.

4.4 Routing Algorithm

At each PE, the routing decision for a packet with destination d depends on whether d is in the same submesh:

1. **Same submesh** (submesh(src) = submesh(d)): Route using the intra-submesh algorithm (e.g., XY) on either of DLR’s two VC sets. This is the common case.
2. **Different submesh:** The packet traverses submeshes along a path that respects the *topological order*: it first ascends (moves to higher-order submeshes), then descends (moves to lower-order submeshes), with no re-ascend after the first descent. The path may use any inter-submesh link in the adjacency graph — not only spanning tree edges — as long as the up*/down* constraint holds. Each hop consists of: (a) Route within the current submesh to the gateway PE for the next submesh, and (b) Cross the submesh boundary via the gateway channel.
3. **VC assignment:** The packet starts on **VC set 0**. While ascending, it remains on VC set 0. On the first descending link (next submesh has a lower order), it switches to **VC set 1** and stays on VC set 1 for all remaining hops — including the final intra-submesh leg to d . No VC set 1 $\rightarrow$ VC set 0 transition is permitted to ensure deadlock freedom.

Path computation. Inter-submesh paths are computed offline via Dijkstra on the submesh adjacency graph, subject to the up*/down* constraint (no ascending after any descent). The edge weight between two adjacent submeshes reflects the physical hop count of the intra-submesh leg from the entry gateway to the exit gateway — i.e., the cost of traversing the submesh interior. This ensures Dijkstra minimizes the

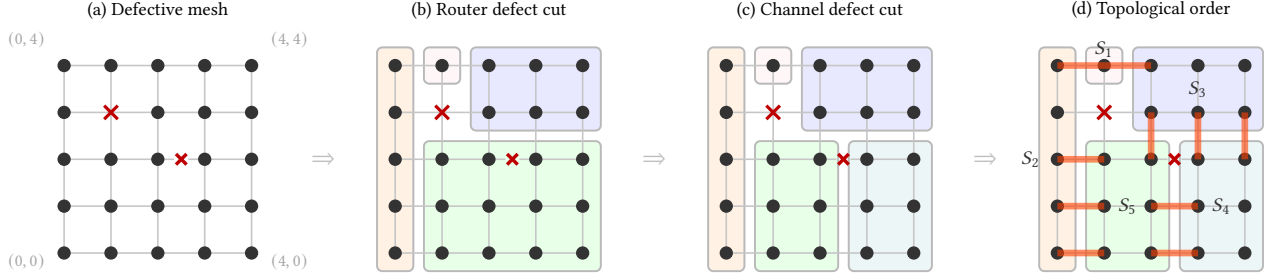


Figure 5. Submesh partitioning and indexing on a 5×5 mesh with one router defect ($\times$) and one channel defect (small $\times$ on broken link). (a) Defective mesh. (b) Router defect splits the mesh into four regions. (c) Channel defect further splits one region. (d) Topological order with inter-submesh links highlighted (orange): the root submesh (S_5) receives the highest order; order decreases outward. Each highlighted link connects boundary PEs of adjacent submeshes.

total physical hop count of the end-to-end path, not just the number of submesh crossings.

For example, in Figure 5d, consider a packet from PE (1, 4) in S_1 to PE (4, 0) in S_4 . The spanning tree path ascends to the root S_5 and then descends into S_4 . However, the adjacency graph also offers $S_1 \rightarrow S_3 \rightarrow S_4$, which bypasses the root and may take fewer total hops. Dijkstra selects the path with the lowest total hop count among all valid up*/down* paths.

Per-PE routing state. Each PE stores: (1) *Own submesh bounding box*: used to determine whether a destination is intra- or inter-submesh. If the destination coordinates fall within this box, the packet is routed using the intra-submesh algorithm (e.g., XY) without any table lookup. Otherwise, the forwarding table is consulted, and (2) *Forwarding table*: one entry per destination submesh. Each entry maps a destination submesh’s bounding box to an output port and VC set. This is sufficient because all destinations within the same submesh share the same next-hop submesh (determined by the submesh-level Dijkstra path) — only the final intra-submesh leg differs, and that requires no table entry. The forwarding table therefore has one entry per submesh. The partitioning produces at most $4F$ submeshes (§4.3), so the table scales in proportion to the defect count, not the mesh size. Note that, on an intact wafer ($F = 0$), no forwarding table is needed — every PE uses intra-submesh routing.

4.5 Correctness and Analysis

We have proved three properties for DLR: deadlock-freedom, routability, and a storage bound, where the latter two are omitted for page limit.

Theorem 4.1. *DLR is deadlock-free using only 2 VC sets.*

Proof. Deadlock occurs when packets form a circular chain of channel dependencies — each packet holds a channel and waits for the next. We show no such cycle can exist.

Case 1: within a single submesh. Within each defect-free rectangular submesh, the intra-submesh routing algorithm (e.g., XY) is deadlock-free on either VC set — a standard

result for dimension-order routing on rectangular meshes. Channels in different submeshes are physically disjoint, so dimension changes across submeshes (e.g., Y in S_A then X in S_B) cannot create cross-submesh intra-routing cycles.

Case 2: across submeshes. Consider a hypothetical cycle $S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_k \rightarrow S_1$ visiting k submeshes. On VC set 0, every inter-submesh link must be ascending, so $\text{ord}(S_1) < \text{ord}(S_2) < \dots < \text{ord}(S_k) < \text{ord}(S_1)$, implying $\text{ord}(S_1) < \text{ord}(S_1)$ — a contradiction. Similarly, on VC set 1, every link must be descending, so $\text{ord}(S_1) > \text{ord}(S_2) > \dots > \text{ord}(S_k) > \text{ord}(S_1)$ — also a contradiction. A cycle mixing ascending and descending links would require a VC set $1 \rightarrow$ VC set 0 transition, which the up*/down* rule forbids. Therefore, no inter-submesh cycle can exist for any cycle length k .

Since neither intra-submesh nor inter-submesh cycles are possible, and no VC set $1 \rightarrow$ VC set 0 transition is permitted, the channel dependency graph is acyclic. By the Dally–Seitz theorem [8], DLR is deadlock-free. $\square$

5 Evaluation

The evaluation proceeds in four aspects:

1. **Scalability** (§5.1): per-router storage and route computation cost across all baselines. Narrows the field to algorithms that can run at wafer scale.
2. **Routing quality** (§5.2): PE utilization (DLR vs. fault-ringing) and path quality (DLR vs. Schley (VC=8)) under uniform random, broadcast, and tree all-reduce traffic.
3. **Simulation** (§5.3): flow-level simulation of contention, decomposing latency into propagation, transmission, and queuing.
4. **Hardware cost** (§5.4): router area and power from RTL synthesis; sizing the routing-table capacity from the per-wafer defect-count distribution, and the fall-back when a wafer exceeds it.

