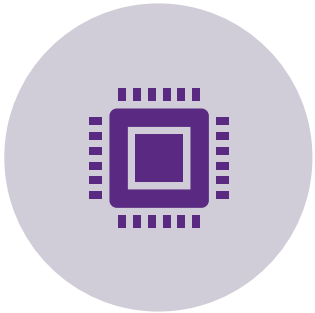
Distributed-STA: A scalable solution for accurate STA signoff in large multi-billion transistor designs

Hima Arumbakar, Microsoft Corporation
Arvind Sridhar, Microsoft Corporation

Agenda

- FullChip STA Challenges
- Traditional hierarchical methods
- HyperGrid- DSTA Flow
- Runtime & QoR comparisons
- Key Findings
- Highlights
- Limitations & Future Enhancements
- Q & A

Full-chip Timing SignOff Challenges



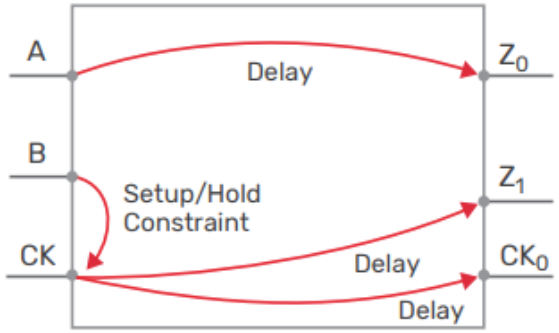
Modern-day designs are increasing in **size and complexity**, putting strain on high-capacity compute resources to run **Full-chip STA**. Sometimes, it is impossible when design size is scaled-up.

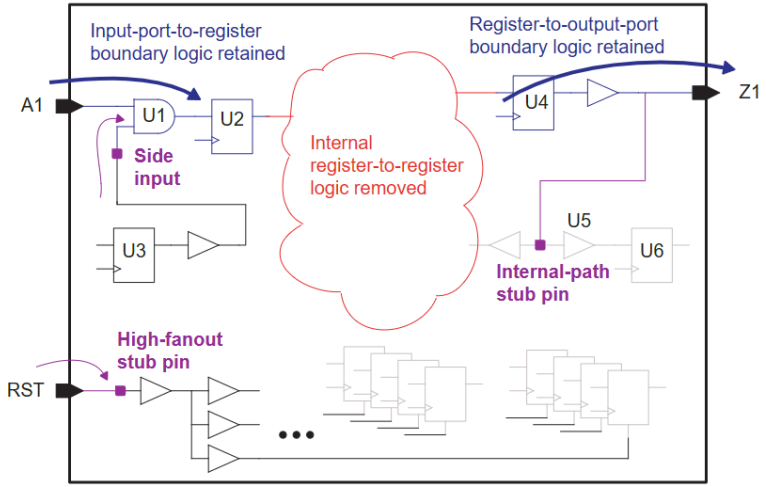
Traditional hierarchical methods like **ETM and Hyperscale modeling** may not provide the accuracy and confidence required for final sign-off STA

We need a **scalable technology** for running Signoff runs without compromising **accuracy**.

Distributed-STA is a scalable solution that partitions large designs and performs multi-threaded analysis, providing both uncompromising accuracy and runtime benefits.

Traditional hierarchical methods





Extracted Timing Models

- Highly abstracted interface timing model
- Extracts only Input/Output timing arcs and related clk port information
- Compact, but less accurate
- Significant effort required to ensure generation of high quality ETM

