RAMTERA TECHNICAL WHITE PAPER SERIES

WP-11

Scaling Across Cores, Honestly

Thread scaling in a tiled physical verification engine, and why the curve bends

A speedup curve that bends is not necessarily a scalability defect. Sometimes it is the machine.

6 min	2 min	52 s	6.9×
ONE CORE	FOUR CORES	SIXTEEN CORES	OVERALL SPEEDUP

Document	WP-11 — Scaling Across Cores, Honestly
Author	Sameer Pawanekar, Founder & Chief Executive Officer, Ramtera Inc.
Revision	v1.0
Date	August 2026
Classification	Public

ABSTRACT

Physical verification parallelises well in principle: the design partitions spatially, and most checks are local. In practice the measured speedup curve bends well short of linear, and the reasons are worth separating carefully — because some of them are properties of the software and some are properties of the machine, and the two call for entirely different responses. This paper reports thread scaling of a non-density design rule check on a one-million-polygon design, from one core to sixteen, and attributes the shape of the curve.

01 The measurement

The design is **daccore**, approximately one million polygons. The workload is the non-density design rule check, with an identical rule deck at every thread count. Violation counts are identical across all runs.

Threads	Runtime	Speedup vs one thread	Parallel efficiency
1	6 min	1.0×	100%
4	2 min	3.0×	75%
16	**52 s**	**6.9×**	**43%**

Table 1 — Thread scaling. daccore non-density check, ~1M polygons, identical deck. Environment in Appendix A.

MEASUREMENT PRECISION

Precision note. The one-thread and four-thread figures are reported to the nearest minute, as measured. The sixteen-thread figure is to the second. Speedup and efficiency derived from the rounded figures therefore carry corresponding uncertainty: the four-thread speedup is between approximately 2.7× and 3.4× on the measurement precision available. This is stated rather than concealed behind derived figures quoted to two decimal places.

02 Why the curve bends

Sixteen threads deliver 6.9×, not 16×. Four separate effects contribute, and they are worth separating because only two of them are addressable in software.

One: the cores are not identical

The measurement machine has sixteen physical cores in a hybrid arrangement — eight performance cores and eight efficient cores — plus simultaneous multithreading giving twenty-four logical processors. The first eight threads land on performance cores. Threads nine through sixteen land on efficient cores, which are materially slower per thread.

A sixteen-thread run on this machine is therefore not sixteen equivalent workers. It is eight fast workers and eight slower ones, and the achievable speedup is bounded well below sixteen before any software consideration enters.

THE KNEE IS IN THE HARDWARE

This is the single most common misreading of scaling data on modern desktop processors. A curve that bends at eight threads on a hybrid machine is measuring the machine, not the software. Publishing such a curve without stating the topology invites a reader to conclude the engine does not scale.

Two: the serial fraction

Rule deck parsing, layer generation, and the ordering of derived layer dependencies are performed once and are not parallelised across the check phase. On a design of this size that setup work is a small but non-zero fraction of total runtime, and it bounds the achievable speedup regardless of core count.

The bound is a hard one. If setup is a tenth of the single-threaded runtime, no amount of parallelism reduces total runtime below a tenth — and the fraction grows in relative terms as the parallel portion shrinks, which is exactly what a bending curve looks like.

Three: work is not uniform across tiles

Spatial partitioning divides the design into regions. It does not divide the *work* evenly, because geometry is not evenly distributed. A tile over dense standard-cell area contains far more shapes than a tile over a routing channel or an empty region.

The run completes when the last tile completes. A partitioning that is balanced by area is imbalanced by work, and the imbalance shows up directly as idle threads at the end of the run. This is addressable, and it is where the remaining headroom on this curve lies.

Four: memory bandwidth

Geometric processing is memory-intensive. Sixteen threads traversing polygon sets contend for a shared last-level cache and for main memory bandwidth. Beyond some thread count, additional threads wait on memory rather than compute, and the marginal return falls. This is a property of the machine and the workload jointly, not of the partitioning.

Contributing effect	Property of	Addressable in software?
Hybrid core topology (performance vs efficient)	The machine	**No**
Serial setup fraction	The software	**Partially** — some setup can be parallelised
Work imbalance across tiles	The partitioning	**Yes** — the main remaining headroom
Memory bandwidth contention	Machine and workload	**Indirectly** — via locality, not thread count

Table 2 — Attribution of the scaling shortfall.

03 What scaling data should always state

Thread scaling figures are quoted frequently and qualified rarely. The parameters below change the interpretation materially, and a curve without them cannot be read.