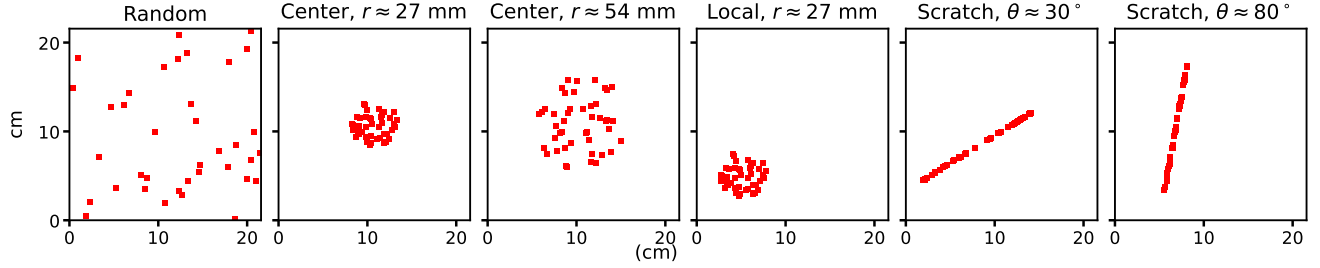


Figure 6. Six defect patterns on a 256×256 mesh at $D_0 = 0.1/\text{cm}^2$. Center and Local place defects uniformly within a disc of radius r , at the chip center and at a random location respectively; Scratch places them along a line at angle θ .

DLR implementation and baselines. All routing algorithms are implemented in Rust. Build time measures routing-table construction, which must complete before the chip can route traffic. Hardware overhead is estimated by synthesizing a baseline 5-port mesh router and each fault-tolerance addition using Yosys with FreePDK45 (45nm), with power estimated by OpenSTA. We compare DLR against various state-of-the-art FT routing baselines (§2.4) covering both table-based and table-less: *Autonet* [29], *Segment-based* [20], *Schley* [28] and *Fault-ring* [3].

Defect patterns and system scales. We model various defect patterns. A defect can be a *router defect*: the defective PE can neither compute nor forward traffic (§3). We evaluate six defect patterns (Figure 6) at $D_0 = 0.1, 0.5$, and 1.0 defects/ cm^2 : Random (Poisson), Center ($r \approx 27$ mm, $r \approx 54$ mm), Local ($r \approx 27$ mm), and Scratch ($\theta \approx 30^\circ$, $\theta \approx 80^\circ$). Edge-ring is omitted as wafer-scale chips use an inscribed square.

In all experiments, we evaluate DLR and the baselines across wafer sizes ranging from $K = 32$ to $K = 1024$, where varying K changes the number of PEs and $K = 1024$ corresponds to roughly one million PEs. Due to page limits, we omit part of the full results and report only the cases for $K = 128$ and $K = 1024$ unless otherwise specified.

Implementation and simulator validation. Our DLR and baseline implementation and simulator have been validated against the official cycle-accurate Cerebras simulator (Cerebras SDK version 1.4). Our multi-hop communication latency measurement and modelling (cycle-level) is further cross validated on the real Cerebras WSE-3 accelerator. This gives us confidence that our modeling of the PE and NoC architectures in principle reflects real wafer-scale accelerators.

5.1 Scalability

We evaluate scalability along two axes: (1) per-router storage – does the forwarding state fit in on-chip SRAM? – and (2) route computation cost – can routing tables be built offline in practical time and memory? Table 1 reports the results across all baselines. Like Autonet, DLR uses up*/down* routing on a spanning tree – but over $O(F)$ submeshes (typically

tens) rather than all N PEs (up to 10^6). This structural difference drives all scalability results below: at $D_0 = 0.1/\text{cm}^2$, $F \approx 46$ while $N = 10^6$ – a $\sim 20,000\times$ gap that DLR exploits at every stage of route computation.

The data reveals that scaling to wafer-scale meshes acts as a *two-stage filter*: algorithms must clear both a storage wall and a computation wall, and most existing approaches fail at one or the other.

Storage wall: most table-based algorithms exceed SRAM at wafer scale. At $K=1,024$ ($N = 10^6$ PEs), Autonet requires 1.9 MB per router, segment-based 384 KB, and Schley (VC=2) 512 KB (Table 1) – all exceeding the tens-of-KB per-PE SRAM budget by orders of magnitude, because their $O(N)$ storage grows linearly with mesh size. Schley’s hierarchical scheme implies a storage trade-off: VC=4 reduces to $O(N^{1/2})$, VC=8 to $O(N^{1/4})$, but by VC=8 the VC buffers outweigh the table several times over. DLR sidesteps this tradeoff entirely: its $O(F \log N)$ storage depends on defect count F , not mesh size N . At $D_0 = 0.1/\text{cm}^2$, this yields **99 B** per router – comparable to Schley (VC=8) but with $4\times$ fewer VC sets. Even at $D_0 = 1.0/\text{cm}^2$, DLR stays under 1 KB.

Offline aglorithmic complexity. Table 1 also reports route computation (offline) in time and peak memory, measured on an AMD EPYC 7453 capped at 64 GB of RAM. Route computation is the process that produces per-PE forwarding tables, which are loaded at boot time. At $K=128$, segment-based and Schley (VC=2) take 1,734 and 2,166 s respectively, while DLR completes in 0.005 s and fault-ring in 0.003 s. Schley (VC=2) consumes **28.4 GB** of peak memory at $K=128$ – already nearly half the machine’s RAM at only 16K PEs – and is infeasible at $K=1,024$ (OOM). Schley (VC=4) also exhausts memory before $K=1,024$. At wafer scale ($K=1,024$), DLR completes in under 1 s with 11–81 MB of memory depending on D_0 ; fault-ring finishes in 0.004 s. Schley (VC=8) is feasible but requires 58 s and 4.3 GB.

Three algorithms survive, with fundamentally different tradeoffs. The two-stage filter eliminates four of the

Table 1. Scalability at $K = 128$ and $1,024$ ($N = K^2$). *Routing state* is the VC buffers and forwarding table each router holds at run time. Buffer bytes are given at flit width $w=64$, with the VC-set count in parentheses, one VC per set and two slots per VC. *Offline route computation* is paid once, before the chip runs; $\times$ marks a run that did not finish and OOM one that ran out of memory. Fault-ring needs no forwarding table, computing its routes online from fault-block geometry.

