RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning

By Haoyuan Wang, Scott Beamer

Presenter: Boyang Zhang, Department of ECE, UW Madison

bzhang523@wisc.edu

About me

- Education
  - BS in School of Electronic and Information, South China University of Technology, China, 2020.
  - MS in Department of ECE, Rutgers University, 2021.
  - Second-year PhD student (supervisor: Prof. Tsung-Wei Huang).
- Current research
  - Incremental task graph partitioning.
  - A paper[1] of task graph partitioning was accepted to DAC'24.
  - I am submitting a paper to ICCAD'24.

[1] Boyang Zhang, Dian-Lun Lin, Che Chang, Cheng-Hsiang Chiu, Bojue Wang, Wan Luan Lee, Chih-Chun Chang, Donghao Fang, and Tsung-Wei Huang, "G-PASTA: GPU Accelerated Partitioning Algorithm for Static Timing Analysis," *ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2024

Outline

- Introduction to Register Transfer Level(RTL) simulation
  - What is RTL simulation?
  - Why partitioning?
  - Partitioning challenges?
- Introduction to RepCut
  - Partitioning algorithm walkthrough
  - Scheduling the partitions
  - Experimental results
- Current research
  - Limitations of RepCut partitioning
  - Our partitioning

Introduction to RTL simulation

What is RTL simulation?

- A process to simulate the behaviors of a Register Transfer Level(RTL) design to verify its functionality.
- **Event-driven**(more accurate) or **full-cycle**(less time-consuming).
- Full-cycle simulator: Verilator[2], **ESSENT**[3](unoptimized).

[2] https://github.com/verilator/verilator
[3] https://github.com/ucsc-vama/essent

Why partitioning?

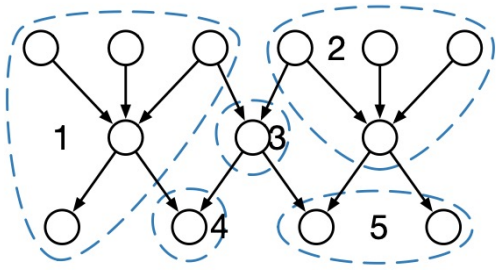
- RTL simulation is very **time-consuming** for large designs.

- RTL simulation in parallel?

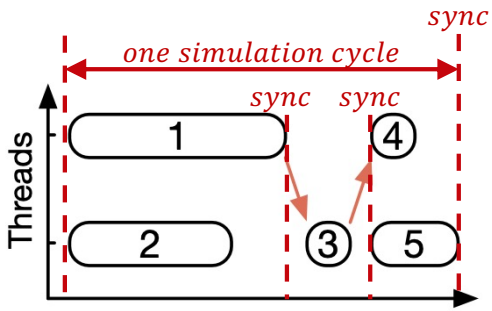
- Represent the design as a task dependency graph(**TDG)**.
  → Node: the simulation task of a logic gate/register.
  → Edge: the connection between logic gates/registers.
  → Too **fine-grained.**
  → Partition the TDG. (Output: A coarsened TDG)

Partitioning challenges?

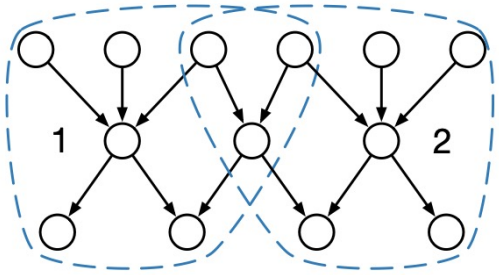
- Goal: **Balanced partitions** and **minimal synchronization effort**.



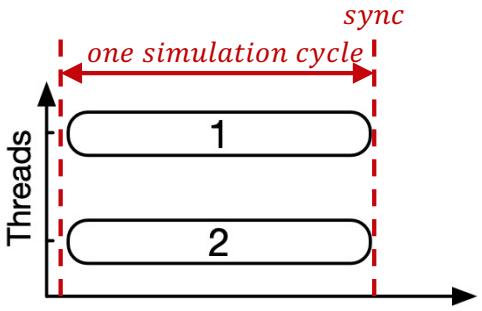
(a) Verilator Partitioning



(b) Verilator Schedule



(c) RepCut Partitioning



(d) RepCut Schedule

Introduction to RepCut

Introduction to RepCut

- An example of RepCut partitioning.
- Schedule the partitions.
- Experimental results.

Introduction to RepCut – An example

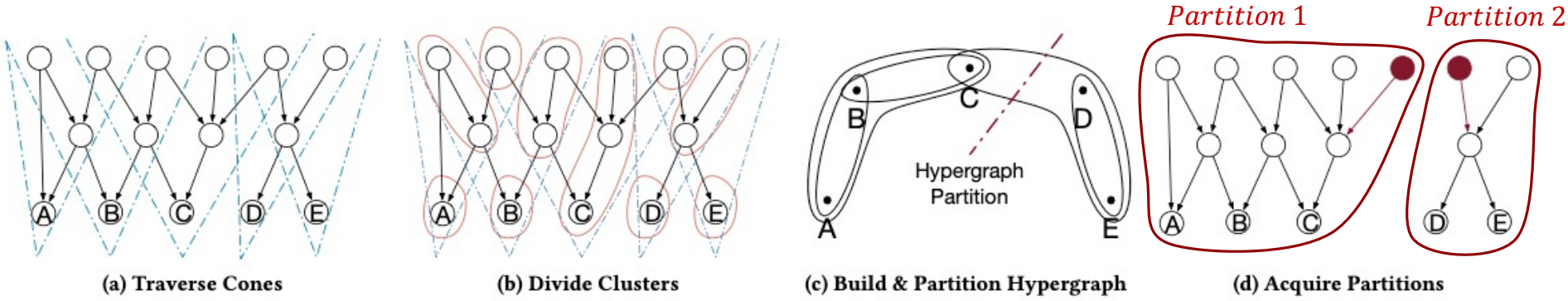
- Input: A fine-grained TDG.
- Output: A coarsened TDG.

