Improving Runtime Scaling in the EDA Flow for Designs with Millions of Gates

David Chinnery
R&D Architect
Aprisa RTL2GDS
Siemens EDA

SIEMENS

Agenda

- Motivation: Flow runtime scaling issues for million gate IC designs
- Standard industry software techniques to speed up synthesis and place-and-route (SAPR)
- What further can be done?
- Example: Speed up scan chain wire length minimization by splitting large scan partitions
- Conclusions

Motivation

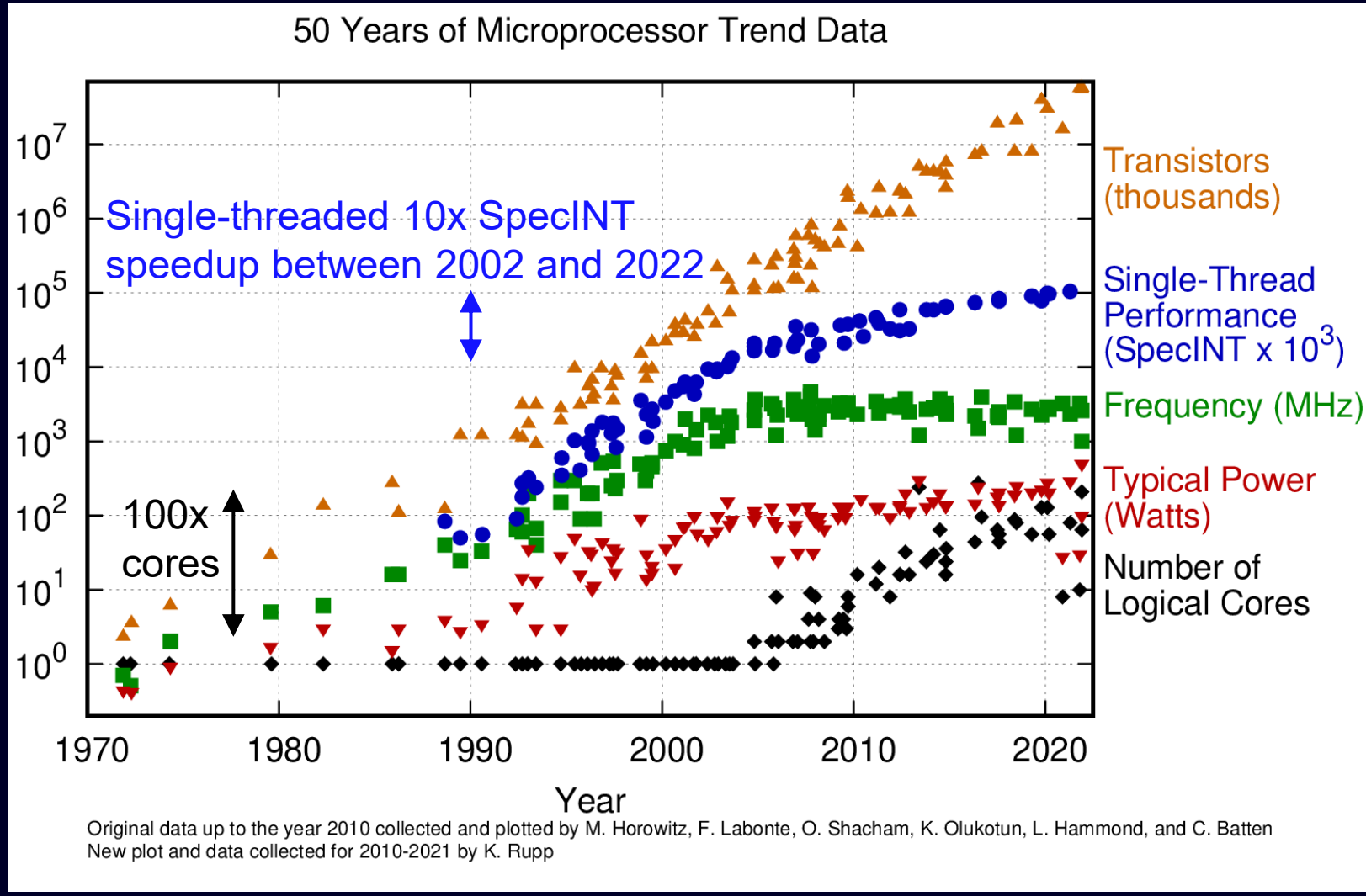
"Why don't commercial EDA tools scale better?"

