

Accelerating EDA from 1 to N

Tribute to Prof. Martin D.F. Wong
ISPD 2024

*Presented by Evangeline F.Y. Young
March 14, 2024*

Martin's Students

1. Shinichiro Haruyama 1990
2. Khe-Sing The 1991
3. Yang Cai 1992
4. Mohankumar Guruswamy 1992
5. Ting-Chi Wang 1993
6. Tsong-Wen Her 1994
7. Kai Zhu 1994
8. Kai-Yuan Chao 1995
9. Honghua Yang 1995
10. Glenn Lai 1995
11. Shashidha Thakur 1996
12. Yao-Ping Chen 1996
13. Yung-Ming Fang 1996
14. Yao-Wen Chang 1996
15. Chung-Ping Chen 1998
16. Arthur Mak 1998
17. Hai Zhou 1999
18. Evangeline Young 1999
19. Chris Chu 1999
20. Huiqun Liu 1999
21. Youxin Gao 2000
22. James Lee 2000
23. Xiaoping Tang 2002
24. Minghorng Lai 2002
25. Ruiqi Tian 2002
26. John Croix 2002
27. Hung-Ming Chen 2003
28. Li-Da Huang 2003
29. Seokjin Lee 2003
30. Muzhou Shao 2004
31. Hua Xiang 2004
32. Yongseok Cheon 2004
33. Mustafa Ozdal 2005
34. Huaizhi Wu 2007
35. Gang Xu 2007
36. Lei Cheng 2007
37. Liang Deng 2007
38. Yu Zhong 2008
39. Hui Kong 2010
40. Tan Yan 2010
41. Lijuan Luo 2011
42. Hongbo Zhang 2012
43. Qiang Ma 2012
44. Ting Yu 2014
45. Yuelin Du 2014
46. Pei-Ci Wu 2015
47. Zhigang Xiao 2015
48. Haitong Tian 2016
49. Tsung-Wei Huang 2017
50. Leslie K. Hwang 2018
51. Daifeng Guo 2019
52. Chun-Xun Lin 2020
53. Guannan Guo 2023
54. Tin-Yin Lai 2023

Martin's Works on EDA Acceleration

- An effective **GPU** implementation of breadth-first search DAC 2010 (324)
- Accelerating aerial image simulation with **GPU** 2011
- Efficient **parallel** power grid analysis via Additive Schwarz Method 2012
- Parallel implementation of R-trees on the **GPU**, ASP-DAC 2012
- Accelerating aerial image simulation using improved **CPU/GPU** collaborative computing 2015
- Accelerate Path-based Timing Analysis with MapReduce ISPD 2015
- Accelerate analytical placement with **GPU**: A generic approach DATE 2018
- A General-purpose Distributed Programming System using **Data-parallel** Streams 2018
- A Modern C++ **Parallel** Task Programming Library 2019
- A High-Performance Accelerator for Super-Resolution Processing on Embedded **GPU** 2021
- GAMER: **GPU** Accelerated Maze Routing 2021
- **GPU-accelerated** Critical Path Generation with Path Constraints 2021
- **GPU-accelerated** Path-based Timing Analysis 2021
- Cpp-Taskflow: A General-Purpose **Parallel** Task Programming System at Scale 2021
- OpenTimer v2: A **Parallel** Incremental Timing Analysis Engine. 2021
- NovelRewrite: node-level **parallel** AIG rewriting, 2022
- A2-ILT: **GPU accelerated** ILT with spatial attention mechanism 2022
- Fast STA Graph Partitioning Framework for **Multi-GPU** Acceleration 2023
- A **GPU-Accelerated** Framework for Path-Based Timing Analysis 2023
- GAMER: **GPU-Accelerated** Maze Routing 2023

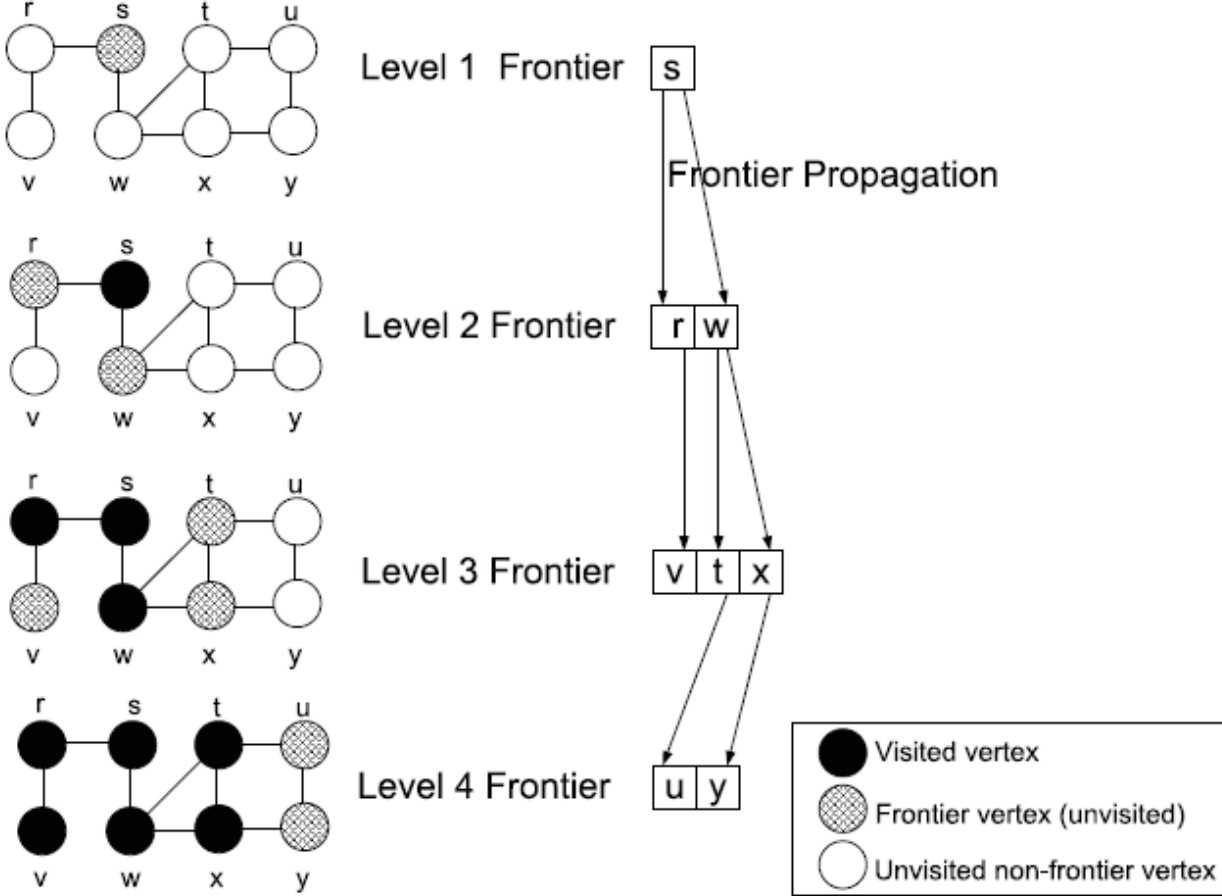


An Effective GPU Implementation of Breadth-First Search

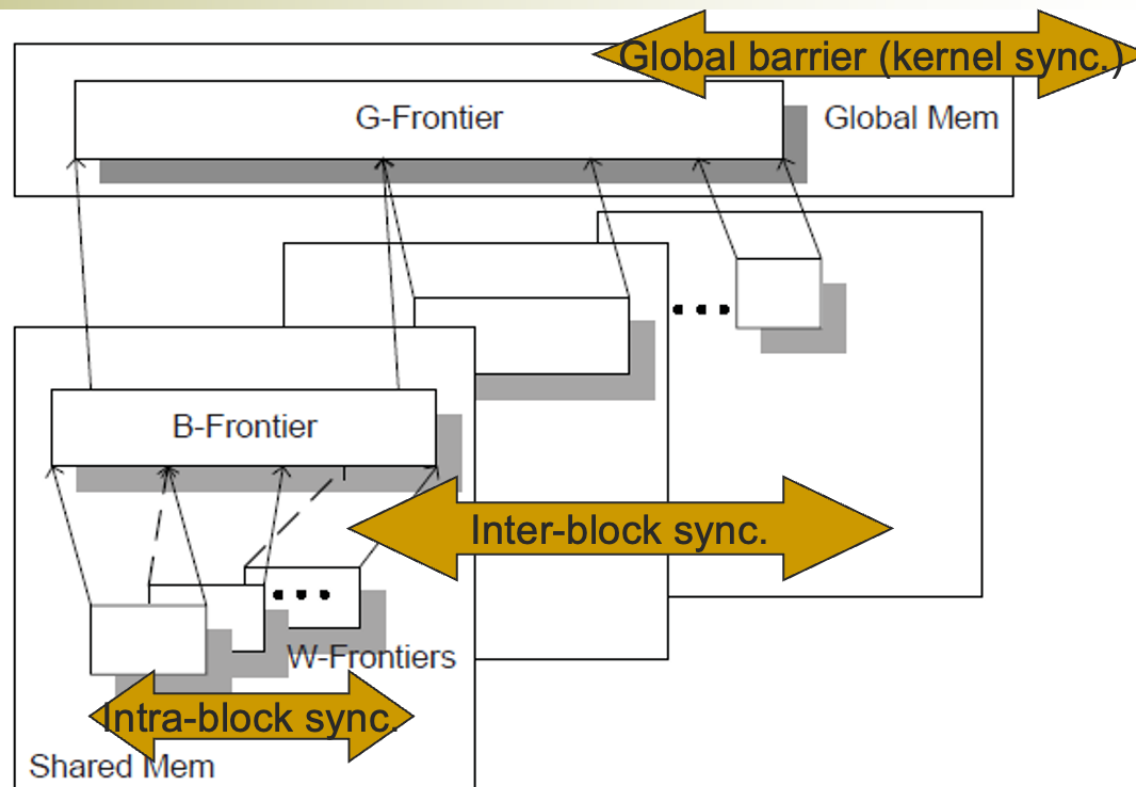
Lijuan Luo, Martin Wong and Wen-mei Hwu
Department of Electrical and Computer
Engineering, UIUC

From DAC 2010

Breadth First Search on Sparse Graph



[Hierarchical Queue Management]



Experimental results (cont.)