Algorithm	Routing state (per router)				Offline route computation				
	VC buffer (sets)	Forwarding table			Time (s)			Peak memory	
		Complexity	$K=128$	$K=1024$	Complexity	$K=128$	$K=1024$	$K=128$	$K=1024$
Autonet [29]	80 B (1)	$O(N)$	30 KB	1.9 MB	$O(N^2)$	37.5	$\times$	5.6 MB	OOM
Segment-based [20]	80 B (1)	$O(N)$	6 KB	384 KB	$O(N^2)$	1,734	$\times$	11.9 MB	OOM
Schley [28] (VC=2)	160 B (2)	$O(N)$	8 KB	512 KB	$O(N^2 \log N)$	2,166	$\times$	28.4 GB	OOM
Schley [28] (VC=4)	320 B (4)	$O(N^{1/2})$	159 B	1.2 KB	$O(N^{3/2} \log N)$	4.3	$\times$	223 MB	OOM
Schley [28] (VC=8)	640 B (8)	$O(N^{1/4})$	34 B	94 B	$O(N^{5/4} \log N)$	0.36	58.0	40.5 MB	4.3 GB
Fault-ring [3]	160 B (2)	$O(1)$	—	—	$O(F^2)$	0.003	0.004	316 KB	80.8 MB
DLR ($D_0=0.1$)	160 B (2)	$O(F \log N)$	72 B	99 B	$O(N + F^2 \log F)$	0.005	0.077	1.3 MB	10.9 MB
DLR ($D_0=0.5$)	160 B (2)	$O(F \log N)$	359 B	494 B	$O(N + F^2 \log F)$	0.005	0.428	1.3 MB	39.5 MB
DLR ($D_0=1.0$)	160 B (2)	$O(F \log N)$	718 B	989 B	$O(N + F^2 \log F)$	0.005	0.814	1.3 MB	80.5 MB

seven evaluated algorithms: Autonet, segment-based, Schley (VC=2), and Schley (VC=4) all fail at $K=1,024$, either exceeding per-PE SRAM or exhausting compute memory. Three survive to wafer scale: (1) **Fault-ring**: tableless, fastest to compute – but may disable healthy PEs; (2) **Schley (VC=8)**: fully scalable, but requires $4\times$ the VC sets and $750\times$ ($D_0=0.1$) to $70\times$ ($D_0=1.0$) the computation time of DLR; and (3) **DLR**: sub-KB storage, sub-second computation, 2 VC sets.

The key takeaway is that DLR is the only algorithm that simultaneously achieves low storage, low computation cost, and few VC sets – all because its complexity scales with defect count F , not mesh size N . We evaluate routing quality among these three survivors in §5.2.

5.2 Routing Quality

We evaluate routing quality along two axes: *PE utilization* (DLR vs. fault-ring) and *path quality* (DLR vs. Schley (VC=8)). All experiments use six defect patterns (Figure 6) at $D_0 = 0.1$, 0.5, and 1.0 defects/cm², representing typical defect patterns in wafer manufacturing processes.

PE utilization: DLR vs. fault-ring. DLR and Schley (VC=8) achieve **100% PE utilization** across all configurations: every healthy PE participates in both computation and routing. Fault-ring, by contrast, disables healthy PEs to form rectangular fault blocks, and its PE loss exhibits two problematic characteristics revealed by Table 2.

Defect pattern sensitivity. At a fixed defect density, fault-ring’s PE loss varies dramatically with spatial distribution. At $K=128$ and $D_0=0.5/\text{cm}^2$, random defects incur only 1.2% average PE loss, yet center-concentrated defects ($r\approx 54$ mm) raise this to **23%** – a $19\times$ difference for the *same* number of defects. At wafer scale ($K=1,024$), the *average* PE loss drops below 0.1% for most patterns because defects become

Table 2. Fault-ring healthy PE loss (%) across six defect patterns, two defect densities, and two mesh sizes. DLR and Schley (VC=8) achieve 0% loss in all configurations. At $D_0=0.1$, loss is below 2% for all patterns (omitted).

Pattern		$D_0 = 0.5$		$D_0 = 1.0$	
		$K=128$	1024	$K=128$	1024
Random	avg	1.2	<0.1	4.6	<0.1
	worst	10	<0.1	19	<0.1
Center, $r\approx 27$ mm	avg	5.4	<0.1	4.3	<0.1
	worst	25	0.1	25	0.5
Center, $r\approx 54$ mm	avg	23	<0.1	22	<0.1
	worst	49	0.2	48	0.2
Local, $r\approx 27$ mm	avg	5.4	<0.1	4.2	<0.1
	worst	24	<0.1	24	0.2
Scratch, $\theta\approx 30^\circ$	avg	19	0.1	19	0.6
	worst	55	2.6	55	5.0
Scratch, $\theta\approx 80^\circ$	avg	7.0	<0.1	7.0	0.1
	worst	63	1.0	63	2.7

sparse relative to mesh size. However, scratch defects remain an outlier: at $D_0=1.0/\text{cm}^2$, scratch ($\theta\approx 30^\circ$) still causes 0.6% average loss and, more critically, up to **5.0%** worst-case loss on a single row or column.

Average PE loss can understate the loss on individual rows and columns. We therefore also report the worst-case loss for a single row or column (*worst* in Table 2). Even at $K=1,024$ where averages appear benign, fault-ring’s worst-case per-row loss reaches 2.6–5.0% under scratch patterns. At $K=128$, this tail is severe: under scratch ($\theta\approx 80^\circ$), fault-ring loses up to **63%** of healthy PEs in the worst row. Since manufacturers

Table 3. DLR/ Schley (VC=8) average hop count ratio (the lower the better) across traffic patterns, defect patterns, and mesh sizes. Values below 1.0 indicate DLR uses fewer hops; values above 1.0 indicate longer paths. DLR uses 2 VC sets; Schley uses 8 VC sets (4×). Broadcast is the center-rooted forwarding chain, averaged over row- and column-direction.

