Toward Practical 3D NoCs: Topology Design, Routing Bias, and Cost–Performance Trade-offs

A gem5/Garnet 3.0 Study of 3D Mesh and Sparse-Vertical Variants

Your Name Your Collaborator

Institution / Course / Term

August 19, 2025

Abstract

While 2D Network-on-Chip (NoC) fabrics are mature and widely deployed, industry is actively exploring 3D NoC organizations to unlock higher bisection bandwidth and lower hop counts without growing chip footprint. Using `gem5` with Garnet 3.0, we first reproduce the well-known efficiency advantage of a 3D Mesh against a 2D Mesh with the *same number of routers* (e.g., 64 vs. 64). However, dense per-router TSV columns exacerbate manufacturing yield risks and area/power overhead, as reported by prior cluster mesh studies. Motivated by these practical constraints, we implement several 3D topologies with *sparser Z connectivity*: (i) Cluster3D with hub-based vertical sharing, (ii) Sparse3D Pillars, (iii) 3D Small-World Express links, and (iv) Hierarchical 3D Chiplets. We provide configurable Python topologies, weight-based TABLE routing to bias paths, and a measurement methodology. Our experiments sweep injection rates across traffic patterns, exposing how each design trades off performance (latency/throughput) versus TSV budget and routing simplicity.

1 Introduction

2D NoCs have benefited from years of architectural optimization and tooling. As transistor scaling slows and multi-die integration rises, 3D NoCs promise better path lengths and bandwidth density. Yet, deploying dense vertical connections (TSVs or micro-bumps) at each router often faces yield, cost, and thermal constraints. As a result, practical systems seek *structured sparsity* in the Z dimension: not every router needs its own vertical link, so long as the network can route efficiently to shared vertical resources.

This project asks: *(1) How much efficiency do we retain when we replace a fully dense 3D Mesh with more practical, sparse-Z topologies? (2) Which sparse patterns strike the best balance between performance and implementation cost?* We build four representative families in `gem5`/Garnet 3.0 and evaluate them against classic 3D/2D Mesh baselines.

Contributions.

- A clean set of `configs/topologies` for 3D Mesh and three sparse-Z designs (Cluster3D-Hub, Sparse3D-Pillars, SW3D-Express) plus a hierarchical 3D Chiplet fabric.
- Weight-based TABLE routing schemes that bias traffic toward designated vertical conduits or express links while preserving deadlock freedom.
- A reproducible measurement pipeline (sweeps, CSV collection, plotting) to compare latency, throughput, link utilization, and saturation points under uniform and non-uniform traffic.
- Case studies of cost–performance trade-offs under fixed "vertical budget," highlighting where sparse designs approach dense mesh performance.

2 Background and Related Work

2.1 Dense 3D Mesh vs. 2D Mesh

3D Mesh reduces average hop count by adding Z edges to the 2D grid, which can improve latency and throughput at a fixed router count. We briefly review dimension-order routing (DOR) and weight-based TABLE routing in Garnet 3.0, and how link/router latencies model physical effects.

2.2 Manufacturability and TSV Economics

Prior work (e.g., cluster mesh proposals) notes that a TSV per router can hurt yield and area. We discuss TSV pitch, thermal implications, and common mitigations (sharing, pillars, chiplets).

2.3 Sparse-Vertical and Hierarchical 3D Fabrics

We summarize key ideas behind: hub-sharing (Cluster3D), pillar-based sparsity, small-world express links, and chiplet-level hierarchies, motivating our four concrete designs.

3 Topologies Implemented

This work adds Python topologies under `configs/topologies/`. All use `IntLink`/`ExtLink` wiring with explicit port names (e.g., `East`, `UpGW`, `EastExp`) and *weights* to guide TABLE routing.

3.1 3D Mesh (XYZ DOR baseline)

We use a standard 3D Mesh (e.g., 4×4×4) with fully populated Z links and DOR (XYZ) or TABLE routing with $W_X < W_Y < W_Z$.

- **Goal**: establish the upper-bound efficiency for a given router count.
- **Knobs**: link/router latency; optional "vertical speedup" to emulate DVFS on Z links; vlink parallelization (bandwidth).

3.2 Cluster3D_Hub

A 2×2 submesh per layer shares a central hub; hubs connect vertically. The hub is transit-only (no endpoint injection). We bias routes via weights to funnel local traffic through its hub for Z.

- **Design knobs**: cluster size (e.g., 2×2×1, 2×2×2), vertical speedup, hub router latency reduction.
- **Rationale**: cut TSV count while preserving short detours to vertical resources.

3.3 Sparse3D_Pillars

Place vertical pillars every (P_x, P_y) tiles. Only routers aligned to a pillar get Up/Down links. Traffic uses XY detours to the nearest pillar, then moves vertically.

- **Design knobs**: P_x, P_y spacing, aligned vs. staggered patterns (future work).
- **Routing**: TABLE with $W_X < W_Y < W_Z$ and no direct Up/Down from non-pillar routers.

3.4 SW3D_Express

Add a budget of long-range *express* links (in-plane or cross-layer) on top of 3D Mesh or sparse-Z base. We use special port names (`EastExp`, `UpExp`, ...) and assign them the lowest weight (W_{EXP}).

Table 1: Topology summary and intended routing bias (lower weight is preferred). Edit with your final parameters.

Topology	Z budget	Extra links	Routing mode	Weights (X/Y/Z/extra)
3D Mesh	dense	none	TABLE or XYZ DOR	1/2/3/–
Cluster3D_Hub	sparse hubs	hub vertical	TABLE	1/2/3/– (hub Z sped-up)
Sparse3D_Pillars	pillars (P_x, P_y)	none	TABLE	1/2/3/–
SW3D_Express	dense/sparse	express	TABLE	1/2/3/0 (exp)
Hier3D_Chiplet	per-gateway	backbone	TABLE	1/2/3/– (+GW tier)

- **Design knobs**: grid-based placement period K (rule-based); total budget B; (optional) randomized placement.
- **Routing**: TABLE prefers express edges first, then normal links.

3.5 Hier3D_Chiplet

Partition the die into chiplets (e.g., 4×4×1 each). Within a chiplet: local mesh. Gateways (1–2 per chiplet) connect to a backbone (2D in-layer and/or vertical). Gateways use `*GW` ports and higher weights at the backbone to avoid overuse.

- **Design knobs**: chiplet dimensions, #gateways, backbone topology (ring/mesh), vertical gateway connectivity.
- **Routing**: TABLE with tiered weights: $W_{\text{intra}} < W_{\text{backbone}} < W_Z$.

4 Routing and Deadlock Considerations

We primarily use TABLE-based routing in Garnet 3.0 with weights to induce preferences:

- **Strict ordering**: Distinct weights per dimension/tier maintain a partial order and avoid cycles typical in oblivious shortest-path selection.
- **Express edges**: assign the smallest weight (e.g., W=0) to let the router choose them deterministically before normal links.
- **Gateway tiers**: use intermediate weights for backbone/GW hops to avoid accidental oscillations.

We validated reachability via `config.ini` inspections and by eliminating "No Route exists from this Router" fatals during warmup runs.

5 Methodology

5.1 Simulator and Network

We use `gem5` [**?**] with Garnet 3.0 (Garnet 3.0). Unless noted, system and Ruby clocks are 1 GHz. Link width 128 bits, link latency 2, router latency 2.

5.2 Traffic and Sweeps

We exercise `garnet_synth_traffic.py` with:

- Patterns: `uniform_random`, `transpose`, `hotspot`, `shuffle`.
- Injection rate sweep: 0.01 to 0.25 (step 0.01).
- Simulation length: 2,000,000 cycles (extend near saturation).

5.3 Metrics

We collect average packet/flit latency, average hops, total injected/received packets (to derive accepted rate), and average link utilization. Saturation is indicated by the knee where latency diverges and accepted rate flattens.

5.4 Reproducibility

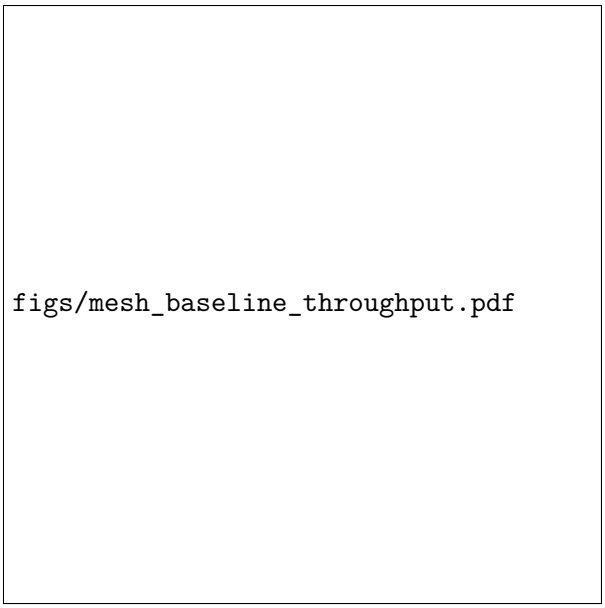
We provide sweep scripts, a post-processing `collect.sh` that converts `stats.txt` into CSV, and `plot_metrics.py` for latency/throughput curves. See **??**.

6 Experimental Results

6.1 3D Mesh vs. 2D Mesh (same router count)

Setup. Compare 4×4×4 vs. 8×8 with identical #routers and similar link/router latencies. **Expectation.** 3D Mesh achieves lower hop counts and better throughput at a given injection rate.

(a) Latency vs. injection.



(b) Throughput vs. injection.

Figure 1: Dense 3D Mesh vs. 2D Mesh with same router count.

6.2 Sparse-Z Designs under a Fixed Vertical Budget

We normalize each design to a comparable TSV count (or pillar density) and evaluate:

- Latency/throughput curves across traffic patterns.
- Avg. link utilization and hop distributions.
- Sensitivity to vertical speedup and gateway count.

6.3 Ablations

Cluster size & hub speedup. How 2x2 vs. 3x3 clusters and hub DVFS change the knee point.

Pillar spacing (P_x, P_y). Trade-offs between detour cost and TSV count.

Express budget B and period K. Diminishing returns of additional long links.

Chiplet gateways. One vs. two gateways per chiplet and backbone shape.

Figure 2: Latency vs. injection across sparse-Z designs (Cluster3D-Hub, Pillars, SW3D-Express, Chiplet).

7 Discussion

7.1 Which Sparse Pattern Scales Best?

We interpret the results through the lens of manufacturing and thermal feasibility, highlighting topologies that get "close enough" to dense 3D Mesh while using far fewer vertical links.

7.2 Routing Simplicity vs. Performance

TABLE routing with weights gives strong baselines without custom algorithms; we note when custom adaptive schemes might help (left as future work).

7.3 Limitations

Cycle-accurate microarchitectural models (buffers, VC allocation) follow Garnet defaults; real silicon may differ in timing closure and power. Our synthetic traffic stresses the network but does not model specific workloads.

8 Conclusion

3D NoCs are promising, but dense Z wiring is difficult to manufacture at scale. Our study shows that structured sparsity—hub sharing, pillars, express links, and chiplets—retains much of the performance benefit while substantially reducing vertical resources. Weight-based TABLE routing is a practical mechanism to guide traffic along these sparse conduits.

A Topology Implementation Details

We place our Python files in `configs/topologies/`:

- `Mesh3D_XYZ_.py`: 3D Mesh baseline (XYZ DOR or TABLE).
- `Cluster3D_Hub.py`: cluster hub + vertical hub links (with configurable hub speedup).
- `Sparse3D_Pillars.py`: pillar spacing (P_x, P_y); optional staggered layout (future).
- `SW3D_Express.py`: rule-based express placement (period K) with W_{EXP}=0.
- `Hier3D_Chiplet.py`: chiplet size, #gateways, backbone wiring.

We followed gem5's `Mesh_XY` style: create routers, wire `ExtLink`s to NIs, then build `IntLink`s with port names and weights.

B Command Lines and Sweeps

Single Run Example

```
./build/NULL/gem5.opt configs/example/garnet_synth_traffic.py \
  --network=garnet --topology=Sparse3D_Pillars \
```

```
  --num-cpus=64 --num-dirs=64 --mesh-rows=4 \
  --routing-algorithm=0 \
  --synthetic=uniform_random --injectionrate=0.05 \
  --sim-cycles=2000000 \
  --sys-clock=1GHz --ruby-clock=1GHz \
  --link-width-bits=128 --link-latency=2 --router-latency=2 \
  --mem-type=SimpleMemory --mem-channels=64 --mem-size=8192MB
```

Sweep Script (bash)

```
for r in $(seq 0.01 0.01 0.25); do
  ./build/NULL/gem5.opt \
    -d runs/pillars_uniform_r${r} \
    configs/example/garnet_synth_traffic.py \
    --network=garnet --topology=Sparse3D_Pillars \
    --num-cpus=64 --num-dirs=64 --mesh-rows=4 \
    --routing-algorithm=0 \
    --synthetic=uniform_random --injectionrate=${r} \
    --sim-cycles=2000000 \
    --sys-clock=1GHz --ruby-clock=1GHz \
    --link-width-bits=128 --link-latency=2 --router-latency=2 \
    --mem-type=SimpleMemory --mem-channels=64 --mem-size=8192MB
done
```

C Data Collection and Plotting

Collect CSV.

```
./collect.sh runs results/pillars_uniform.csv
```

Plot curves.

```
python3 plot_metrics.py \
  --out results/plots \
  --series "Pillars:results/pillars_uniform.csv" \
  --series "3D Mesh:results/mesh3d_uniform.csv"
```

D Sanity Checks

We grep `config.ini` to confirm topology name, routing mode, #IntLinks, port names (e.g., `UpGW`, `EastExp`) and weight histograms; we ensure every router has outgoing `IntLink`s.

References

References