"Why don't you speed up industrial SAPR software with GPU acceleration?"

Digital IC design flow throughput has scaled poorly vs. compute speedup

- Single-threaded performance has increased 3x to 10x in the past 20 years.
- Multi-threaded max performance has increased 100x in the past 20 years.
- And now we have GPU acceleration …

- But only 10x SAPR speedup in 20 years.
- Commercial EDA flows still do not achieve 24 hours per million gates from RTL high-level description to GDS layout.

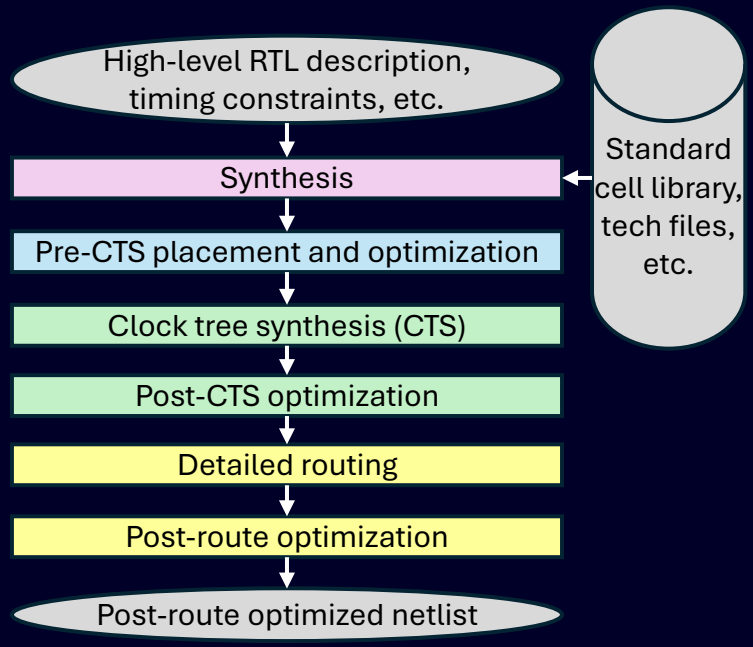


K. Rupp. 50 years of microprocessor trend data. https://github.com/karlrupp/microprocessor-trend-data

What limits speed up of commercial SAPR tools?

- Most SAPR algorithms are not highly parallelizable, with a few exceptions such as global placement, static timing analysis (STA), and track routing.
  - Parts are highly multi-threaded, e.g., STA using 40 threads, but average thread usage in a flow may be only 10 threads.
- Must synchronize, avoid multi-threaded conflicts, avoid non-determinism
  - Mutual exclusion (mutex) locks when making database changes to prevent conflicts, avoid read/write races for timing & resource utilization updates, etc.
- Various overheads such as file I/O for database read/write, & cache misses.

