FineMap: A Fine-grained GPU-parallel LUT Mapping Engine

Tianji Liu, Lei Chen, Xing Li, Mingxuan Yuan, Evangeline F.Y. Young

香 港 中 文 大 學
The Chinese University of Hong Kong

Outline

- Motivation
- Background
- GPU LUT mapping engine
  - Fine-grained Parallel Mapping
  - Local Area Evaluation
  - Dynamic Memory Management
  - Parallel Timing Analysis & Cut Expansion
- Experimental Results
- Summary

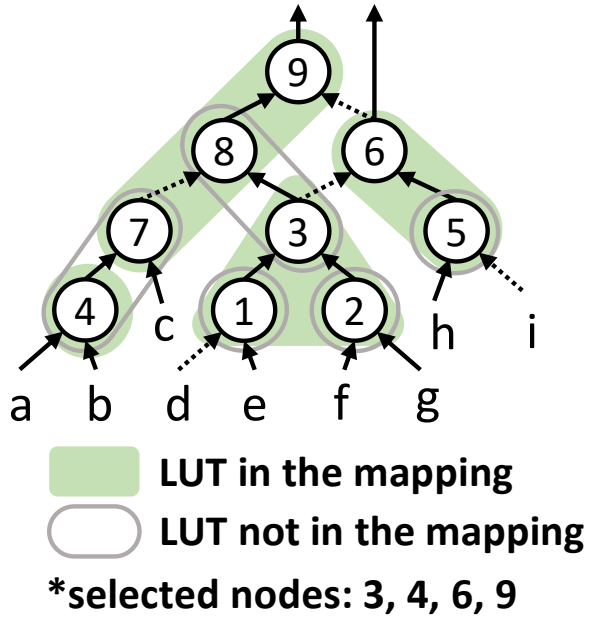
Motivation

- Logic synthesis becomes time demanding
  - Large-scale designs
  - High-effort optimization flows for better QoR
- LUT mapping is widely used in modern synthesis flows
  - Necessary for FPGA design
  - Also adopted in high-effort tech-independent optimization flows, e.g., ABC `&deepsyn`, LUT-based optimization [1]
- Previous works on parallel LUT mapping show limited performance

[1] L. Amaru et al., "LUT-based optimization for ASIC design flow", Proc. DAC'21.

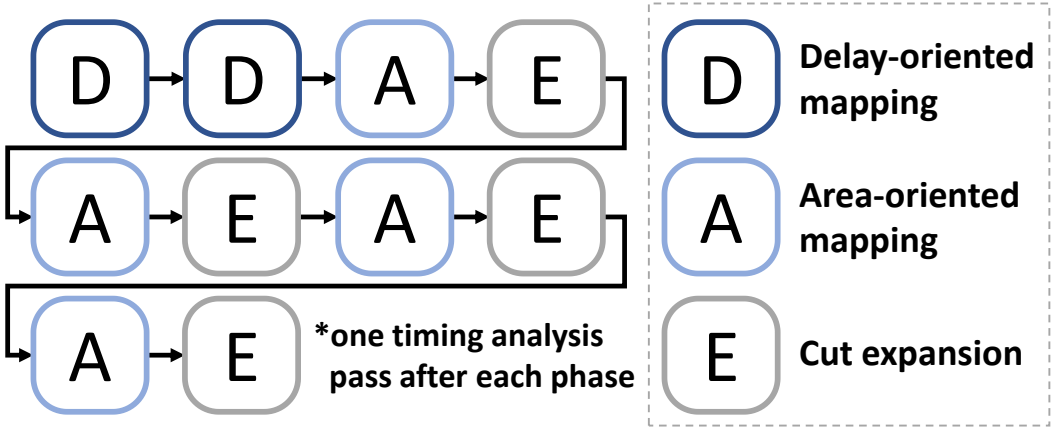
Background - LUT Mapping

- 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 s.t. their cones cover the entire network



