Euler Meets GPU

Practical Graph Algorithms with Theoretical Guarantees

Adam Polak Adrian Siwiec Michał Stobierski

Jagiellonian University in Kraków

IPDPS 2021

GPU $\approx$ PRAM

GPU $\approx$ PRAM

but

GPU $\neq$ PRAM

GPU $\approx$ PRAM

but

GPU $\neq$ PRAM

Folk wisdom: Often better to implement simple, worst-case* quadratic algorithm, rather than complex theoretically optimal linear algorithm

*Especially for graph algorithms, nice input structure can save us from worst-case

GPU $\approx$ PRAM

but

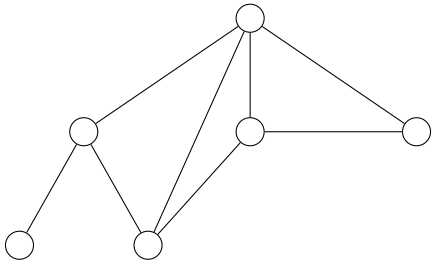
GPU $\neq$ PRAM

Folk wisdom: Often better to implement simple, worst-case* quadratic algorithm, rather than complex theoretically optimal linear algorithm

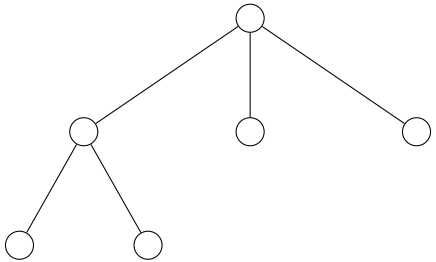
*Especially for graph algorithms, nice input structure can save us from worst-case

Let's aim for **best of both worlds**:
Good performance in practice and theoretical worst-case guarantees