HyperScale Models

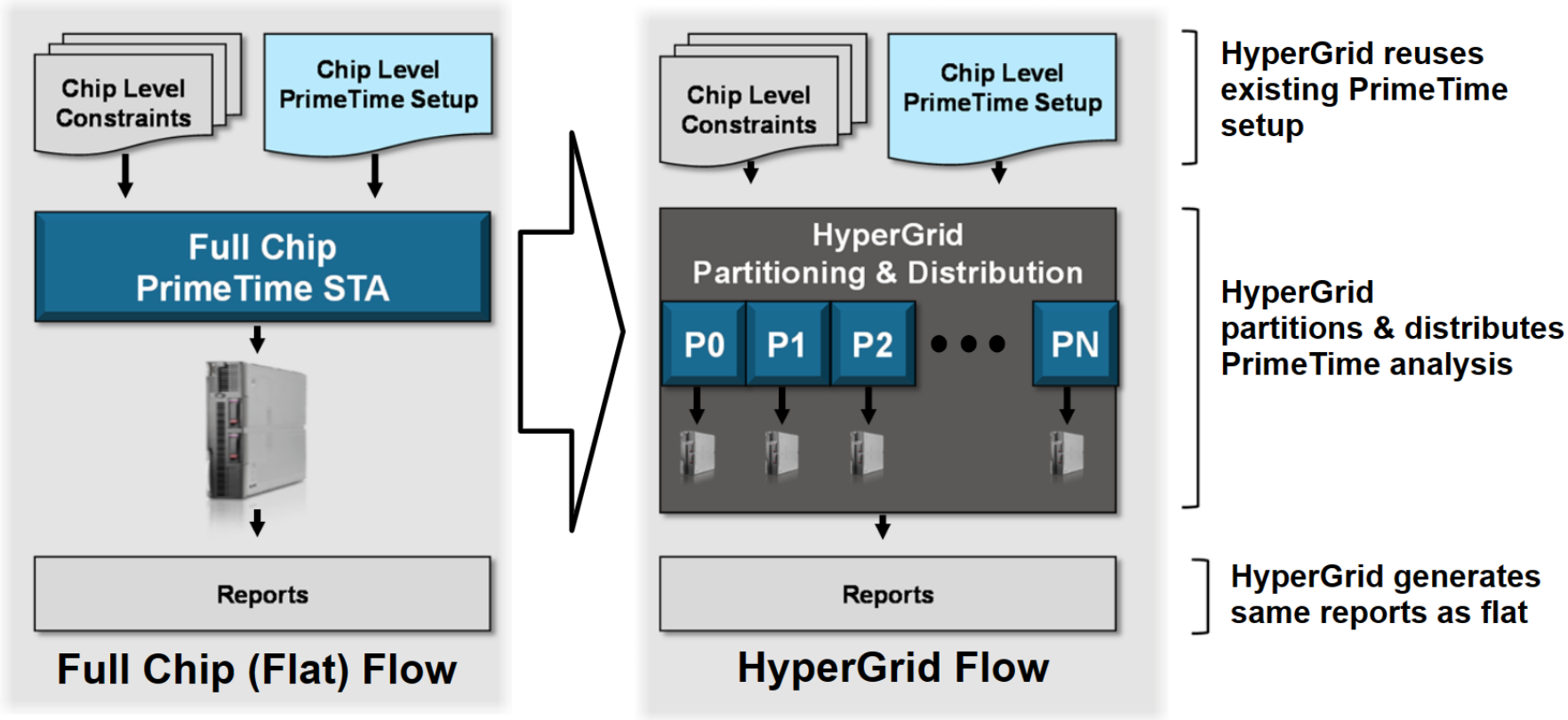
- Abstracted model highly dependent on physical hierarchies and their interactions, affected by MIM dominant logic.
- Extracts in-depth timing information pertinent to boundary logic
- Larger and more accurate than ETM
- Needs manual database and context management required for reliable rollups

HyperGrid Distributed Analysis

HyperGrid Distributed Static Timing Analysis (DSTA)

HyperGrid is a highly scalable technology which automatically partitions and distributes the design and performs timing analysis. It uses distributed and reusable hardware resources to reduce runtime, memory usage and cost.

