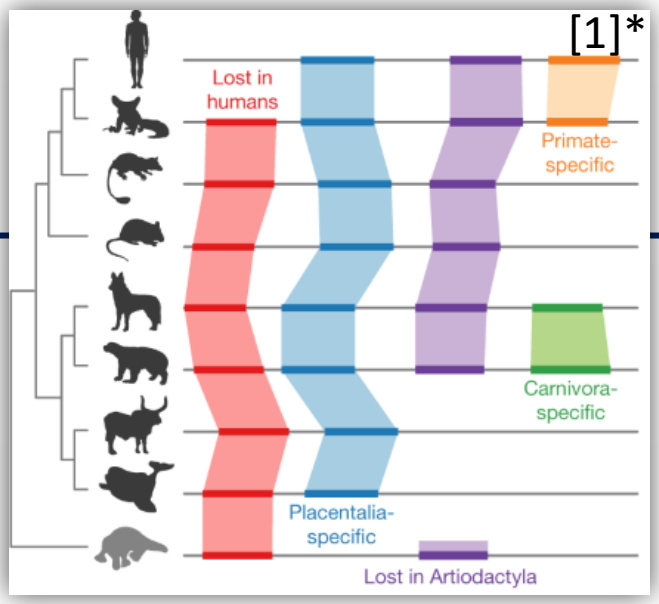
FastZ: Accelerating Gapped Whole Genome Alignment on GPUs

Sree Charan Gundabolu, T. N. Vijaykumar, and Mithuna Thottethodi

PURDUE UNIVERSITY

Gene Sequence Alignment



Basic computational kernel of genomics

- Genome assembly
- Genome alignment for comparative genomics

Our focus: Whole Genome Alignment (WGA)

- Genome alignment between two (or more) organisms
- Answers key evolutionary and genetic questions
  - based on genome (dis)similarity

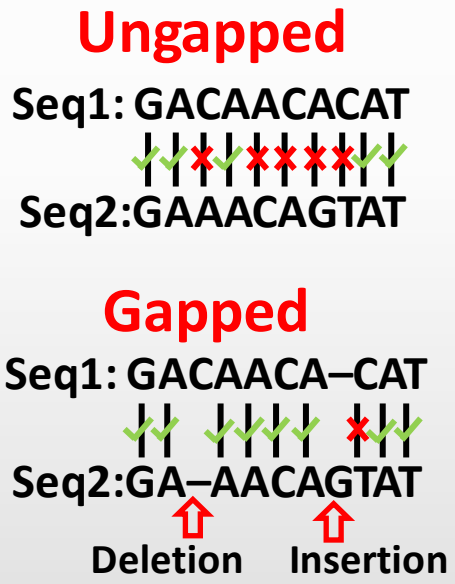
LASTZ is the state-of-the-art WGA application

- National Institutes of Health (NIH) support LASTZ servers
- Sequential

WGA is a key bioinformatics kernel

Need for Scalable High-Performance WGA

- Availability of genomic data
  - Faster and cheaper sequencing
  - Increasing number of fully sequenced genomes
  - Genome assemblies of complex organisms
- Existing techniques in LASTZ
  - Ungapped filtering (exact matches)
    - faster but fewer high-scoring alignments
  - Gapped alignment (matches with insertions/deletions)
    - slower but more high-scoring alignments



Our focus: Fast, sensitive gapped WGA on commodity GPU

LASTZ: WGA State-of-the-Art

Heuristic based seed-and-extend algorithm

- Identify seeds: short, exact matches
- Extend each seed with gaps
  - Smith-Waterman dynamic programming (SWDP)

Previous Work

- Gapped WGA using custom ASICs [Darwin-WGA, HPCA'19]
- Low-sensitivity WGA with ungapped filtering using GPU [SegAlign, SC'20]
- Parallelize single DP on GPU [ICPADS'09]

We accelerate gapped LASTZ on GPUs

WGA Workload Characteristics (1 of 2)