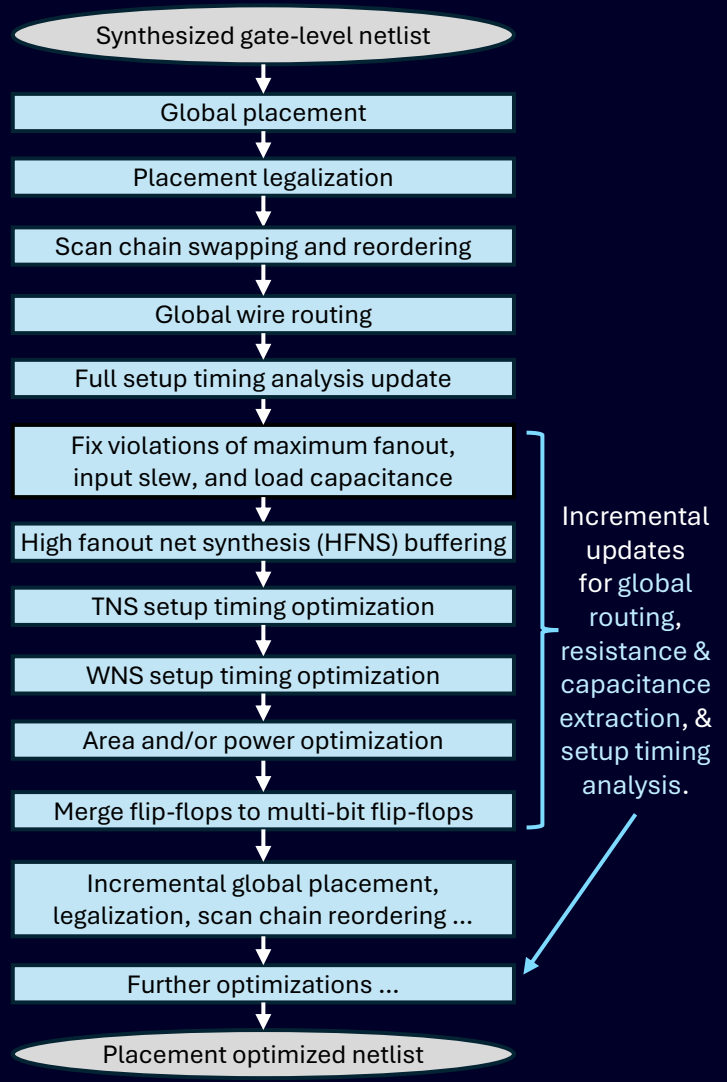
- GPU acceleration can degrade results vs. commercial, though it is improving: Y.Lu et al., C3PO: Commercial-Quality Global Placement, ASPDAC 2026.
- Customers will not usually accept worse quality of results to speed up the flow.



High-level SAPR flow diagram (omits floor planning, sign-off analysis, layout verification ...)

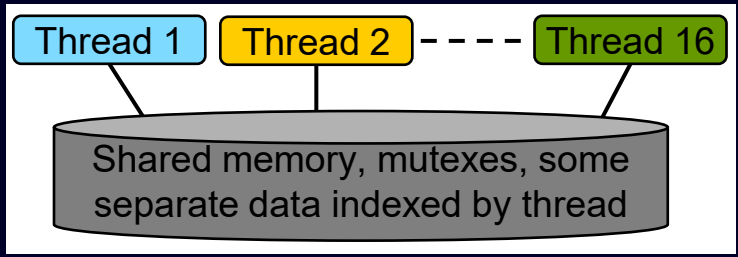
Speeding up one SAPR engine provides little overall benefit

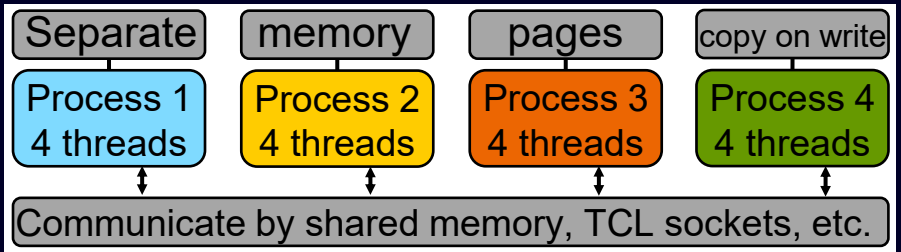
- Commercial SAPR code has millions of lines of code & 100+ different engines.
- Each engine usually contributes at most 5% to 10% of the total flow runtime.
- Amdahl's Law: $Speedup = \frac{1}{(1-P)+\frac{P}{n}}$, fraction P parallelized by speedup n.
- If 90% of global placer runtime is sped up **10x** on GPU, and if it is 10% of pre-CTS (clock tree synthesis) runtime, so **11%** pre-CTS speedup, but pre-CTS is about 30% of SAPR runtime, so **3%** full flow speedup.
- Must speed up many engines to really speed up the full flow!