→ Build “cones”.

→ Coarsen the graph.

→ Build hypergraph and partition.

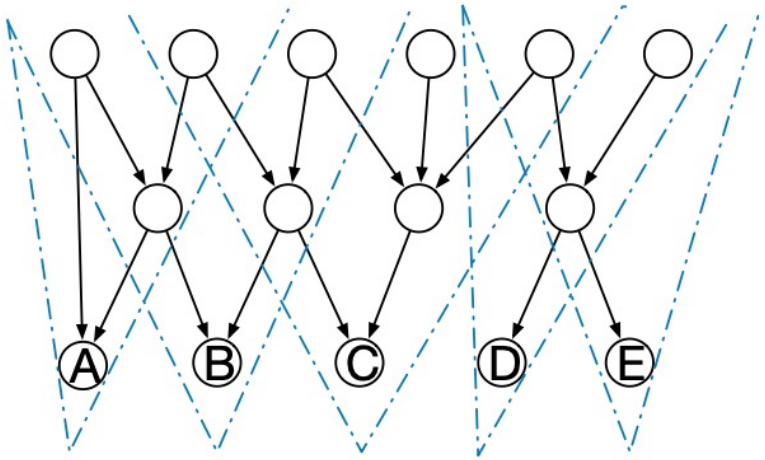
→ Acquire partitions.



(a) Traverse Cones (b) Divide Clusters (c) Build & Partition Hypergraph (d) Acquire Partitions

Introduction to RepCut – Build “Cones”

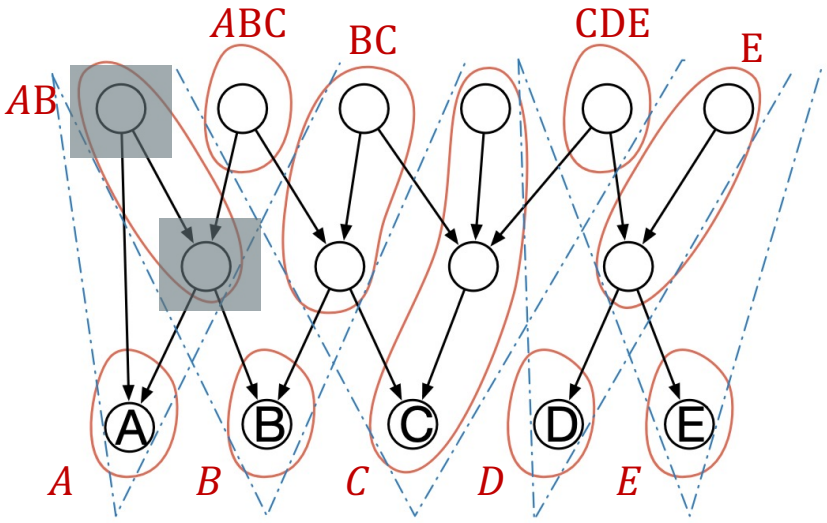
- Start from sink nodes, traverse nodes within its ”cone”.



(a) Traverse Cones

Partitioning Walkthrough – Coarsen the graph

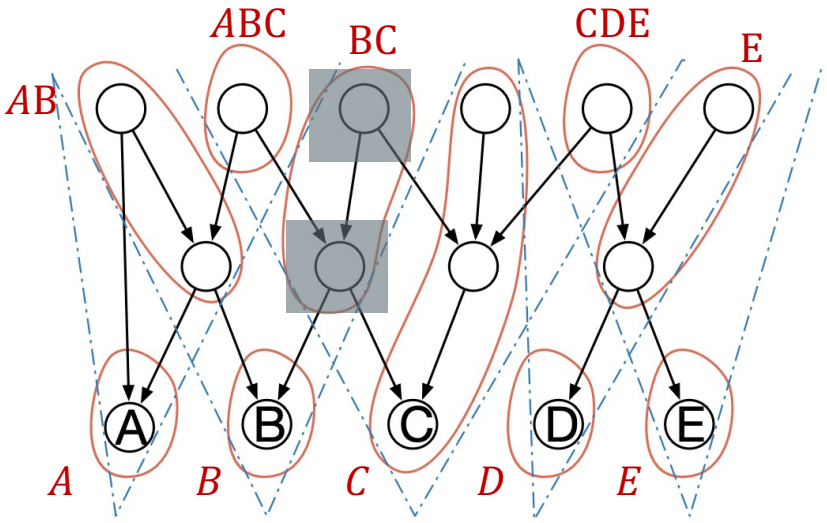
- Cluster nodes according to ”cones”.



(b) Divide Clusters

Partitioning Walkthrough – Coarsen the graph

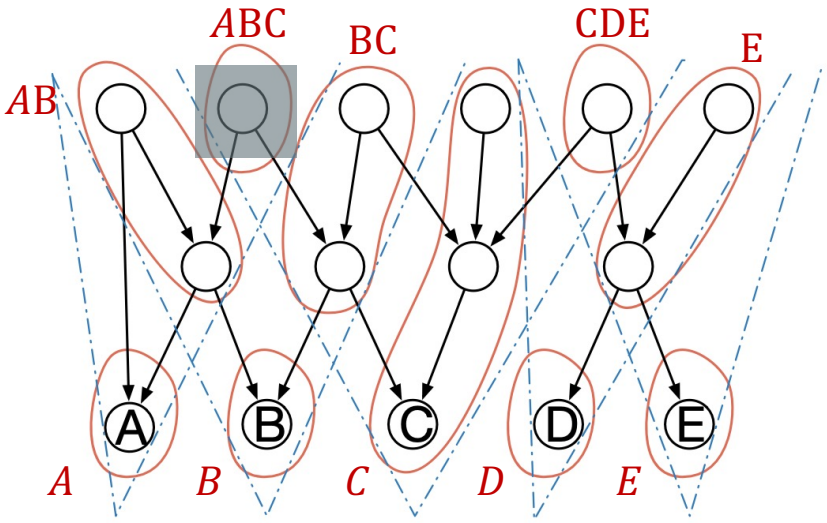
- Cluster nodes according to ”cones”.