Gapped LASTZ execution time

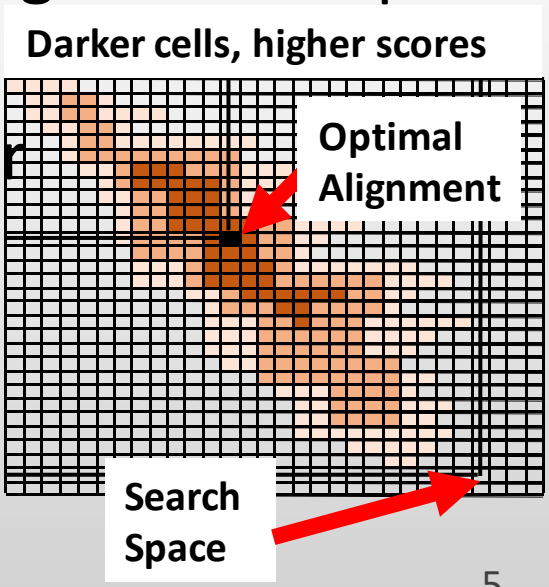
- SWDP accounts for most runtime (>99%)

Numerous seeds/DP problems (millions)

- Independent seeds → abundant inter-DP parallelism
- Intra-DP parallelism along each anti-diagonal

SWDP explores well beyond optimal alignment despite pruning

- Track scores and traceback often for much larger search space
  - E.g., OAL < 128 base pairs (bp), search up to 5700 bp



WGA Workload Characteristics (2 of 2)

Alignment lengths vary highly (e.g., 16 - 10K bp)

- GPU dynamic memory allocation slow
- Worst case allocation → reduce parallelism
- Load imbalance

Low compute per byte

- 4 comparisons + 5 additions
- Parallelization → memory bandwidth-bound

Short lifetime of DP scores unlike traceback data

- 5 reads (by two later anti-diagonals), then dead

We employ Inspector-Executor to exploit these characteristics

FastZ: Contributions

Lightweight inspector finds optimal alignment length (OAL)

- elides traceback state to avoid large and dynamic mem. alloc.
- except extremely short alignments

Inspector ***eagerly traces back*** extremely short alignments

- Entirely eliminates executor in common case

Executor statically allocates memory using inspector's OALs

Executor trimming computes DP and traceback only for OALs

- not larger search space

Cyclic-use-and-discard holds short-lived DP scores in registers

- eliminates most memory accesses in inspector and executor

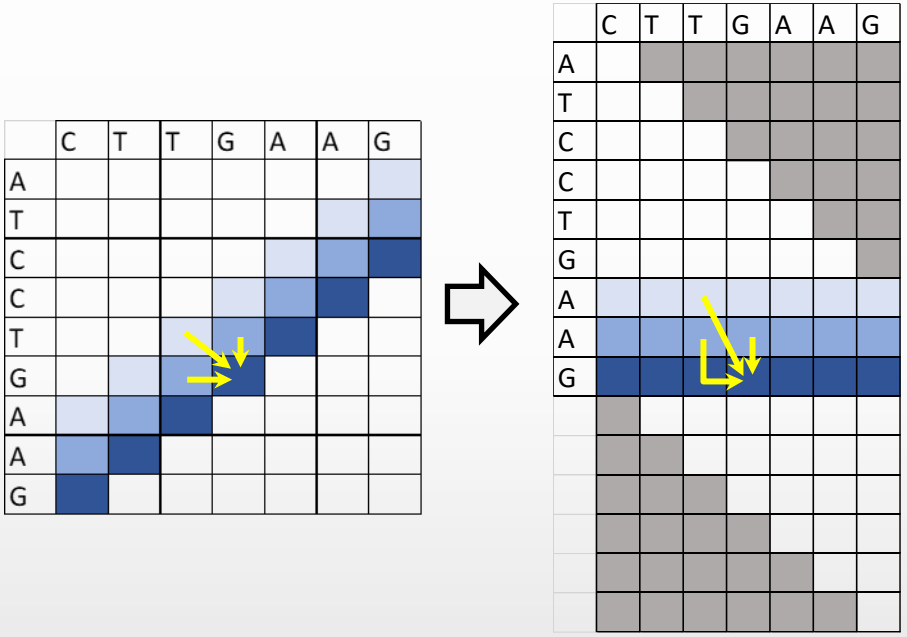
Load balancing in the executor by binning seeds based on OALs

FastZ achieves 111x speedup over the sequential LastZ on an Nvidia Ampere GPU

Outline

- Introduction
- Background
- FastZ
  - Lightweight inspector
    - Eager traceback
  - Executor trimming
  - Cyclic-use-and-discard
  - Load balancing
- Methodology
- Results
- Conclusion