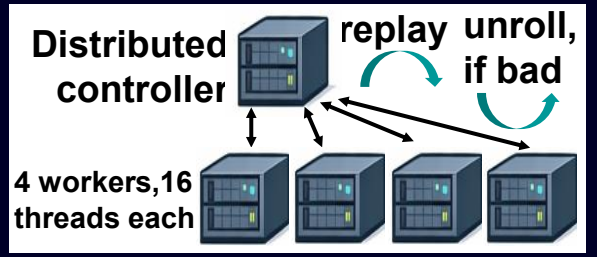


Simplified pre-CTS flow with some engines used in blue

Commercial SAPR software speed up techniques







- Multi-threading, multi-processing, and distributed computing are the main parallelism approaches in SAPR.
- Multi-threading is faster, but difficult to implement if legacy code is single-threaded.
  - Parallel optimization of small windows of logic is a typical approach.
  - Threads have separate copies of some data to avoid contention and the overhead for mutexes.
- Spatially separate regions to avoid contention.
- Connected paths in same region to optimize better.
- Boundary crossings must have be constrained to avoid mismatched optimizations on either side.
- Need to avoid conflicting changes, or quickly fix them – e.g., fast TNS timing optimization after area opt.
- Multi-process and distributed approaches often use slower replay and unroll to prevent problems.

What further can be done to better speedup SAPR software?

- Monolithic SAPR flows, running a sequence of steps, waiting for each to complete, greatly limit parallelism.
- Instead, we could run different optimizations in different parts of a design in parallel. For example, perform area or hold minimization on paths with setup slack, while doing setup timing optimization on critical paths.
  - If thread utilization is low and there's available memory bandwidth, these can be done on same server.
  - Otherwise, a distributed computing approach would be needed, with higher overhead costs.
- Can also do multi-objective optimization, e.g., gate-sizing for area, power, and timing simultaneously.

- Large designs could be automatically split to run sequences of steps in parallel on each, autonomously.
  - Manual approach in industry for many years, but lots of work for constraint budgeting et al.
- Minimum cut partitioning is a simple approach to separate regions, but can split timing critical paths.
  - Current state-of-the-art min-cut approaches are TritonPart ICCAD 2023, available in OpenROAD, SHyPar TCAD 2026, and DAC 2026 also has a good paper on GPU acceleration of min-cut.
  - And see Wednesday's ISPD 2026 talk on Modern Hypergraph Partitioning.
- Recombine + repartition to optimize further, or perform top-level chiplet optimization for global paths.

- It is an open academic research problem to better automate this.

Example:

Split large scan partitions to speed up swapping between scan chains to minimize scan chain wire length