(b) Divide Clusters

Partitioning Walkthrough – Coarsen the graph

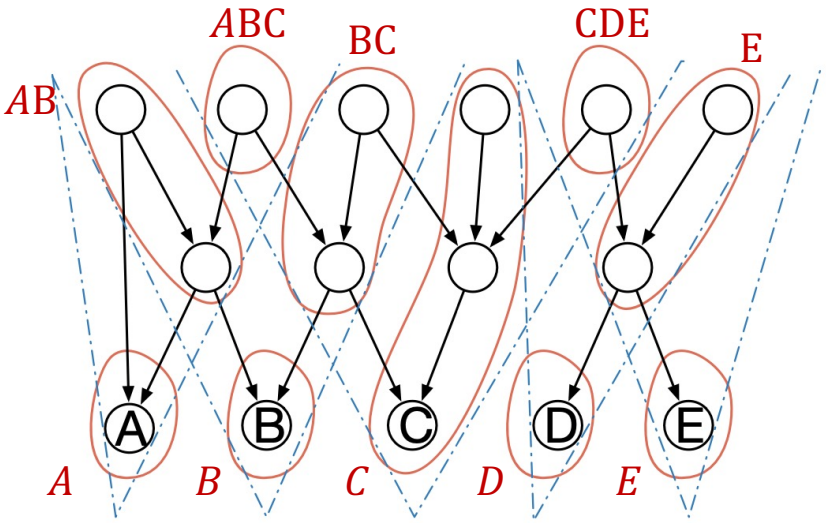
- Cluster nodes according to ”cones”.



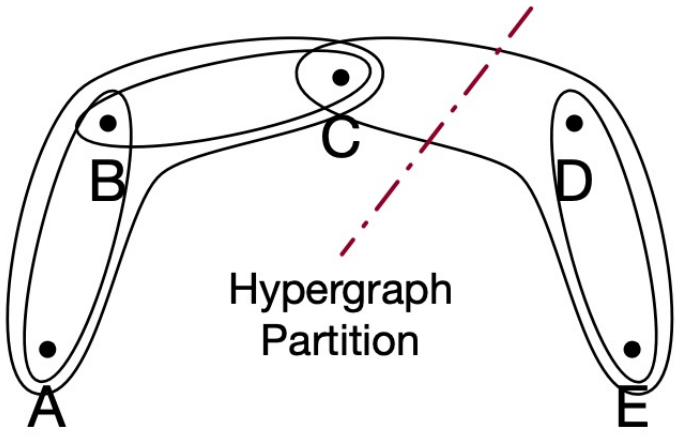
(b) Divide Clusters

Partitioning Walkthrough – Build hypergraph

- Build hypergraph according to clusters.



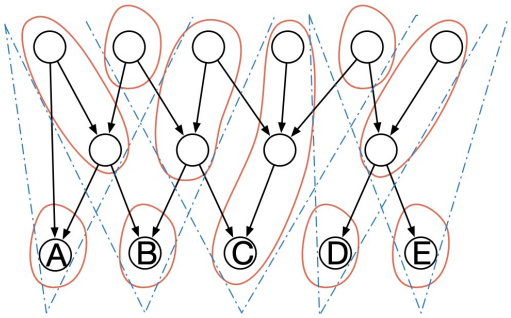
(b) Divide Clusters



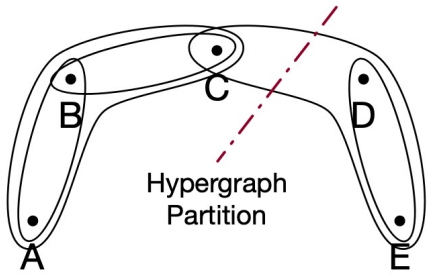
(c) Build & Partition Hypergraph

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



(b) Divide Clusters



(c) Build & Partition Hypergraph

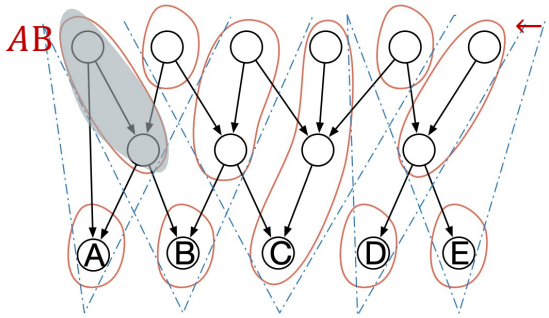
η: *predicted weight of clusters*
$\Gamma(v)$: *the set of hyperedges connected to* v
$|e|$: *the number of end points (pin counts) of a hyperedge* e