Pattern	D_0	Nearest Neighbor		Broadcast		Tree Reduce (col)		Tree Reduce (row)		Uniform Random	
		$K=128$	$K=1024$	$K=128$	$K=1024$	$K=128$	$K=1024$	$K=128$	$K=1024$	$K=128$	$K=1024$
Random	0.1	1.01	1.00	1.00	1.00	1.00	1.00	3.47	3.52	1.68	1.60
	0.5	1.04	1.00	1.04	1.00	1.00	1.00	6.24	22.4	1.77	1.91
	1.0	1.10	1.00	1.08	1.00	0.99	1.00	7.24	20.3	1.86	1.92
Center, $r \approx 27$ mm	0.1	1.01	1.00	1.01	1.00	1.00	1.00	2.02	3.70	1.13	1.12
	0.5	1.03	1.00	1.03	1.00	1.02	1.00	1.58	9.04	1.14	1.17
	1.0	1.00	1.00	0.99	1.01	0.99	1.05	0.99	11.5	1.14	1.18
Center, $r \approx 54$ mm	0.1	1.00	1.00	1.00	1.00	1.00	1.00	2.42	4.01	1.23	1.25
	0.5	1.03	1.00	1.04	1.00	1.00	1.00	2.97	11.9	1.24	1.33
	1.0	1.07	1.00	1.05	1.00	1.03	1.00	2.53	16.5	1.26	1.34
Local, $r \approx 27$ mm	0.1	1.00	1.00	1.00	1.00	1.00	1.00	3.46	4.55	1.31	1.32
	0.5	1.05	1.00	1.05	1.00	1.03	1.00	2.29	11.5	1.24	1.41
	1.0	1.00	1.00	0.99	1.07	1.00	1.01	1.65	13.6	1.20	1.42
Scratch, $\theta \approx 30^\circ$	0.1	1.00	1.00	1.00	1.00	1.08	1.18	1.87	1.09	1.22	1.09
	0.5	1.10	1.00	1.10	1.00	1.06	1.00	1.09	7.97	1.31	1.29
	1.0	1.10	1.00	1.10	1.00	1.06	1.01	1.09	11.4	1.31	1.31
Scratch, $\theta \approx 80^\circ$	0.1	1.01	1.00	1.02	1.00	1.00	1.04	1.20	1.56	1.11	1.19
	0.5	1.01	1.00	1.01	1.00	1.01	1.15	1.58	1.44	1.19	1.15
	1.0	1.01	1.00	1.01	1.00	1.01	1.00	1.58	1.20	1.19	1.14

cannot control which defect pattern a wafer exhibits, the fault-ring's PE loss is unpredictable at deployment time.

Fault-ring's PE loss is not merely a function of defect *count* but of defect *geometry* – a quantity that varies from wafer to wafer and cannot be controlled. As a key takeaway, DLR eliminates this uncertainty entirely: regardless of defect pattern, density, or mesh size, every healthy PE is utilized, enabling deterministic capacity planning with zero waste.

Path quality: DLR vs. Schley (VC=8). Both DLR and Schley (VC=8) achieve 100% PE utilization, enabling a direct comparison on identical PE sets. The key hardware difference is the number of VC sets: DLR uses 2 VC sets; Schley requires 4× as many. Table 3 reports the ratio of DLR's average hop count to Schley's across five traffic patterns, six defect distributions, and two mesh sizes. A ratio of 1.0 means identical path length; below 1.0 means DLR routes shorter. Section 5.3 evaluates the end-to-end latency of the same routes under contention.

5.3 Simulation

We evaluate each pattern's end-to-end latency under contention at $K=1,024$, the wafer-scale mesh DLR targets. To reach this scale, we use an analytical flow-level simulation with max-min fair bandwidth sharing: flows on a shared link split its bandwidth equally. Each pattern of Table 3 is simulated over the routes each scheme computes, with one

16B message per flow. A collective's flows are dependency-ordered: each starts only after the flows feeding it complete. Uniform random keeps 1,024 messages in flight, one per mesh row on average, drawn from 16,384 sampled source-destination pairs. Per-flow latency decomposes into three components: **propagation** – the per-hop latency accumulated along the route; **transmission** – the message's serialization time; and **queuing** – the time spent waiting. Propagation and transmission follow the WSE-3's specifications.

Figure 7 shows the resulting decomposition against a defect-free mesh (Ideal). Broadcast and tree reduce, whose flows are dependency-chained, report each collective's makespan (completion time), decomposed along its critical path and averaged over the concurrent collectives. Nearest-neighbor and uniform random, whose flows are independent, report mean per-flow latency.

Two facts explain both the hop ratios and the latencies. First, a path could take extra hops only when it crosses sub-mesh boundaries; traffic that stays inside one submesh routes identically to a fault-free mesh. Second, wafer-scale kernels consist mostly of local traffic – stencil and nearest-neighbor exchanges between adjacent PEs, and broadcasts or reductions that run along single rows or columns [4, 5, 12, 19] – and local traffic rarely crosses a boundary. Three observations follow.

Observation 1: DLR matches the defect-free ideal on most patterns wafer-scale workloads use, in both hops