Source: Synopsys

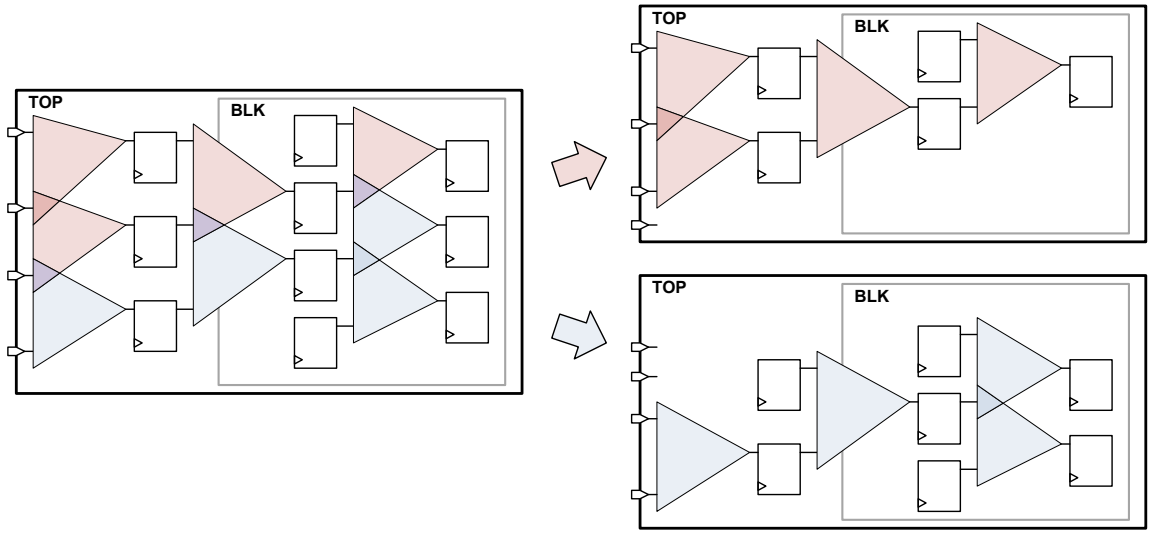


DSTA Flow

Partitioning

Splits the design into partitions, using fanin logic cones to clock and data endpoints, transcending physical hierarchies.

Pre-scans the clock network and the constraints to create well-balanced partitions in terms of gate-count and memory



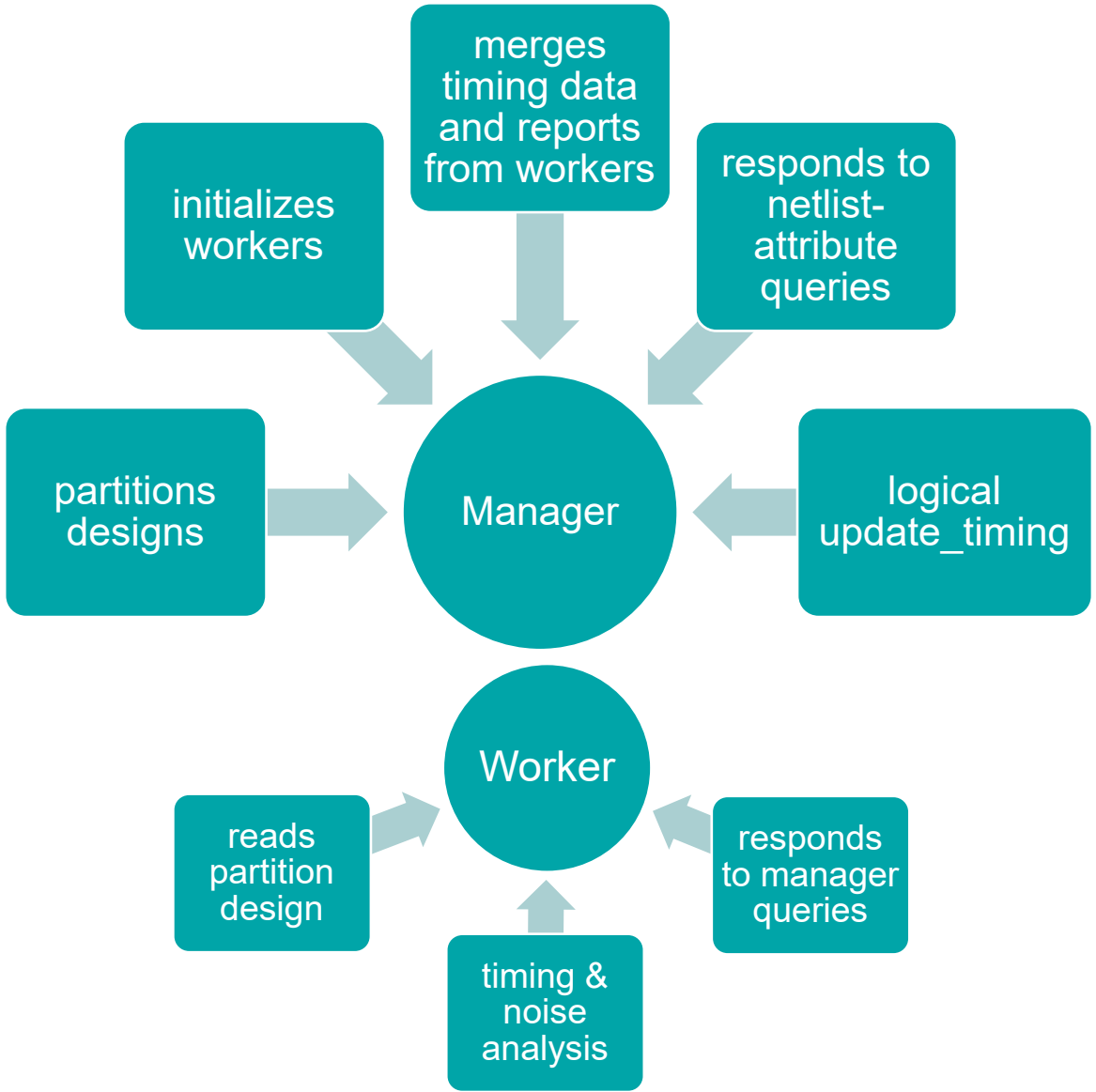
Fanin logic cone include:

- Side load stages: models receiver loads outside the current partition
- Aggressor stages: models coupling capacitance for accurate noise and timing analysis

Due to the nature of the partitioning, same startpoints could be included in multiple partitions. And also, different partitions can contain different portions of the same block

DSTA Flow

Manager and Worker Processes



- **Manager** retains logical netlist representation and **Worker** contains all timing information
- **Attribute caching** – HyperGrid utilizes caching of frequently accessed attributes within iterative constructs at the manager level allowing worker partitions to retrieve and execute faster loop iterations.
- The **`cache_distribute_attribute_data`** command enables explicit attribute caching at the manager processes
- **Distributed steps:**
  - read parasitics
  - update_timing
  - report generation
- **Partially distributed steps:**
  - read and link designs
  - read sdc

DSTA Flow

Running DSTA

- **distributed_enable_analysis**: Enables HyperGrid's distributed analysis feature.
- **distributed_working_directory**: Sets the path for the working directory; defaults to **dsta_working_dir**.
- **set_host_options** Configures remote worker processes and **start_hosts** starts the workers
- **start_dsta**: Runs distributed analysis and executes the analysis script with master-worker coordination.
- Restoring distributed session re-creates original configuration with same partitions and worker resources

HyperGrid Script

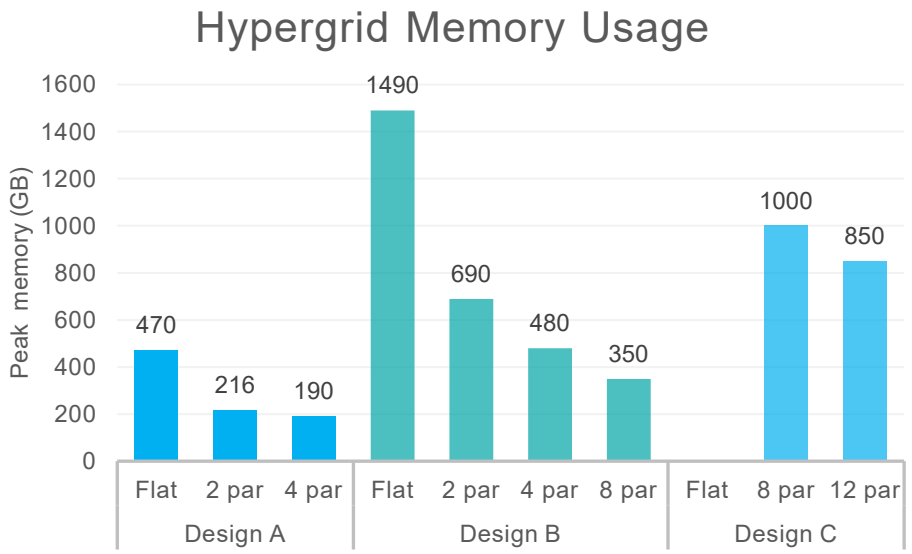
```
# Enable partitioning and distribution
set distributed_enable_analysis true
set distributed_working_directory $dsta_work_dir

# distribution setup
set_host_options -num_process $nProc -max_core $mCore …
…
start_hosts

start_dsta -script flat_run.pt -no_of_partitions $nProc
```

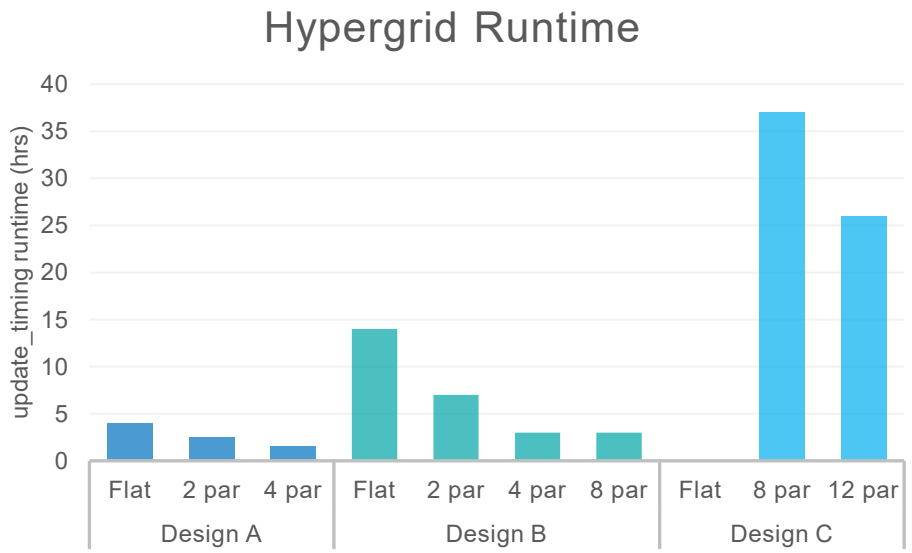
Existing Flat Script

```
# Existing STA flow
set_link_path “* lib_0p5v.db”
read_verilog TOP.v

read_parasitics …

read_sdc TOP.sdc

update_timing

report_timing -max 10000 -path full_clock

save_session my_session
```

Results

Runtime Compare





	Gate count (million)	Partitions	Peak Memory (GB)	Overall runtime (hrs)	update_timing runtime (hrs)
Design A	102	Flat	470	33.5	4
		2	216	24	2.5
		4	190	20	1.5
Design B	390	Flat	1490	71	14
		2	690	41	7
		4	480	37	3
		8	350	33	3
Design C	1700	Flat	~6500	~336	60
		8	1000	148	37
		12	850	144	26

- 3x faster update_timing
- 1.5x faster overall runtime
- 65% less Peak memory consumption

QoR Compare

- +/- 2 ps accuracy on setup
- +/-1 ps accuracy on hold
- 100% match for DRCs, Noise, check_timing and parasitic annotation

Max Delta Slack (ps)	No of Endpoints
0	97538
0 to 1	2359
1 to 2	103

Min Delta Slack (ps)	No of Endpoints
0	99117
0 to 1	875
1 to 2	8

Flat GBA

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-1.5661	-1.5661	-0.1478	-0.1036	0.0000
TNS	-2412.6240	-2324.2338	-14.5420	-73.8482	0.0000
NUM	368213	364883	858	2472	0

Hold violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.2147	-0.2147	0.0000	0.0000	0.0000
TNS	-2424.2157	-2424.2157	0.0000	0.0000	0.0000
NUM	61303	61303	0	0	0

