北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

GPU Acceleration in VLSI Back-end Design: Overview and Case Studies

Yibo Lin

Department of Computer Science, Peking University

Website: https://yibolin.com

Email: yibolin@pku.edu.cn

Outline

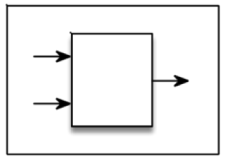
- Introduction
- Challenges of GPU Acceleration
- Current Status
  - Placement
  - Routing
  - Timing analysis
- Case Studies
  - Placement
  - Timing analysis
- Conclusion & Future Work

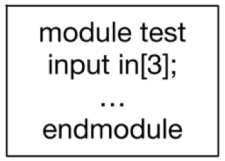
IC Design Flow

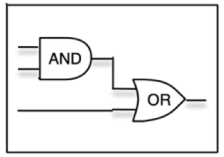
Fabless Design House

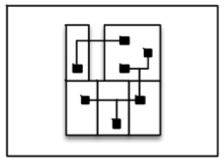
Fab/Foundry

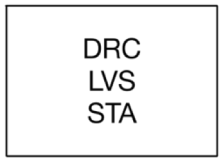
System Design | Logic Design | Backend Design

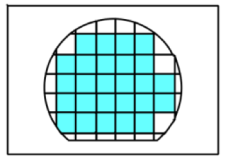


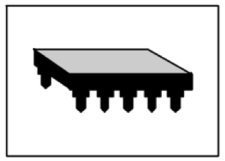




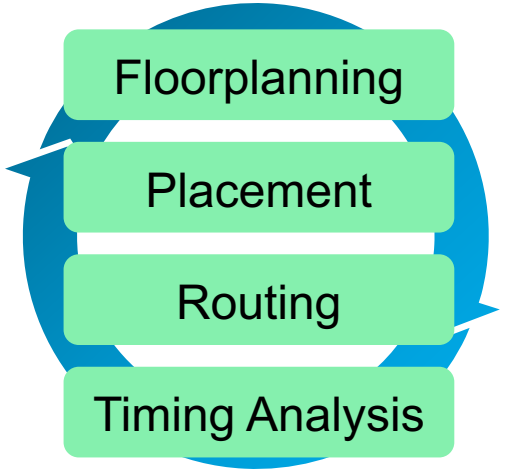






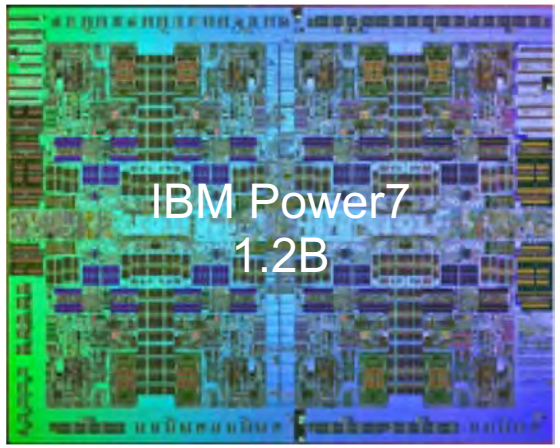


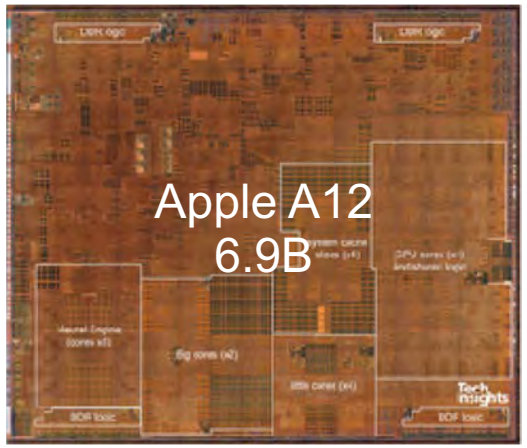
System Level Synthesis → Logic Synthesis → Physical Design → Physical Verification → Fabricate → Package Test

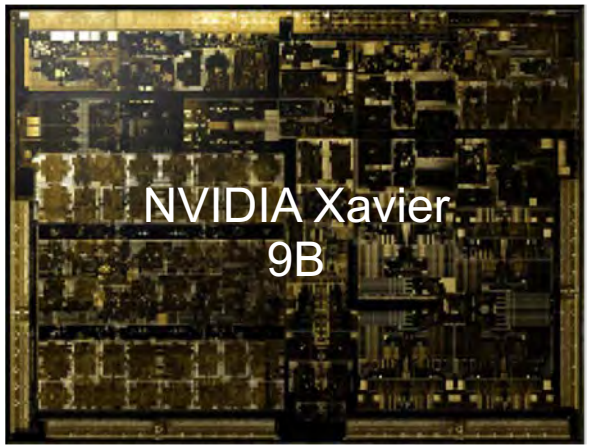


Timing Closure
DFM Closure

Challenges in VLSI Design Automation

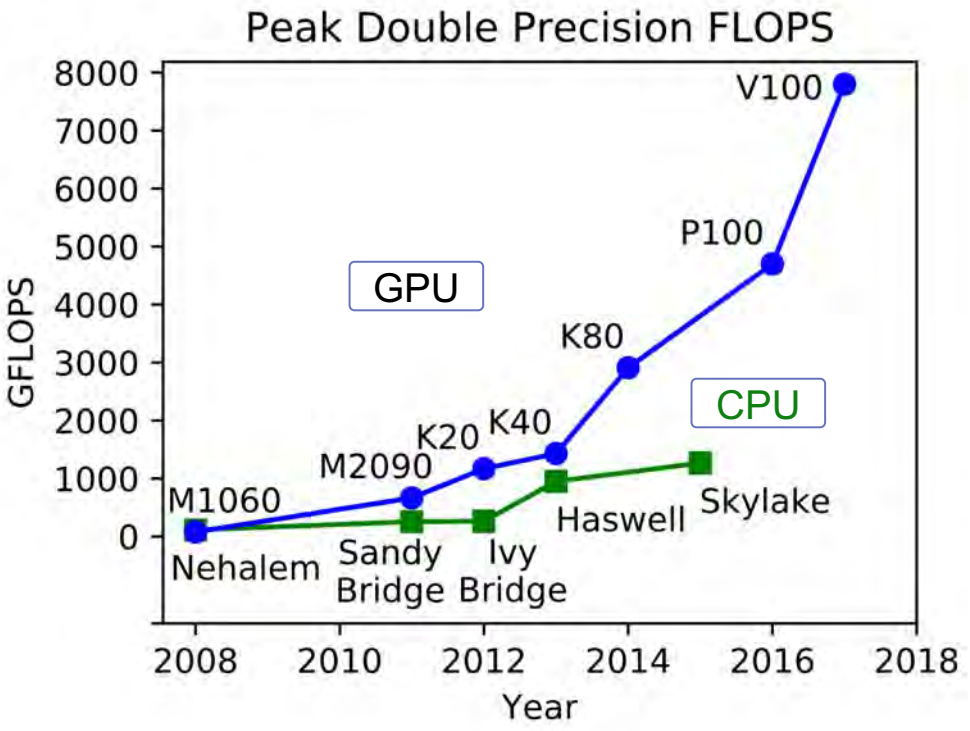






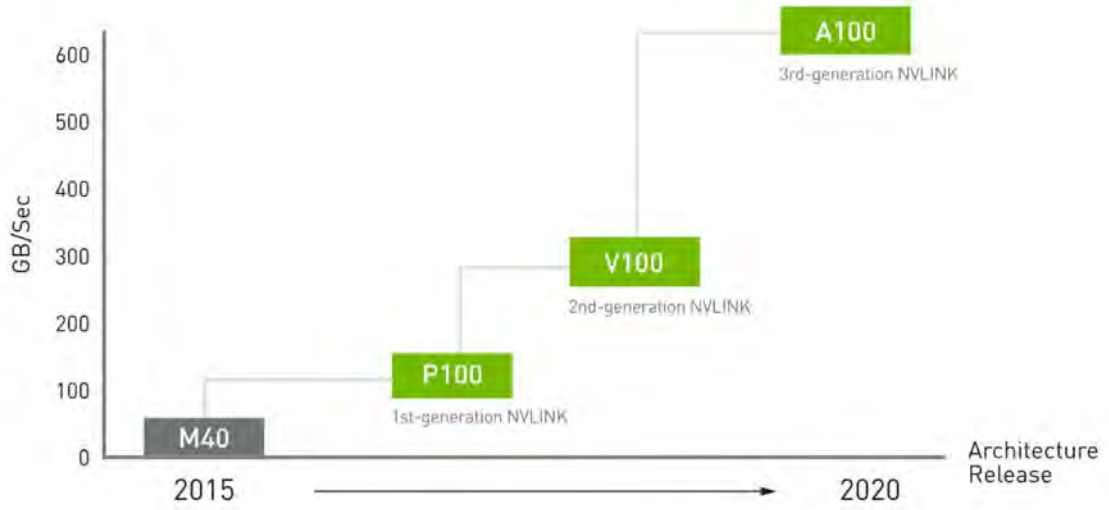
- Large scale: billions of transistors
- Numerous constraints from low-level manufacturing & high-level architecture
- Complicated design flow
- Long design cycles