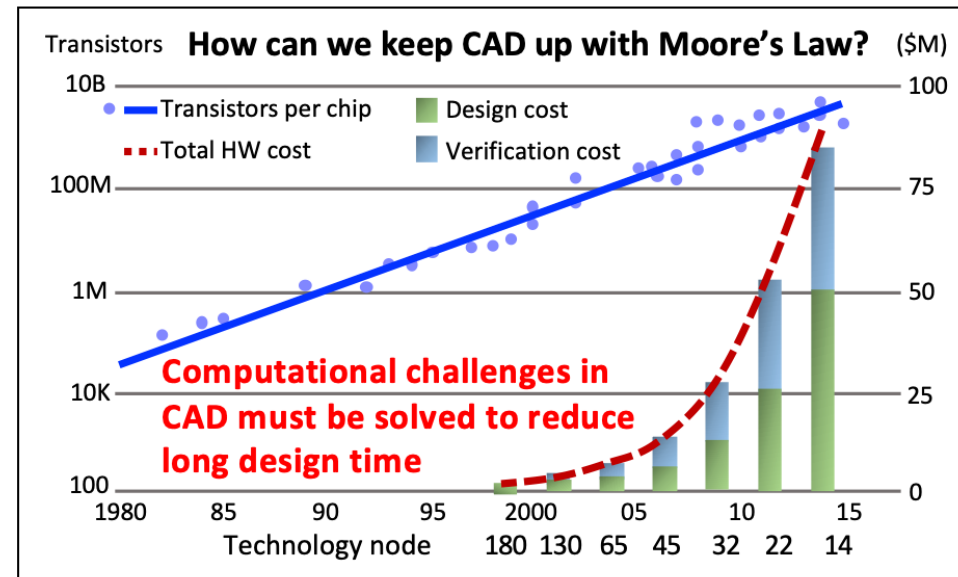
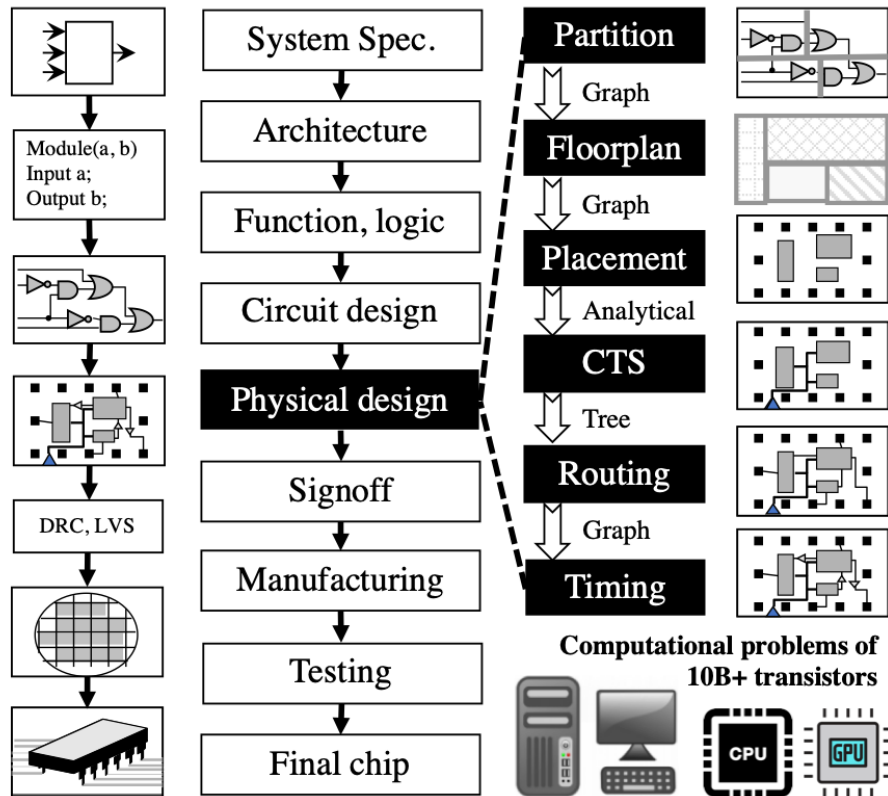
- The results on real world graphs
 - Average $\text{deg}(V)=2$, maximum $\text{deg}(V)=8$ or 9

Table 2: BFS results on real world graphs

	#Vertex	IIIT-BFS	CPU-BFS	UIUC-BFS	Sp.
New York	264,346	79.9ms	41.6ms	19.4ms	2.1
Florida	1,070,376	372.0ms	120.7ms	61.7ms	2.0
USA-East	3,598,623	1471.1ms	581.4ms	158.5ms	3.7
USA-West	6,262,104	2579.4ms	1323.0ms	236.6ms	5.6



DARPA IDEA Project¹: Parallelizing CAD



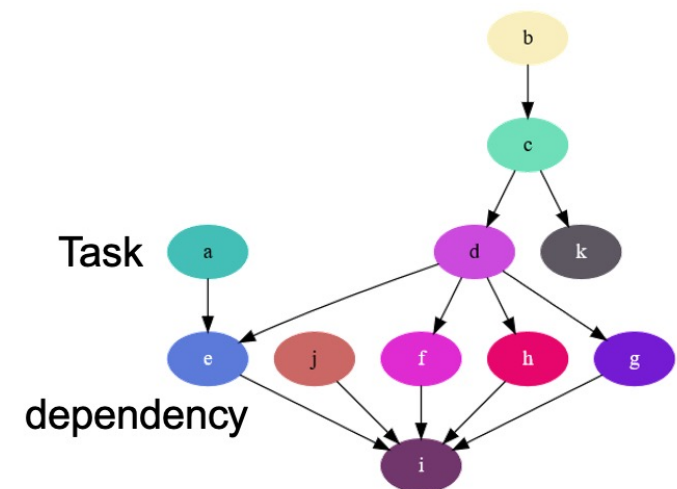
DARPA Electronic Resurgence Initiative (ERI): <https://eri-summit.darpa.mil/>

¹: "OpenTimer and DtCraft," \$427K, 06/2018-07/2019, DARPA Intelligent Design of Electronic Assets (IDEA) Program, FA 8650-18-2-7843



Need a Parallel Programming Abstraction

- From user's perspective, the biggest challenge is *transparency*
 - Programming abstraction, runtime optimization, load balancing, etc.
- Observing from the evolution of parallel programming:
 - **Task graph parallelism** (TGP) is the best model for future parallel arch
 - Capture programmers' intention in decomposing a heterogeneous algorithm into a top-down task graph scalable to different processing units
- Plenty of challenges to be solved ...
 - New applications demand new tasking models
 - Cost of control flow becomes more important
 - New accelerators demand new schedulers
 - Must value performance portability
 - Sustainability over hardware generations
 - ...

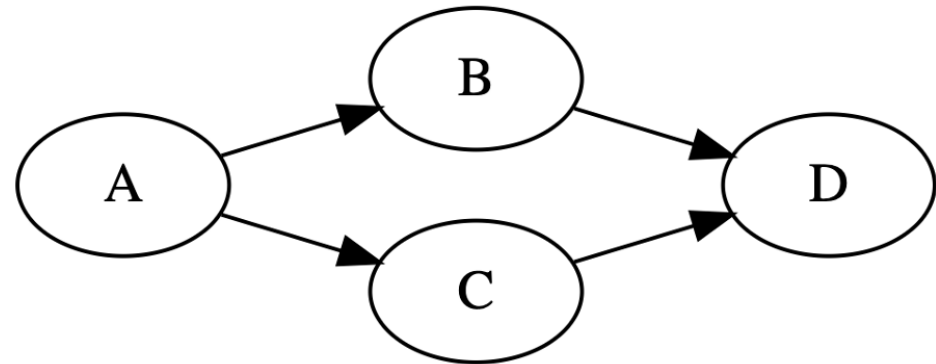




“Hello World” in Taskflow¹

```
#include <taskflow/taskflow.hpp>
int main(){
    tf::Taskflow taskflow;
    tf::Executor executor;
    auto [A, B, C, D] = taskflow.emplace(
        [] () { std::cout << "TaskA\n"; },
        [] () { std::cout << "TaskB\n"; },
        [] () { std::cout << "TaskC\n"; },
        [] () { std::cout << "TaskD\n"; }
    );
    A.precede(B, C);
    D.succeed(B, C);
    executor.run(taskflow).wait();
    return 0;
}
```

// live: <https://godbolt.org/z/j8hx3xnnx>



¹: T.-W. Huang, et. al, “Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System,” *IEEE TPDS*, vol. 33, no. 6, pp. 1303-1320, June 2022



Control Taskflow Graph Programming (CTFG)

// CTFG goes beyond the limitation of traditional DAG-based models

```
auto cond_1 = taskflow.emplace([](){ return run_B() ? 0 : 1; }); // 0: is the index of B
```

```
auto cond_2 = taskflow.emplace([](){ return run_G() ? 0 : 1; }); // 0: is the index of G
```

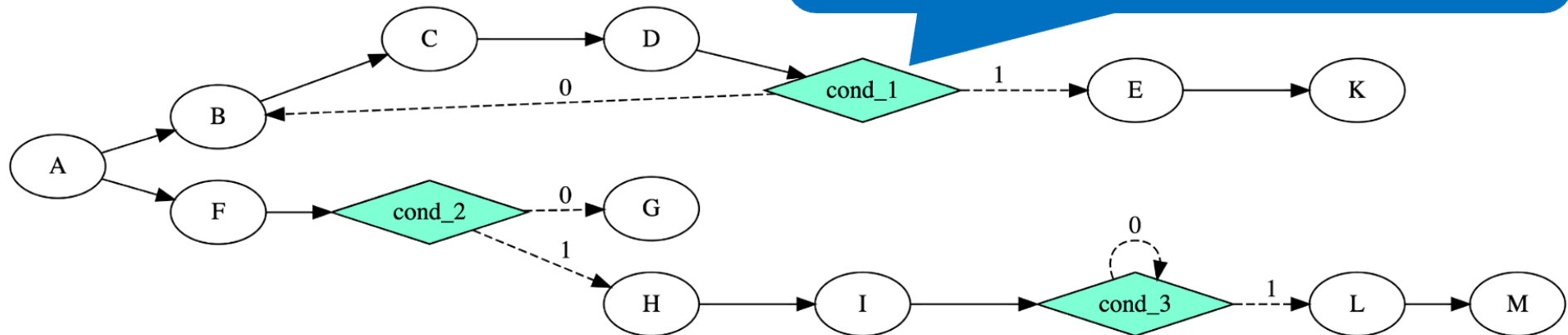
```
auto cond_3 = taskflow.emplace([](){ return loop() ? 0 : 1; }); // 0: is the index of cond_3
```

```
cond_1.precede(B, E); // cycle
```

```
cond_2.precede(G, H); // if-else
```