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

[4] https://kahypar.org

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



$\leftarrow weight\ of\ nodes = 1$

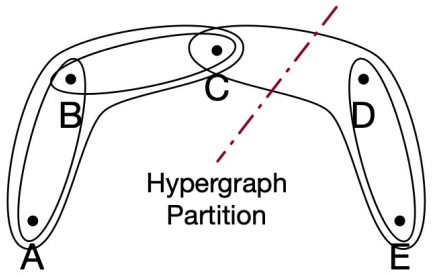
η: predicted weight of clusters

$\Gamma(v)$: the set of hyperedges connected to v

$|e|$: the number of end points (pin counts) of a hyperedge e

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

(b) Divide Clusters



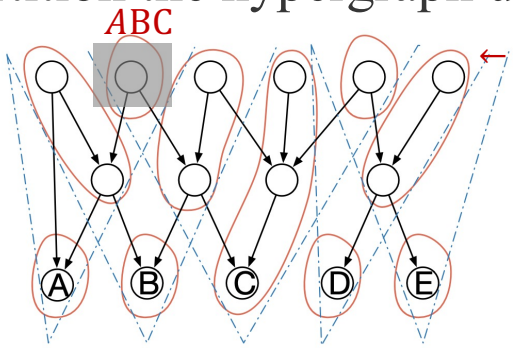
$w_e(e^{AB}) = 2$

(c) Build & Partition Hypergraph

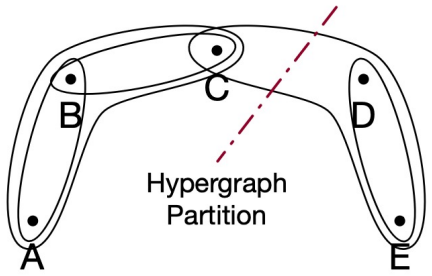
[4] https://kahypar.org

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



(b) Divide Clusters



(c) Build & Partition Hypergraph

$\leftarrow weight\ of\ nodes = 1$

$\eta: predicted\ weight\ of\ clusters$

$\Gamma(v): the\ set\ of\ hyperedges\ connected\ to\ v$

$|e|: the\ number\ of\ end\ points\ (pin\ counts)\ of\ a\ hyperedge\ e$

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

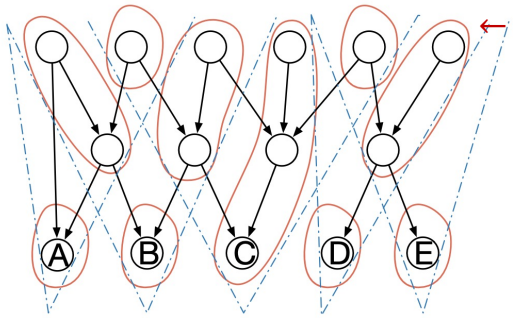
$w_e(e^{AB}) = 2$

$w_e(e^{ABC}) = 1$

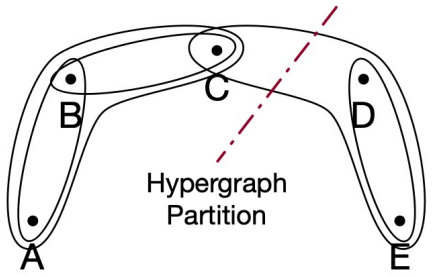
[4] https://kahypar.org

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



(b) Divide Clusters



(c) Build & Partition Hypergraph

$\leftarrow$ *weight of nodes* $= 1$

η: *predicted weight of clusters*

$\Gamma(v)$: *the set of hyperedges connected to* v

$|e|$: *the number of end points (pin counts) of a hyperedge* e

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

weight of itself

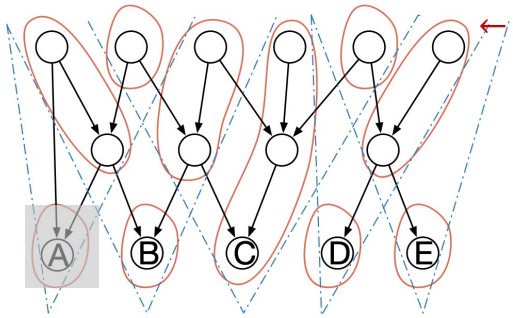
sum of proportional share in hyperedges

$$w_e(e^{AB}) = 2$$
$$w_e(e^{ABC}) = 1$$
$$w_e(e^{BC}) = 2$$
$$w_e(e^{CDE}) = 1$$
$$w_e(e^{DE}) = 2$$

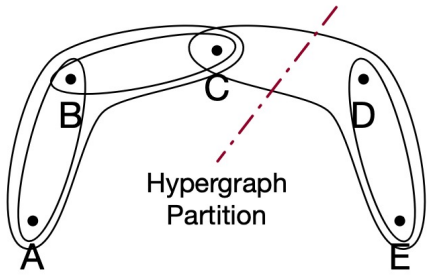
[4] https://kahypar.org

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



(b) Divide Clusters



(c) Build & Partition Hypergraph

$\leftarrow weight\ of\ nodes = 1$

$\eta: predicted\ weight\ of\ clusters$

$\Gamma(v): the\ set\ of\ hyperedges\ connected\ to\ v$

$|e|: the\ number\ of\ end\ points\ (pin\ counts)\ of\ a\ hyperedge\ e$

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

weight of itself

sum of proportional share in hyperedges

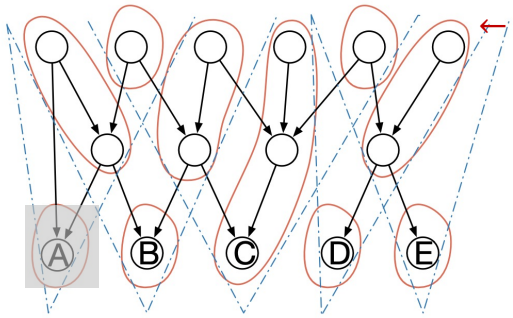
$$w_e(e^{AB}) = 2$$
$$w_e(e^{ABC}) = 1$$
$$w_e(e^{BC}) = 2$$
$$w_e(e^{CDE}) = 1$$
$$w_e(e^{DE}) = 2$$

$$w_v(A) = \eta(A) + \left(\frac{W_e(e^{AB})}{2} + \frac{W_e(e^{ABC})}{3}\right) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3}$$

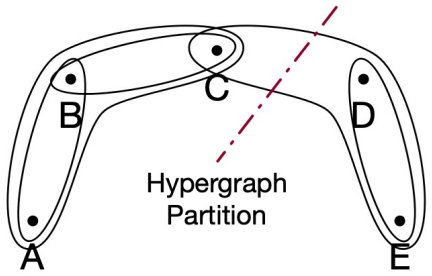
[4] https://kahypar.org

Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].