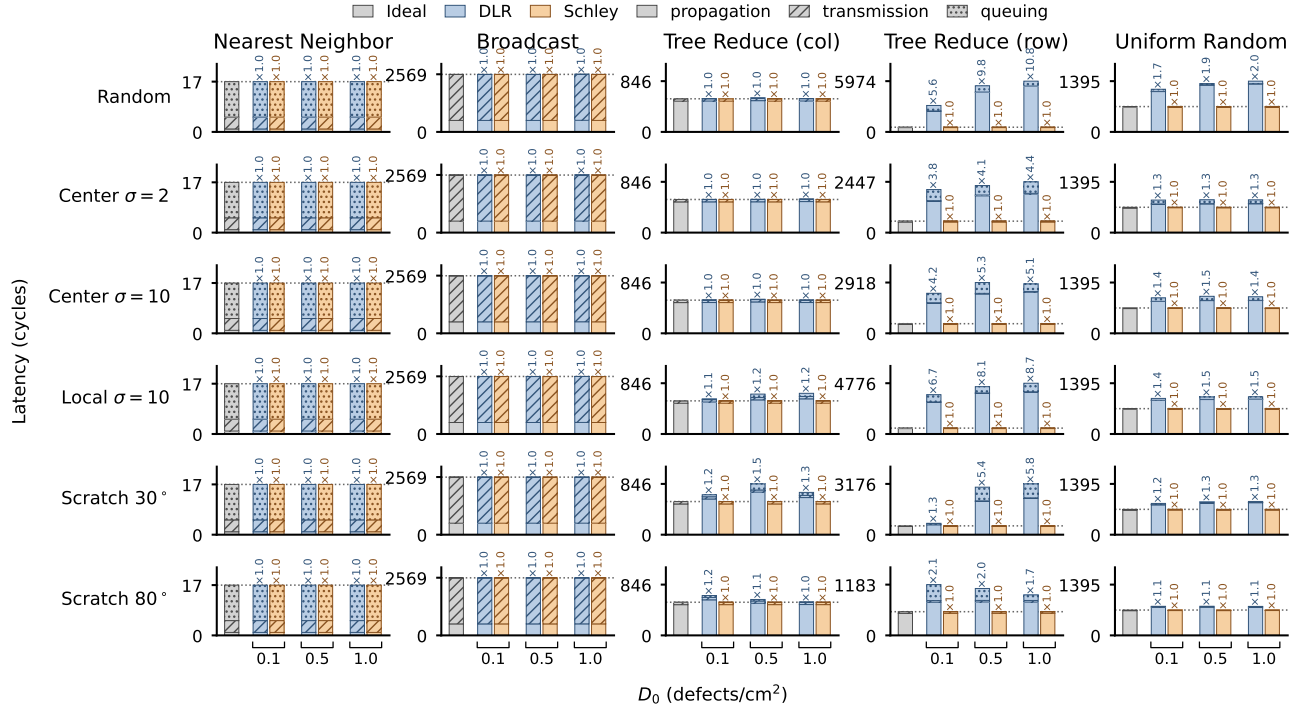


Figure 7. Latency decomposition under contention at $K=1,024$ with 16B messages. Broadcast and tree reduce report each collective’s makespan along its critical path, averaged over the concurrent collectives; nearest-neighbor and uniform random report mean per-flow latency; each bar is labeled with its ratio to the ideal. In each panel, the leftmost bar is the defect-free ideal; within each braced pair, the left bar is DLR and the right is Schley.

and latency. For nearest-neighbor and broadcast, DLR matches Schley at the hop level: ratios of 1.00–1.10 $\times$ and 0.99–1.10 $\times$, and 1.00 in nearly every $K=1,024$ entry (Table 3), because both patterns stay almost entirely intra-submesh. The simulation shows that this parity holds under contention. Nearest-neighbor latency is comparable for DLR, Schley, and the ideal in every configuration; most of it is queuing: each PE’s four incoming flows arrive nearly simultaneously and contend for its single local port to eject. Broadcast forwards along a chain (each participant receives the message and re-sends it to its next neighbor). Most flows stay inside one submesh. At most one flow per node is in flight, so queuing is ≈ 0 and the makespan matches the ideal at every density.

Observation 2: latency depends on the defect pattern and partitioning policy. The partitioner cuts tight around each defect (§4.2), so every defect could leave a submesh strip just **one PE wide** (e.g., S_1 in Figure 5d). If many such strips fall in the same direction – for instance, along adjacent columns, due to the $\theta \approx 30^\circ$ scratch in Figure 6 – traffic routing row-wise will be forced to cross one such strip after another, potentially resulting in non-minimal routes depending on traversal order of the submeshes using up/down routing. In contrast, traffic running along a column in this scenario will

cross far fewer submesh boundaries, naturally leading to minimal or near-minimal paths.

Tree reduce demonstrates this. Tree reduce proceeds in stages, whereby in stage s , PEs 2^{s-1} hops apart combine their partial results [12], so the communication distance doubles from one stage to the next. The later stages, which have the longest communication distances, therefore dominate the critical path. A row-direction reduce crosses these one-PE-wide strips one after another, while a column-direction reduce runs along them. In hop count, the row direction reaches 22.4 $\times$ of Schley, whereas the column direction is at most 1.18 $\times$ of Schley (Table 3). The makespans follow, with the row direction at 6.7 $\times$ the ideal at $D_0=0.1$ and 10.8 $\times$ at $D_0=1.0$, against 1.2 $\times$ for the column direction at $D_0=0.1$ (Figure 7).

Note that our partitioner cuts each defect’s column before its row, leaving columnar strips that affect row-wise routing; cutting rows first would leave row-wise strips and disrupt columnar routing instead. The key takeaway is that the asymmetry is a partitioning artifact, not a property of the routing scheme.

Observation 3: traffic without locality cannot always take the shortest path. Communication between distant

PEs often crosses several submesh boundaries. These crossings must follow the same ascend-then-descend order, so the path can take extra hops detouring through additional submeshes. Uniform random traffic consists largely of such communication. The extra hops depend heavily on defect geometry: clustered defect patterns (center, local, scratch) stay within $1.09\text{--}1.42\times$ of Schley, while purely random defects fragment the mesh into more submeshes and push the ratio to $1.60\text{--}1.92\times$ in Table 3. The latencies follow the same trend, as shown in Figure 7. Propagation dominates the latency in every configuration. Inter-submesh flows concentrate on the links near the highest-order submeshes, and at higher loads random traffic would congest these links. In practice, the workloads that run on wafer-scale meshes are locality-aware [4, 12, 19, 26, 37] and rarely generate such traffic.

Takeaway and future work. The simulation confirms the hop-level results under contention: DLR stays close to the defect-free ideal on nearest-neighbor, broadcast, and reduce – the patterns most critical to wafer-scale workloads – while using a quarter of Schley’s VC sets. The one exception, the row-direction tree, is a partitioning artifact rather than a routing limitation. Two approaches could address it: (1) a balance-aware partitioner, and (2) partition-aware relaying, which ejects and re-injects at boundary PEs to split multi-submesh paths [32]. We leave both as future work.

5.4 Hardware Cost

We quantify the silicon cost of DLR’s routing in two steps. First, we size DLR’s routing table based on the per-wafer defect-count distribution. Following that, we compare the total storage required for the DLR router and Schley (VC=8), the baseline with the best routing quality (§5.2, §5.3), against a vanilla XY router (Table 4).

Table capacity. The routing-table capacity must be fixed at design time, but the actual defect count is not known then: it varies from wafer to wafer around the mean $D_0 \times A_{\text{chip}}$ (§2.2). Purely random defects are modeled by a Poisson distribution, and spatially clustered defects by the negative binomial model with clustering parameter α [6, 30]; we set $\alpha = 2$ as a representative clustering strength, following ITRS [14]. Figure 8 plots the resulting distributions at $D_0 = 0.1/\text{cm}^2$: the mean is 46.5 defects, and clustering raises the p95 to 112 ($\alpha=2$) – still only about a hundred defects.