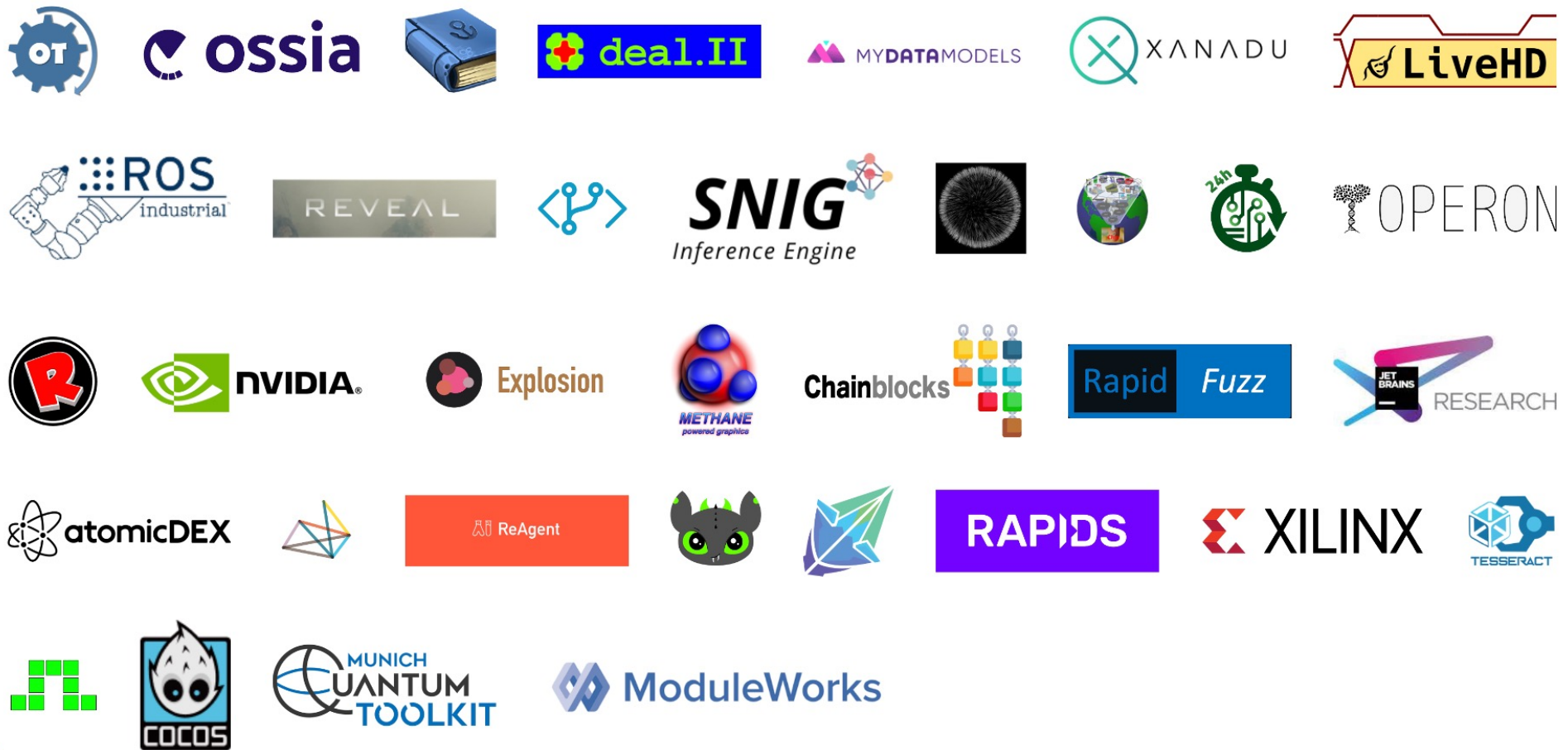
```
cond_3.precede(cond_3, L); // loop
```

Very difficult for existing DAG-based systems to express an efficient overlap between tasks and control flow ...





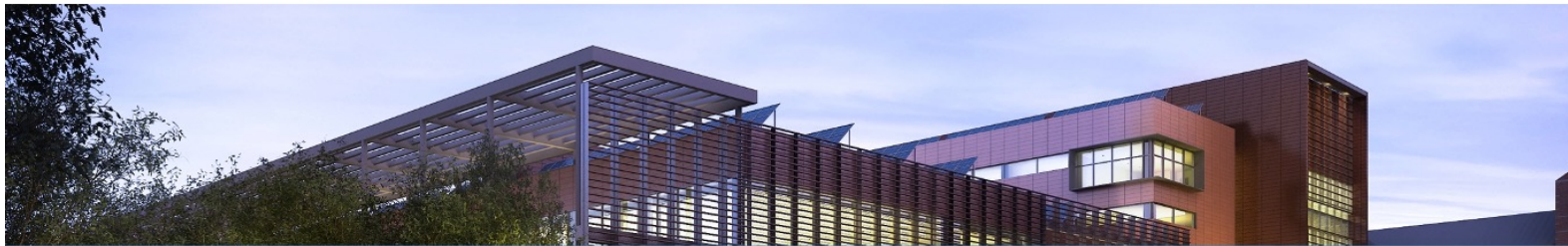
Taskflow is Being Used by 1000+ Projects