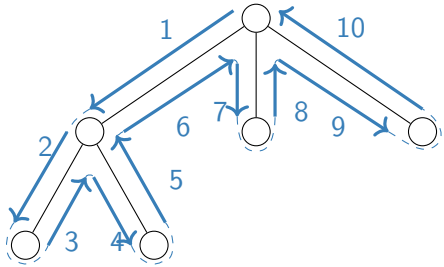
Euler Tour



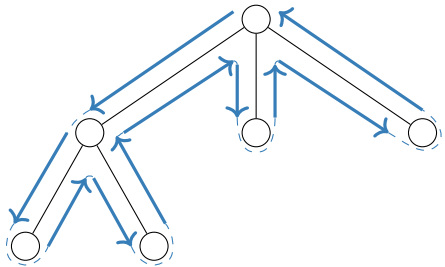
Euler Tour



Euler Tour

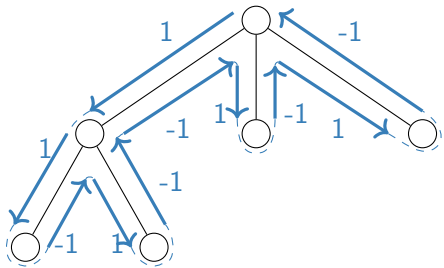


Euler Tour



Simple application: computing node depths

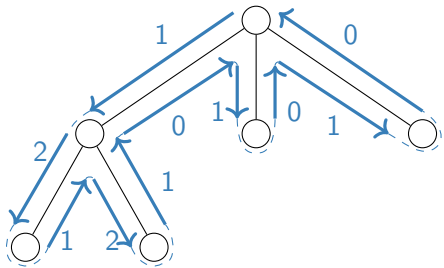
Euler Tour



Simple application: computing node depths

1. Put $+1$ on each edge if it goes down, -1 if it goes up

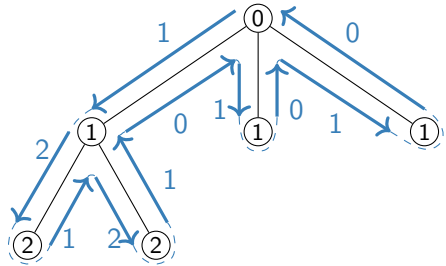
Euler Tour



Simple application: computing node depths

1. Put +1 on each edge if it goes down, -1 if it goes up
2. Compute prefix sums

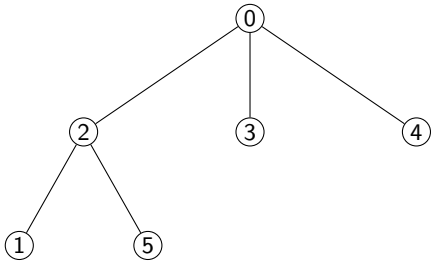
Euler Tour



Simple application: computing node depths

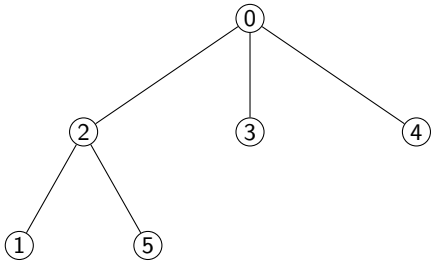
1. Put $+1$ on each edge if it goes down, -1 if it goes up
2. Compute prefix sums
3. Edge (u, v) contains the depth of v

Constructing Euler Tour



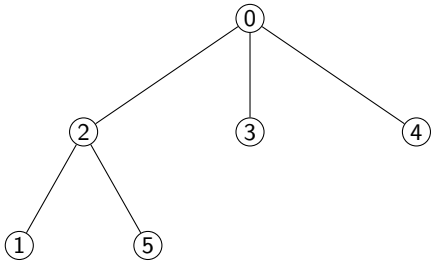
edges:	(0,2)	(0,3)	(0,4)	(2,1)	(2,5)

Constructing Euler Tour



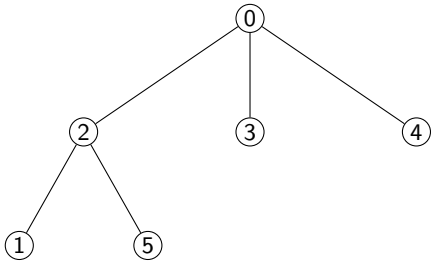
edges: (0,2) (0,3) (0,4) (2,1) (2,5)

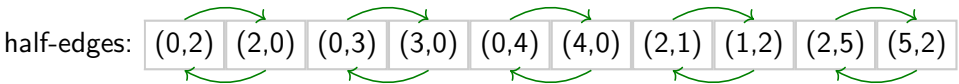
Constructing Euler Tour



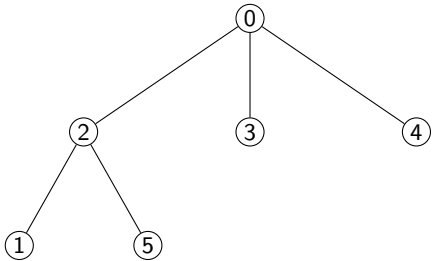
half-edges:	(0,2)	(2,0)	(0,3)	(3,0)	(0,4)	(4,0)	(2,1)	(1,2)	(2,5)	(5,2)