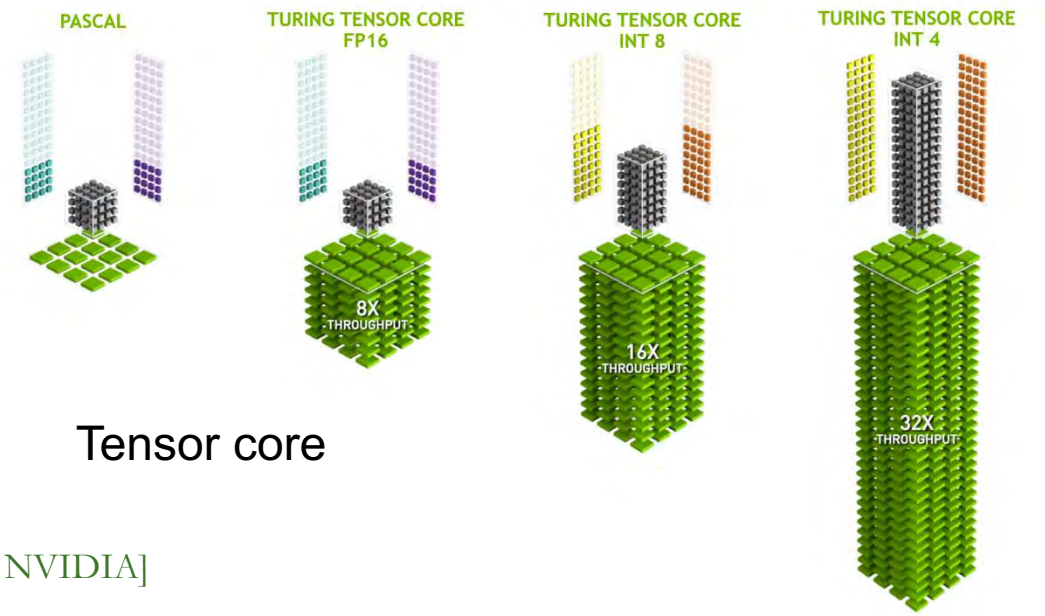
Advances in GPU



Over **60x** speedup in neural network training since 2013

NVLink Performance





Tensor core

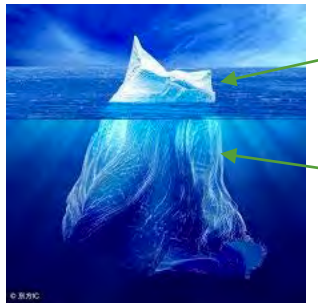
[Courtesy NVIDIA]

Hardware Acceleration is NOT New for Physical Design

- Tom Black @ Standford, 1984
  - IEEE Design & Test of Computers
  - A Survey of Hardware Accelerators Used in Computer-Aided Design
  - Roughly 20 machines built and tested simulation, design rule checking, placement, and routing
- What has changed
  - CPU & GPU performance
  - Physical design algorithms
  - Problem scales
  - Development environment like CUDA since 2007, Tensorflow since 2015, PyTorch since 2016
  - …

Challenges in GPU Acceleration

- Successful acceleration in deep learning
  - Conv., gemm/gemv, BLAS, …
  - Conv takes >80% runtime in CNN inference
  - Multiply-accumulate (MAC), systolic array, …
- HPC community
  - BLAS kernels
  - Graph kernels: SSSP, page rank, BFS, …



- Physical Design
  - No dominating steps
  - Placement (~20%), routing (~40%)
  - Timing analysis, routing congestion
  - Iterative algorithms: ~1K to ~10K
  - Single iteration is not slow
  - Random memory access
  - ~500ms data transfer, ~50ms computation
  - All customized kernels, lack of parallelism
  - BLAS, graph, heuristics, …

Outline

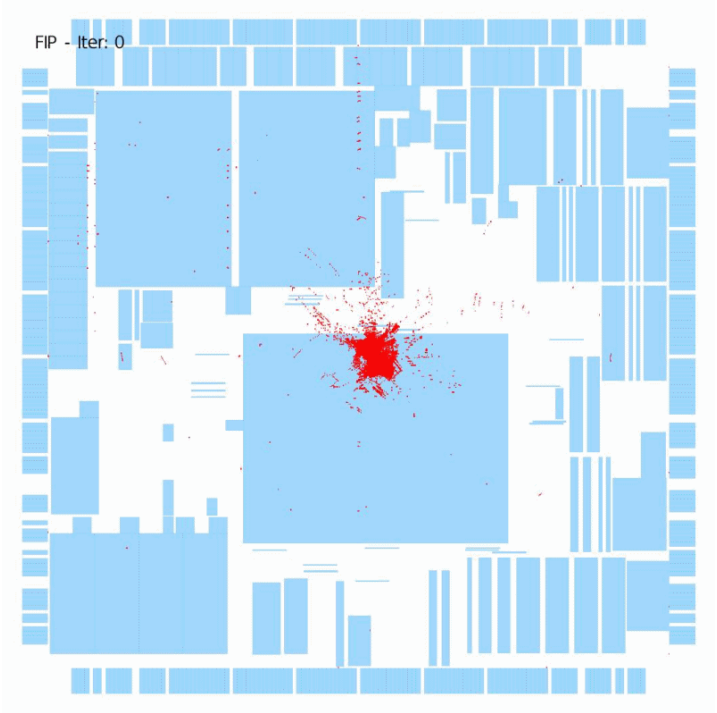
- Introduction
- Challenges of GPU Acceleration
- Current Status
  - Placement
  - Routing
  - Timing analysis
- Case Studies
  - Placement
  - Timing analysis
- Conclusion & Future Work

Current Status – Placement

Input
Gate-level netlist
Standard cell library

Output
Legal placement solution

Objective
Optimize wirelength, routability



Cell Spreading in Placement
[RePlAce]

Current Status – Global Placement

- Mathematical formulation

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & WL(\mathbf{x},\mathbf{y}), \\ \text{s.t.} \quad & D(\mathbf{x},\mathbf{y}) \leq t_d \end{aligned}$$

- Nonlinear placement objective

$$\min \quad \underbrace{\left(\sum_{e\in E} WL(e;\mathbf{x},\mathbf{y})\right)}_{\text{Wirelength}} + \lambda \underbrace{D(\mathbf{x},\mathbf{y})}_{\text{Density}}$$

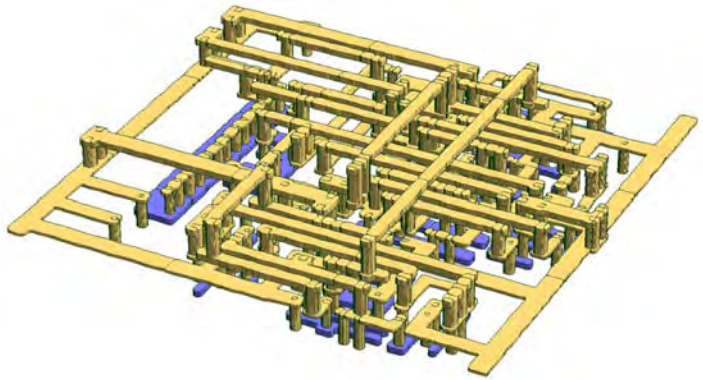
- Wirelength & density map acceleration
  - [Lin+, DATE'18]
- Nonlinear placement with clustering
  - mPL6 [Cong+, ICCAD'09] **15x** speedup
- Force-directed TimerWolf
  - [Kawam+, ICITC'15]
  - [Bredthauer+, ISPDC'18]
  - **2-5x** speedup
- Nonlinear placement w. electrostatic analogy ePlace
  - DREAMPlace [Lin+, DAC'19, TCAD'20]
  - **30-40x** speedup

Current Status – Detailed Placement

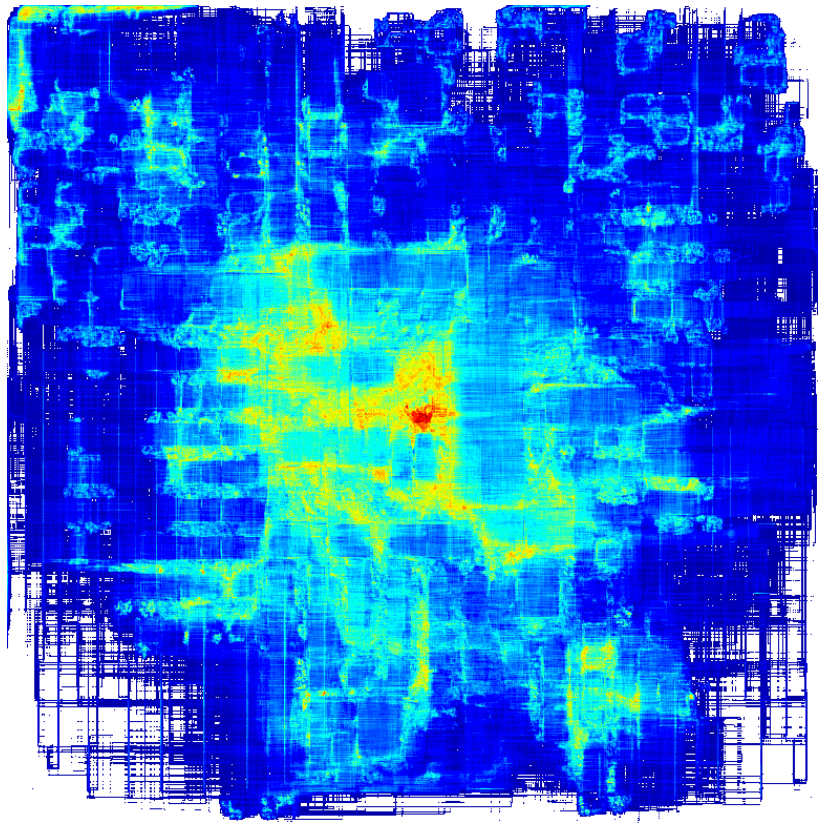
- Greedy and iterative nature
- Row-based interleaving algorithm
  - GDP [Dhar+, HPEC'18], **7x** speedup over 20-thread CPU
  - Fill 3D dynamic programming table
- Misc. algorithms
  - ABCDPlace [Lin+, TCAD'20], **15x** speedup over 20-thread CPU
  - Independent set matching, global swap, local reordering

Current Status – Routing

- Variations of shortest path problems
  - Single-source multiple targets
  - 3D grids with millions of nets
  - 10+ metal layers
  - May be highly congested
  - Minimize wirelength



A zoom-in 3D view
[curtesy samyzaf]



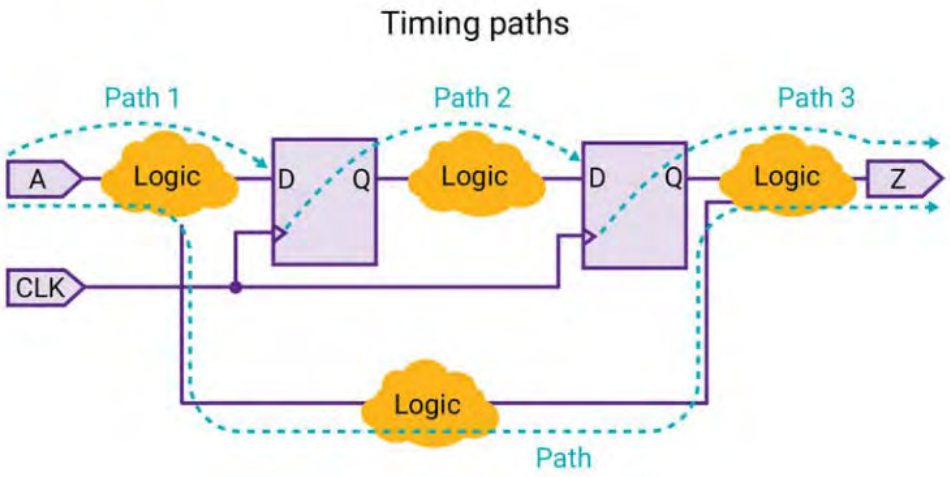
[Curtesy Umich]

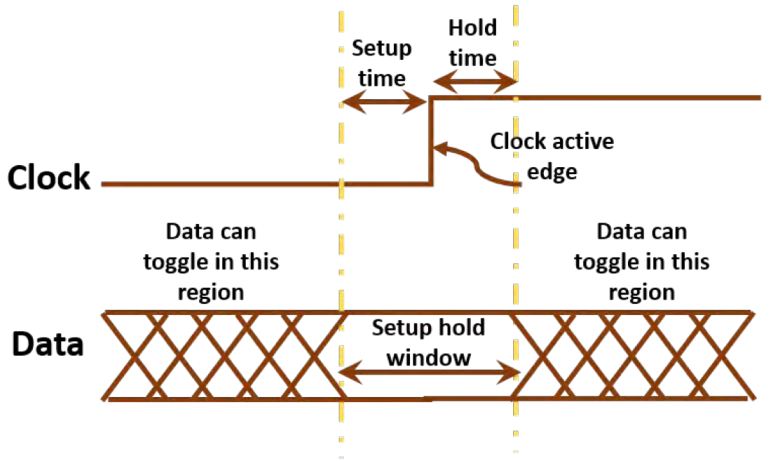
Current Status – Routing

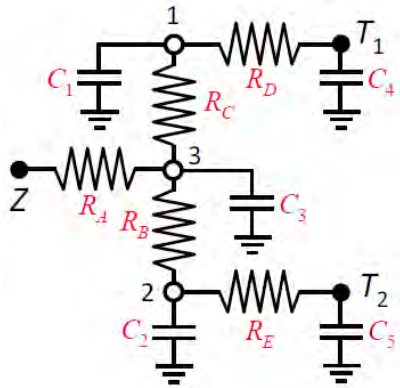
- ASIC routing
  - Global router [Han+, ICCD'13], decompose multi-pin nets to 2-pin nets, GPU-accelerated SSSP kernels
  - **2.5-3.9x** speedup with 2.5% WL degradation, over NCTUgr 2.0
  - Improve scheduling of nets [Han+, TVLSI'13]
  - **4x** speedup with 1% WL degradation, over NTHU-Route 2.0
- FPGA routing
  - GPU-accelerated SSSP kernels based on Bellman-Ford algorithm [Shen+, FPGA'17, TPDS'18]
  - **21x** speedup over sequential VPR 7.0 [Luu+, TRETS'14]
- Challenges
  - Lack of parallelism
  - Divergence of computation patterns between nets
  - Huge random memory access

Current Status – Timing Analysis

- Critical for performance and correct functionality of the chips
- Cell delay: non-linear delay model (NLDM)
- Net delay: Elmore delay model (Parasitic RC Tree)
- Timing propagation





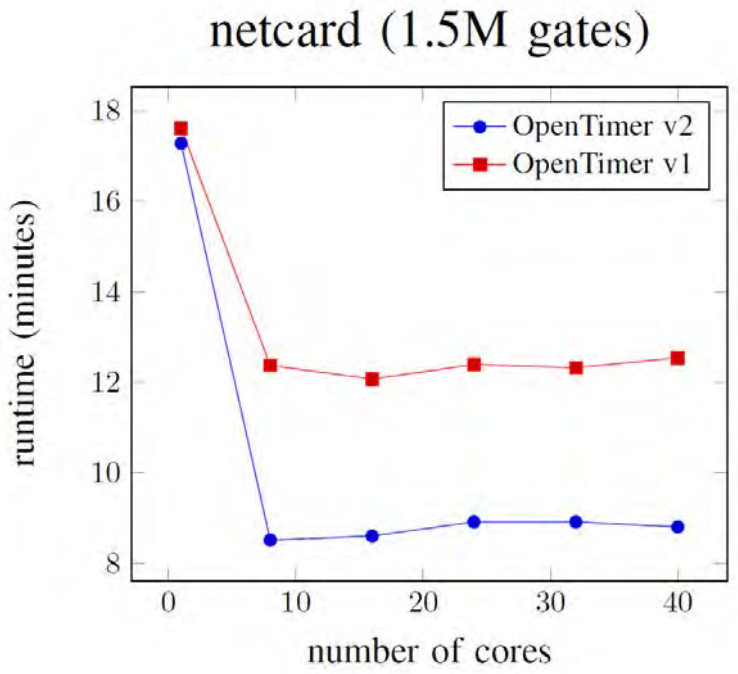


Delay for wires

[Courtesy Synopsys]

Current Status – Timing Analysis

- Parallelization on CPU by multithreading
  - [Huang+, ICCAD'15] [Lee+, ASP-DAC'18]…
  - cannot scale beyond 8-16 threads
- Leveraging GPU?
  - Single instruction multiple thread (SIMT) architecture
- Statistical STA acceleration using GPU
  - [Gulati+, ASPDAC'09] [Cong+, FPGA'10]…
  - Less challenging than STA based on pessimism
- Accelerate STA using modern GPU
  - Lookup table query and timing propagation [Wang, ICPP'14] [Murray, FPT'18]
  - 6.2x kernel time speed-up, but 0.9x of entire time because of data copying
- Leveraging GPU is challenging
  - Graph-oriented: diverse computational patterns and irregular memory access
  - Data copy overhead



Outline

- Introduction
- Challenges of GPU Acceleration
- Current Status
  - Placement
  - Routing
  - Timing analysis
- Case Studies
  - Placement
  - Timing analysis
- Conclusion & Future Work

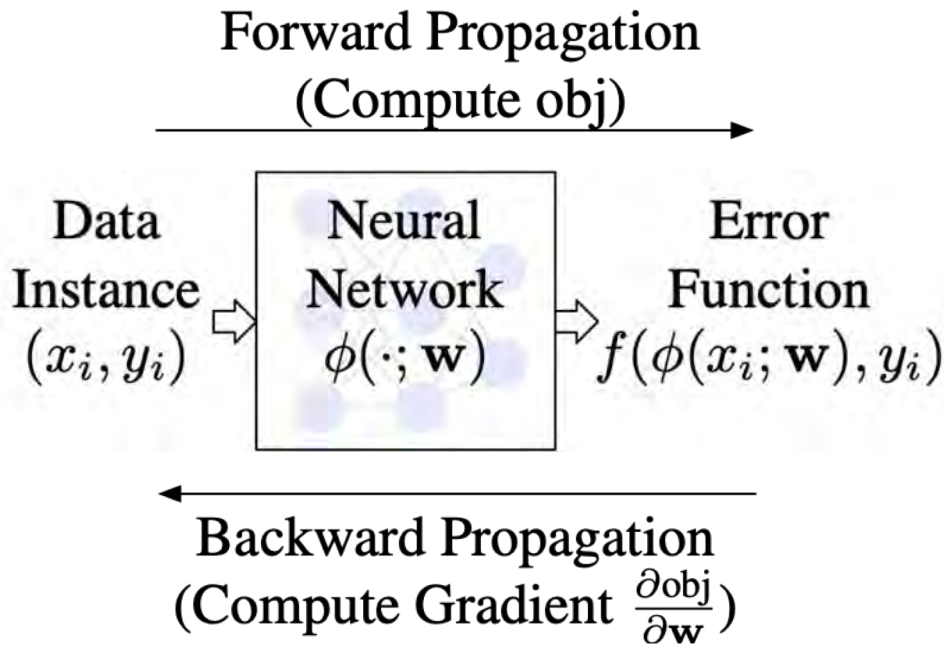
Case Studies – DREAMPlace

- We propose a novel analogy by casting the **nonlinear placement optimization** into a **neural network training** problem
- Greatly leverage deep learning hardware (GPU) and software toolkit (e.g., PyTorch)
- Enable ultra-high parallelism and acceleration while getting the state-of-the-art results

Analogy between Neural Network Training and Placement

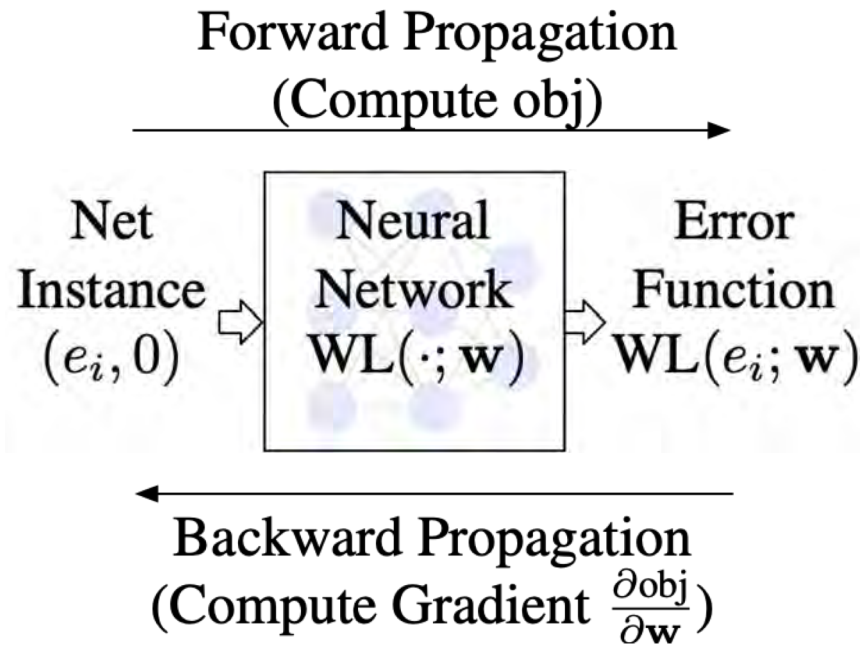
Casting the placement problem into neural network training

$$\min_{\mathbf{w}} \sum_{i}^{n} f(\phi(x_i; \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$



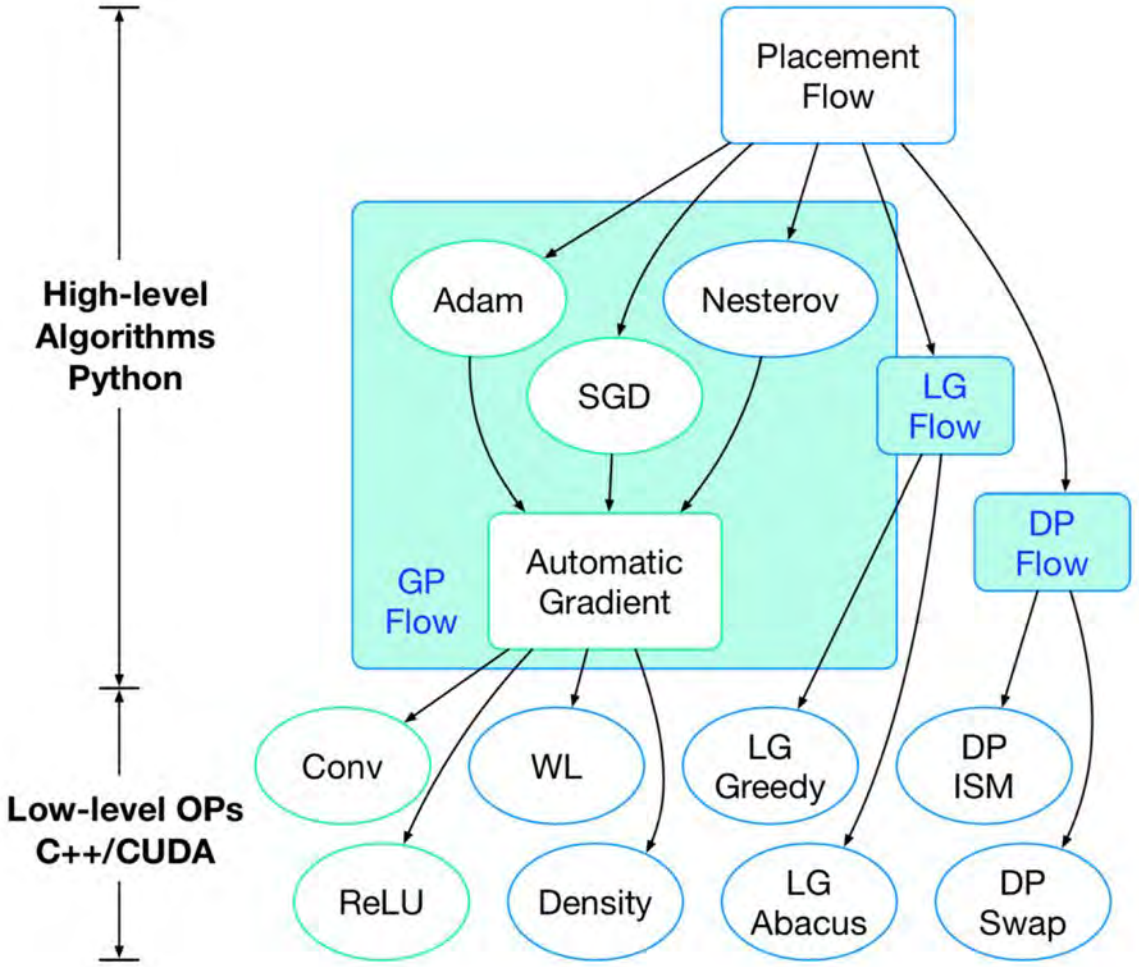
Train a neural network

$$\min_{\mathbf{w}} \sum_{i}^{n} \text{WL}(e_i; \mathbf{w}) + \lambda D(\mathbf{w})$$



Solve a placement

Develop Placement Engine with Deep Learning Toolkit

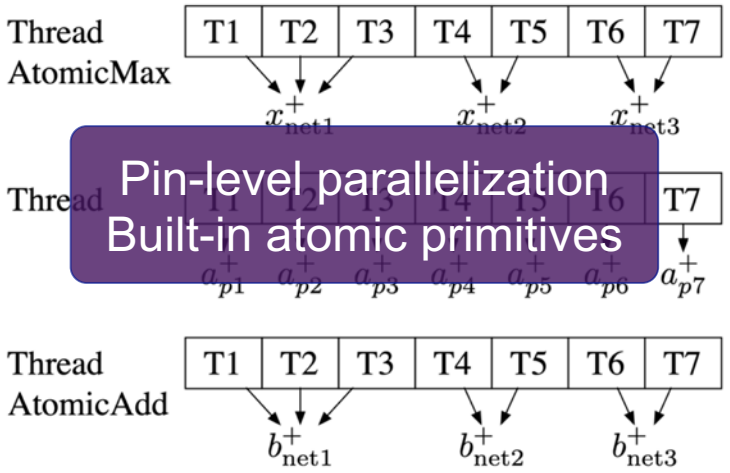


Customized C++/CUDA Kernels

Wirelength OP

Smoothing HPWL by weighted average wirelength

$$\mathrm{WA}_e = \frac{\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{-\frac{x_i}{\gamma}}}$$



Pin-level parallelization
Built-in atomic primitives

1.9x faster than net-by-net parallelization

Density Penalty OP

Electrostatic system analogy

Cell instance Electric particle

$$a_{u,v} = \mathrm{DCT}(\mathrm{DCT}(\rho)^T)^T$$

$$\psi_{\mathrm{DCT}} = \mathrm{IDCT}(\mathrm{IDCT}(\{\frac{a_{u,v}}{w_u^2 + w_v^2}\})^T)^T$$

$$\xi^X_{\mathrm{DSCT}} = \mathrm{IDXST}(\mathrm{IDCT}(\{\frac{a_{u,v} w_u}{w_u^2 + w_v^2}\})^T)^T$$

$$\xi^Y_{\mathrm{DCST}} = \mathrm{IDCT}(\mathrm{IDXST}(\{\frac{a_{u,v} w_v}{w_u^2 + w_v^2}\})^T)^T$$

1.4x faster than native DCT implementation used in TensorFlow

Experimental Results for Global Placement

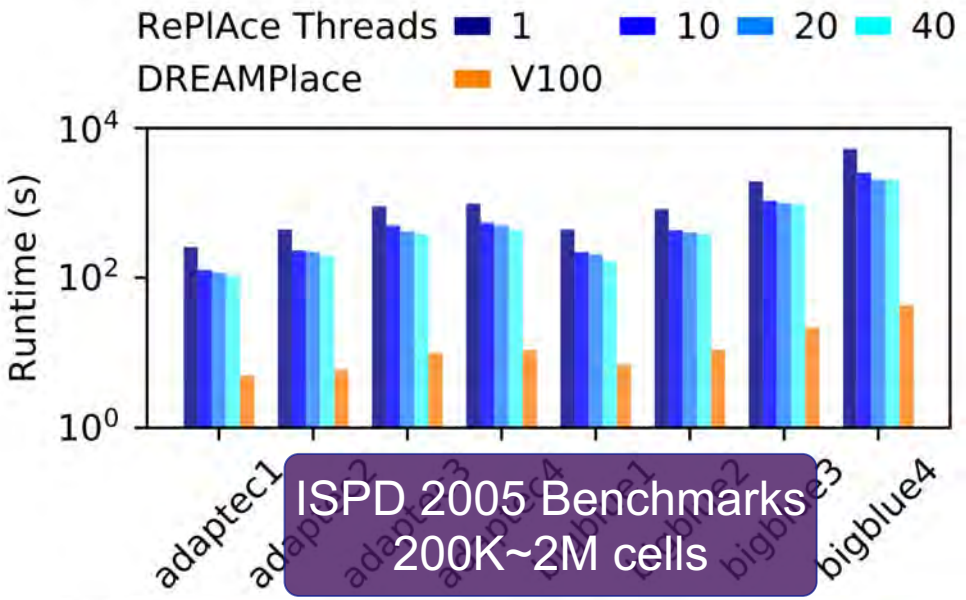
DREAMPlace
- CPU: Intel E5-2698 v4 @2.20GHz
- GPU: 1 NVIDIA Tesla V100
- Single CPU thread was used

RePlAce [TCAD'18,Cheng+]
- CPU: 24-core 3.0 GHz Intel Xeon
- 64GB memory allocated

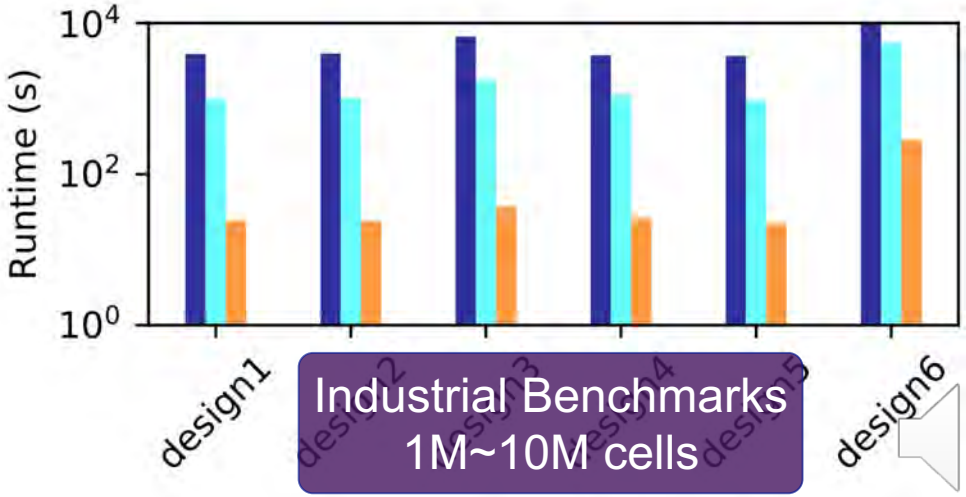
Same quality of results!

10M-cell design finishes within **5min c.f. 3h**

34x speedup



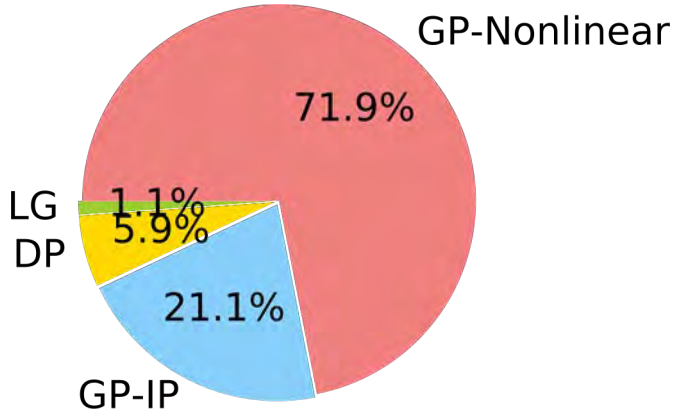
43x speedup



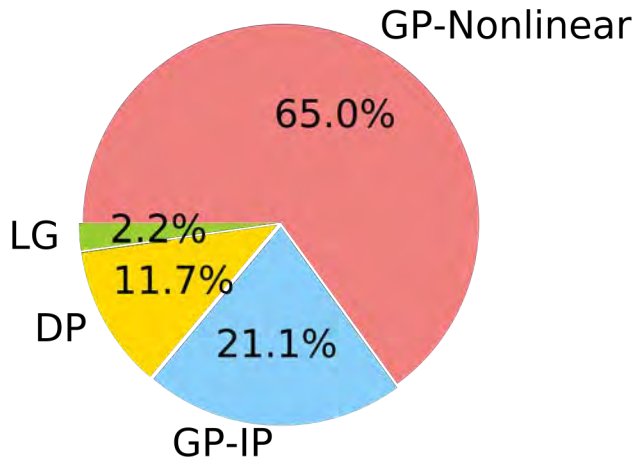
Runtime Breakdown on Bigblue4 (2M-Cell Design)

RePlAce

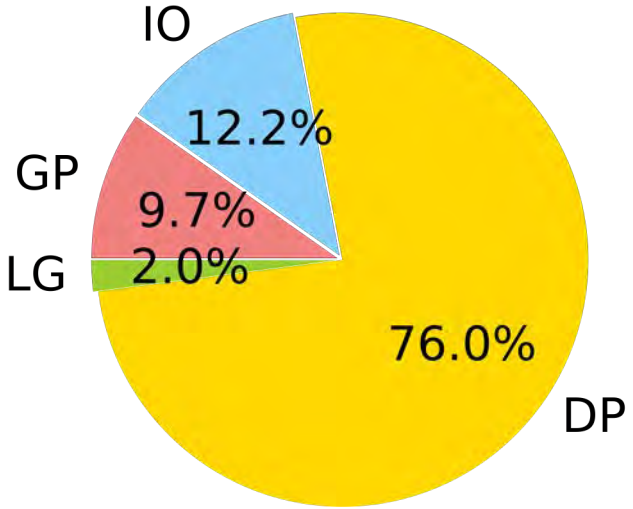
1 thread



10 threads



DREAMPlace
with GPU acceleration



DREAMPlace 2.0
ABCDPlace

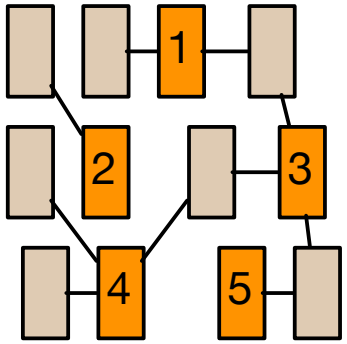
15X speedup on DP
with GPU acceleration

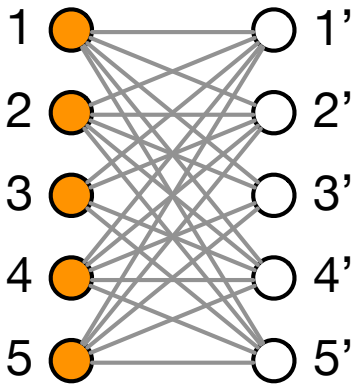
1min for 10M-cell design

Detailed Placement – Sequential and Iterative Nature

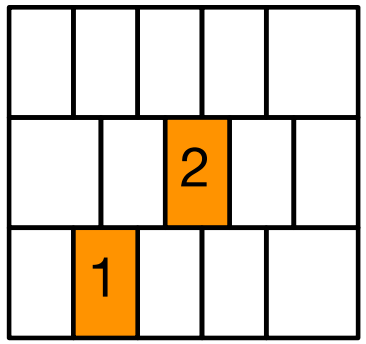
- Local and greedy algorithms
  - Iterate between a subset of cells
- Lack of parallelism
  - Interdependency due to connectivity
- Irregular

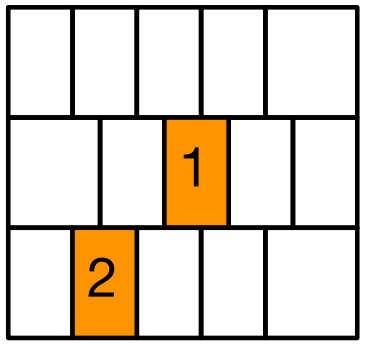
For each window
  Collect a subset of cells
  Find the best permutation
  Apply the movement
  …



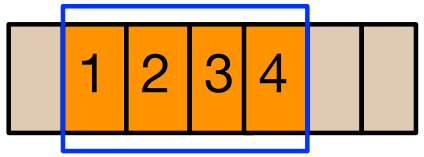


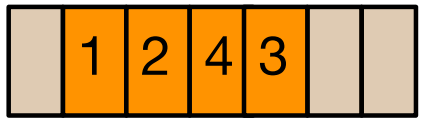
Independent Set Matching

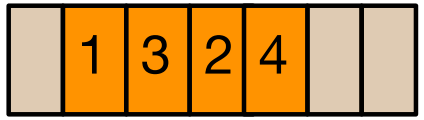




Global Swap





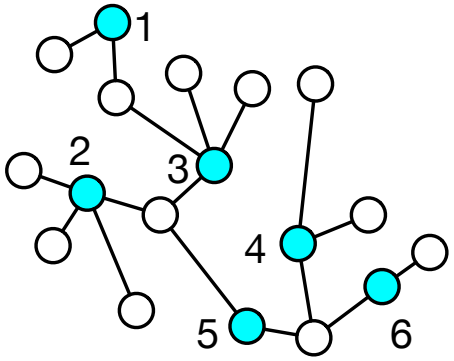


…

Local Reordering

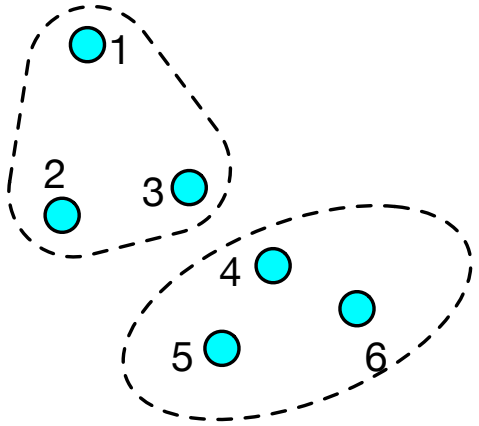
Concurrent Independent Set Matching

- Moving the refinement from a window to the entire layout
- Complicated graph analytics
  - Suitable algorithms for parallelization
  - Specialized parallelization scheme for GPU: know GPU architecture



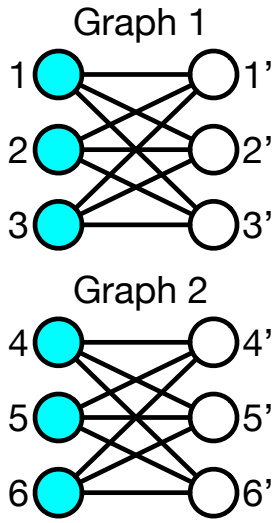
Maximal independent set

Blelloch's algorithm



Balanced Partitioning

K-means clustering



Bipartite Matching
(Batched)

Auction algorithm

Experimental Results for Detailed Placement

GPU friendly DP algorithms

GPU-accelerated graph solvers

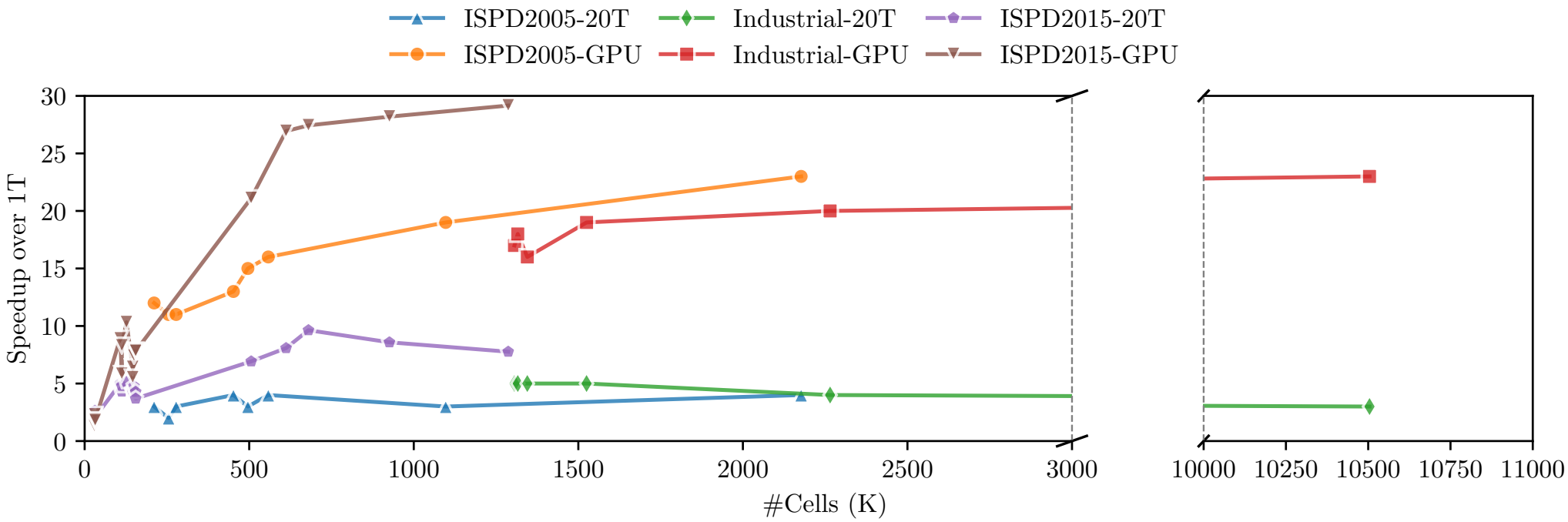
ABCDPlace
- CPU: Intel E5-2698 v4 @2.20GHz
- GPU: 1 NVIDIA Tesla V100

NTUplace3 [TCAD'08,Chen+]
- CPU: Intel E5-2698 v4 @2.20GHz

Same quality of results!

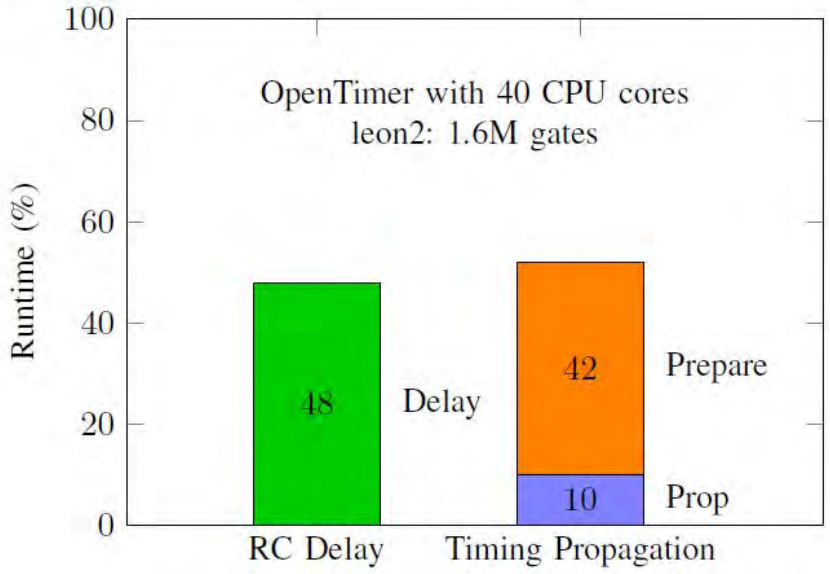
10~15x speedup

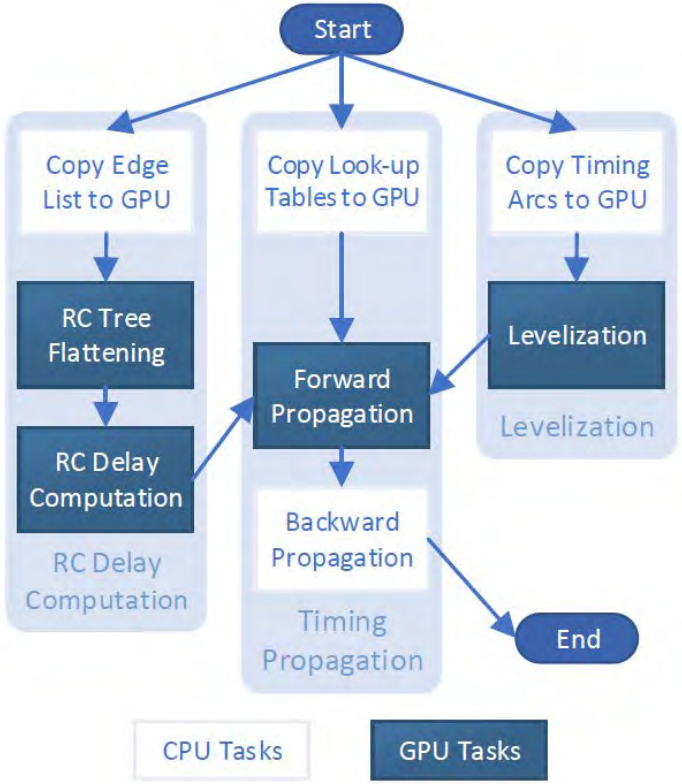
10M-cell design finishes within **58s c.f. 26min**



Case Studies – Timing Analysis

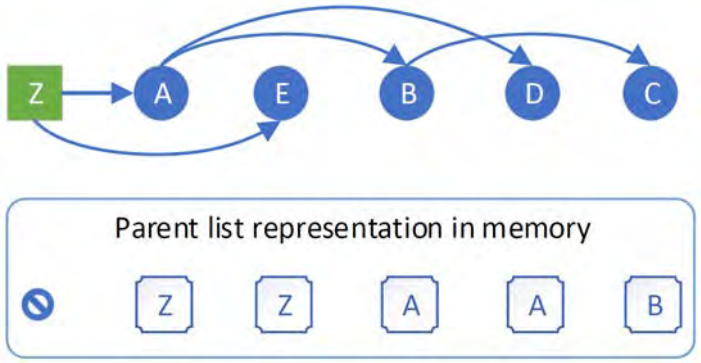
- RC delay computation, task graph levelization, and timing propagation
  - Covers the runtime bottlenecks
- Implementation based on open source STA engine OpenTimer [Huang+, ICCAD2015]

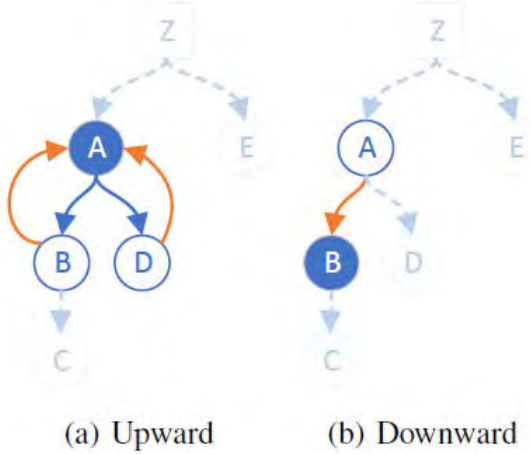


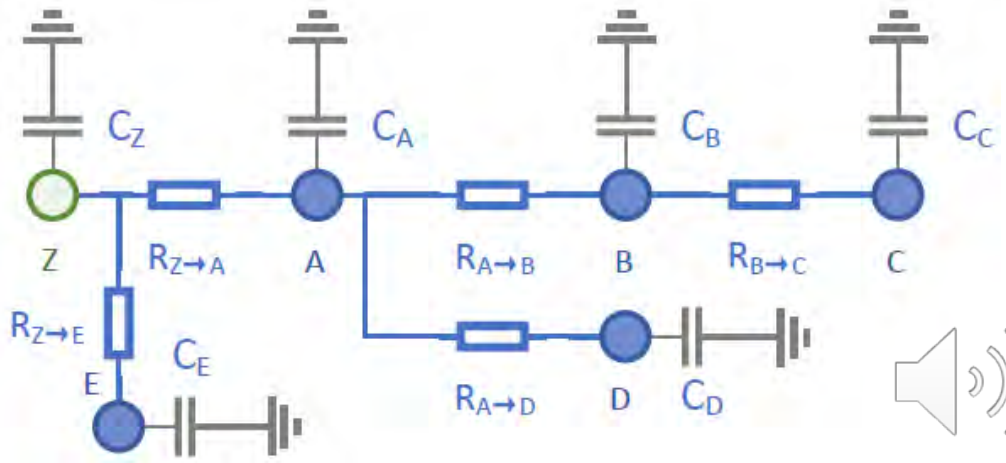


RC Delay Computation

- The Elmore delay model
- $load_u = \sum_{v \text{ is child of } u} cap_v$
  - eg. $load_A = cap_A + cap_B + cap_C + cap_D = cap_A + load_B + load_D$
- $delay_u = \sum_{v \text{ is any node}} load_v \times R_{Z \to LCA(u,v)}$
  - eg. $delay_B = load_A R_{Z \to A} + load_D R_{Z \to A} + load_B R_{Z \to B} + load_C R_{Z \to B}$
    $= delay_A + R_{A \to B} load_B$
- $ldelay_u = \sum_{v \text{ is child of } u} cap_v \times delay_v$
- $\beta_v = \sum_{v \text{ is any node}} ldelay_v \times R_{Z \to LCA(u,v)}$

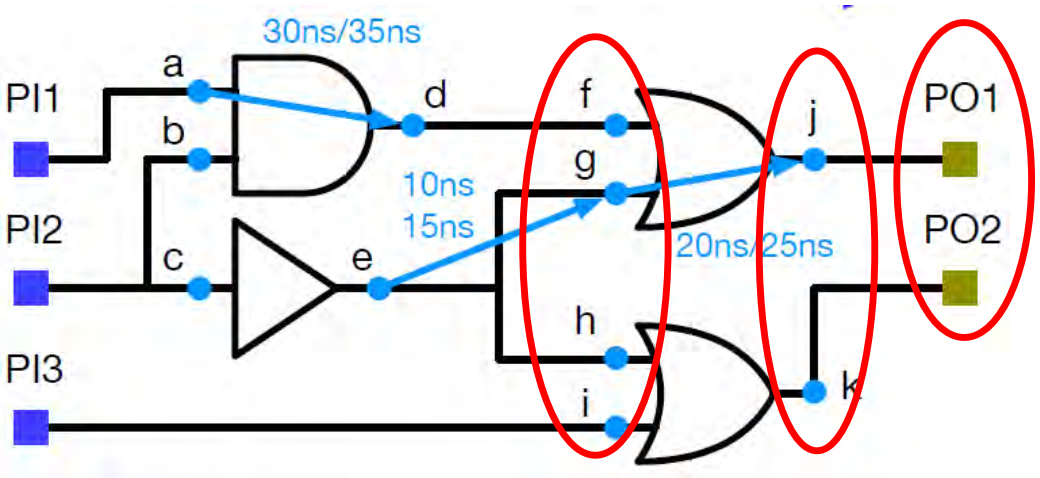


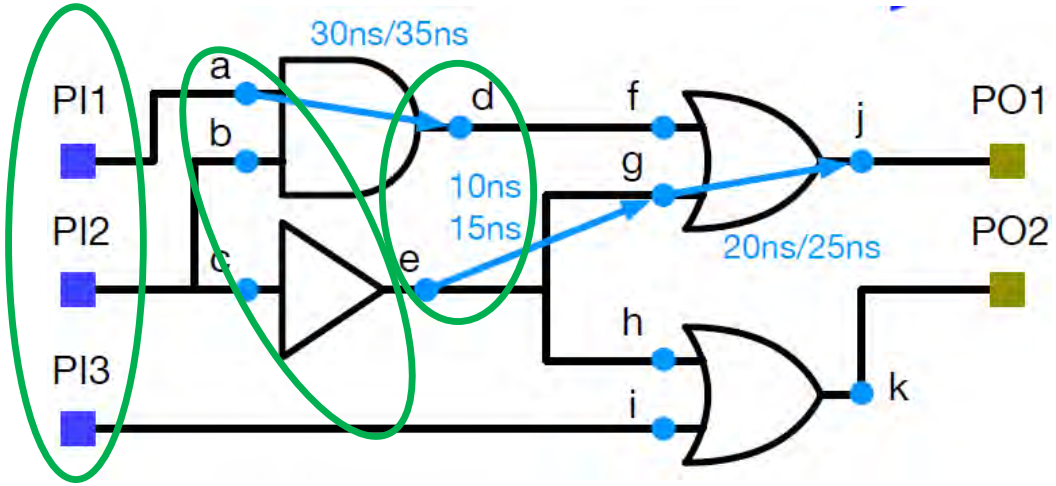




Task Graph Levelization

- Build level-by-level dependencies for timing propagation tasks.
  - Essentially a parallel topological sorting.
- Maintain a set of nodes called frontiers, and update the set using “advance” operation.

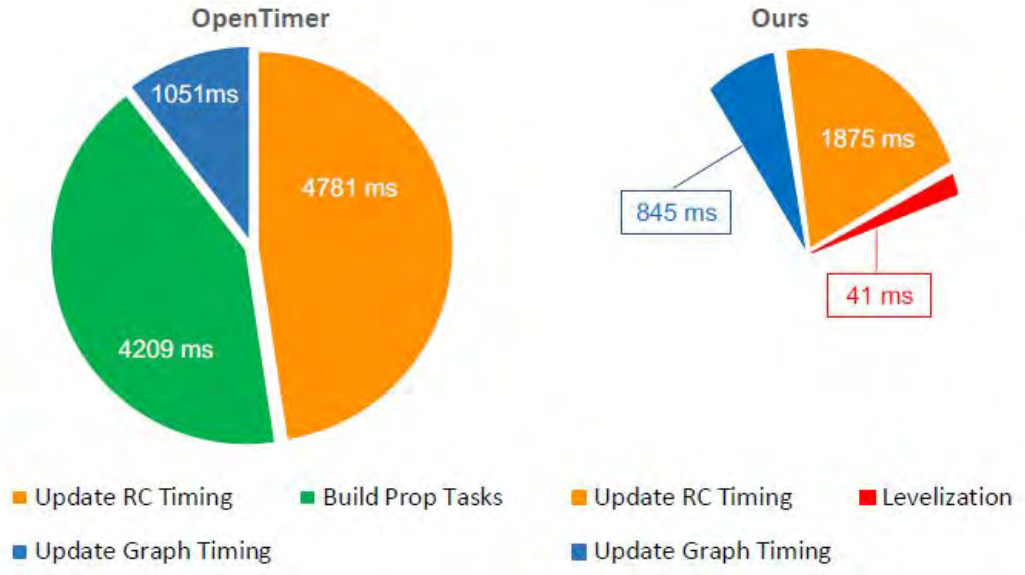




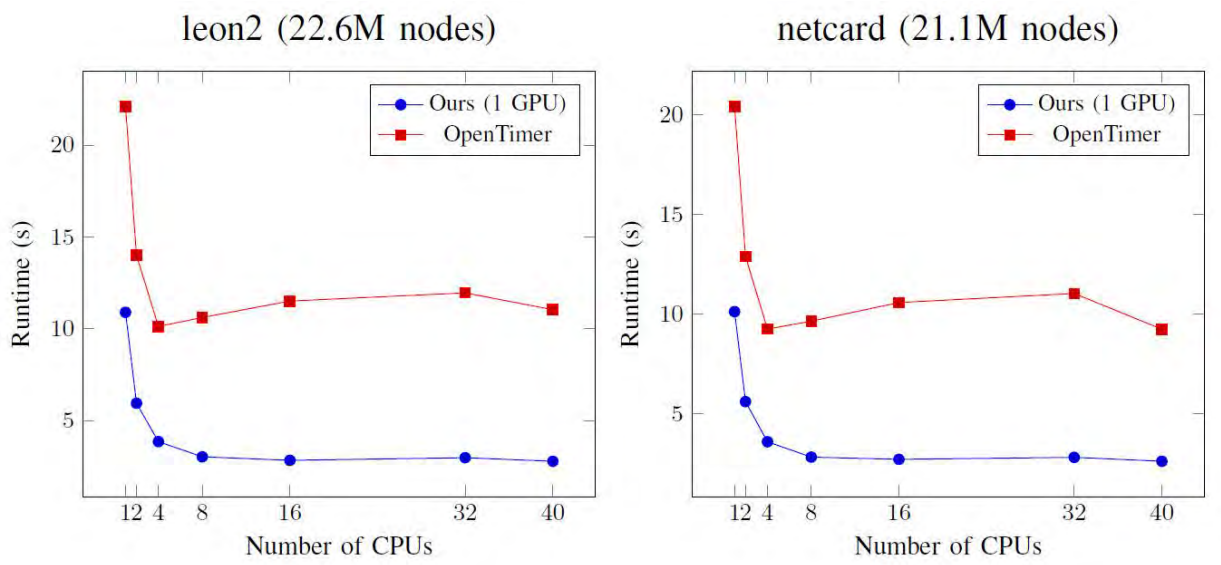
Benchmark	#nodes	Max In-degree	Max Out-degree
netcard	3999174	8	260
vga_lcd	397809	12	329
wb_dma	13125	12	95

Experimental Results for Timing Analysis

- NVIDIA CUDA, RTX 2080, 40 Intel Xeon Gold 6138 CPU cores
- Up to **3.69×** speed-up (including data copy)
- Bigger performance margin with bigger problem size



leon2 (21M nodes)



Single-core CPU with 1 GPU is close to 40-core C[illegible]

Conclusion and Future Work

- Recent efforts on accelerating backend design with GPU
  - Placement, routing, and timing analysis
- High-level challenges in physical design
  - Lack of parallelism and irregular computation patterns
  - high expectation to quality and inevitable quality degradation
  - lack of available baseline implementations and high development overhead
- Future work
  - Algorithmic innovation to accelerate practical design stages, routability- or timing-driven PnR
  - Push limits on really hard kernels, e.g., bipartite matching, mazing routing, timing propagation
  - Universal frameworks or programming models that can support CPU/GPU programming naturally

Acknowledgement

David Z. Pan @ UT Austin
Mark Ren & Brucek Khailany @ NVIDIA

Tsung-Wei Huang @ Utah
Zizheng Guo @ PKU

Thanks!

Questions are welcome

Website: https://yibolin.com
Email: yibolin@pku.edu.cn