Background - LUT Mapping

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



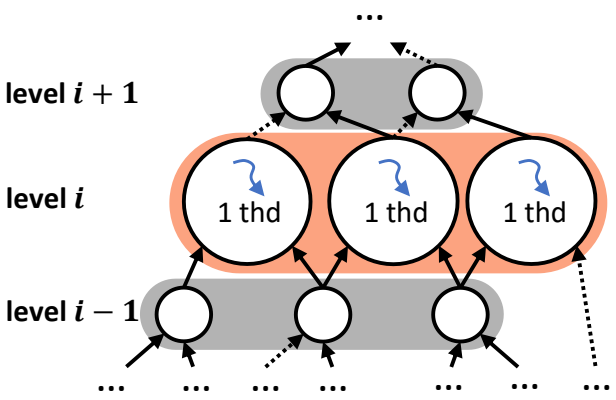
- Each phase incrementally improves the mapped network
- A timing analysis pass after each mapping/cut expansion phase
  - Compute the required time of each node

Fine-grained Parallel Mapping

- What needs to be computed in a mapping phase?
  - For each node n, a priority cut set $P(n)$. The representative cut is the best in $P(n)$
  - $P(n)$ = the trivial cut $\{n\}$ + the best C cuts in the candidate cut set $E(n)$
  - The candidate cuts (maximally $(C+1)^2$ cuts) are computed by cut enumeration:

$$E(n) = \{u \cup v \ : \ u \in P\big(fanin0(n)\big), v \in P\big(fanin1(n)\big), |u \cup v| \leq k\}$$

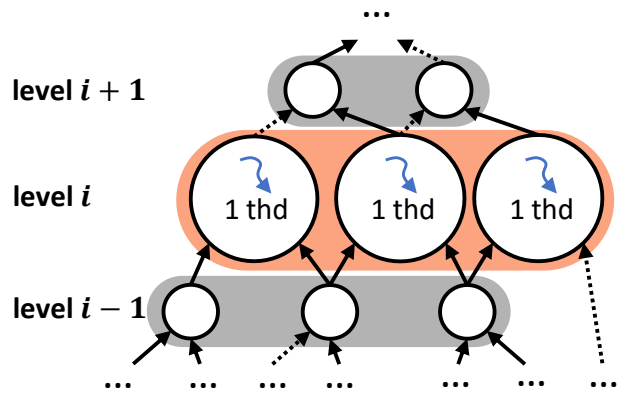
  - The candidate cut ranking is evaluated by cut metrics
    - cut delay, area-flow and exact area
- Previous work on GPU LUT mapping [2] (FPL'10)
  - Coarse-grained, level-wise parallel
  - Only the mapping phase is parallelized



[2] D. Chen and D. Singh, "Parallelizing FPGA technology mapping using Graphics Processing Units (GPUs)", Proc. FPL'10.

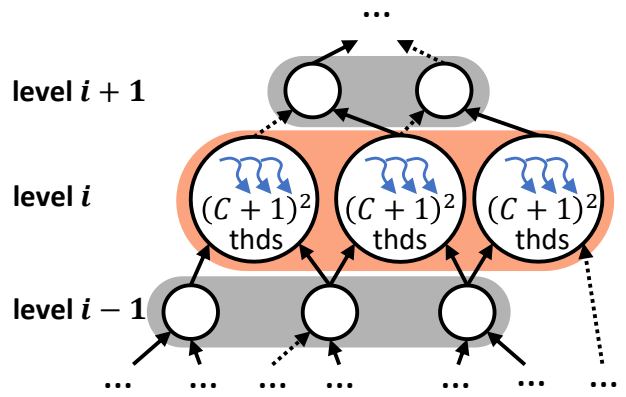
Fine-grained Parallel Mapping

- Fine-grained approach for **one** node
  - $(C+1)^2$ threads for each node at this level
  - Each thread enumerates and evaluates one candidate cut of $E(n)$



Coarse-grained parallel

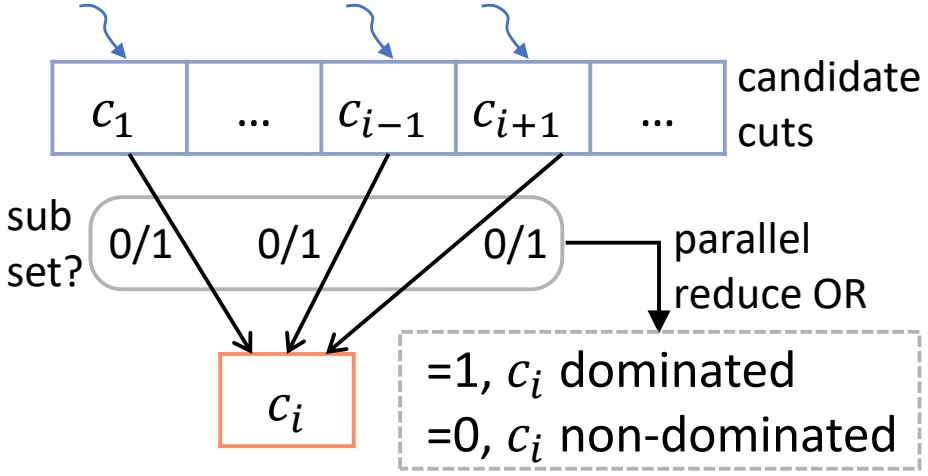
- Challenge 1: select the top-C from $(C+1)^2$ cuts
  - Perform parallel reduction C times
  - After each iteration, mask out the selected best one
  - Experimentally, much faster than parallel sorting