(b) Divide Clusters



(c) Build & Partition Hypergraph

$\leftarrow$ *weight of nodes* $= 1$

η: *predicted weight of clusters*

$\Gamma(v)$: *the set of hyperedges connected to* v

$|e|$: *the number of end points (pin counts) of a hyperedge* e

$$\omega_v(v) = \eta(v) + \sum_{e \in \Gamma(v)} \frac{\eta(e)}{|e|} \qquad \omega_e(e) = \eta(e) \qquad (1)$$

weight of itself

sum of proportional share in hyperedges

$$\begin{cases} w_e(e^{AB}) = 2 \\ w_e(e^{ABC}) = 1 \\ w_e(e^{BC}) = 2 \\ w_e(e^{CDE}) = 1 \\ w_e(e^{DE}) = 2 \end{cases}$$

$$\begin{cases} w_v(A) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \\ w_v(B) = 1 + \left(\frac{2}{2} + \frac{2}{2} + \frac{1}{3}\right) = \frac{10}{3} \\ w_v(C) = 3 + \left(\frac{2}{2} + \frac{1}{3} + \frac{1}{3}\right) = \frac{14}{3} \\ w_v(D) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \\ w_v(E) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \end{cases}$$

[4] https://kahypar.org

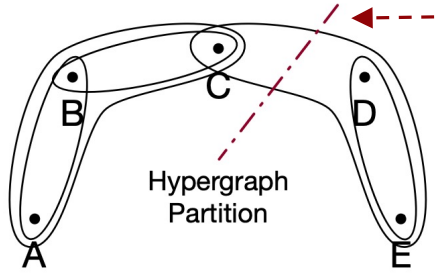
Partitioning Walkthrough – Partition hypergraph

- Partition the hypergraph using Kahypar[4].

→ Constraint: balanced partitions.

→ Cost function: min-cut.

→ 2-way partition.



(c) Build & Partition Hypergraph

$$\begin{bmatrix} w_e(e^{AB}) = 2 \\ w_e(e^{ABC}) = 1 \\ w_e(e^{BC}) = 2 \\ w_e(e^{CDE}) = 1 \\ w_e(e^{DE}) = 2 \end{bmatrix}$$

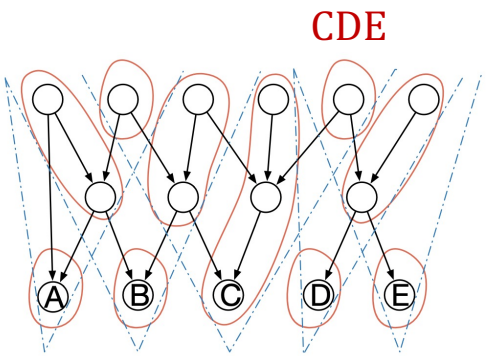
$$\begin{bmatrix} w_v(A) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \\ w_v(B) = 1 + \left(\frac{2}{2} + \frac{2}{2} + \frac{1}{3}\right) = \frac{10}{3} \\ w_v(C) = 3 + \left(\frac{2}{2} + \frac{1}{3} + \frac{1}{3}\right) = \frac{14}{3} \\ w_v(D) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \\ w_v(E) = 1 + \left(\frac{2}{2} + \frac{1}{3}\right) = \frac{7}{3} \end{bmatrix}$$

Kahypar

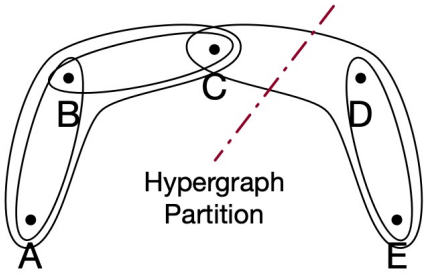
[4] https://kahypar.org

Partitioning Walkthrough – Acquire partitions

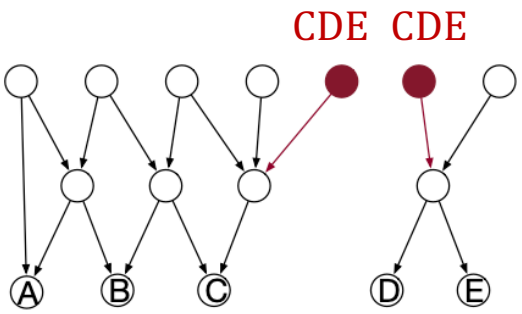
- Acquire the final partitions by replicating the nodes in cut edge.



(b) Divide Clusters



(c) Build & Partition Hypergraph



(d) Acquire Partitions

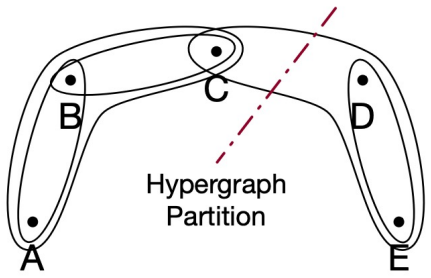
Partitioning Walkthrough – Acquire partitions

- Acquire the final partitions by replicating the nodes in cut edge.

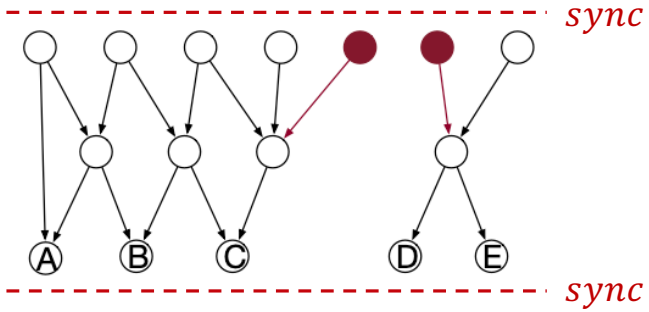
→ Our goal: **Balanced partitions** and **minimal synchronization effort**.

→ Constraint: balanced partitions.

→ Cost function: min-cut.



(c) Build & Partition Hypergraph



(d) Acquire Partitions

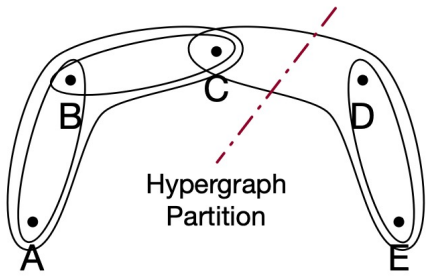
Partitioning Walkthrough – Acquire partitions

- Acquire the final partitions by replicating the nodes in cut edge.