1. **Core topology.** Homogeneous or hybrid; physical cores or logical threads; whether simultaneous multithreading is engaged at the higher counts. On a hybrid machine this is the dominant explanation for the curve's shape.
2. **Whether the thread count refers to physical cores or logical processors.** These differ by a factor of two on most contemporary processors, and conflating them roughly halves an apparent efficiency figure.
3. **The workload class.** Density and non-density checks on the same design have different runtimes and different parallel characteristics. A figure without its check class is ambiguous.
4. **Design scale.** A small design will scale badly at high thread counts simply because there is insufficient work per tile to amortise partitioning overhead.
5. **Measurement precision.** A speedup quoted to two decimals from runtimes measured to the nearest minute is false precision.
6. **That results are identical across thread counts.** A parallel run that produces different results at different thread counts is not faster; it is broken. This should be asserted explicitly, not assumed.

> **DETERMINISM IS A PRECONDITION, NOT A FEATURE**
> The last point is the one that matters most for a verification tool and is almost never stated. Thread count must not be observable in the output. If it is, the engine has a race, a boundary-ownership defect, or an accumulation order dependency — and the speedup figure is irrelevant until that is fixed.

04 Where the remaining headroom is

Of the four effects above, only tile work imbalance offers substantial recoverable performance. The hybrid topology is fixed, the serial fraction is small and only partially reducible, and memory bandwidth is addressed through locality rather than scheduling.

Work imbalance is addressed by partitioning against a work estimate rather than against area — for instance, by shape count within a region — and by scheduling tiles dynamically rather than assigning them statically, so that a thread finishing a sparse tile takes the next available dense one rather than idling.

Both changes have a correctness constraint attached: results must remain identical to the single-threaded run, and identical across thread counts. Any partitioning scheme is only admissible if it preserves that property, which rules out approaches that would otherwise be attractive.

05 Conclusion

A 6.9× speedup on sixteen physical cores is a 43% parallel efficiency figure, and stated bare it looks poor. Attributed, it is largely explained by a hybrid core topology in which half the cores are substantially slower than the other half, with a small serial fraction and a tile work imbalance accounting for most of the remainder.

The general point is that scaling curves are frequently published without the topology that produced them, and are consequently read as software defects when they are hardware characteristics. Stating the topology costs one line and is the difference between a curve a reader can interpret and one they will misinterpret.

Appendix A — Measurement environment

Component	Specification
Processor	Intel Core i7-13700K (13th generation, Raptor Lake)
Core topology	**16 physical: 8 performance + 8 efficient**; 24 logical with SMT
Clock	800 MHz minimum, 5,400 MHz maximum
Cache	L1d 640 KiB, L1i 768 KiB, L2 24 MiB, L3 30 MiB shared
NUMA	Single node
Memory	64 GB DDR5-4800 (2 × 32 GB), 1 GB swap
Storage	Kingston SNV2S500G NVMe SSD; all design data on NVMe
Operating system	Ubuntu 22.04.3 LTS (jammy), kernel 6.8.0-136-generic
Compiler	GCC 14.0.0 20231104 (experimental snapshot), `-O3`
Design	daccore, approximately 1,000,000 polygons
Check class	**Non-density** design rule check
Rule deck	Identical at every thread count
Thread counts	1, 4, 16 — physical cores
Result verification	Violation counts identical at all thread counts

Table A1 — Measurement environment.

> **WHICH CHECK CLASS**
>
> **Relationship to other papers in this series.** The density design rule check on this same design completes in 120 seconds at sixteen cores and is reported in WP-02. Density and non-density checks execute different rules over different derived layers and should not be compared directly.

Runtimes at one and four threads were recorded to the nearest minute; the sixteen-thread runtime to the second. Derived speedup and efficiency figures inherit that precision and should be read as approximate at the lower thread counts. Memory headroom of 64 GB against 1 GB of swap means these measurements are compute-bound, never paging-bound.

About Ramtera

Ramtera Inc. develops **LVR**, a physical verification engine that executes foundry SVRF rule decks natively, covering design rule checking, layout versus schematic, electrical rule checking, layout XOR, density and fill, and parasitic extraction. LVR is written in modern C++ with a Qt6 interface, and has been validated against publicly available process design kits including SKY130 and IHP SG13G2.

Ramtera also develops **PRE**, a place-and-route engine sharing LVR's geometry kernel, and maintains a portfolio of granted patents, registered copyrights and trademarks across its verification and implementation technologies.

Ramtera Inc. is incorporated in the United States, with research and development conducted by Ramtera Design Automation India Private Limited in Bangalore, India.

Contact	
Web	https://ramtera.com
General enquiries	info@ramtera.com
Technical support	support@ramtera.com
Series	Ramtera Technical White Paper Series
This document	WP-11, revision v1.0, August 2026