Fine-grained parallel

Fine-grained Parallel Mapping

- Challenge 2: filter out dominated cuts
  - If a candidate cut is a superset of (dominated by) another, it should be filtered out
  - Parallel checking the dominance of all the other cuts against the selected one



Fine-grained Parallel Mapping

Algorithm 1 Fine-grained Parallel Mapping (Per-thread)

Input: AIG node n, max cut size k, thread id tid
Updates: priority cut set $P(n)$, representative cut $RC(n)$

1: $c_0 \leftarrow getCut(P(fanin_0(n)), tid \ / \ (C+1))$
2: $c_1 \leftarrow getCut(P(fanin_1(n)), tid \ \% \ (C+1))$
3: $c \leftarrow c_0 \cup c_1$
4: $valid \leftarrow (c_0 \neq \emptyset$ and $c_1 \neq \emptyset$ and $|c| \leq k)$, $selected \leftarrow false$
5: **if** $valid$ **then**
6: Compute the delay, area-flow or exact area of c
7: **if** $d_C(c) > t_{req}(n)$ **then** $valid \leftarrow false$

enumeration & evaluation

8: **if** $tid = 0$ **then** $P(n) \leftarrow \{RC(n)\}$
9: $sync\_threads()$ ▷ synchronizing the $(C+1)^2$ threads
10: **while** $|P(n)| < C$ **do**
11: $c_b \leftarrow reduceBest(c, valid$ and $!selected)$ ▷ parallel reduction
12: **if** $c_b = \emptyset$ **then break** ▷ no valid and non-selected cut left

top-C selection

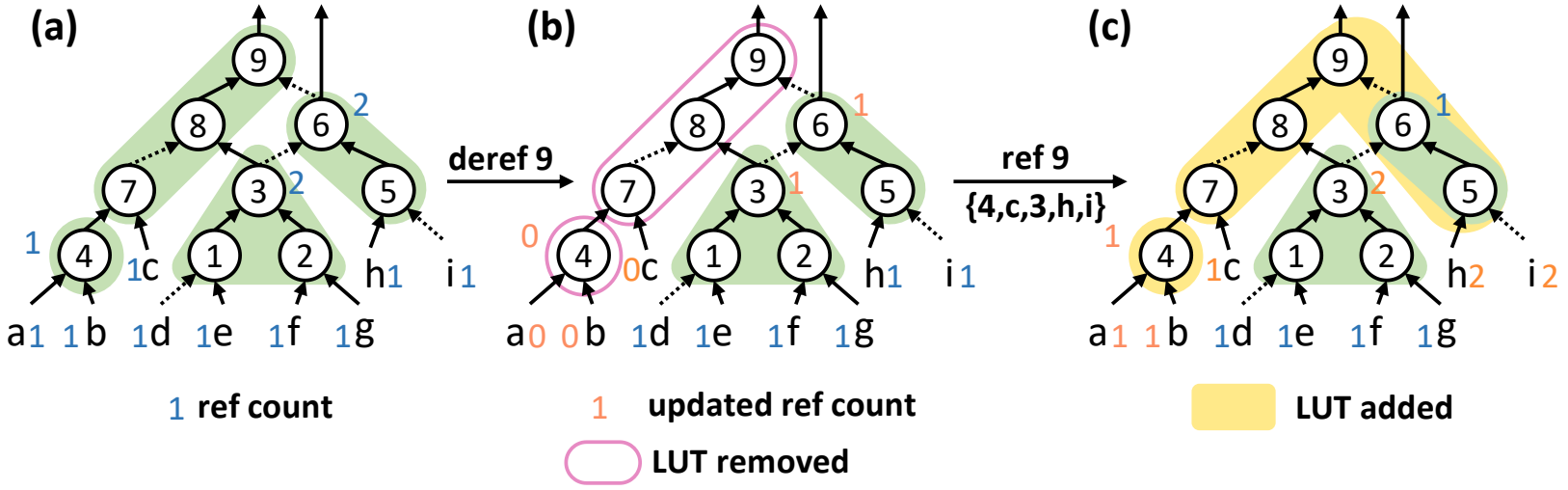
13: **if** $valid$ and $c \neq c_b$ **then** $f \leftarrow isSubset(c, c_b)$
14: **else** $f \leftarrow false$
15: $dom \leftarrow reduceOr(f)$ ▷ parallel reduction
16: **if** $tid = 0$ and $!dom$ **then** $P(n) \leftarrow P(n) \cup c_b$
17: **if** $c = c_b$ **then** $selected \leftarrow true$ ▷ mask out c_b

dominance checking

18: $sync\_threads()$
19: **if** $tid = 0$ **then**
20: $P(n) \leftarrow P(n) \cup \{n\}$ ▷ add the trivial cut
21: $RC(n) \leftarrow$ the first (best) cut in $P(n)$

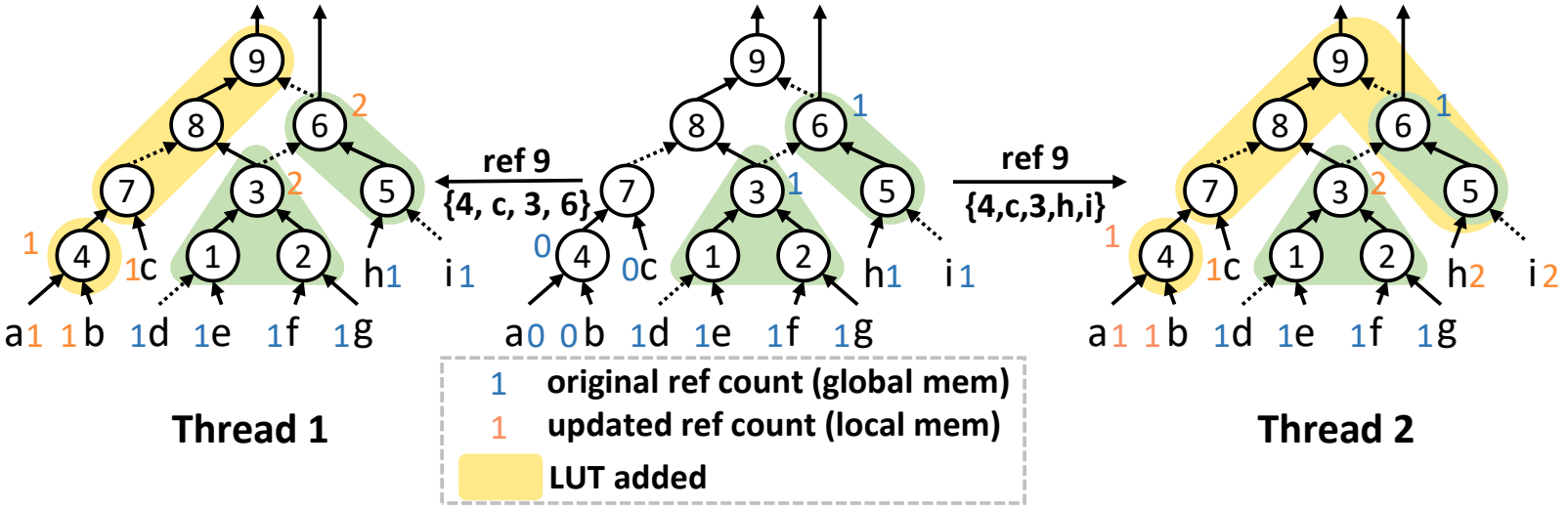
Local Area Evaluation in Parallel Settings

- Ref count of a node: #fanouts in the current mapped LUT network
- (de)reference a cut (LUT)
  - Ref: recursively add (reference) all the LUTs needed to drive the LUT
  - Deref: recursively remove (dereference) all the LUTs dedicated to driving the LUT
- Exact area of a cut: #LUTs traversed during (de)ref

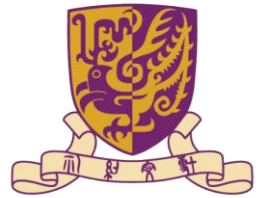


Local Area Evaluation in Parallel Settings

- Challenge 1: data race during concurrent exact area computation
  - Multiple threads attempts to manipulate a single ref count
- Data-race-free solution
  - Each thread records a copy of its updated ref counts in thread-local memory
  - The original ref counts are saved in global memory, read-only

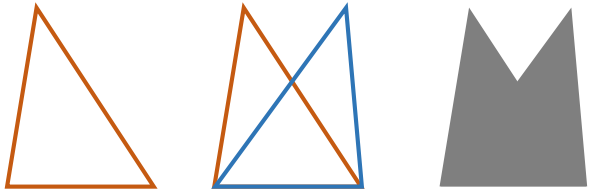


Local Area Evaluation in Parallel Settings

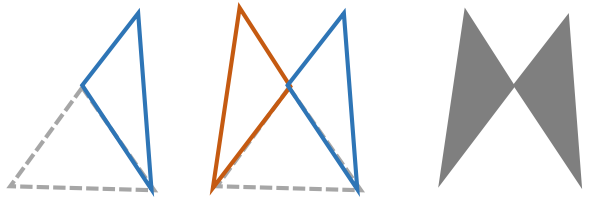


- Challenge 2: some candidate cuts having very large exact areas
  - Lead to slow execution of some threads, and insufficient thread-local memory
- Original exact area evaluation strategy
  - Deref original representative cut c first, then ref candidate cut (under eval) c'
- Our new strategy
  - Ref c' first, then deref c
  - Local area metric = #LUTs traversed during ref c' - #LUTs traversed during deref c

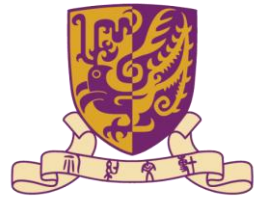
original strategy



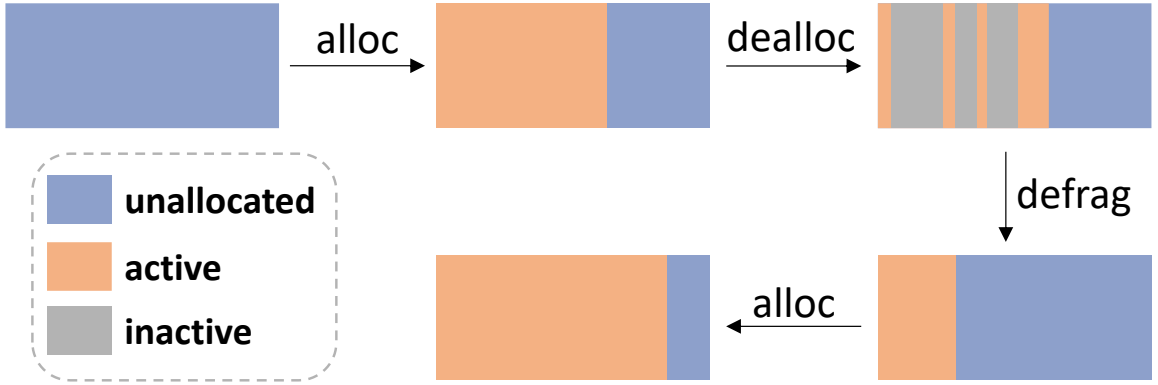
new strategy



Dynamic Memory Management



- Priority cut sets required huge amount of memory
  - Allocating all the cut sets at once leads to OOM on GPUs
- GPU memory pool for managing cut sets
  - Functionalities: batched alloc, batched dealloc (free), defragmentation (gc)
  - All implemented by GPU-parallel algorithms



Dynamic Memory Management

- Cut set (de)allocation scheme
  - The level to allocate the cuts for node $n = n.level$
  - The level to deallocate $n = \max_{n' \in fanout(n)} n'.level$
  - Perform batched allocation and deallocation once respectively per-level

- Reduction on memory footprint
  - On our largest case: 39.3GB (alloc once) → 6.7GB (mem pool), 5.9x reduction
- Runtime overhead
  - On average: <3% of total runtime

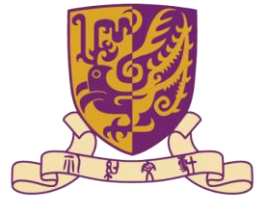
Parallel Timing Analysis

- Updates the required time of each node
  - During mapping, a candidate cut will be rejected if its delay exceeds the required time
- Top-down propagation

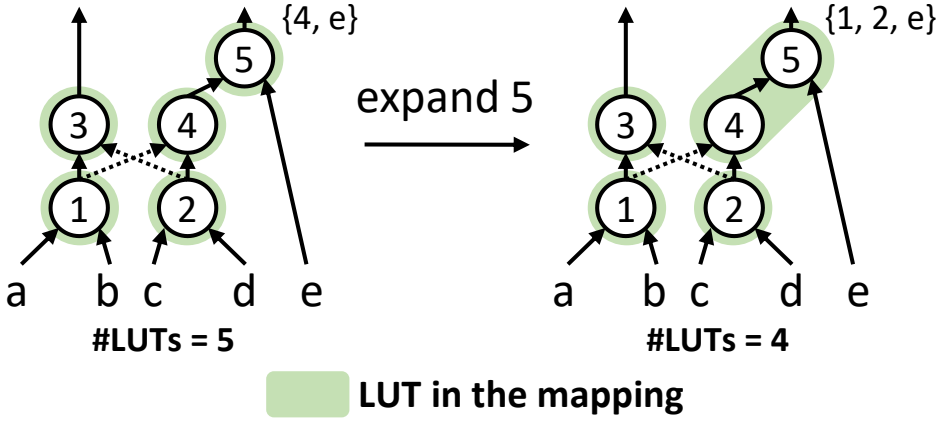
$$t_{req}(n) = \min_{n' \in lut\_fanout(n)} \left(t_{req}(n') - 1 \right)$$

- Parallel case
  - Perform the top-down propagation through reversed level-wise parallel sweeping

Parallel Cut Expansion



- Expands the representative cuts towards PIs and enable LUT sharing
- Iteratively replace a node n' in a cut of n with its two fanins n'_0, n'_1, if
  - n' exclusively drives n in the mapping; and
  - n'_0, n'_1 also drives other LUTs in the mapping
- Parallel case
  - All the expansions are performed concurrently
  - An additional level-wise parallel sweeping of all nodes to commit/reject the updates, along with recomputing delay and checking the required time



Experimental Results

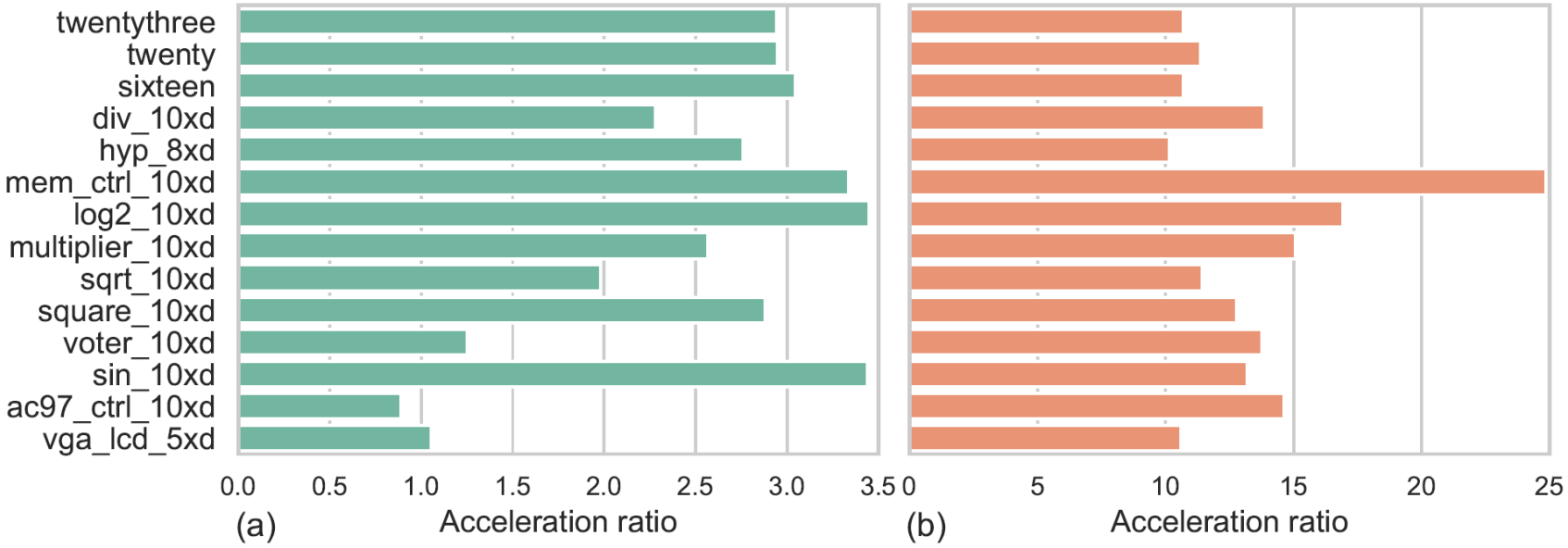
- Tested on enlarged benchmarks from the EPFL and IWLS'05 Suite
- 128.7x acceleration over the ABC `if` mapper, with slightly better QoR