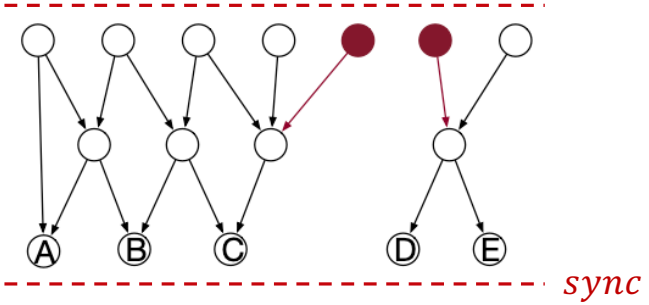
→ Our goal: **Balanced partitions** and **minimal synchronization effort**.

→ Constraint: balanced partitions.

→ Cost function: min-cut. → minimize replication cost.



(c) Build & Partition Hypergraph



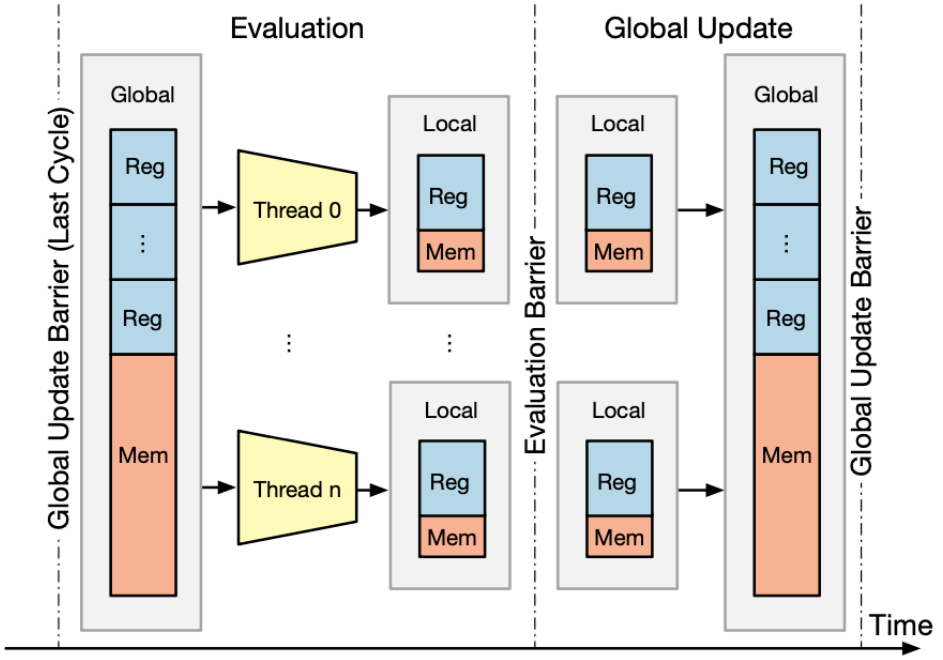
(d) Acquire Partitions

Schedule the partitions

- Divide the circuit into multiple regions.

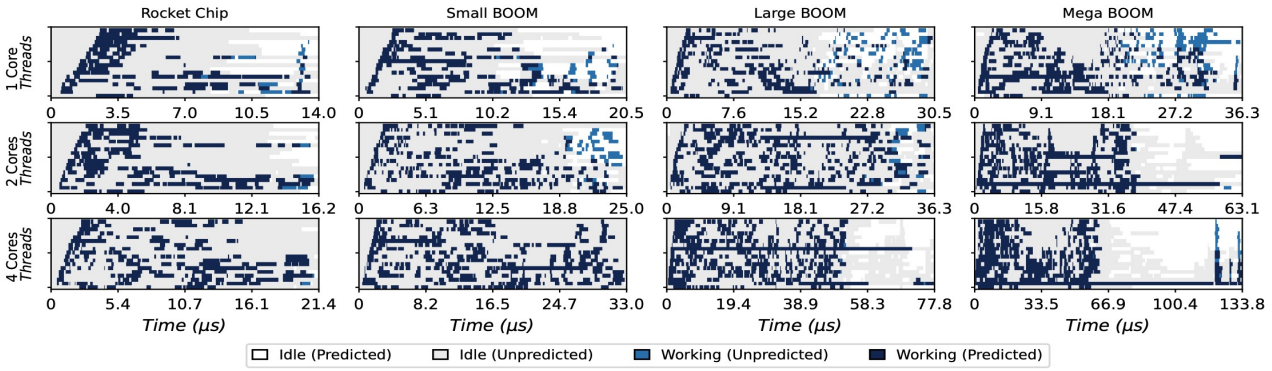
→ Boundary: registers and memory.

- Apply partitioning to each region.
- Schedule the partitions.

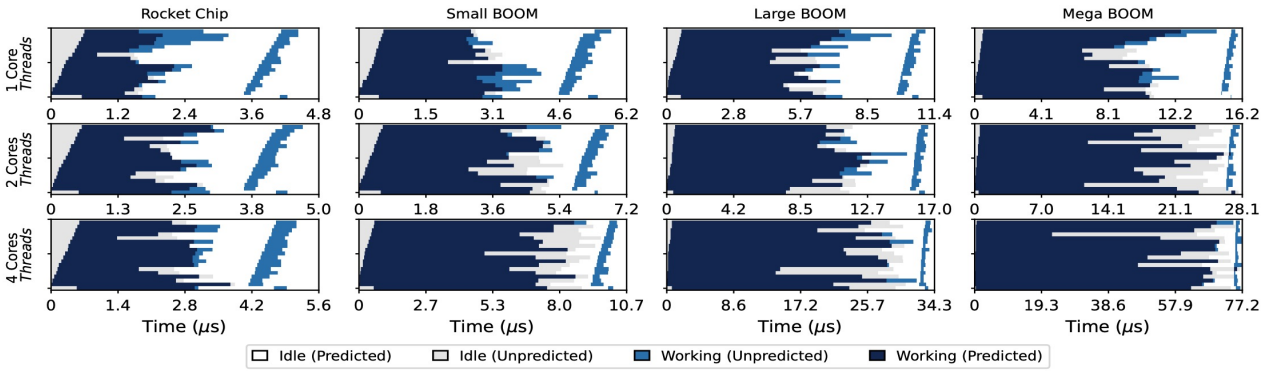


Experimental results – Balanced partitions

- ESSENT (RepCut) vs Verilator (Vivek's[5])