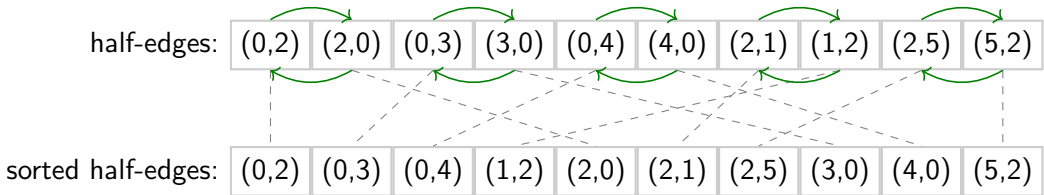
Constructing Euler Tour



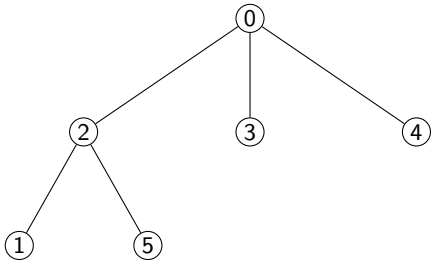


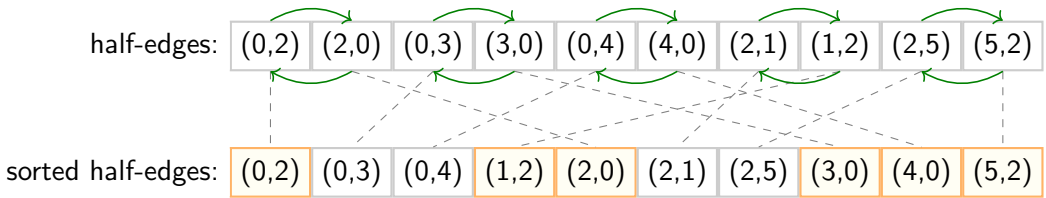
Constructing Euler Tour



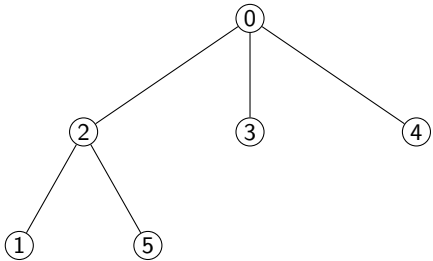


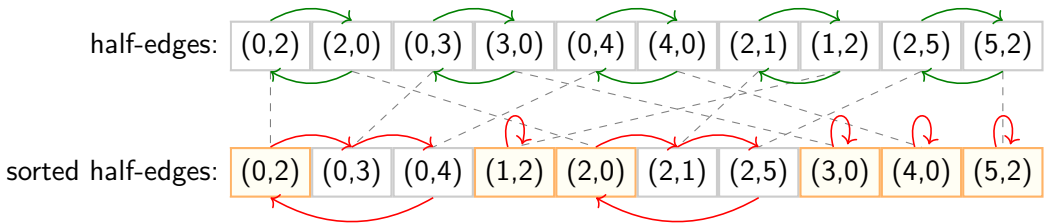
Constructing Euler Tour



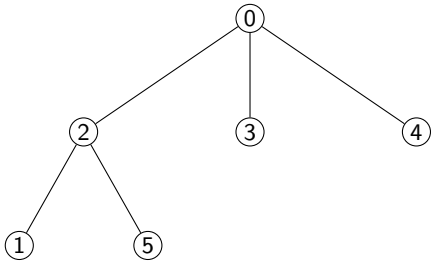


Constructing Euler Tour

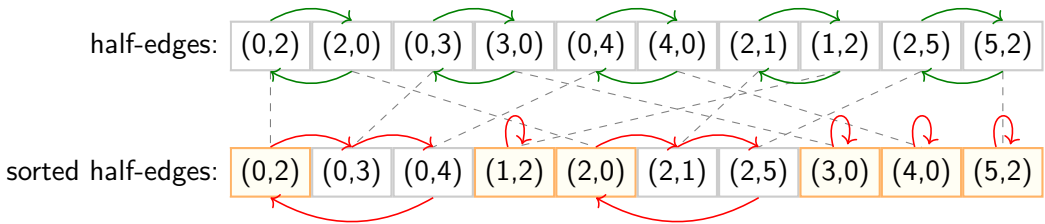




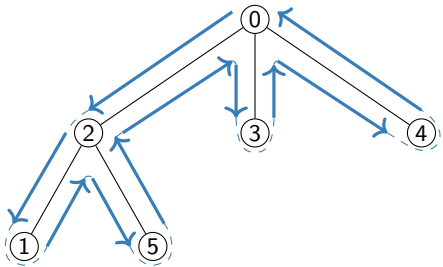
Constructing Euler Tour



succ(e) = next(twin(e))



Constructing Euler Tour



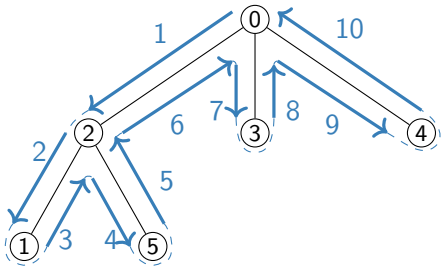
twin first next

succ(e) = next(twin(e))

half-edges: (0,2) (2,0) (0,3) (3,0) (0,4) (4,0) (2,1) (1,2) (2,5) (5,2)

sorted half-edges: (0,2) (0,3) (0,4) (1,2) (2,0) (2,1) (2,5) (3,0) (4,0) (5,2)

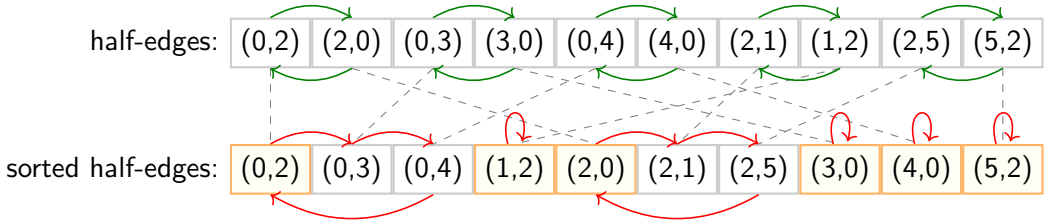
Constructing Euler Tour



twin first next

succ(e) = next(twin(e))

run list-rank on succ



Two example applications: LCA and bridge-finding

For both problems:

- theoretically optimal parallel algorithms in PRAM model
  - using Euler Tour
  - logarithmic time & (near-)linear work
- in practice simpler algorithms state-of-the-art on GPU
  - fast on real-world instances
  - worst-case linear time & quadratic work

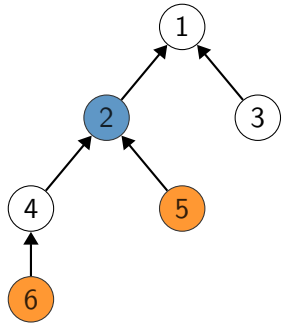
First example: Lowest Common Ancestors (LCA) in trees

Input:

- a rooted tree T,
- queries: $(x_1, y_1), (x_2, y_2), \ldots, (x_q, y_q) \in T \times T$

Output:

- $\mathrm{lca}(x_1, y_1), \mathrm{lca}(x_2, y_2), \ldots, \mathrm{lca}(x_q, y_q)$

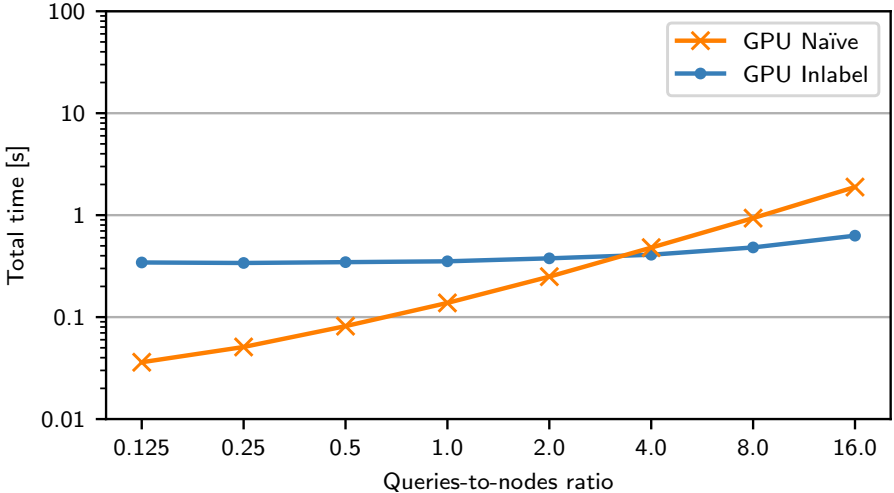


LCA algorithms

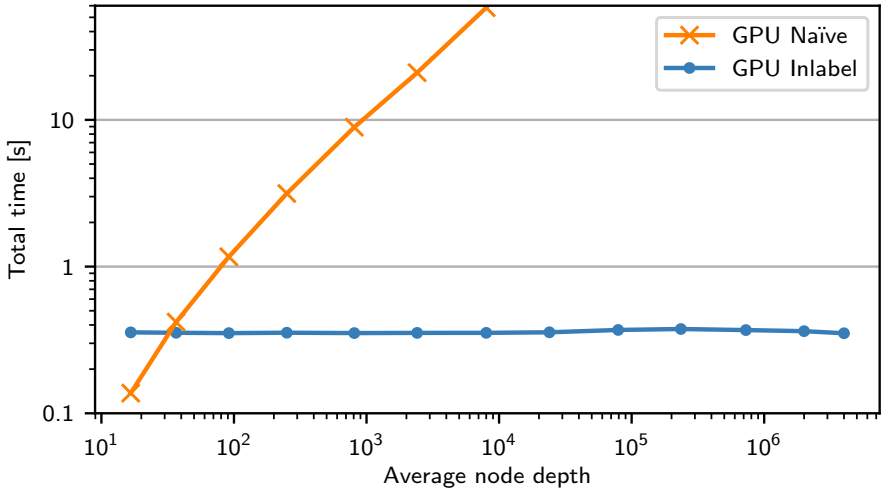
	Preprocessing			
	Time	**Work**	**Query time**	
Naïve	$O(\log n)$	$O(n \log n)$	$O(\text{dist}(x, y))$	folklore*
Inlabel	$O(\log n)$	$O(n)$	$O(1)$	[Schieber, Vishkin, 1988]

uses Euler Tour

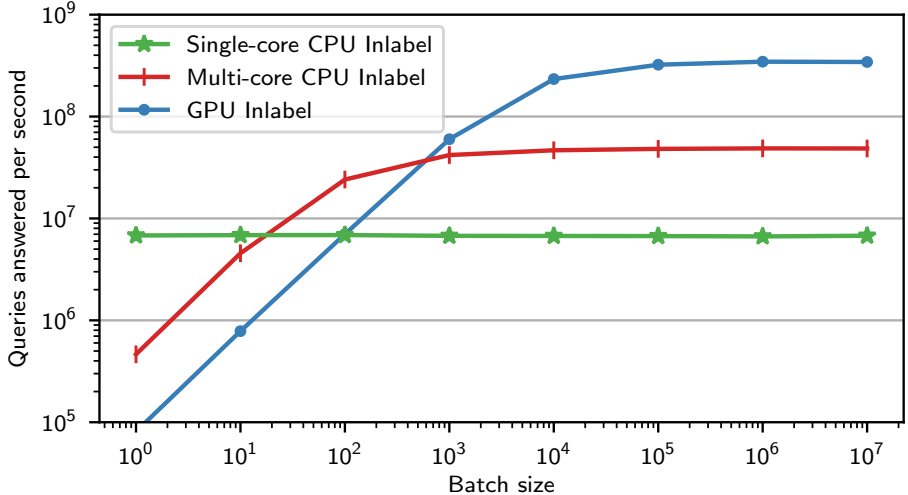
*used in Martins et al., *Phylogenetic distance computation using CUDA*, 2012



LCA total time by queries-to-nodes ratio
shallow trees, average node depth ≈ 16, $\#\text{nodes} = 8 \cdot 10^6$



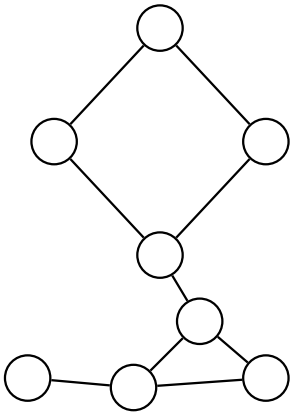
LCA total time by depth, #nodes = #queries = $8 \cdot 10^6$



LCA queries throughput* by batch size, $\#\text{nodes} = 8 \cdot 10^6$

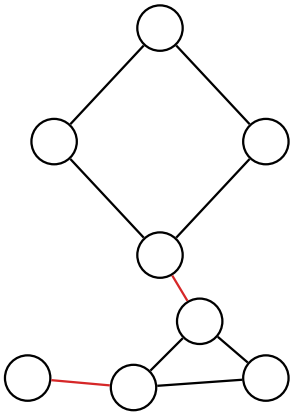
*higher is better

Second example: bridge-finding



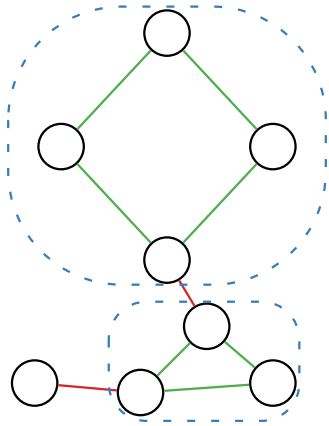
Given a graph $G = (V, E)$, determine for each edge $e \in E$ whether e is a bridge

Second example: bridge-finding



Given a graph $G = (V, E)$, determine for each edge $e \in E$ whether e is a bridge

Second example: bridge-finding



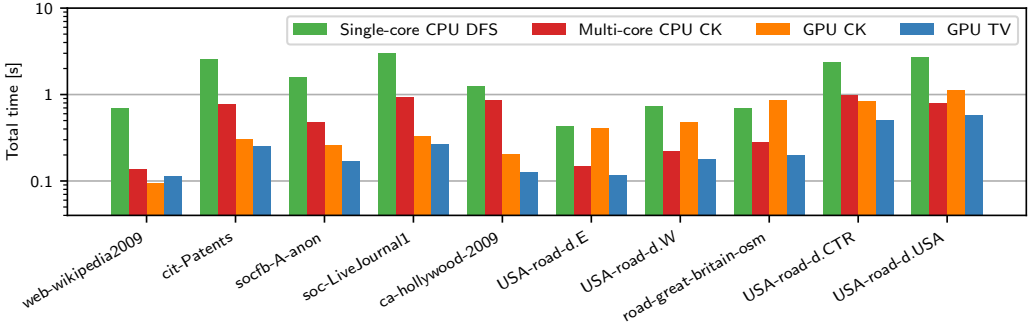
Given a graph $G = (V, E)$, determine for each edge $e \in E$ whether e is a bridge

Bridge-finding algorithms

	Time	**Work**	
Single-core CPU DFS	$O(m)$	$O(m)$	[Hopcroft, Tarjan, 1971]
Multi-core CPU CK	$O(\text{diam})$	$O(m \cdot \text{diam})$	[Chaitanya, Kothapalli, 2016]
GPU CK	$O(\text{diam})$	$O(m \cdot \text{diam})$	[Chaitanya, Kothapalli, 2016]
GPU TV	$O(\log m)$	$O(m)$	[Tarjan, Vishkin, 1985]

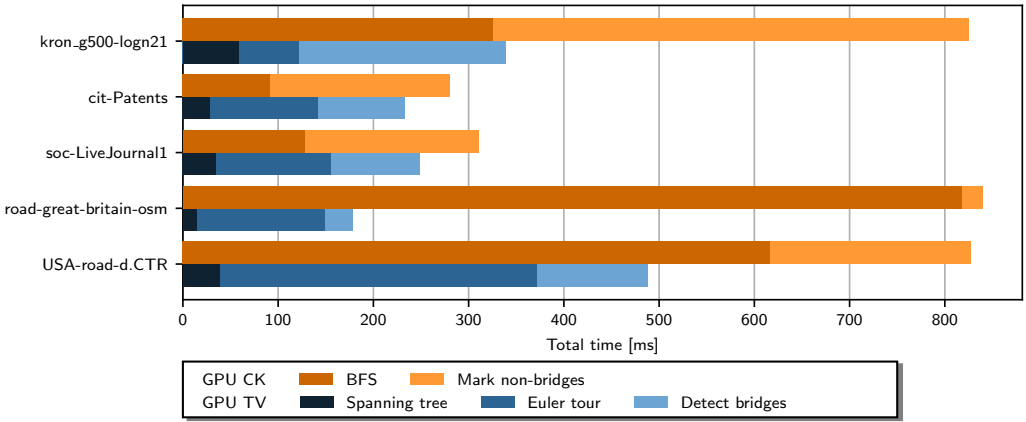
uses Euler Tour

Comparison of bridge-finding algorithms on real-world graphs



GPU TV speedup vs.	Single-core CPU DFS	4-12x
	Multi-core CPU CK	up to 8x
	GPU CK	up to 4.7x

Running time breakdown of GPU bridge-finding algorithms



Open question: How to **fast** get a **rooted** spanning tree?

Take-home message:

Don't be afraid of seemingly complex Euler Tour-based algorithms!

Thank you :)