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

- Comparison against the previous work FPL'10 on GPU LUT mapping

LUT mapping flow	Mapping	Timing Analysis	Cut Expansion
FPL'10 full-flow (est.)	coarse-grained par.	ABC (sequential)	ABC (sequential)
Ours	fine-grained par.	parallel	parallel

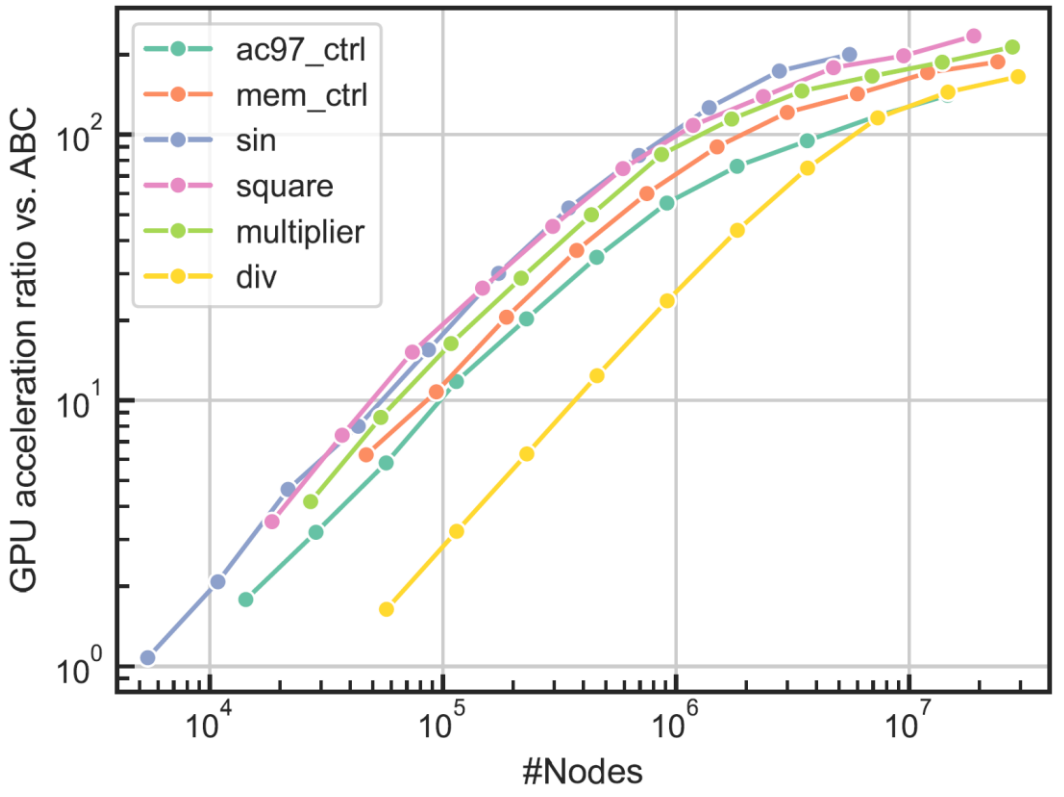


mapping func. only: 2.3x

full-flow: 13.2x

Experimental Results

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



Summary

- Propose an ultra-fast GPU LUT mapping engine, consisting of
  - fine-grained parallel mapping, with a new local area evaluation strategy and dynamic memory management methods
  - parallel cut expansion and timing analysis
- Achieve 128.7x acceleration over ABC with slightly better QoR