(a) Verilator Thread Profile

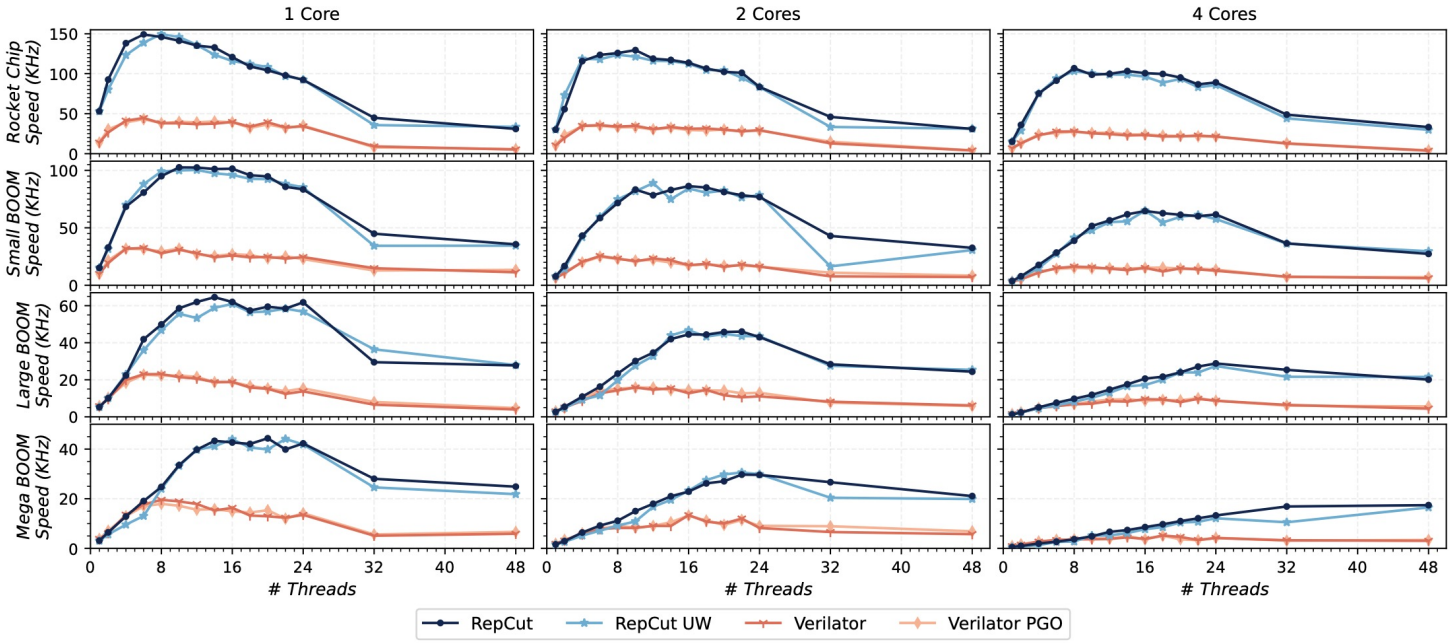


(b) RepCut Thread Profile

[5] Sarkar, Vivek. Partitioning and scheduling parallel programs for execution on multiprocessors. Stanford University, 1987.

Experimental results – Overall speed of RTL simulation

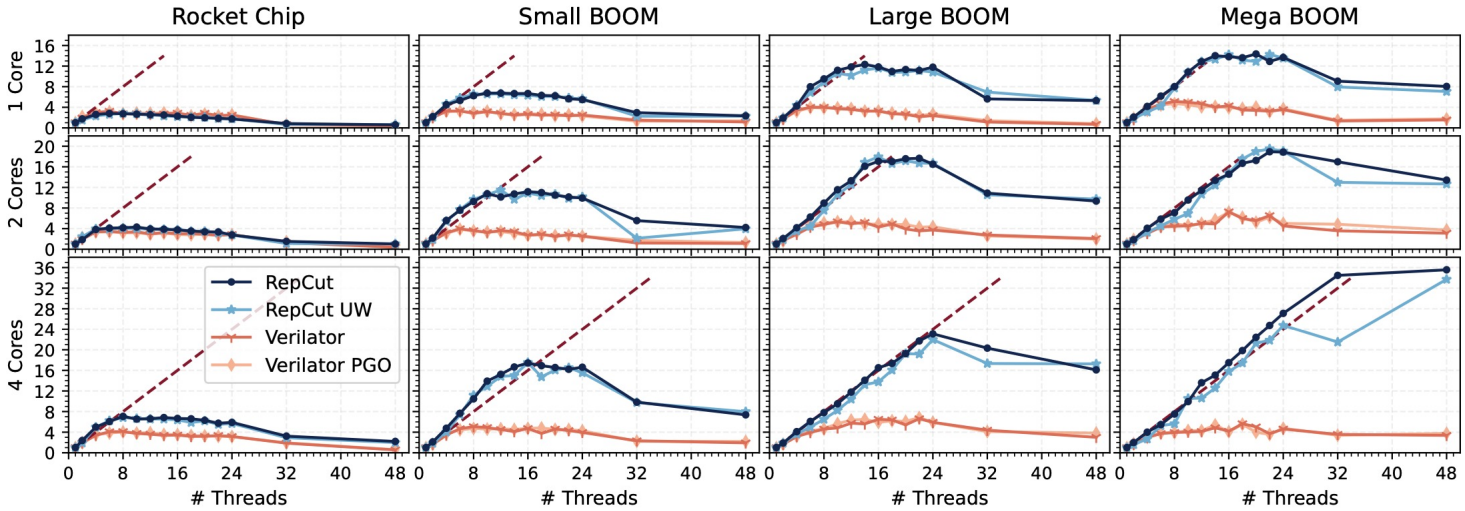
- ESSENT (RepCut) vs Verilator (Vivek's[5])



[5] Sarkar, Vivek. Partitioning and scheduling parallel programs for execution on multiprocessors. Stanford University, 1987.