DSTA GBA

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-1.5661	-1.5661	-0.1478	-0.1036	0.0000
TNS	-2412.6199	-2324.2296	-14.5420	-73.8482	0.0000
NUM	368213	364883	858	2472	0

Hold violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.2147	-0.2147	0.0000	0.0000	0.0000
TNS	-2424.2156	-2424.2156	0.0000	0.0000	0.0000
NUM	61303	61303	0	0	0

Flat PBA

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-1.5480	-1.5480	-0.1450	-0.0899	0.0000
TNS	-163.9540	-77.8353	-14.0995	-72.0192	0.0000
NUM	15731	12477	811	2443	0

Hold violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.2124	-0.2124	0.0000	0.0000	0.0000
TNS	-2195.3896	-2195.3896	0.0000	0.0000	0.0000
NUM	34536	34536	0	0	0

DSTA PBA

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-1.5480	-1.5480	-0.1450	-0.0899	0.0000
TNS	-163.9533	-77.8345	-14.0995	-72.0192	0.0000
NUM	15731	12477	811	2443	0

Hold violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.2124	-0.2124	0.0000	0.0000	0.0000
TNS	-2195.3920	-2195.3920	0.0000	0.0000	0.0000
NUM	34536	34536	0	0	0

Key Findings

Our comprehensive analysis of various gate-count designs has led to preliminary estimations for optimal partitioning and computational needs. These guidelines aim to provide a solid starting point for users, keeping in mind the specificities of each design's clocking, constraints, and modeling requirements.

- ❑ Optimal gate count per partition: **100-150 million**
- ❑ Optimal memory requirements per partition: **400-500GB**
- ❑ expected link_design runtime: **1 hr * number of partitions**
- ❑ Linking runtime increases with increased partitioning
- ❑ Partitions less than 100 millions gate count do not yield faster runtimes for distributed steps, but degraded linking runtime as the manager is working harder to do more partitioning
- ❑ Although further partitioning beyond optimal range flatlined with respect to runtimes, it may be preferred in some cases as lower memory resources might be more readily available in your server pool
- ❑ Accuracy was not affected by the number of partitions from our **QoR comparison** of different designs

The provided calculations are highly dependent on the specific design and its unique modelling. Users are encouraged to adjust these guidelines based on their project's needs and available resources.

Summary

Highlights

Highlights

- Highly scalable technology that can handle multi-billion gate count designs
- 3x faster timing update and report generation runtime
- 65% lower memory consumption
- +/- 2ps accurate with full visibility to the design
- Automatic partitioning and distribution under the hood
- Same setup as flat STA, no need to manipulate parasitics, constraints, etc.
- Quick and easy to launch DSTA runs without child database/context dependencies
- Supports GPD stitching and avoids large flat SPEF
- All latest features are supported like DSLG, distance POCV, ECO fixing with PrimeClosure, Hyperscale models and context, etc.

Challenges & Future Enhancements

Limitations/Challenges

- Linking in design takes significantly longer compared to standard flat runs due to the detailed pre-scanning of clock networks and SDC for efficient partitioning.
- Requires multiple iterations to optimize the partitioning and compute resource needs specific to your design and machine pool
- Understanding the manager-worker interactions is crucial for runtime optimization.
- DSTA is heavily multi-threaded and is very sensitive to job-scheduling queues and policies which has its own multi-threading algorithms
- Certain PrimeTime commands like report_qor and get_timing are notably slower in DSTA.
- Addressing issues or performance bottlenecks in DSTA flow can take longer given the size of the design
- For sourcing large tcls after update_timing, load_constraints is essential for significant speedup in DSTA
- Does not currently support features like SMVA/DVFS, ECO fixing (automatic or manual), netlist editing commands and HyperTrace PBA

Planned Enhancements in 2023.12-SP*

- HyperScale context with MIM merging is available, but there are further enhancements in progress for this
- Improvements aimed at get_timing_paths, report_qor, and similar commands for better runtimes.
- Enhanced pre-scanning of SDC during partitioning is in development to improve both memory usage and runtime performance.


THANK YOU
YOUR
INNOVATION
YOUR
COMMUNITY