GPU-ACCELERATED PATH-BASED TIMING ANALYSIS

Guannan Guo, Tsung-Wei Huang, Yibo Lin and Martin D.F. Wong

ACM/IEEE Design Automation Conference 2021



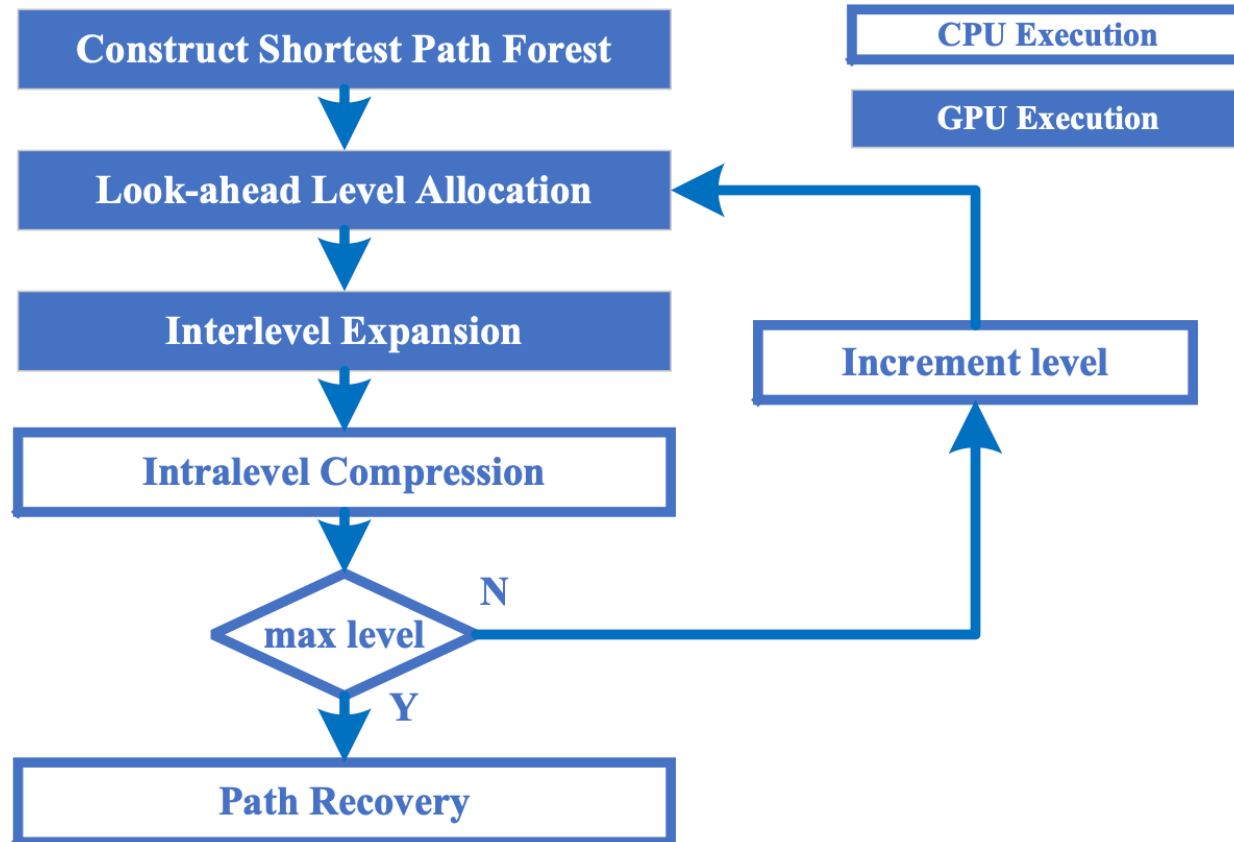
Mar 14, 2024

ECE ILLINOIS

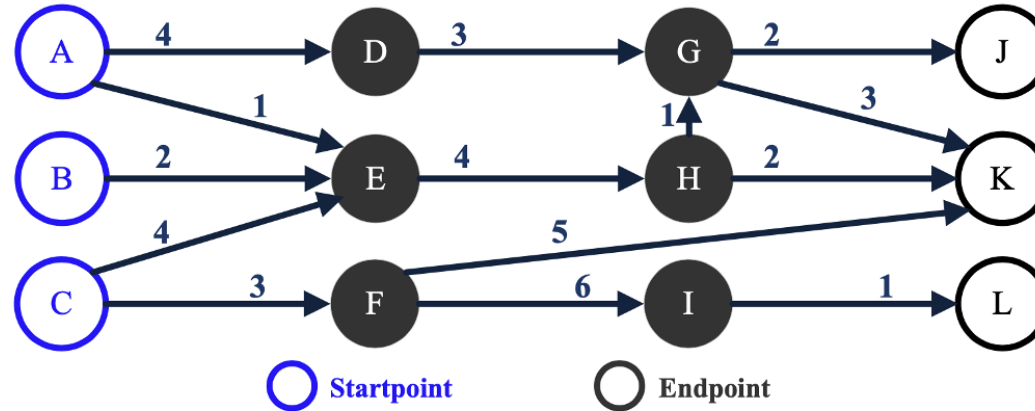
 ILLINOIS

14

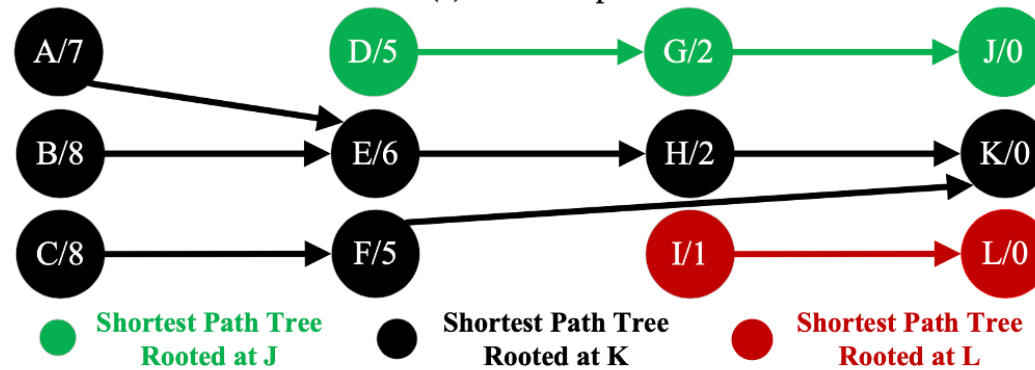
GPU-Accelerated PBA Algorithm Flow



GPU-Generated Shortest Path Forest

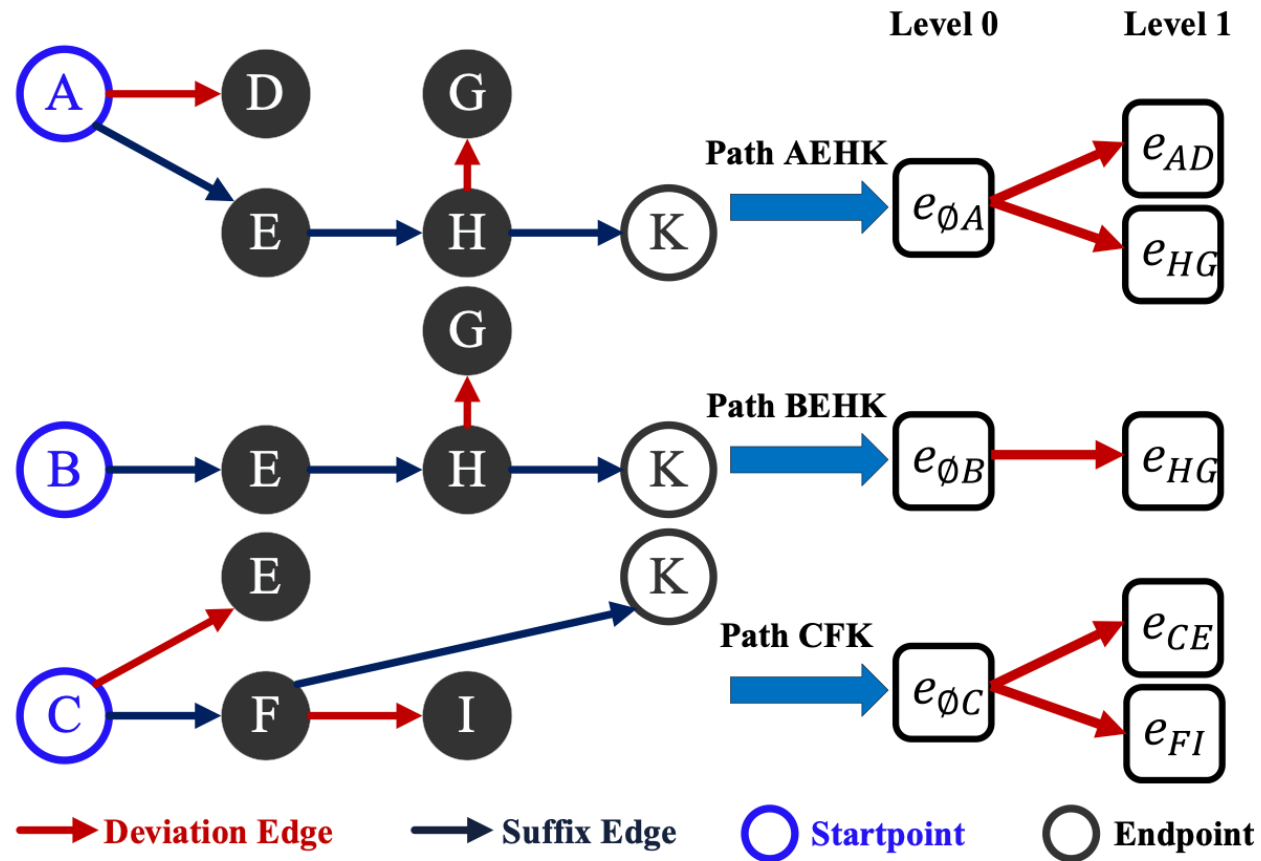


(a) STA Graph.



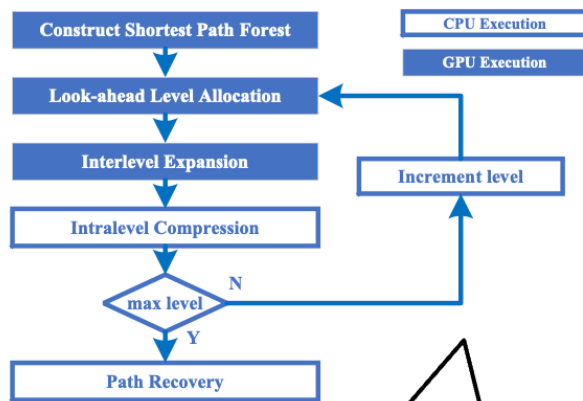
(b) Shortest path forest.

Parallel Prefix Forest Level Allocation

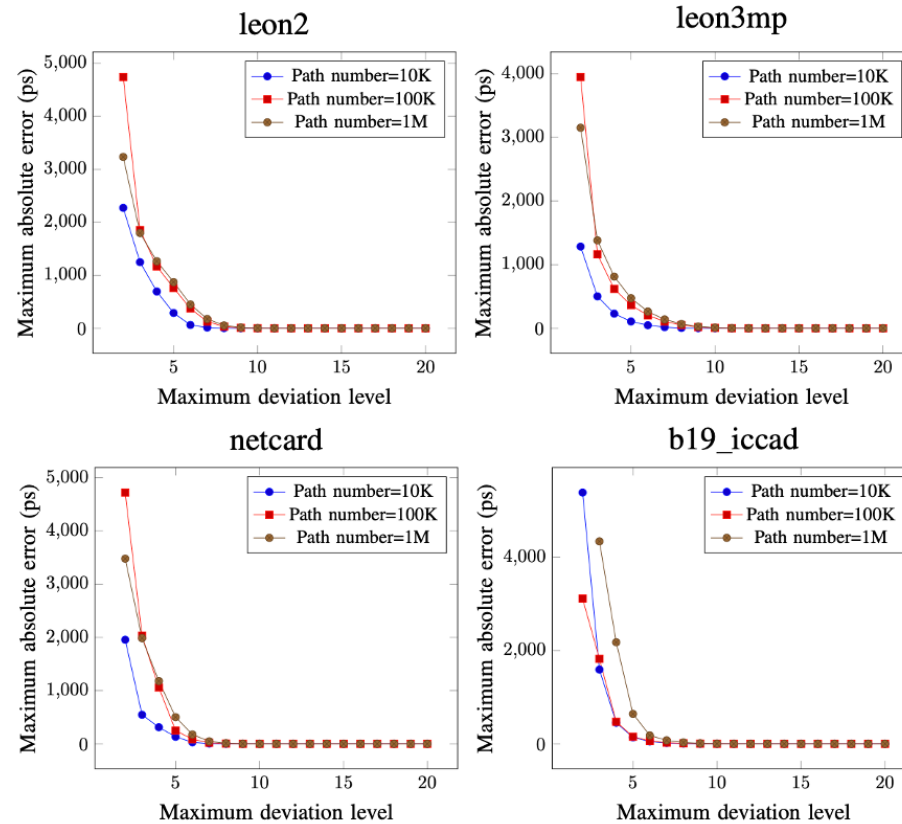


Critical Path Accuracy vs Iteration of Path Enumeration

□ Achieve decent accuracy at 10-12 GPU iterations



More GPU expansions (iterations) lead to higher numbers of paths and thus better accuracy