Experimental results – Speedup over itself

- ESSENT (RepCut) vs Verilator (Vivek's[5])



[5] Sarkar, Vivek. Partitioning and scheduling parallel programs for execution on multiprocessors. Stanford University, 1987.

My current research

Limitations of RepCut

- High replication cost.

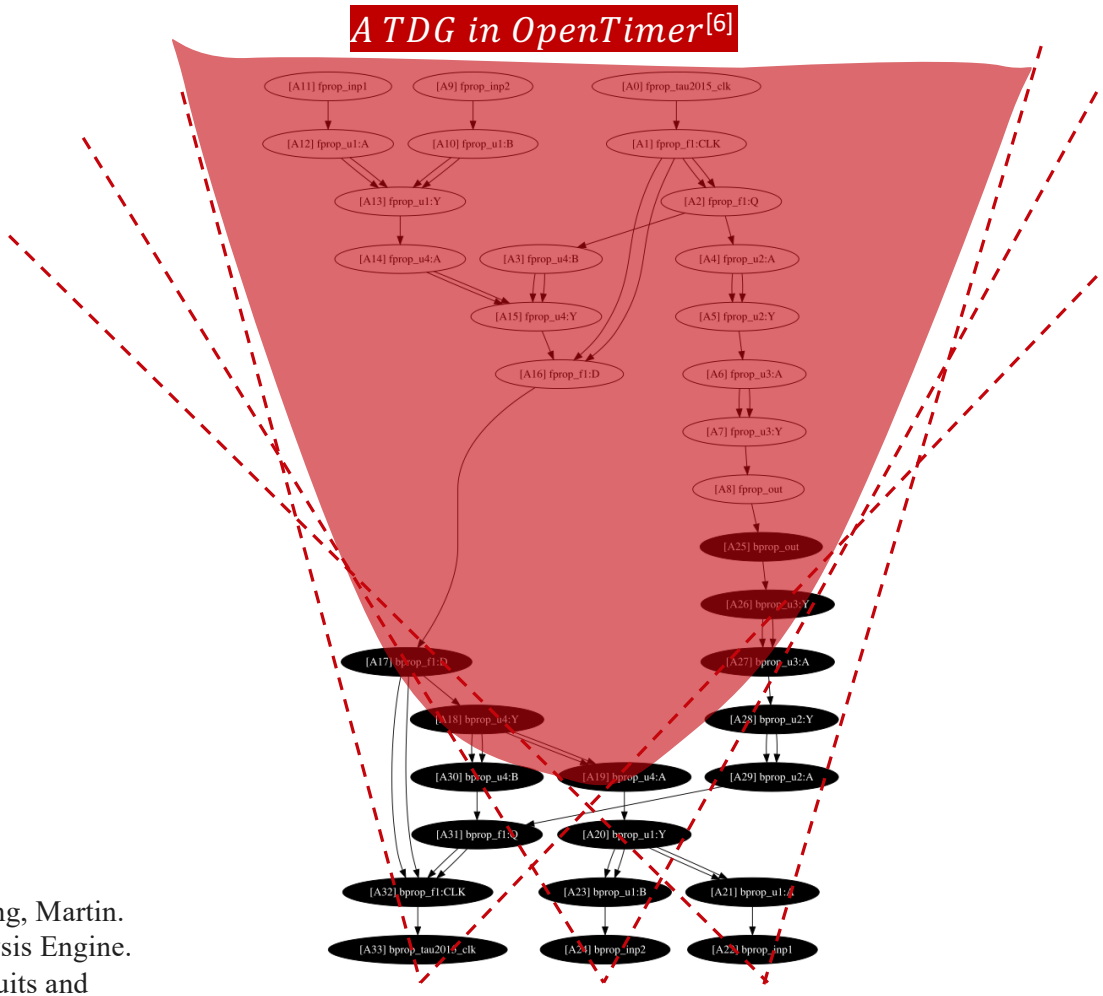
→ TDGs with a long critical path.

- Large partitioning overhead.

→ TDGs can change each iteration.

→ Invoke external hypergraph partitioner.

→ Single-thread partitioning.



[6] Huang, Tsung-Wei & Guo, Guannan & Lin, Chun-Xun & Wong, Martin. (2020). OpenTimer v2: A New Parallel Incremental Timing Analysis Engine. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. PP. 1-1. 10.1109/TCAD.2020.3007319.

Our research

- TDG partitioning in parallel?

→ Put it on GPU[1].

- Incrementally update TDG partitions?

[1] Boyang Zhang, Dian-Lun Lin, Che Chang, Cheng-Hsiang Chiu, Bojue Wang, Wan Luan Lee, Chih-Chun Chang, Donghao Fang, and Tsung-Wei Huang, "G-PASTA: GPU Accelerated Partitioning Algorithm for Static Timing Analysis," *ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2024

Conclusion

- We introduced RTL simulations.
- We introduced RepCut's partitioning methods.
- We talked about RepCut's limitations.
- We summarized our research.

Reference

[1] Boyang Zhang, Dian-Lun Lin, Che Chang, Cheng-Hsiang Chiu, Bojue Wang, Wan Luan Lee, Chih-Chun Chang, Donghao Fang, and Tsung-Wei Huang, "G-PASTA: GPU Accelerated Partitioning Algorithm for Static Timing Analysis," *ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2024

[2] https://github.com/verilator/verilator

[3] https://github.com/ucsc-vama/essent

[4] https://kahypar.org

[5] Sarkar, Vivek. Partitioning and scheduling parallel programs for execution on multiprocessors. Stanford University, 1987.

[6] Huang, Tsung-Wei & Guo, Guannan & Lin, Chun-Xun & Wong, Martin. (2020). OpenTimer v2: A New Parallel Incremental Timing Analysis Engine. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. PP. 1-1. 10.1109/TCAD.2020.3007319.


WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON