

ParaYosys: Tail-Aware Partition-Based Parallel RTL Synthesis

A partition-based parallel synthesis flow for Yosys — and why structural balance does not predict backend cost.

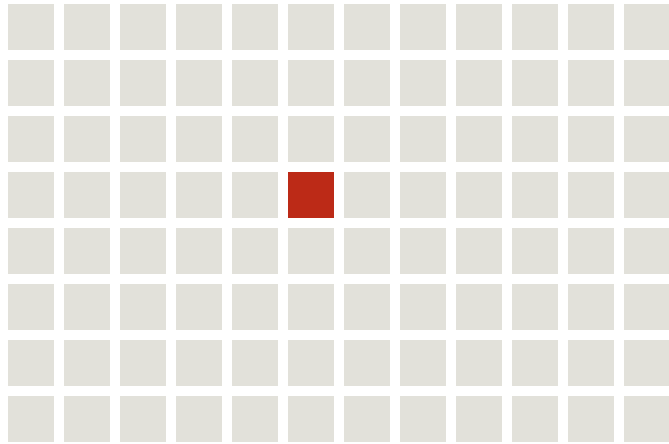
Zhengyi Zhang · Sijing Yang · Lingli Wang

State Key Laboratory of Integrated Chips and Systems
Fudan University, Shanghai



servant at P24: eleven workers finish in seconds,
one shard sets the makespan.

RTL synthesis stays serial on machines with dozens of hardware threads



Passes exchange global information. Rewriting crosses module boundaries and cleanup depends on whole-design structure.

So thread-safety means rewriting the tool, not adding a feature — every pass in a mature pipeline would have to change.

But the post-flatten backend is local. Optimization and mapping act on a region of logic, which is a granularity a partitioner can expose.

96 threads available.

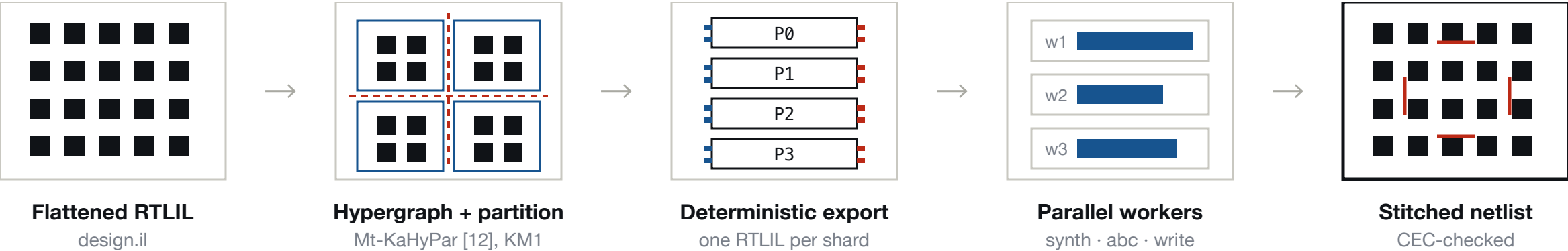
One does the work.

1. Framework. Partition-based parallel synthesis inside Yosys [1].

2. Metrics. Cost-model gap and critical-shard share.

3. Cost model. Cheap features that predict backend cost.

Four problems a partitioned flow must solve



Problem	Mechanism
Partition selection	Hypergraph over logic clusters [9, 10]
Boundary ownership	One driver per cut bit
Execution isolation	Separate OS processes
Reconstruction	Mask-driven stitch

Nodes are logic clusters — a cell, or a memory cluster kept intact.

A net with n terminals becomes one hyperedge over n nodes. Pairwise edges over-count wide fanout and mis-price the cut.

k partitions are scheduled onto w worker processes, with $k \geq w$ permitted.

Bit-level ownership makes each shard a legal module

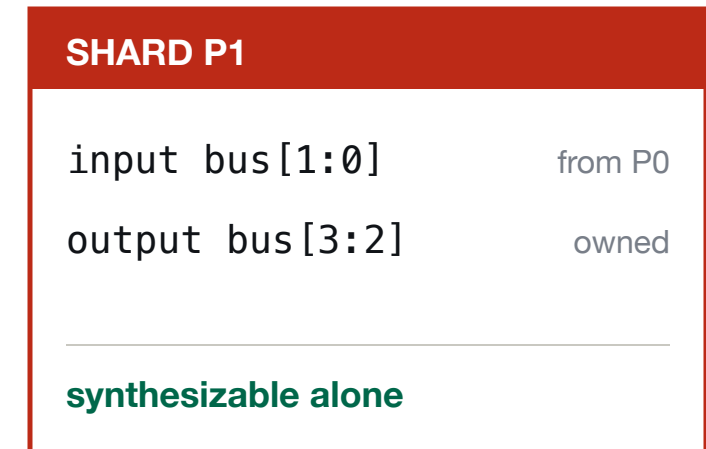
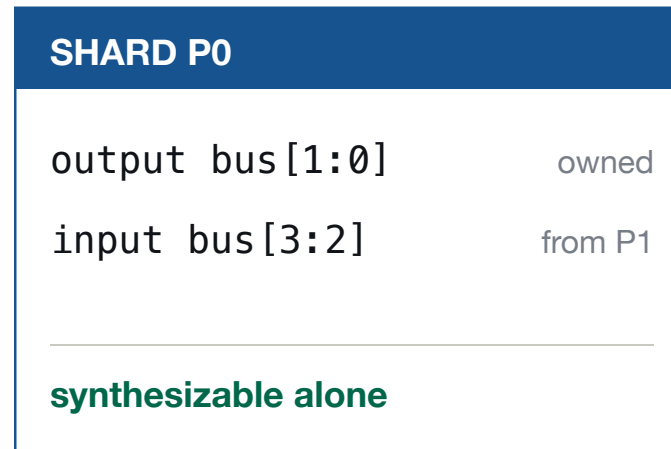
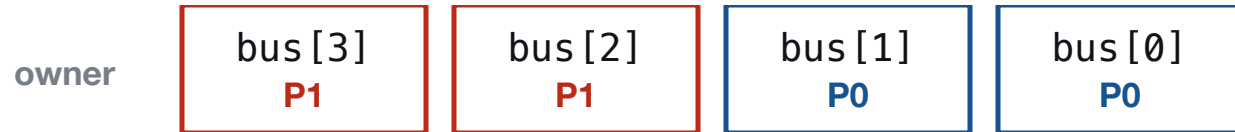
Export, per partition

1. **Clone** the flattened top
2. **Remove** foreign cells
3. **Prune** unreferenced wires
4. **Preserve** module references
5. **Rewrite ports** by ownership

Ownership rule

A cut bit belongs to the partition holding its **driver**. Readers declare an input port. **One driver per bit, by construction.**

bus[3:0] crossing one cut



Export is **deterministic**: the same input and partition count give byte-identical shards.

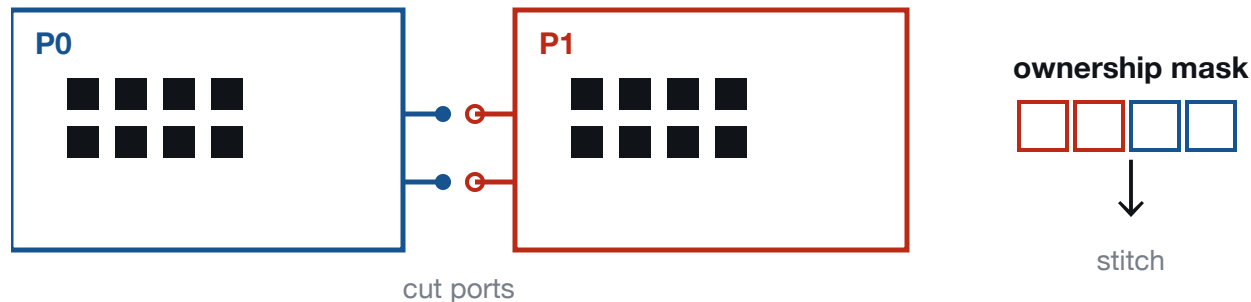
Isolated workers, then a mask-driven stitch

One process per worker, not one thread. Shared-state hazards disappear and Yosys runs unmodified inside each worker.

"Backend" is a defined scope — post-flatten optimization and mapping, including ABC [2]. Parsing and flatten sit outside it.

After worker synthesis

two netlists, cut ports unconnected

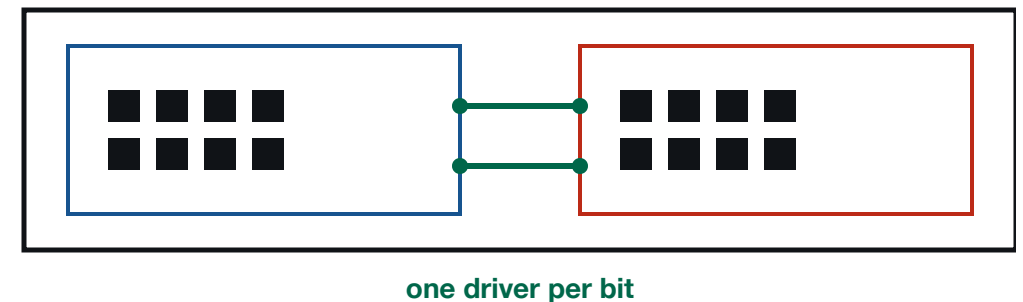


The stitcher reconnects cut nets from the ownership masks recorded at export, then materializes one composition module.

It also records what each partition cost. That instrumentation is what the second half of this talk rests on.

After stitch

one composition module, cuts reconnected



Every stitched netlist is CEC-equivalent to its paired serial reference. CEC runs separately, excluded from all timings.

What the partitioner should minimize

$$\min \max_i \hat{T}(P_i) + \lambda \text{Cut}(P_1, \dots, P_k) + \mu \text{Overhead}$$

Wall time is decided by the first term alone — the makespan of the slowest partition.

The substitution everyone makes

Set $\hat{T}$ to **structural size** and this collapses into ordinary node-balanced partitioning, which is what an off-the-shelf partitioner gives you. **The rest of the talk measures what that costs.**

Tail-reduction mechanisms

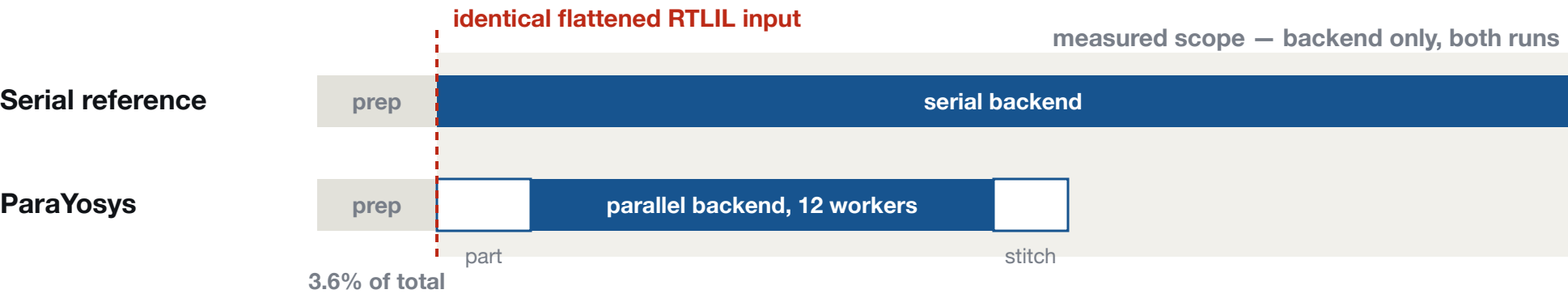
Cost-aware weighting — bias weights toward memory-heavy, arithmetic-heavy and wide-interface regions so they get split.

Overpartitioning — more partitions than workers; the pool absorbs residual estimation error at schedule time.

Memory sidecar — take memory clusters out of ordinary partitioning altogether.

Cost proxies come from the shard artifact alone — cell counts by category, wire bits, port bits. **No trial synthesis.**

Scope, baseline and benchmarks

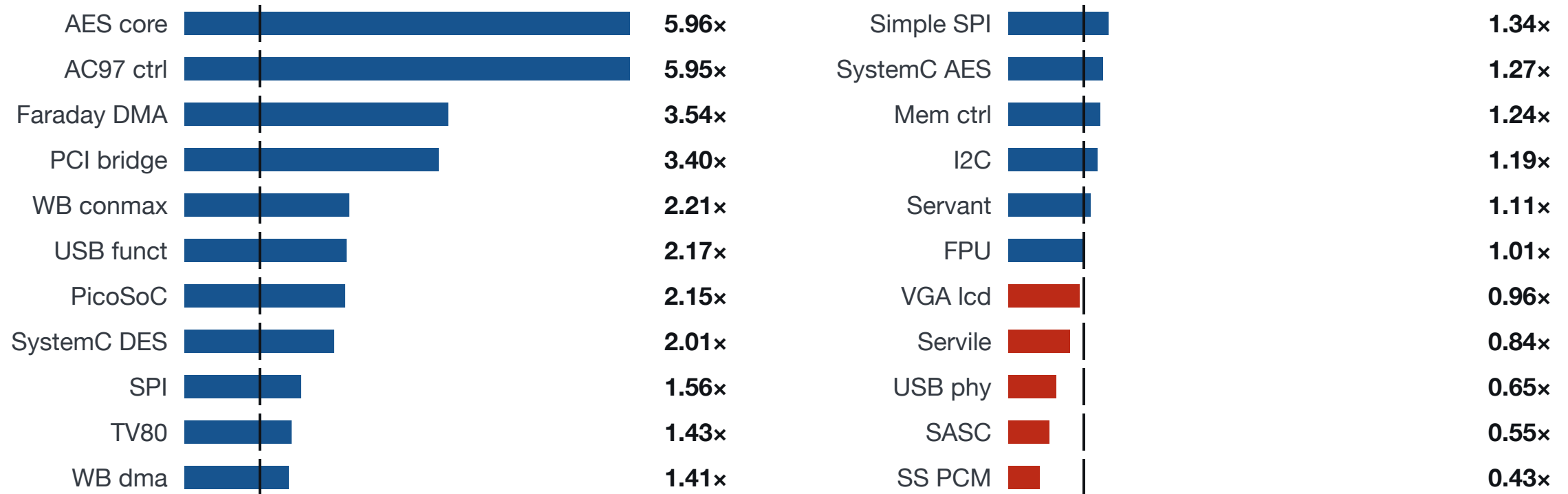


Setup	
Host	4 × Xeon Gold 6126, 48 cores
Workers	12 processes, fixed throughout
Partitioner	Mt-KaHyPar [12], KM1, 5% imbalance
Default point	P24 — 24 partitions, 12 workers

Benchmarks	Cases	
IWLS 2005 / OpenCores [5]	22	bus, memory, arithmetic RTL
EPFL combinational [4]	4	to 483,682 cells
SoC wrappers	3	picosoc, servile, servant
CPU / DMA IP	3	mor1kx, zipcore, zipdma

23 RTL designs attempted, 22 evaluated. Flattened input median **500** cells; paired serial output median **5,579** — that gap is slide 11.

Backend speedup, 22 public RTL designs at P24



Median **1.38x** · mean **1.93x** · best **5.96x** · **13x spread** across the set.

Same flow, host and worker count in every case; the vertical rule marks 1x.

At scale, and the choice of 24 partitions

Large public designs, same P24 point

Design	Flat. in	Serial	Parallel	Speedup	S
EPFL hyp	483,682	1144.5 s	213.2 s	5.37x	0.04
EPFL mem_ctrl	103,337	155.5 s	39.5 s	3.94x	0.05
EPFL mult	59,148	118.0 s	31.0 s	3.81x	0.06
EPFL sqrt	57,104	78.3 s	24.6 s	3.18x	0.06

Core-scale IP on 15–64 s backends: **mor1kx 1.59x**, **zipdma 2.26x**, **zipcore 2.47x**.

The large designs keep runtime tails near **1.3x** and critical-shard shares of **0.04–0.06** — consistent with the stronger speedups.

Operating points

Design	P12	P24	P36
AC97 ctrl	1.82	5.95	4.15
USB funct	1.81	2.17	2.45
WB conmax	2.18	2.21	1.45
WB dma	1.19	1.41	1.11
VGA lcd	0.90	0.96	1.13
FPU	1.08	1.01	1.06
USB phy	1.74	0.65	0.55
Mean	1.53	2.05	1.70

Each sweep has its **own** paired baseline, so this is an operating-point comparison, not a scaling curve.

Structural balance is achieved.

Runtime balance is **not**.

servant — 24 shards of one run

Structural weight

median		1.00
slowest		1.04

Backend runtime

median		0.35 s
slowest		127.5 s

Across the whole sweep: median structural tail **1.04x**, median runtime tail **5.34x**, worst case **365x**.

Worker schedule



makespan 127.5 s

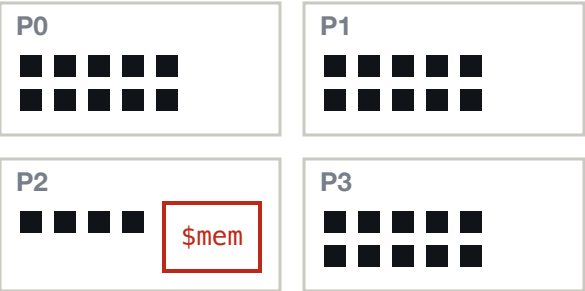
Eleven workers drain in seconds and then idle. The run ends when one shard ends — adding workers changes nothing.

The same holds on FPU (110 s vs 0.40 s) and PicoSoC (18.2 s vs 0.59 s). **In every case the structural tail stays near 1x.**

The partitioner weighs the netlist **before** lowering; the backend pays for it **after**

What the partitioner weighs

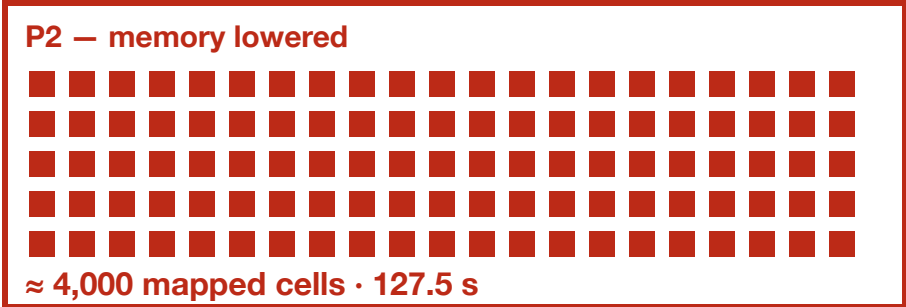
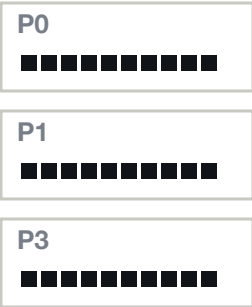
flattened RTLIL — four balanced partitions



structural tail 1.04x

What the backend runs

after lowering — three shards at 0.35 s median, one at 127.5 s



Design	Flat. input	Serial output	Expansion	S	Speedup
VGA lcd	459	37,902	82.6x	0.88	0.96x
FPU	1,085	21,047	19.4x	0.96	1.01x
AC97 ctrl	877	5,673	6.5x	0.11	5.95x
EPFL hyp	483,682	166,754	0.34x	0.04	5.37x

Concentration, not expansion

mor1kx and zipdma expand 56x and 72x and still reach 1.59x and 2.26x. What breaks the schedule is expansion **concentrated in one region** — exactly what an input-side balance constraint cannot see.

Median expansion **11x**. Cost is driven by memory lowering, DAG-aware rewriting [3] and boundary width.

Two tail metrics, and the bound they imply

$$G = B_T / B_W$$

Realized runtime skew over the structural skew the partitioner optimized.

Median gap **5.23x**; worst case **352x** on servant.

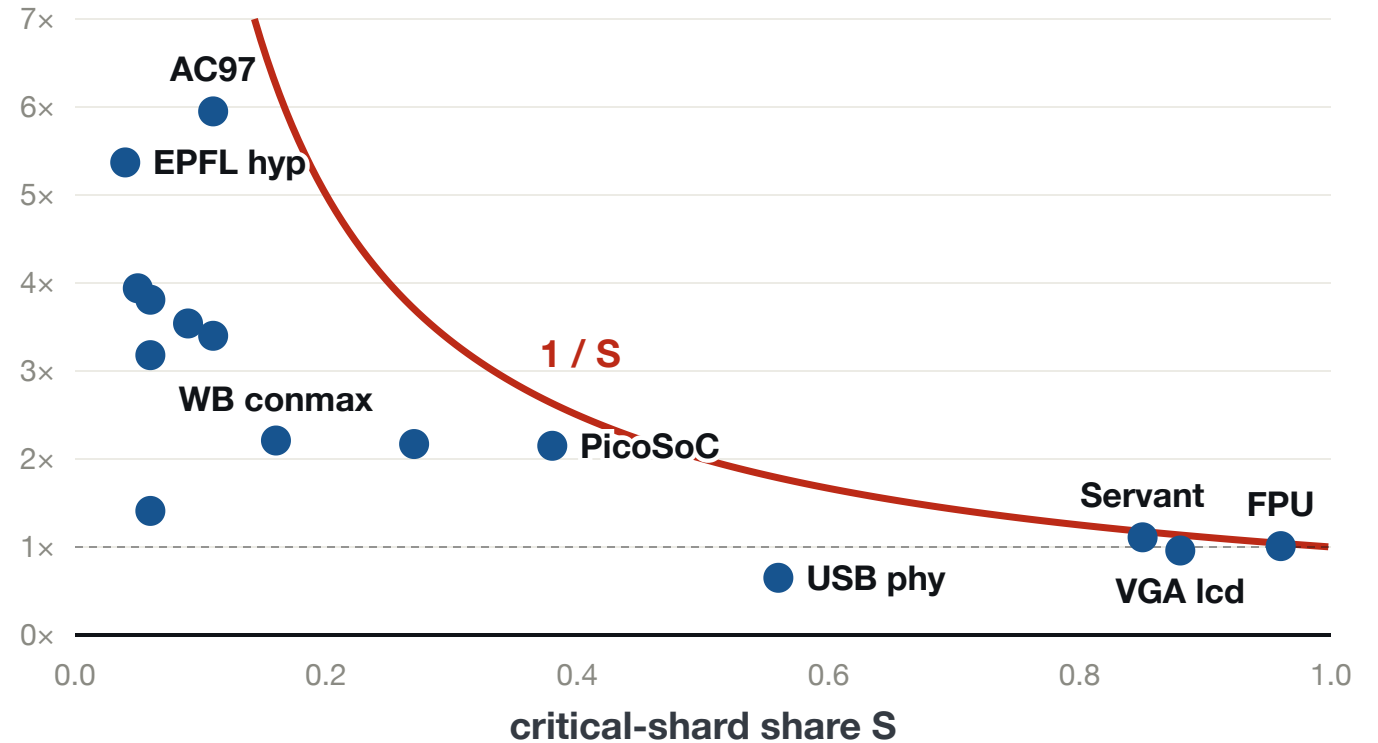
$$S = \max_i T_i / T_{\text{serial}}$$

How much of the serial backend survives on **one** parallel path.

$$T_{\text{par}} \geq \max_i T_i \Rightarrow \text{speedup} \leq 1/S$$

Every measured point sits under the curve. **One instrumented run gives the ceiling in advance.**

backend speedup



Cheap features predict which shard becomes critical

Metric	Structural	Fitted proxy
Spearman ρ vs. runtime	0.753	0.850
Critical-shard hit rate	9.1%	45.5%
Median log tail error	1.65	0.68

Leave-one-**benchmark**-out fit over **528 partitions** from 22 benchmarks. Features come from the shard artifact, so the proxy costs **no synthesis**.

AC97 ctrl

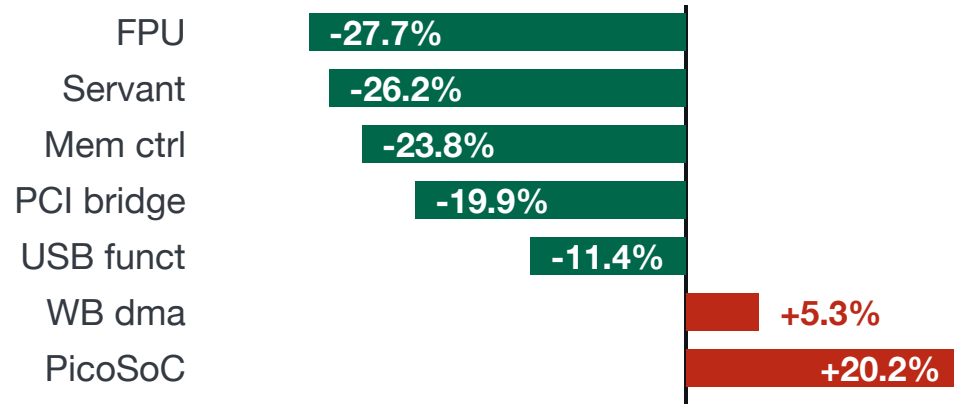
Structural weighting predicts a **1.05×** tail. The proxy predicts **6.44×** and names the right shard. Realized: **6.89×**.

Fitted coefficients		
memory cells		0.801
arithmetic cells		0.185
port bits		0.158
wire bits		0.060
sequential cells		0.023
structural weight		0
cell count		0

The two signals the partitioner uses today shrink to **exactly zero**. Runtime skew is predictable structure, not noise.

Feeding the cost signal back into the weights

10-benchmark P24 rerun — change in parallel wall time



8 of 10 reruns improved, median reduction **11.3%**. Each rerun carries its own fresh paired serial baseline.

A **better default**, not a replacement for per-design tuning — and it helps most where the tail was worst.

Why two regress

wb_dma has the widest cut interface in the subset — 5,220 bits — so boundary emphasis fragments bus paths instead of relieving a tail.

picosoc couples CPU, memory and peripherals tightly enough that the heuristic **relocates** the critical shard rather than splitting it.

Memory sidecar on servant: the slowest partition falls from **92 s to 1.02 s**. A logic-only diagnostic, reported outside the main tables.

Equivalence and downstream QoR

Mapped QoR drift vs. serial	Median	Mean	Worst $ \Delta $	Within band
Area	+0.16%	+2.23%	24.3%	16 / 21 in $\pm 3\%$
Delay	+0.87%	+0.55%	34.6%	11 / 21 in $\pm 5\%$
Sequential area	0.00%	+0.39%	13.3%	17 / 21 in $\pm 5\%$

Both outputs remapped to the **same** liberty under identical ABC constraints; 21 of 22 P24 pairs completed the flow. The omitted parse and flatten step is **3.6%** of total time, so backend-only speedup is a tight upper bound.

Correctness. Every stitched netlist is CEC-equivalent to its paired serial reference. No exceptions.

Stability. Seven cases rerun five times. CV **5.8–19.4%**; the design-level ordering never changes.

The outliers are diagnosable. mem_ctrl duplicates controller logic near boundaries; wb_dma has the widest cut interface.

Conclusion

Framework. Partition-based parallel synthesis inside Yosys — deterministic export, bit-level cut ownership, isolated workers, mask-driven stitch. All CEC-clean, median mapped-area drift **+0.16%**, P24 median **1.38x** and up to **5.96x**.

Mechanism. A structural tail of **1.04x** hides a runtime tail of **5.34x**: the partitioner weighs the **pre-lowering** netlist while the backend pays for the **post-lowering** one. Speedup is bounded by **1/S**, not by worker count.

Cost model. Cheap RTLIL features lift rank correlation to **0.850** and critical-shard identification from 9.1% to **45.5%**, cutting parallel wall time on **8 of 10** reruns.

Limitations

Backend-only scope is an **upper bound**.

The proxy still misses the critical shard **more than half the time**.

Two of ten reruns regress.

Next: make $\hat{T}$ good enough to put **makespan** in the partitioner's objective.

References

- [1] C. Wolf and J. Glaser. Yosys — a free Verilog synthesis suite. *Austrochip*, 2013.
- [2] R. Brayton and A. Mishchenko. ABC: an academic industrial-strength verification tool. *CAV*, 2010.
- [3] A. Mishchenko, S. Chatterjee, R. Brayton. DAG-aware AIG rewriting. *DAC*, 2006.
- [4] L. Amarú et al. The EPFL combinational benchmark suite. 2019.
- [5] IWLS 2005 benchmarks. iwls.org, 2005.
- [6] J. Rose et al. The VTR project. *FPGA*, 2012.
- [7] T. Ajayi et al. Toward an open-source digital flow: OpenROAD. *DAC*, 2019.
- [8] D. Shah et al. Yosys+nextpnr. *FCCM*, 2019.
- [9] G. Karypis et al. Multilevel hypergraph partitioning in the VLSI domain. *DAC*, 1997.
- [10] S. Dey, F. Brglez, Z. Kedem. Circuit partitioning for logic synthesis. *JSSC*, 1991.
- [11] L. Gottesbüren et al. Scalable shared-memory hypergraph partitioning. *ALENEX*, 2021.
- [12] L. Gottesbüren et al. Scalable high-quality hypergraph partitioning. *ACM TALG*, 2024.
- [13] R. Liang, A. Agnesina, H. Ren. MedPart. *ISPD*, 2024.
- [14] S. Tong et al. EasyPart. *ICCAD*, 2024.
- [15] K. De and P. Banerjee. Parallel logic synthesis using partitioning. *ICPP*, 1994.
- [16] Synopsys. Design Compiler Graphical. 2025.
- [17] Y. Li et al. DAG-aware synthesis orchestration. *TCAD*, 2024.
- [18] H. Wang and S. Beamer. RepCut. *ASPLOS*, 2023.
- [19] M. Emami, T. Bourgeat, J. R. Larus. Parendi. *ASPLOS*, 2025.
- [20] YH. Chung et al. SimPart. *Euro-Par*, 2025.

Numbering follows the paper. github.com/0xtaruhi/yosys-parallel