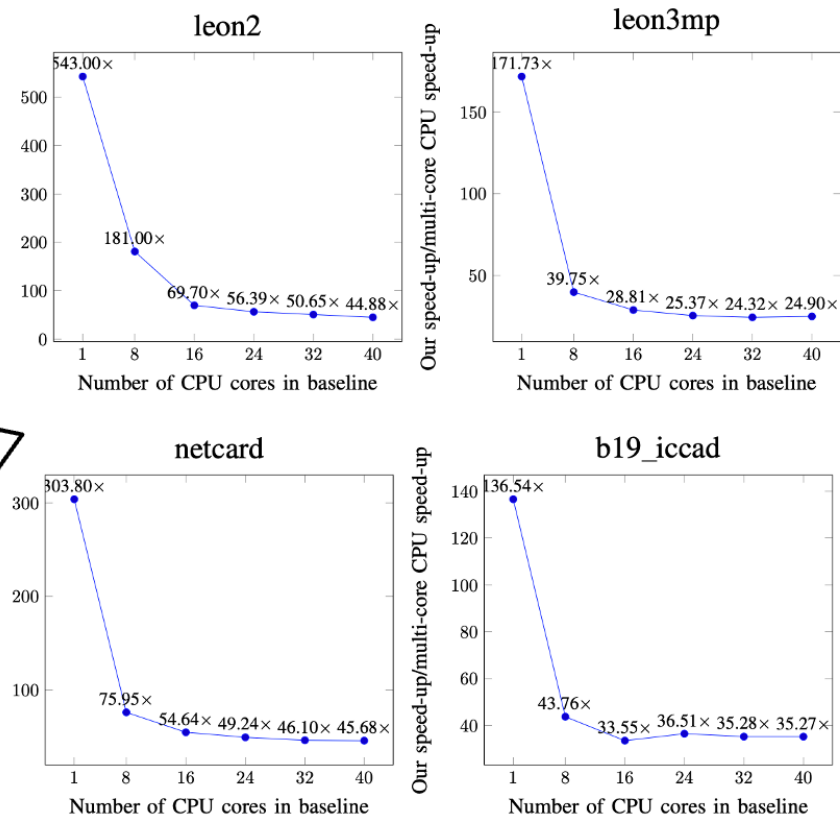


Performance vs Many-core CPUs

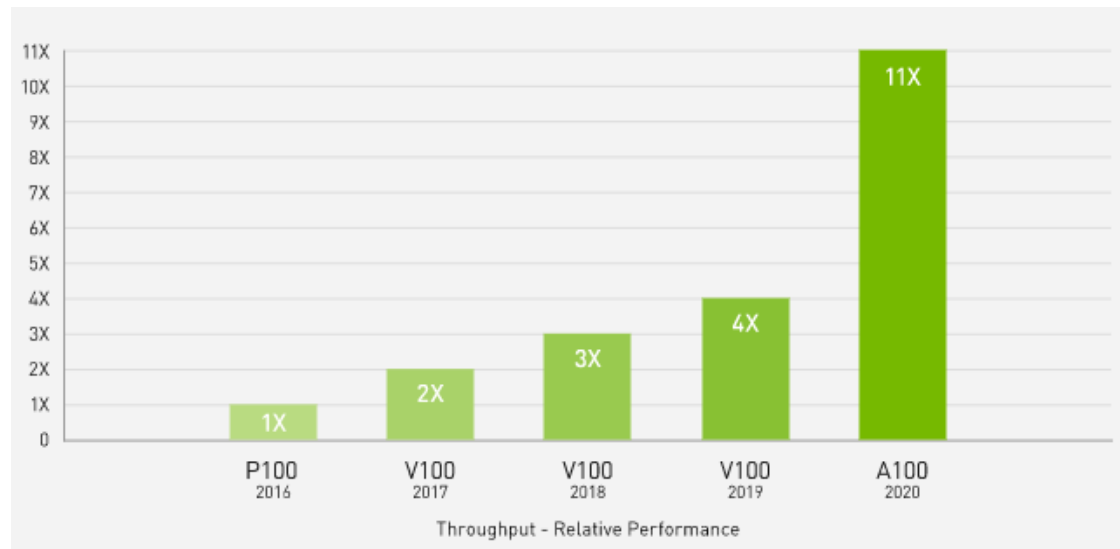
□ One GPU is even faster than OpenTimer with 40 CPUs

- 44x on leon2
- 25x on leon3mp
- 46x on netcard
- 35x on b19

In fact, according to our experiments, our GPU-accelerated PBA is always faster than OpenTimer's CPU baseline regardless of the core count



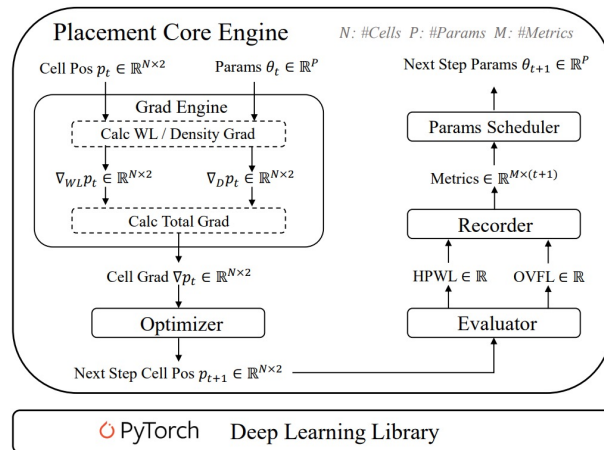
Time to Go for Massively Parallel EDA Tool!



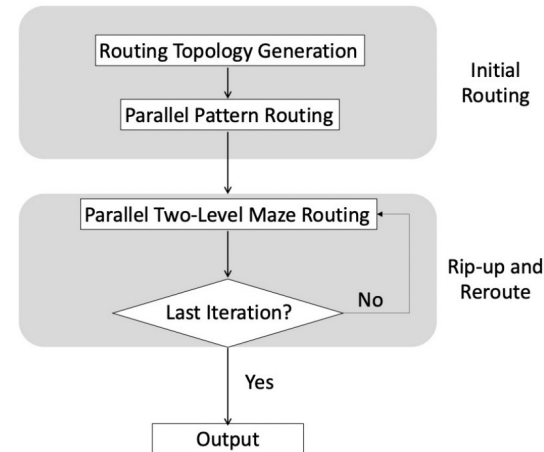
<https://www.nvidia.com/en-in/data-center/a100/>

GPU Acceleration in Place and Route

- Rapid development of GPU's computational power
- GPU acceleration becomes an important direction



GPU-Accelerated Placer Xplace [1]



GPU-Accelerated Global Router GGR [2]

[1] Lixin Liu, et al. "Xplace: an extremely fast and extensible global placement framework" Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC). 2022.

[2] Shiju Lin and Martin DF Wong. "Superfast Full-Scale GPU-Accelerated Global Routing" Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD). 2022.

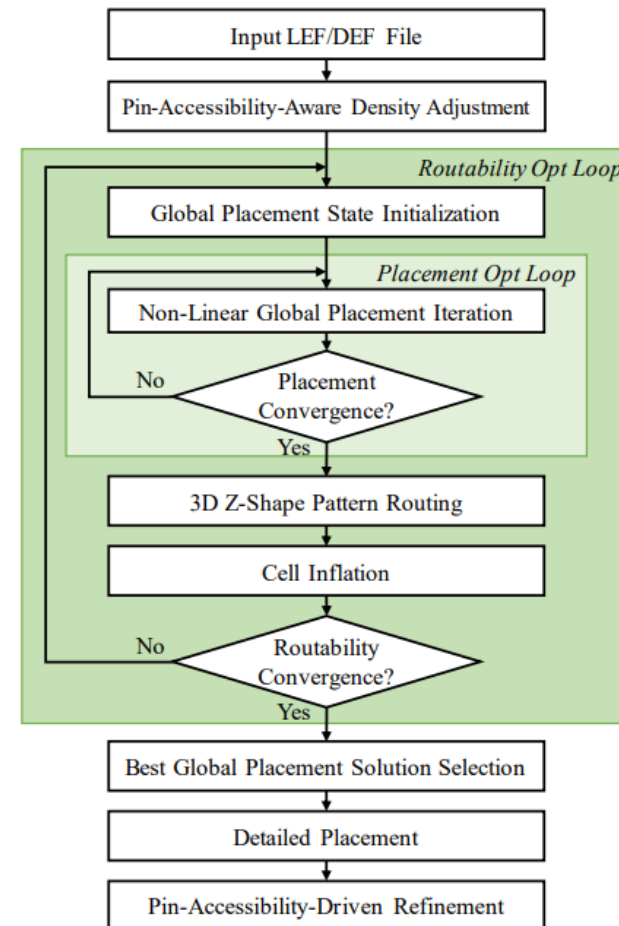
Xplace-R: Detailed-Routability-Driven Placement

Components

1. GPU-accelerated placement engine
2. GPU-accelerated routing engine
3. Routability-optimization techniques

Xplace-Route achieves

- Superior detailed-routability
- Remarkable runtime speedup



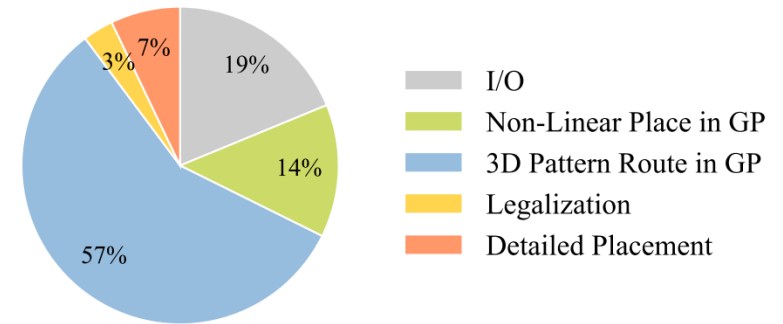
Design	Xplace-Route (DM)					DREAMPlace (NonDM)					NTUplace4dr (DM)						
	DR	WL/um	#DR Vias	#DRVs	PL/s	*DR/s	DR	WL/um	#DR Vias	#DRVs	PL/s	*DR/s	DR	WL/um	#DR Vias	#DRVs	PL/s
des_perf_1	1439840	564970	10306	10	1801	1449286	570616	19945	12	2391	1509013	630344	2931	357	2721		
des_perf_a†	2488402	575632	43854	23	645	2366671	565391	32381	12	652	2397042	569311	2602	302	4480		
des_perf_b†	1802951	543049	1323	10	415	1810931	555033	14920	12	2018	1769864	551066	1887	332	468		
edit_dist_a†	5750598	1015469	422480	26	2521	5687881	1012545	426136	14	2640	6576184	1125780	1188119	457	5736		
fft_1	513869	185766	3478	13	769	522764	188497	8011	9	1375	572738	199139	1397	91	1376		
fft_2	621300	191162	1186	12	953	599272	187689	8886	8	423	729743	195227	993	99	753		
fft_a	1137087	192259	781	13	1127	1079780	192945	4533	8	1569	1176459	199888	1888	96	2668		
fft_b	1282342	213953	16661	14	941	1252724	211800	21013	8	895	1267726	208975	39082	111	835		
matrix_mult_1	2648037	828540	12373	26	6933	2712367	812150	79049	14	1344	2699517	892013	4186	339	3517		
matrix_mult_2	2678569	857402	11421	26	8809	2723576	842283	69799	15	1482	2733713	906449	5062	360	8217		
matrix_mult_a	3941724	848261	7054	12	6536	3881128	864485	26127	16	5073	4190876	869526	4089	381	12186		
matrix_mult_b†	3649667	782850	45894	10	1301	3670952	789352	73368	15	1280	3921403	804760	61470	318	1188		
matrix_mult_c†	3685351	791702	7518	10	3952	3712692	816093	26761	16	6080	4331793	855656	3784	340	5535		
pci_bridge32_a†	651346	145663	4001	5	3163	650836	149042	5969	8	2649	606431	143725	1729	125	2022		
pci_bridge32_b†	1007617	147953	347	5	162	1012909	150805	2158	11	320	811280	137266	426	88	119		
superblue11_a†	40316503	5659845	1202	78	6906	39973449	5645598	1182	73	6953	64214794	7218914	533505	14321	16325		
superblue12	42780736	10508482	29569	311	22614	42477241	11254375	3283142	91	27160	48273446	11768924	31293	7493	24229		
superblue14	27959928	4341696	366	70	11901	28386548	4435721	418	53	14639	31959227	5340941	1343	3166	11635		
superblue16_a†	31460619	4644486	4489	82	16539	31455744	4732382	3871	55	15284	36908305	5632423	14586	3654	30615		
superblue19	20451060	3582347	8255	58	9310	20556809	3612264	7266	42	7285	22402103	4035778	8041	3245	7839		
Mean	9813377	1831074	31628	41	5365	9799178	1879453	205747	25	5076	11952583	2114305	95421	1784	7123		
Ratio	1.00	1.00	1.00	1.00	1.00	1.00	1.03	6.51	0.60	0.95	1.22	1.15	3.02	43.82	1.33		
Design	Xplace (DM)					DREAMPlace + CUGR (NonDM)					Enhanced DREAMPlace + CUGR (NonDM)						
	DR	WL/um	#DR Vias	#DRVs	PL/s	*DR/s	DR	WL/um	#DR Vias	#DRVs	PL/s	*DR/s	DR	WL/um	#DR Vias	#DRVs	PL/s
des_perf_1	1447376	569167	20064	7	2726	1447442	568831	19829	204	2400	1440466	561512	19976	205	2499		
des_perf_a†	2338413	558472	28028	7	589	2866082	605591	240085	115	1181	2366966	565054	33102	218	629		
des_perf_b†	1816033	557017	14768	6	2310	1828936	566050	14253	173	1937	1820022	546426	13966	251	1901		
edit_dist_a†	5671918	1008135	405727	8	2820	6045810	1059270	608067	164	2985	5838307	1080238	493280	202	2876		
fft_1	517999	187120	7802	4	1291	516922	188519	8157	44	1317	524240	188222	8485	97	1316		
fft_2	598808	188353	9200	3	394	608394	196025	5552	65	1739	629227	194278	6105	89	1552		
fft_a	1093206	193126	4677	4	1476	1345219	219195	7641	49	1277	1076563	192419	4855	86	1585		
fft_b	1249450	203681	32851	4	928	1405437	224950	44052	41	1224	1255215	211532	21385	90	1090		
matrix_mult_1	2703662	807649	76976	7	1306	2711580	821025	79519	84	1318	2722410	812788	79271	164	1325		
matrix_mult_2	2718941	840570	68624	7	1694	2710509	845701	56748	125	1681	2709817	833468	54706	160	1655		
matrix_mult_a	3877049	861600	25646	9	5540	4708376	914328	41858	113	1400	3892047	865240	25770	154	5268		
matrix_mult_b†	3657963	791901	66643	8	1327	4465439	864227	77742	112	1332	3674444	785578	47308	147	1178		
matrix_mult_c†	3725954	814420	26195	8	6008	4831883	919329	26265	115	8360	3706182	816044	26340	144	5857		
pci_bridge32_a†	642818	148475	5631	3	3130	653934	149437	5941	16	2786	646469	148451	5613	106	2877		
pci_bridge32_b†	981717	149362	2076	4	276	1029276	152185	2123	23	350	1028863	151654	2148	127	338		
superblue11_a†	40316503	5659845	1202	42	7399	45092596	5907673	1314	1043	7428	40335827	5637829	967	1224	6464		
superblue12	42734765	10840792	3122560	58	25770	45921282	11068935	21863	2020	18623	46529015	11083954	15744	2477	21085		
superblue14	27955769	4339771	347	29	13082	28925588	4234243	415	943	5143	28548859	4184970	365	1121	5816		
superblue16_a†	31460619	4644486	4489	31	16771	31953926	4634390	2495	677	12834	31082724	4568077	2410	815	12601		
superblue19	20468106	3584954	8289	23	9165	22266531	3709667	6116	869	5766	22642721	3733021	19036	1326	55805		
Mean	9798853	1847445	196590	14	5200	10566758	1892479	63502	350	4054	10123519	1858038	44042	460	6686		
Ratio	1.00	1.01	6.22	0.33	0.97	1.08	1.03	2.01	8.59	0.76	1.03	1.01	1.39	11.30	1.25		

We use Innovus to run detailed routing under the same setting on the same machine and report DR WL, #DR Vias, #DRVs and DR time.

* The DR time (DR/s) may not precisely reflect the placement routability because Innovus will early terminate its detailed router if a design has low routability.

TABLE I Detailed Routing Metrics and Runtime Results on the ISPD 2015 Contest Benchmarks.

Experimental Results (Summary)



Settings:


- Benchmarks: ISPD 2015
- Commercial Detailed Router and Metric Evaluator: Cadence Innovus

Summary: **Only take 41 seconds (on average) to conduct DR-driven placement**

- Compared to **Xplace** (Deterministic version):
Only take 27 seconds extra runtime, achieving **622%** #DRVs reduction
- Compared to **DREAMPlace + CUGR**:
9× PL time speedup, 8% shorten WL, 3% fewer #DR Vias, **200%** #DRVs reduction
- Compared to **NTUplace4dr**:
44 × PL time speedup, 22% shorten WL, 15% fewer #DR Vias, **300%** #DRVs reduction

GitHub Code of Xplace-R (aka Xplace 2.0)

<https://github.com/cuhk-eda/Xplace>

 **cuhk-eda / Xplace** Public

Xplace 2.0: An Extremely Fast, Extensible and
Deterministic Placement Framework with Detailed-
Routability Optimization

 BSD-3-Clause license

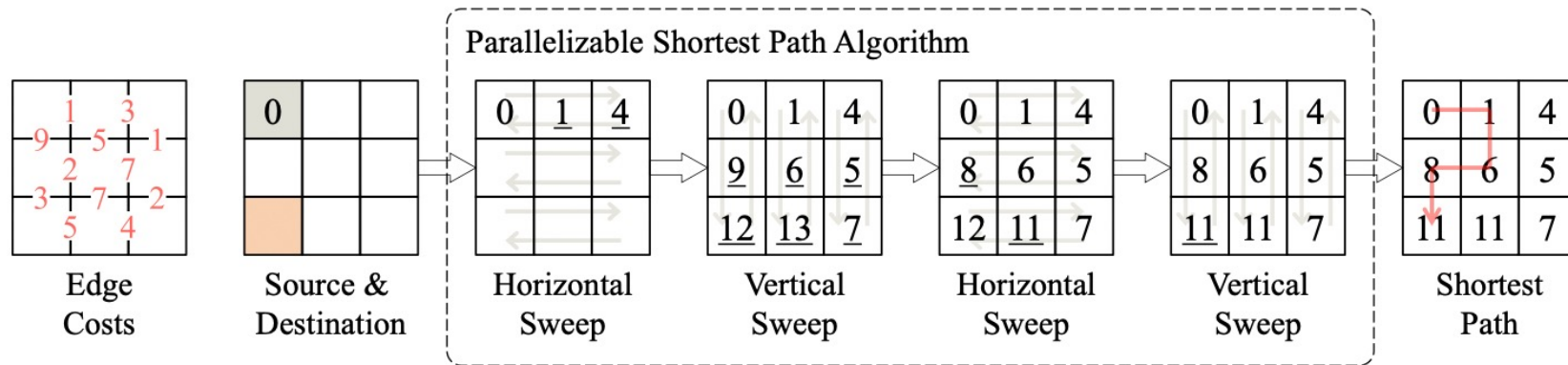
GPU accelerated CUGR

- Parallelize all 3 stages in CUGR 2.0 [2]:
 - Sparse Graph Maze Routing [1]
 - L-shape Routing
 - DAG Routing

[1] Shiju Lin, Jinwei Liu, Evangeline F.Y. Young, & Martin D.F. Wong. "GAMER: GPU accelerated maze routing" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022.

[2] Jinwei Liu and Evangeline F.Y. Young. "EDGE: Efficient DAG-based Global Routing Engine" Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC). 2023.

Sparse Graph Maze Routing – GAMER [1]



Shortest Path via Alternating Sweeps

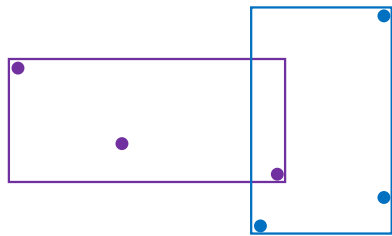
d_i : current shortest distance to G-cell i

c_i : wire cost between G-cell $i - 1$ and G-cell i

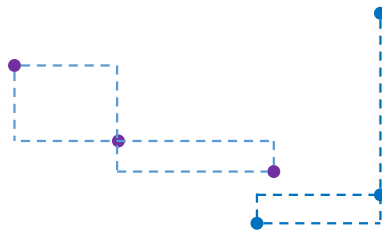
$$d_i^* = \min_{0 \leq j \leq i} \left(d_j + \sum_{k=j+1}^i c_k \right)$$

[1] Shiju Lin, Jinwei Liu, Evangeline F.Y. Young, & Martin D.F. Wong. "GAMER: GPU accelerated maze routing" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022.

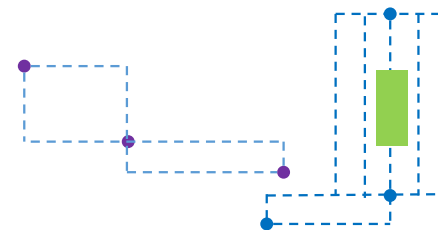
Net-based Parallelism



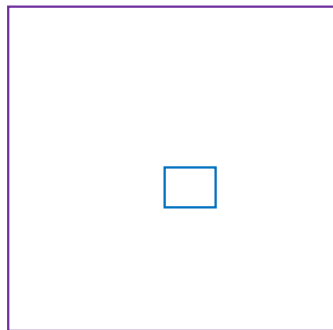
Previous method (BB)
overlap (pessimistic)



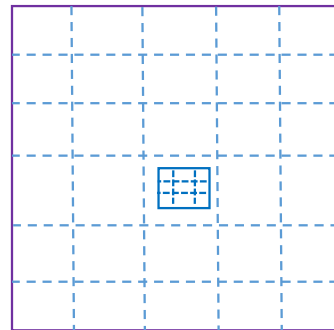
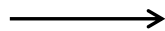
RSMT-based L-shape graph
no overlap (accurate)



Augmented routing graph
(to escape the congested green area)
no overlap (accurate)



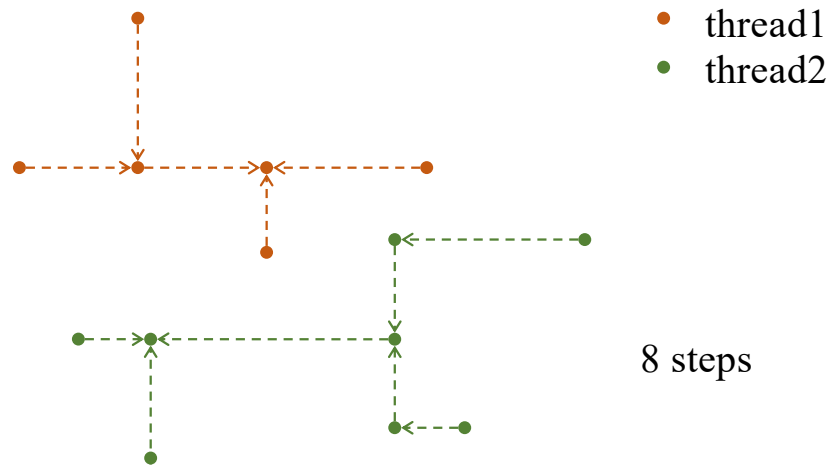
Previous method (BB)
overlap (pessimistic)