The scan chain optimization problem

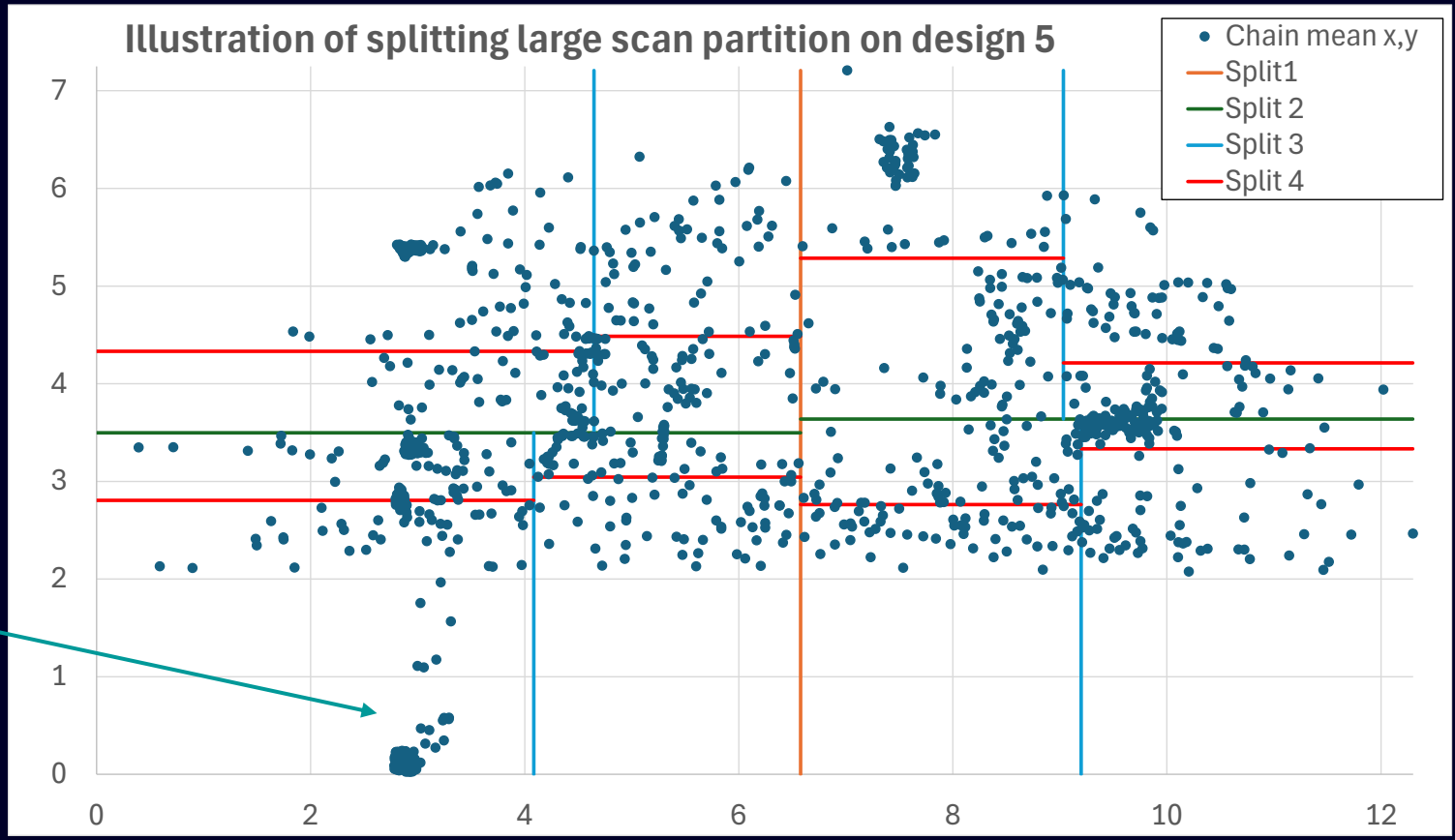
- Scan chain optimization is a multi-depot multiple-traveling salesman NP-hard problem.

- Objective is to minimize Manhattan wire length of the scan chains, while avoiding violating maximum bit counts for each scan chain, and balancing bit counts of chains in each partition.
- Scan DEF format specifies MAXBITS per chain, and which scan partition each scan chain is in.
- Each scan partition can have many scan chains, which each have hundreds of flip-flop cells.
- Can reorder FLOATING cells on a chain, or swap them between chains in the same scan partition.
- ORDERED segments must be kept together.

Simple algorithm for splitting very large scan partitions

- Calculate mean x,y from chain's cell locations.
- Alternate bisecting horizontally and vertically scan chains in each partition halving bit count.
- Stop when below 10^5 bits in sub-partition.
- Swapping between scan chains is limited to within the sub-partitions to save runtime.

- Some scan chains have much lower bit count, so some sub-partitions appear uneven.



Scan reordering/swapping benchmark characteristics

- Large industrial design benchmarks with runtime issues for scan reordering and swapping.
- All of them have very tight maximum bit count (MAXBITS) specified per chain in the scan DEF.
- Some have high standard deviation in MAXBITS per chain, so difficult to balance cells across chains.
- Some have insufficient MAXBITS capacity for the partition, so impossible to meet MAXBITS of each chain.
- Design 2 has 83133 multiplexers in ORDERED pairs with registers, so total scan bit count < # scan nodes.

- Splitting the larger scan partitions to sub-partitions of less than 10^5 bits can save runtime for scan swapping.

	Process	Standard				Average	Std. dev.	# Scan node		Scan chain	Highest	After splitting	
	tech	cell	Macro	# Scan	# Scan	chain	chain	excluding	Total scan	bit count	partition	# Scan	Largest part.
Test	node	count	count	partition	chain	MAXBITS	MAXBITS	buff/inverter	bit count	capacity	bit count	Partition	split into
1	16 nm	6246021	221	228	15693	124.2	12.3	1949611	1949611	1949604	1921860	259	32
2	3 nm	3561829	295	16	2259	440.9	83.8	1011653	996318	995903	527399	25	8
3	5 nm	11104715	25	6	28556	36.1	74.2	1029237	1032056	1032057	991397	21	16
4	5 nm	5205045	25	6	4799	336.3	28.5	1226161	1613775	1613774	1573117	21	16
5	14 nm	6935238	519	1	1093	840.1	429.2	818251	918282	918282	918282	16	16

Scan reordering/swapping runtimes in minutes

Runtimes reported across three different software versions:

- **Initial** with major runtime problems on partitions with scan chains with uneven MAXBITS, or infeasible
- **Fix** to address these runtime issues and stop when the scan chains cannot be balanced to meet MAXBITS.
- **Refactored** to speedup comparing scan nodes to determine if can reorder them, check hierarchy, etc.

- Splitting large partitions gave a big speedup when there were runtime problems, but much less so when these issues were fixed. With the fix and refactoring, runtimes were similar vs. initial with split partitions.

	Unsplit partitions			Split	Split partitions			Split speedup		
Test	Refactored	Fix	Initial	Time	Refactored	Fix	Initial	Refactored	Fix	Initial
1	28.9	38.6	88.1	0.6	16.0	21.5	21.3	1.7x	1.8x	4.0x
2	26.8	1961.5	stuck	0.3	7.3	26.4	57.6	3.5x	73.5x	high
3	50.5	70.3	186.7	1.0	26.2	28.5	42.4	1.9x	2.4x	4.3x
4	31.5	36.8	51.0	0.8	21.9	30.5	29.4	1.4x	1.2x	1.7x
5	12.0	9.8	82.6	0.4	8.4	9.7	9.2	1.4x	1.0x	8.5x
							Geometric mean:	1.8x	3.2x	4.0x

Splitting large scan partitions doesn't improve multi-threading that much due to overheads and the most time consuming sub-partition runtimes (Amdahl's Law!)

- As shown in the table on the left, the CPU to real world runtime ratios were poor.
- Splitting the large scan partitions allows more parallelism, as each sub-partition can do swapping in parallel.
- Though we split the large partitions into 8+ sub-partitions, the speedup by splitting is at most 4.2x.

- The speedup is limited by the most time consuming sub-partition (most difficult MAXBITS constraints), and by Amdahl's Law due to other runtime overheads. As core code is sped up, other overheads dominate.

Ratio of CPU time to real world time for core swap/reorder portion

	Unsplit partitions			Split partitions			Factor vs. unsplit		
Design	Refct	Fix	Init	Refct	Fix	Init	Refct	Fix	Init
1	1.43	1.32	0.95	3.21	3.43	3.93	2.2x	2.6x	4.2x
2	1.52	1.00	n/a	4.29	2.03	1.52	2.8x	2.0x	n/a
3	1.39	1.37	1.11	3.76	4.43	3.60	2.7x	3.2x	3.3x
4	1.81	2.02	1.91	3.85	4.40	4.27	2.1x	2.2x	2.2x
5	1.73	3.72	1.17	3.01	3.73	3.62	1.7x	1.0x	3.1x

Portion of runtime not in core swap/reordering

	Unsplit partitions			Split partitions		
Design	Refct	Fix	Init	Refct	Fix	Init
1	58%	57%	28%	72%	60%	61%
2	21%	0.3%	n/a	57%	22%	12%
3	34%	22%	9%	59%	53%	44%
4	47%	42%	40%	60%	50%	52%
5	53%	63%	8%	69%	66%	64%

Impact on total wire length summed across the scan chains

- Total scan chain Manhattan pin-to-pin wire lengths, normalized to the best result, are tabulated below.
- The largest increase in scan chain wire length due to splitting is 9.7%,
  on average 1.8% higher, excluding worse initial unsplit results (e.g., designs 3 and 4).
- Split sub-partitions reduced wire length on designs 3 & 5 as less issues meeting difficult MAXBITS constraints.
- Refactoring reduced design 5 wire length by fixing inconsistency in (dis)allowing reconnection of scan nodes driven by negative polarity buffer/inverter chain, as 25% of scan nodes had negative polarity.
  - Most designs have relatively few cases of negative polarity buffering, so they benefited little from this.

	Unsplit partitions			Split partitions			Percent higher vs. unsplit		
Test	Refct	Fix	Initial	Refct	Fix	Initial	Refct	Fix	Initial
1	1.00	1.00	1.00	1.10	1.10	1.09	9.5%	9.5%	9.7%
2	1.00	1.00	n/a	1.09	1.09	1.09	8.8%	8.9%	n/a
3	1.00	1.00	1.14	0.89	0.89	0.87	-11.2%	-11.2%	-23.4%
4	1.00	1.00	6.13	1.08	1.08	1.08	8.1%	8.1%	-82.3%
5	1.00	1.88	1.93	0.92	1.80	1.81	-8.5%	-4.4%	-6.2%
					Average:		1.3%	2.2%	-25.5%

Conclusions

- Speeding up SAPR tools requires many fixes with small incremental improvements.
- Long runtime poles in optimization limit multi-threading utilization and the total runtime.
- Difficult for much of the SAPR code to achieve high parallelism, though algorithms & code can be sped up.
- Find & fix test cases with bad runtime scaling! Fixing such issues speeds up the flow on other designs.

Traditional monolithic sequences of SAPR flow steps in SAPR should be reconsidered.

- Make better use of multi-objective optimization, and do more in parallel on sub-portions of the designs.
- Need to sub-partition large designs into pieces that can be done in parallel across multiple servers.
- Recombine + repartition to optimize further, and/or perform top-level “chiplet” optimization for “global” paths.
  - Some sacrifice in quality-of-results may be necessary to achieve good speedups.

Splitting large scan partitions demonstrated some of these trade-offs on cases with runtime issues for scan wire length minimization

- 2.9x geomean speedup by fixing the scan reorder/swap runtime issues with difficult MAXBITS constraints
- 2.6x additional speedup with refactoring to quickly determine if can reorder, check hierarchy constraints, etc.
- Splitting the largest scan partitions gave up to 3.5x speedup, with geomean 1.8x speedup versus the refactored runs, at the cost of 1.8% higher wire length on average.
- Together the above give geomean speedup of an order of a magnitude. This can be mostly achieved by code improvement, avoiding worse scan wire length, rather than from splitting to improve multi-threading and mitigate issues with $O(n^2)$ quadratic runtime growth or worse.
- Development time of about 2.5 worker months for scan code improvements vs. 2 weeks for splitting partitions.

- The thread utilization remained poor, lacking good parallelism, only 2.3 threads on average even with splitting.
- The refactored code gave only 25% average runtime reduction for typical designs.
- The scan partition splitting is not used in Aprisa's commercial flow due to somewhat worse scan wire length.
- With overheads taking 43% of scan reorder/swap runtime, 64% with split scan partitions, further runtime improvement would need to focus increasingly on the overheads.

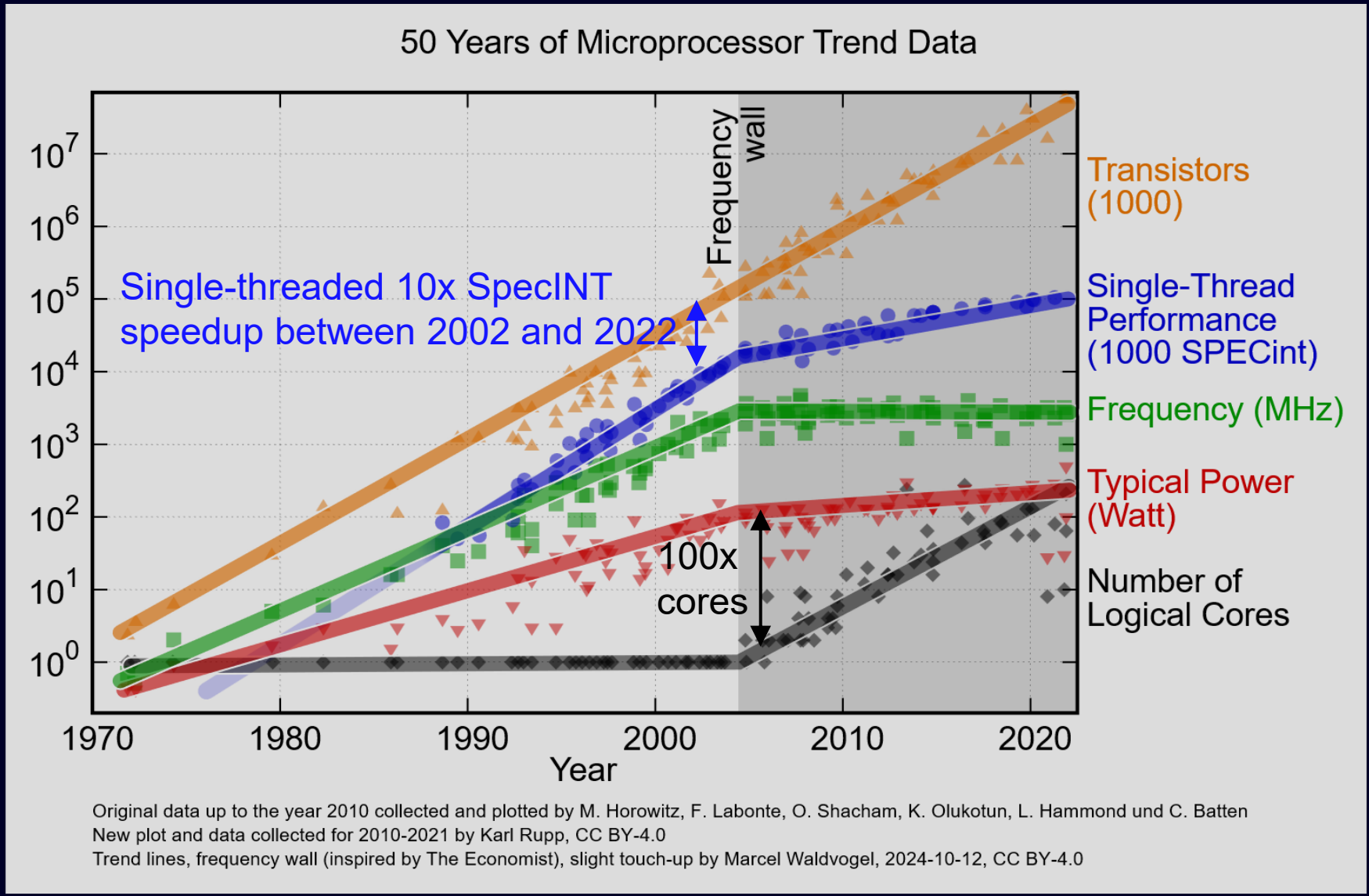
Acknowledgements

- Thanks to everyone from Aprisa place-and-route who helped identify runtime issues, discuss my solutions to some of these, and to fix such bugs.
- Software architectural analysis with Nitro R&D aided seeing solutions beyond standard practice.
- Conversations on runtime scaling challenges with industry and academia have also been very helpful.

Backup slides

Microprocessor scaling trend lines over the past 50 years

- Clock frequency wall of about 5GHz from the mid-2000s, due to power, slowed single-thread performance scaling.
- High-end server 10x speedup in single-thread performance (SPECint) since mid-2000s with improved architecture, better branch prediction, faster memory, etc., improving instructions per cycle.
- With transistor count still increasingly rapidly with Moore's Law, many more logical cores are now on servers, with up to about 100x multi-threaded performance for massively parallel applications.



M.Waldvogel. Was Prozessoren und die Frequenzwand mit der Cloud zu tun haben. https://marcel-waldvogel.ch/2024/10/12/prozessoren-cpu-frequenzwand-cloud/