Not every defect consumes table entries – a defective PE with an intact router and links does not affect routability. Routers occupy only 10–20% of PE area [9, 13, 23], so under uniformly distributed defects only 10–20% disable a router (§3), each adding at most four routing table entries, one per sub-rectangle the cut produces (§4.2). Even a p95 wafer (112 defects, Figure 8) thus needs at most $112 \times 0.2 \times 4 \approx 90$ entries for router defects; we provision 128, the smallest power of two covering this need. A channel defect costs two entries

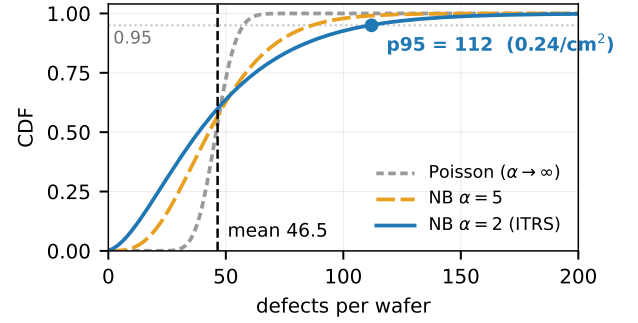


Figure 8. Per-wafer defect-count CDF at $D_0=0.1/\text{cm}^2$. Defect clustering widens the tail and the p95 reaches 112 at $\alpha=2$.

Table 4. DLR router overhead relative to a vanilla XY router as a function of flit width w . Each block’s header row is the total ratio relative to the vanilla router; the rows below break down each component’s share. DLR’s *index-to-port* LUT and Schley (VC=8)’s hold 128 entries each (§5.4); DLR’s *submesh-to-index* table is in the PE’s SRAM and does not require dedicated storage.

(a) Area				
	$w=32$	$w=64$	$w=128$	$w=256$
DLR router	1.77×	1.54×	1.39×	1.31×
Logic	12%	8%	5%	3%
Crossbar	22%	29%	35%	39%
VC buffers	32%	40%	46%	50%
LUT	34%	23%	14%	8%
Schley (VC=8) router	3.76×	3.50×	3.35×	3.26×
Logic	5%	3%	2%	1%
Crossbar	10%	13%	15%	16%
VC buffers	61%	69%	75%	78%
LUT	24%	15%	8%	5%
(b) Power				
	$w=32$	$w=64$	$w=128$	$w=256$
DLR router	1.23×	1.23×	1.19×	1.19×
Logic	27%	16%	9%	4%
Crossbar	7%	9%	11%	13%
VC buffers	61%	72%	79%	82%
LUT	5%	3%	1%	1%
Schley (VC=8) router	3.09×	3.28×	3.40×	3.36×
Logic	11%	6%	3%	2%
Crossbar	3%	3%	4%	4%
VC buffers	83%	89%	92%	94%
LUT	3%	2%	1%	≈0%

(§4.2), so the remaining capacity of the 128-entry LUT can accommodate about 20 channel defects.

Should a wafer’s defects nonetheless exceed the provisioned table capacity, DLR degrades gracefully rather than

forcing the wafer to be discarded. The excess defects are disabled with the row/column mechanism proposed for today’s wafer-scale systems [18], and DLR routes around the remaining defects as usual; §7 discusses this composition. For a $1,024 \times 1,024$ mesh, Schley (VC=8) needs 125 entries, so Table 4 synthesizes both routers’ tables at 128 entries.

Router implementations. The vanilla router consists of a 5-port crossbar, an XY arbiter, five input buffers, and a switch allocator. The DLR router doubles the input buffers to two VC sets. We assume one VC per set and two buffer slots per VC at each port, enough to hold one flit while the other is in flight. More slots, or more VCs per set (e.g., for protocol deadlock avoidance), would scale the buffers of DLR and Schley (VC=8) alike.

A direct implementation of the per-PE routing state (§4.4) needs a content-addressable memory (CAM) to match the bounding boxes at every hop, which is costly in both area and power (Table 4). Instead, DLR splits that state into two tables, a *submesh-to-index* table in each PE’s SRAM and an *index-to-port* LUT in the router. At system boot time, the submeshes are indexed and both tables are filled. When routing a message, the sender queries the submesh-to-index table once, finding the submesh whose bounding box contains the destination, and carries that index in the message header. Each router then reads its LUT at the index encoded in the table, so no bounding-box matching is needed.

For storage efficiency, DLR also optimizes the buffer organization. Rather than giving each VC set its own two slots, each port has one private slot per set plus one slot the two sets share [22, 31]; the sets contend for the shared slot only when both private ones are occupied. We apply the same optimization to the Schley router, allowing every pair of VC sets to share a slot.

Area and power. Table 4 reports router area and power relative to the vanilla router across flit widths. Storage bits are counted at SRAM bit-cell density; power is measured at 1 GHz and 10% input activity, and the routing table is written once at configuration. The shares within each router sum to 100%. Relative to the vanilla router, DLR costs $1.77\times$ the area and $1.23\times$ the power at $w=32$, falling to $1.31\times$ and $1.19\times$ at $w=256$, while Schley (VC=8) costs $3.76\times$ to $3.26\times$ the area and $3.09\times$ to $3.40\times$ the power. Since routers typically occupy 10–20% of PE area, DLR adds at most 15% of a PE, whereas Schley adds up to 55%.

Two components account for the difference between DLR and Schley: the LUT and the VC buffers. For the LUT, each entry indicates (1) an output port, which requires 3 bits in a 5-ported router; and (2) a VC set for the next hop, requiring 3 bits for Schley with 8 VC sets and 1 bit for DLR with 2 sets. Schley’s LUT entries are therefore 6 bits to DLR’s 4, making Schley’s LUT $1.5\times$ the area. For the VC buffers, area scales with the number of sets, so Schley’s are $4\times$ DLR’s. They dominate Schley’s router, taking 61–78% of its area and

83–94% of its power, against 32–50% and 61–82% for DLR. Both shares rise with the flit width, and more slots per port would raise them further, at $4\times$ the cost for Schley.

Putting everything together, we can conclude that on most patterns characteristic of wafer-scale workloads, DLR is competitive with Schley’s path lengths and latency at under **half** its router area and a **third** of its power.

6 Related Work

Table compression. Region-based routing [11] groups destinations into rectangular regions sharing the same output port, reducing table entries per router. uLBDR [27] and d2-LBDR [2] encode forwarding logic into compact per-port registers. These are compression techniques applied on top of existing routing algorithms—they reduce storage but do not change the underlying $O(N^2)$ route computation, which remains infeasible at wafer scale. DLR achieves $O(F \log N)$ storage natively.

Higher-dimensional meshes. Xiang et al. [35] target three-dimensional meshes, routing adaptively through the x – y and y – z planes to stay deadlock-free with 2 VC sets, based on a minimum-connected-component (MCC) fault model that marks the healthy nodes around a fault as unsafe to route through. DLR targets the two-dimensional mesh of a wafer-scale chip, where every healthy PE stays routable.

Dynamic reconfiguration. Deadlock-free dynamic reconfiguration [10, 24], epoch-based schemes [21], runtime routing reconfiguration [15], and ARIADNE [1] address runtime fault recovery by managing transitions between routing configurations. ImmuneNet [25] reconfigures at runtime around any faults that leave the network connected, reserving a safe virtual network in which deadlock cannot occur. DLR is designed for offline computation at boot time; extending it with incremental updates for runtime faults is future work.

7 Discussion

Current limitations. DLR has several current limitations. First, routing quality is sensitive to the partitioning policy. Our current policy cuts along each router defect’s row and column, which favors traffic aligned with the cuts; §5.3 shows the resulting row–column asymmetry. A policy that balances the two directions, or that accounts for the collectives a wafer-scale workload actually runs, remains open. Second, our evaluation shows that DLR scales well across increasing defect densities and a range of defect patterns. Characterizing its worst-case and failure-case behavior under a broader set of patterns requires deeper engagement with frontier wafer-scale system designers.

Mapping and routing. Mapping and routing can in principle be coupled. In this paper, however, we assume that the mapping is provided by the application. This is consistent with today’s wafer-scale systems. For example, WaferLLM

empirically determines how LLM layers are mapped onto physical cores; DLR complements such empirical mapping with the routing support. We leave the joint optimization of mapping and routing to future work.

Composability with industrial repair mechanisms.

Production wafer-scale systems already have defect-repair mechanisms. Cerebras' patent [18] describes two options: physical redundancy, which provisions spare rows of PEs and reconstructs the logical mesh through bypass links, and disabling of the entire rows and columns that contain defects. Both sacrifice healthy PEs. DLR composes with these mechanisms rather than competing: DLR absorbs the expected dozens of defects with zero PE loss, while row/column disabling remains the fallback for the rare wafer whose defects exceed the provisioned table capacity (§5.4). An architect can therefore layer DLR on top of the existing repair mechanisms while retaining that fallback; the PEs DLR preserves are put to use by defect-aware mapping, which places computation only on healthy PEs (§3). Cerebras' WSE already routes over tens of *colors*, akin to virtual channels, and carries routing configuration in every router [5, 18]; DLR's two VC sets and LUT are similar router state, with routes computed automatically from the defect map rather than configured per application. By reducing the reliance on hardware redundancy that currently constrains design choices and sacrifices compute cores, DLR widens the design space available to wafer-scale architects.

8 Conclusion

Wafer-scale systems promise unprecedented efficiency for AI, but defects turn routing into a fundamental scalability bottleneck. We show that this bottleneck can be removed without sacrificing healthy PEs or incurring the heavy costs of prior FT routing schemes. By exploiting the mostly intact structure of real wafers and preserving the locality that wafer-scale workloads depend on, DLR achieves scalable, low-overhead, and fully utilizable FT routing for million-PE meshes. This shifts defect tolerance from a source of silicon loss into an opportunity for better wafer-scale system design.

Acknowledgments

The authors thank the shepherd and the anonymous reviewers for their valuable feedback on this work. We acknowledge the support of the Advanced Research and Invention Agency (ARIA), EPCC, and the Edinburgh International Data Facility (EIDF) at the University of Edinburgh. We also thank EASE Lab and its industry partners and sponsors for their support.

References

- [1] Konstantinos Aisopos, Andrew DeOrio, Li-Shiuan Peh, and Valeria Bertacco. 2011. ARIADNE: Agnostic Reconfiguration in a Disconnected Network Environment. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 298–309. doi:10.1109/PACT.2011.61
- [2] Rimpay Bishnoi, Vijay Laxmi, Manoj Singh Gaur, and José Flich. 2015. d^2 -LBDR: Distance-Driven Routing to Handle Permanent Failures in 2D Mesh NoCs. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*. IEEE, 800–805. doi:10.7873/DATE.2015.0477
- [3] Rajendra V. Boppana and Suresh Chalasani. 1995. Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks. *IEEE Trans. Comput.* 44, 7 (1995), 848–864. doi:10.1109/12.392844
- [4] Nick Brown, Brandon Echols, Justs Zarins, and Tobias Grosser. 2023. Exploring the Suitability of the Cerebras Wafer Scale Engine for Stencil-Based Computation Codes. In *Euro-Par 2022: Parallel Processing Workshops (Lecture Notes in Computer Science, Vol. 13835)*. Springer, Cham, 51–65. doi:10.1007/978-3-031-31209-0_4
- [5] Cerebras Systems. [n.d.]. *Wafer-Scale Engine Architecture — Cerebras SDK Documentation*. Retrieved September 13, 2026 from <https://sdk.cerebras.ai/computing-with-cerebras>
- [6] J. A. Cunningham. 1990. The Use and Evaluation of Yield Models in Integrated Circuit Manufacturing. *IEEE Transactions on Semiconductor Manufacturing* 3, 2 (1990), 60–71. doi:10.1109/66.53188
- [7] Ian Cutress. 2020. 'Better Yield on 5nm than 7nm': TSMC Update on Defect Rates for N5. AnandTech. Retrieved September 13, 2026 from <https://web.archive.org/web/20200825135522/https://www.anandtech.com/show/16028/better-yield-on-5nm-than-7nm-tsmc-update-on-defect-rates-for-n5>
- [8] William J. Dally and Charles L. Seitz. 1987. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.* C-36, 5 (1987), 547–553. doi:10.1109/TC.1987.1676939
- [9] Bhavya K. Daya, Chia-Hsin Owen Chen, Suvinay Subramanian, Woo-Cheol Kwon, Sunghyun Park, Tushar Krishna, Jim Holt, Anantha P. Chandrakasan, and Li-Shiuan Peh. 2014. SCORPIO: A 36-Core Research Chip Demonstrating Snoopy Coherence on a Scalable Mesh NoC with In-Network Ordering. In *Proceedings of the 41st Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 25–36. doi:10.1109/ISCA.2014.6853232
- [10] José Duato, Olav Lysne, Ruoming Pang, and Timothy M. Pinkston. 2005. A Theory for Deadlock-Free Dynamic Network Reconfiguration. Part I. *IEEE Transactions on Parallel and Distributed Systems* 16, 5 (2005), 412–427. doi:10.1109/TPDS.2005.58
- [11] José Flich, Andres Mejia, Pedro López, and José Duato. 2007. Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chips. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS)*. IEEE, 183–194. doi:10.1109/NOCS.2007.39
- [12] Congjie He, Yeqi Huang, Pei Mu, Ziming Miao, Jilong Xue, Lingxiao Ma, Fan Yang, and Luo Mai. 2025. WaferLLM: Large Language Model Inference at Wafer Scale. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Boston, MA, 257–273. <https://www.usenix.org/conference/osdi25/presentation/he>
- [13] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. 2007. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro* 27, 5 (2007), 51–61. doi:10.1109/MM.2007.4378783
- [14] International Technology Roadmap for Semiconductors. 2003. *ITRS 2003 Edition: Yield Enhancement*. Technical Report. Semiconductor Industry Association. <https://www.semiconductors.org/wp-content/uploads/2018/08/YieldEnhanc2003.pdf>
- [15] Anugrah Jain, Vijay Laxmi, Meenakshi Tripathi, Manoj Singh Gaur, and Rimpay Bishnoi. 2019. S2DIO: An Extended Scalable 2D Mesh Network-on-Chip Routing Reconfiguration for Efficient Bypass of Link Failures. *The Journal of Supercomputing* 75, 10 (2019), 6855–6881. doi:10.1007/s11227-019-02915-5
- [16] Sean Lie. 2019. Wafer-Scale Deep Learning. In *IEEE Hot Chips 31 Symposium (HCS)*. IEEE, 1–31. doi:10.1109/HOTCHIPS.2019.8875628

- [17] Sean Lie. 2023. Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning. *IEEE Micro* 43, 3 (2023), 18–30. doi:10.1109/MM.2023.3256384
- [18] Sean Lie, Michael Edwin James, Michael Morrison, Srikanth Arekapudi, and Gary R. Lauterbach. 2022. Processor Element Redundancy for Accelerated Deep Learning. US Patent 11,328,208 B2. <https://patents.google.com/patent/US11328208B2/en>
- [19] Piotr Luczynski, Lukas Gianinazzi, Patrick Iff, Leighton Wilson, Daniele De Sensi, and Torsten Hoefler. 2024. Near-Optimal Wafer-Scale Reduce. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, New York, NY, USA, 334–347. doi:10.1145/3625549.3658693
- [20] Andrés Mejia, José Flich, José Duato, Sven-Arne Reinemo, and Tor Skeie. 2006. Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 10 pages. doi:10.1109/IPDPS.2006.1639341
- [21] José Miguel Montañana, José Flich, and José Duato. 2008. Epoch-Based Reconfiguration: Fast, Simple, and Effective Dynamic Network Reconfiguration. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, 1–12. doi:10.1109/IPDPS.2008.4536298
- [22] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, Mazin S. Younis, and Chita R. Das. 2006. ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 333–346. doi:10.1109/MICRO.2006.50
- [23] Yanghui Ou, Shady Agwa, and Christopher Batten. 2020. Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology. In *Proceedings of the 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 1–8. doi:10.1109/NOCS50636.2020.9241710
- [24] Timothy Mark Pinkston, Ruoming Pang, and José Duato. 2003. Deadlock-Free Dynamic Reconfiguration Schemes for Increased Network Dependability. *IEEE Transactions on Parallel and Distributed Systems* 14, 8 (2003), 780–794. doi:10.1109/TPDS.2003.1225057
- [25] Valentín Puente, José A. Gregorio, Fernando Vallejo, and Ramón Beivide. 2004. ImmuneNet: A Cheap and Robust Fault-Tolerant Packet Routing Mechanism. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 198–209. doi:10.1109/ISCA.2004.1310775
- [26] Kamil Rocki, Dirk Van Essendelft, Ilya Sharapov, Robert Schreiber, Michael Morrison, Vladimir Kibardin, Andrey Portnoy, Jean Francois Dietiker, Madhava Syamlal, and Michael James. 2020. Fast Stencil-Code Computation on a Wafer-Scale Processor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 1–14. doi:10.1109/SC41405.2020.00062
- [27] Samuel Rodrigo, José Flich, Antoni Roca, Simone Medardoni, Davide Bertozzi, Jesús Camacho, Federico Silla, and José Duato. 2011. Cost-Efficient On-Chip Routing Implementations for CMP and MPSoC Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 4 (2011), 534–547. doi:10.1109/TCAD.2011.2119150
- [28] Gert Schley and Martin Radetzki. 2015. Fault Tolerant Routing for Hierarchically Organized Networks-on-Chip. In *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 379–386. doi:10.1109/PDP.2015.36
- [29] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. 1990. *Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links*. SRC Research Report 59. DEC Systems Research Center. <https://web.mit.edu/6.826/archive/S97/21-Schroeder-Autonet-1-paper.pdf>
- [30] C. H. Stapper, F. M. Armstrong, and K. Saji. 1983. Integrated Circuit Yield Statistics. *Proc. IEEE* 71, 4 (1983), 453–470. doi:10.1109/PROC.1983.12619
- [31] Yuval Tamir and Gregory L. Frazier. 1988. High-Performance Multi-queue Buffers for VLSI Communication Switches. In *Proceedings of the 15th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society Press, 343–354. doi:10.1109/ISCA.1988.5245
- [32] Leslie G. Valiant. 1982. A Scheme for Fast Parallel Communication. *SIAM J. Comput.* 11, 2 (1982), 350–361. doi:10.1137/0211027
- [33] James Wang. 2025. *100x Defect Tolerance: How Cerebras Solved the Yield Problem*. Retrieved September 13, 2026 from <https://www.cerebras.ai/blog/100x-defect-tolerance-how-cerebras-solved-the-yield-problem>
- [34] Ming-Ju Wu, Jyh-Shing R. Jang, and Jui-Long Chen. 2015. Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets. *IEEE Transactions on Semiconductor Manufacturing* 28, 1 (Feb. 2015), 1–12. doi:10.1109/TSM.2014.2364237
- [35] Dong Xiang, Yueli Zhang, Yi Pan, and Jie Wu. 2007. Deadlock-Free Adaptive Routing in Meshes Based on Cost-Effective Deadlock Avoidance Schemes. In *Proceedings of the International Conference on Parallel Processing (ICPP)*. IEEE, 41. doi:10.1109/ICPP.2007.30
- [36] Qiao Xu, Naigong Yu, and Firdaous Essaf. 2022. Improved Wafer Map Inspection Using Attention Mechanism and Cosine Normalization. *Machines* 10, 2, Article 146 (2022), 19 pages. doi:10.3390/machines10020146
- [37] Zheng Xu, Dehao Kong, Jiaxin Liu, Jinxi Li, Jingxiang Hou, Xu Dai, Chao Li, Shaojun Wei, Yang Hu, and Shouyi Yin. 2025. WSC-LLM: Efficient LLM Service and Architecture Co-exploration for Wafer-scale Chips. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, New York, NY, USA, 1–17. doi:10.1145/3695053.3731101