Sparse routing graph
no overlap (accurate)

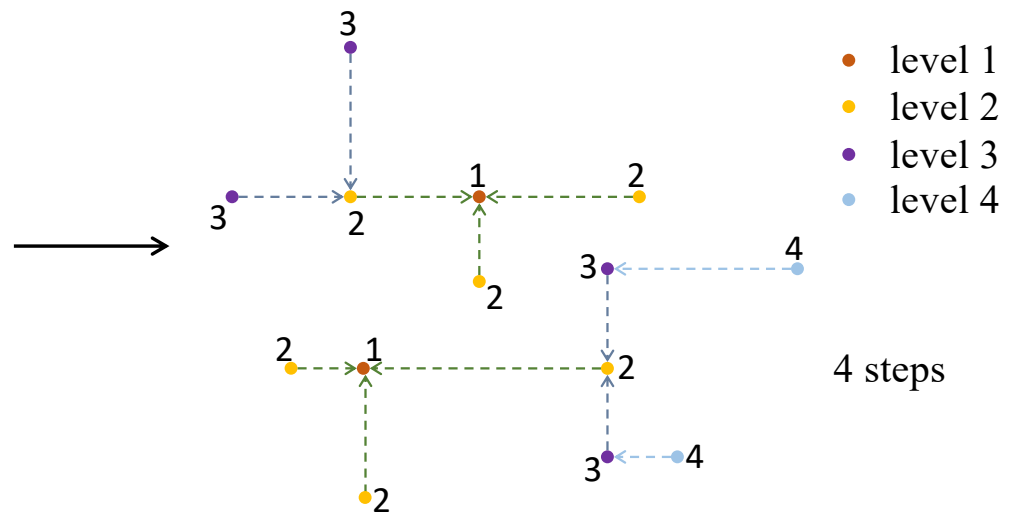
← unselected routing row
← Selected routing row

Level-wise Parallelism in DAG Routing



8 steps

Net-based parallelism:
Use one thread to handle a net.



4 steps

Level-wise parallelism:
All nodes at the same level
will be handled in parallel.

Acceleration in Logic Synthesis

- Rewrite [1] (×32.91)
- Refactor [2] (×42.7)
- Rebalance [2] (×14.8)
- Resyn2 [2] (×45.9)
- Technology map [3] (×128.7)
- K-resub [4] (×41.9)

<https://github.com/cuhk-eda/CULS>

- Standalone mode

To interact with the command prompt, run

```
./gpuls
```

You can also directly execute a script, e.g.,

```
./gpuls -c "read ../abc/i10.aig; resyn2; write i10_resyn2.aig"
```

- ABC patch mode

The usage is the same as ABC. For instance,

```
./abcg -c "read ../abc/i10.aig; gget; gresyn2; gput; print_stats; cec -n"
```

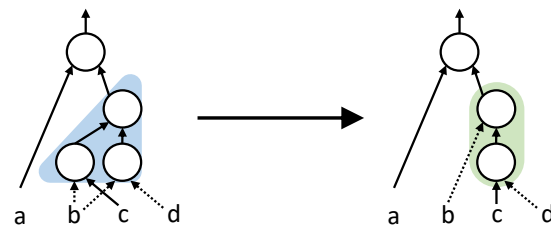
- [1] Shiju Lin, Jinwei Liu, Tianji Liu, Martin D.F. Wong and Evangeline F.Y. Young, "NovelRewrite: Node-Level Parallel AIG Rewriting", DAC 2022.
[2] Tianji Liu et al., Rethinking AIG Resynthesis in Parallel, DAC 2023.
[3] Tianji Liu, et al., "FineMap: A Fine-grained GPU-Parallel LUT Mapping Engine", ASP-DAC 2024.
[4] Yang Sun, et al., "Massively Parallel AIG Resubstitution", DAC 2024.

Background

- AIG resynthesis is equipped with many algorithms
 - rewrite (**rw**), refactor (**rf**), (and-)balance (**b**), etc.
 - Different algorithm applies different optimization strategy
- A commonly used AIG resynthesis flow
 - (**resyn2**) b; rw; rf; b; rw; rw -z; b; rf -z; rw -z; b
- GPU has shown its effectiveness in accelerating rewriting
 - S. Lin et al., "NovelRewrite: node-level parallel AIG rewriting," DAC 2022.
- To fully accelerate resyn2, parallel refactor and balance are indispensable

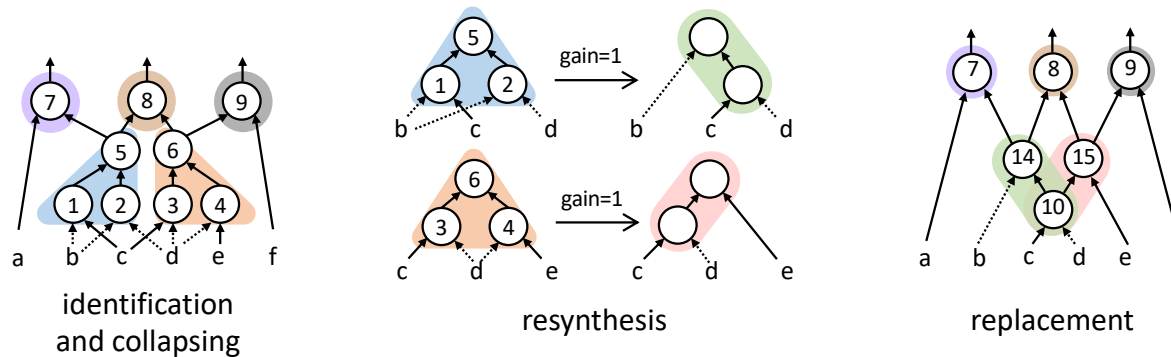
AIG Refactoring

- Optimizes the area of AIG
- Resynthesize large logic cones
 - By algebraic factoring from collapsed SOPs
 - Using one cut per node to reduce time complexity
- Replace the original cone if the resynthesized cone has positive gain



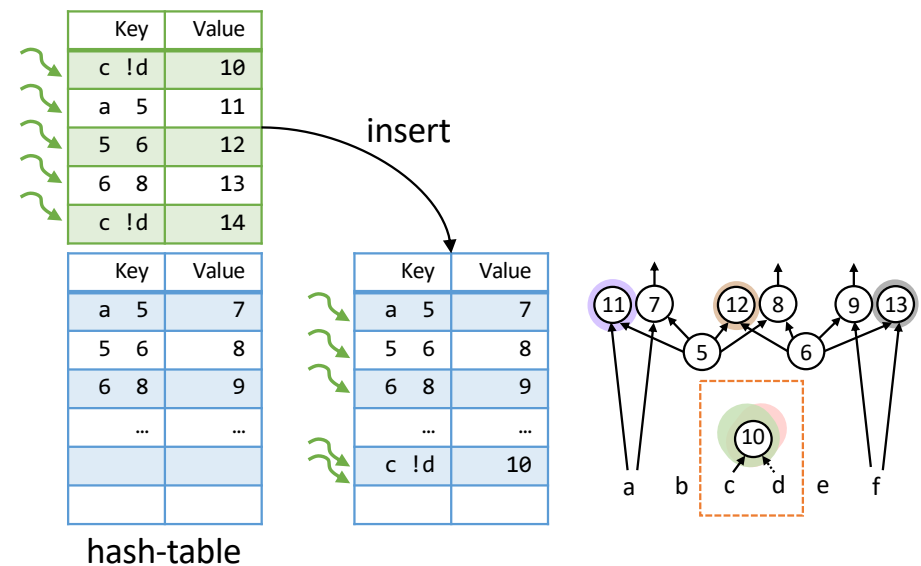
Parallel Refactoring

- Three Stages:
 - **Identify** and collapse logic cones with **level-wise parallelism** that are
 - (1) Disjoint from each other (k-bounded fanout free cone), and
 - (2) Cover all the logic (non-PI nodes) in the AIG
 - **Resynthesize** all local functions **in parallel**
 - **Replace** the original cones by resynthesized cones **in parallel**



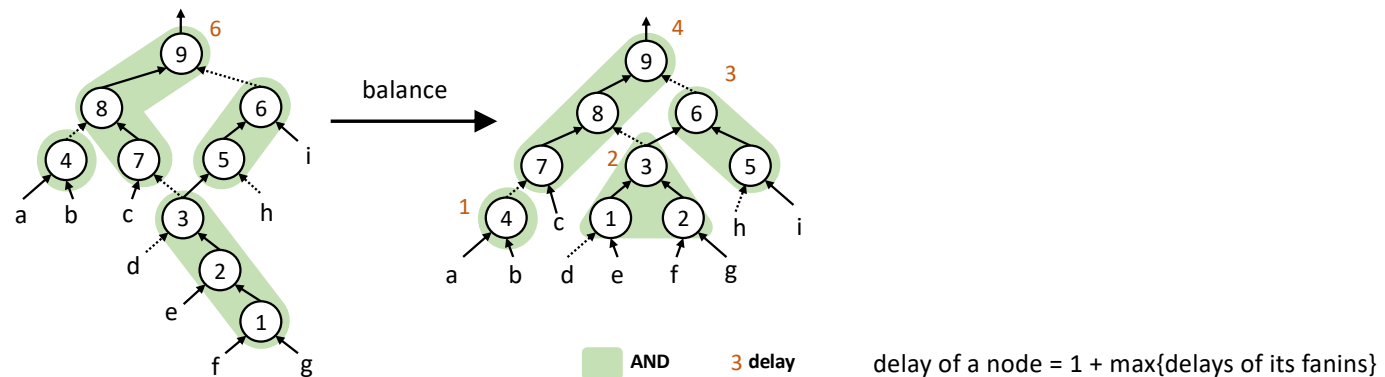
GPU Hash-table

- Enable logic sharing by structural hashing
 - Developed a GPU hash-table supporting batched node insertion and retrieval
 - Ensures that only one node can exist with a particular fanin and negation status



AIG Balancing

- Identify and collapse n-input AND gates with a tree structure
- Recursively balance the cut nodes, with their delays obtained
- Reconstruct the cone by AND-ing the inputs with the order of ascending delay



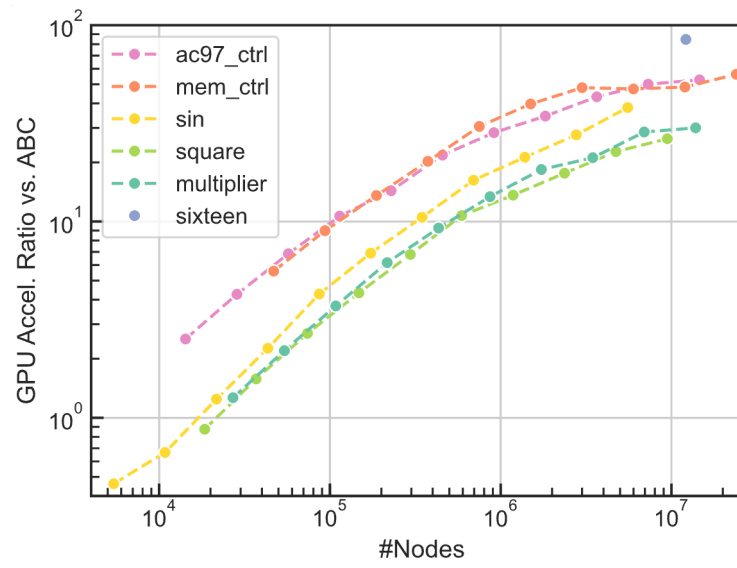
Experimental Results - Acceleration & QoR

- Tested on enlarged benchmarks from the EPFL and IWLS 2005 Suite
- Similar or slightly better QoR (AIG area and delay) compared with ABC

Benchmarks	ABC rf_resyn		GPU rf_resyn		ABC resyn2		GPU resyn2 (rwz ×2)		GPU rf (×2)		
	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time	#Nodes / Levels	Time	
twentythree	17396577 / 104	1490.4	17348255 / 98	17.3	16931332 / 72	4174.5	16915942 / 68	55.2	18397973 / 103	10.7	
twenty	15514368 / 94	1359.7	15480873 / 90	15.2	15095643 / 65	3633.5	15079948 / 65	49.1	16421342 / 95	9.2	
sixteen	12147445 / 99	1115.4	12118520 / 99	13.2	11765351 / 68	2760.7	11757432 / 64	40.4	12911473 / 101	7.6	
div_10xd	56788992 / 4404	2273.1	48652485 / 4373	104.7	41665780 / 4388	8028.8	41689305 / 4422	239.8	48779264 / 4422	20.5	
hyp_8xd	54539008 / 24785	2104.8	54539008 / 24787	345.2	54193719 / 24785	10771.1	54205696 / 24671	653.8	54539008 / 24790	32.7	
mem_ctrl_10xd	46615552 / 105	3448.9	47312818 / 108	54.7	43777052 / 92	6936.8	44821695 / 94	132.9	47444992 / 109	16.9	rf -z;
log2_10xd	31011840 / 366	1434.7	31371816 / 390	34.3	29946093 / 358	4879.4	29966717 / 358	91.5	31552512 / 396	7.9	z; b
multiplier_10xd	26471424 / 265	947.1	26469376 / 265	33.0	24957961 / 262	3509.3	24949760 / 262	77.5	26565632 / 265	5.5	
sqrt_10xd	23618560 / 5182	3904.8	23014400 / 5174	54.8	18884491 / 6020	5639.8	18800640 / 5928	131.5	24862720 / 5365	12.4	
square_10xd	17935360 / 250	606.7	17888256 / 250	22.9	17091593 / 248	2343.3	17052614 / 249	58.0	18081792 / 250	3.7	e in
voter_10xd	9951232 / 59	370.6	10874377 / 63	14.6	8845336 / 66	1432.4	8961831 / 60	33.2	11480064 / 66	2.5	
sin_10xd	5285888 / 179	232.0	5319346 / 187	6.2	5156077 / 163	851.0	5158131 / 161	16.2	5357568 / 213	1.5	
ac97_ctrl_10xd	10956800 / 11	699.6	11020147 / 9	13.5	10490213 / 10	1375.5	10660403 / 10	32.0	11144192 / 12	3.4	
vga_lcd_5xd	2918528 / 18	189.5	2946898 / 20	5.2	2903540 / 24	439.6	2903968 / 23	10.7	2952480 / 26	1.2	
Geomean Ratio vs. ABC	1.000 / 1.000	1.0	0.996 / 1.000	39.5× accel.	1.000 / 1.000	1.0	1.003 / 0.982	45.9× accel.	0.983 / 0.980	42.7× accel.	

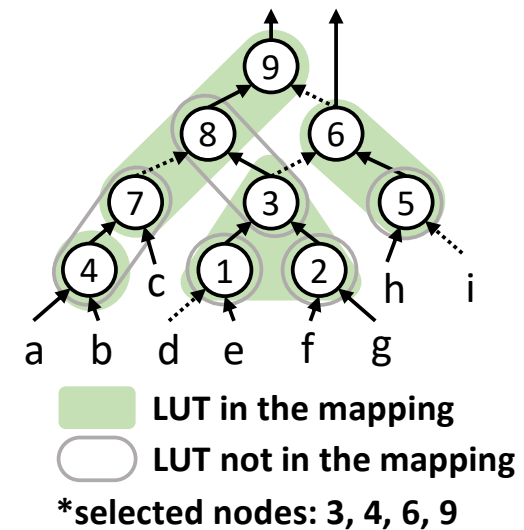
Experimental Results - Scalability

- Speedup with different AIG sizes on the rf_resyn script
- GPU implementation is faster than ABC when #node > 30k



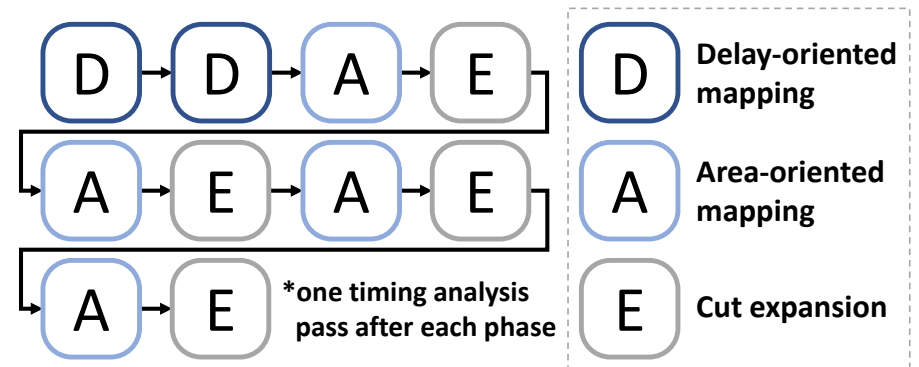
LUT Mapping

- Objective
 - Transform a Boolean network (e.g., AIG) into a k-input LUT network
 - Minimize LUT count (area) & level (delay)
- Common approach
 - Assign each node a **representative cut**
 - Select a subset of representative cuts such that their cones cover the entire network



LUT Mapping

- Flow of state-of-the-art LUT mapper
 - Multiple “mapping phases”, delay- or area-oriented
 - Cut expansion phase



- Each phase incrementally improves the mapped network
- A timing analysis pass after each mapping/cut expansion phase
 - Compute the required time of each node

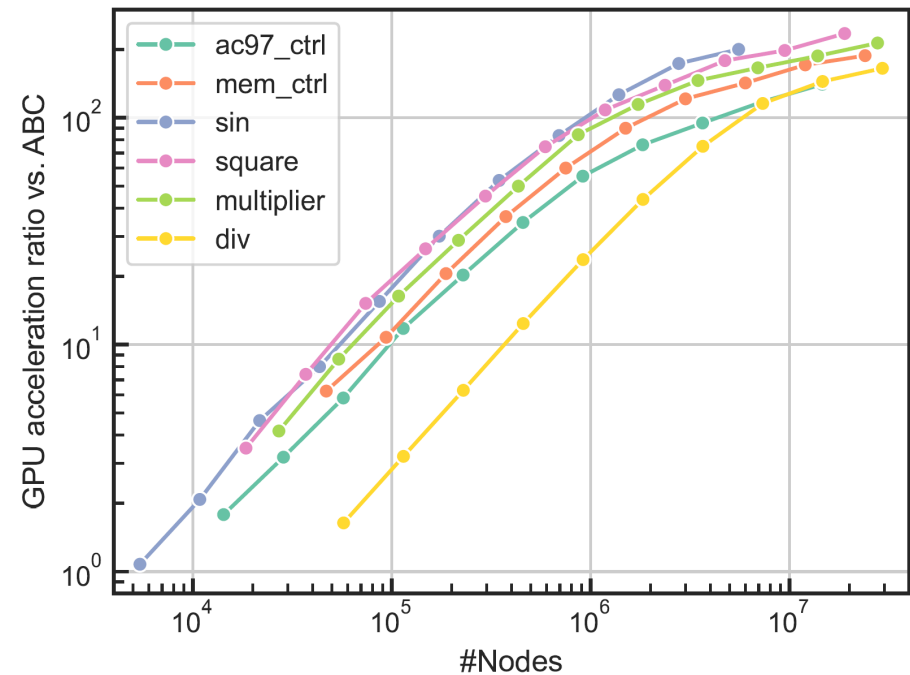
Experimental Results

- Tested on enlarged benchmarks from the EPFL and IWLS'05 Suite
- 128.7x acceleration over the ABC if mapper, with slightly better QoR

Benchmarks	AIG Statistics		ABC if			FineMap (Ours)		
	#AIG Nodes	Levels	#LUTs	Levels	Time	#LUTs	Levels	Time
twentythree	23339737	176	6659071	36	2322.8	6646639	36	71.3
twenty	20732893	162	5929939	33	1888.4	5927717	33	49.4
sixteen	16216836	140	4486446	29	1358.3	4471454	29	37.1
div_10xd	58620928	4372	22559744	864	4100.5	22793216	864	23.2
hyp_8xd	54869760	24801	11392768	4194	3862.1	11461888	4194	23.5
mem_ctrl_10xd	47960064	114	12386304	25	2560.6	12402688	25	11.5
log2_10xd	32829440	444	8200192	77	2462.3	8056832	77	10.6
multiplier_10xd	27711488	274	6054912	53	1869.2	6000640	53	8.4
sqrt_10xd	25208832	5058	5857280	1033	1778.5	5919744	1033	11.1
square_10xd	18927616	250	4080640	50	1358.5	4007936	50	5.8
voter_10xd	14088192	70	2885632	17	760.9	2890752	17	4.2
sin_10xd	5545984	225	1492992	42	401.4	1483776	42	2.2
ac97_ctrl_10xd	14610432	12	2992128	4	441.4	2998272	4	3.2
vga_lcd_5xd	4054752	24	910912	7	191.6	910976	7	1.5
Geomean Ratio			1.000	1.000	128.7	0.998	1.000	1.0

Experimental Results

- Scaling experiments
- Even on small benchmarks, our GPU mapper is faster than ABC



Thank You