Background on Intra-DP Parallelism



- Parallelism along each anti-diagonal --> hard for GPU
- Well-known, GPU-friendly layout transformation
  - Converts anti-diagonals to coalesceable rows

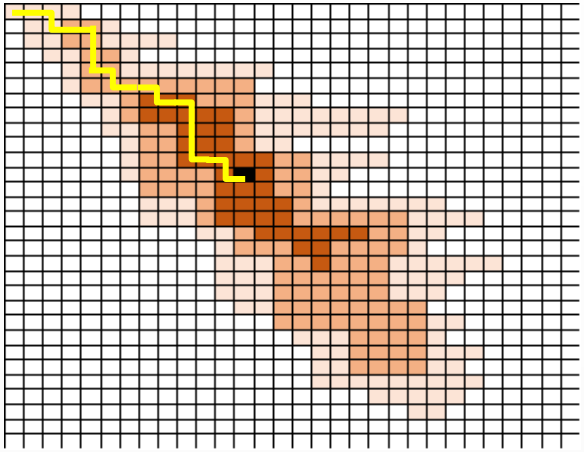
FastZ's Parallelism

- Intra-DP parallelism using previous transformation
  - Each seed extension is assigned to a warp
  - Warp lanes compute diagonal elements in parallel
- Inter-DP parallelism
  - Multiple concurrent seed extensions
    - Multiple warps per threadblock
    - Multiple threadblocks

FastZ's Inspector & Executor benefit from both

Fastz's Inspector-Executor Architecture

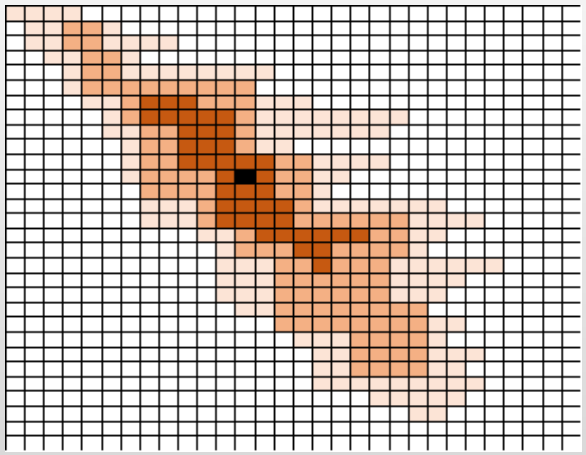
LASTZ's single-pass SWDP on full workload



DP Scores for search space (with y-drop pruning) + Traceback state for search space + Traceback for OAL

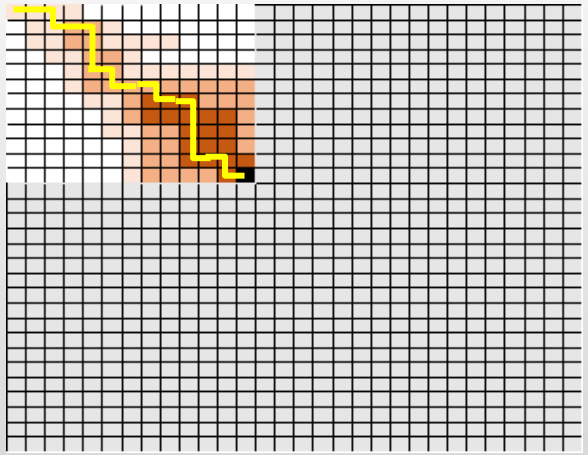
Scores for search-space + 16x16 eager traceback (~80 % of seeds)

FastZ's Inspector



+

FastZ's Executor



~ 20% of seeds

Scores for OAL + Traceback state for OAL + Traceback for OAL

Fastz's Lightweight Inspector

- Identifies optimal alignment, but searches larger space
- Elides traceback except for extremely small alignments
  - No large and dynamic memory allocation for traceback
    - For DP scoring matrix too (later)
  - Reduce inspector's bandwidth and footprint
    - Improves parallelism – more concurrent problems

Mostly eliminate traceback, slow dynamic memory allocation

Fastz's Eager Traceback in Inspector

- Inspector eagerly traces back extremely small alignments (16x16)
  - Most seed extensions are short (> 80%)
    - Part of longer left/right alignment → cannot discard
- Inspector tracks 16x16 traceback data for each seed
- Inspector performs traceback only if OAL within 16x16
  - Low cost – small enough to fit in L1/ Shared Memory

Eliminate executor for common, extremely small problems

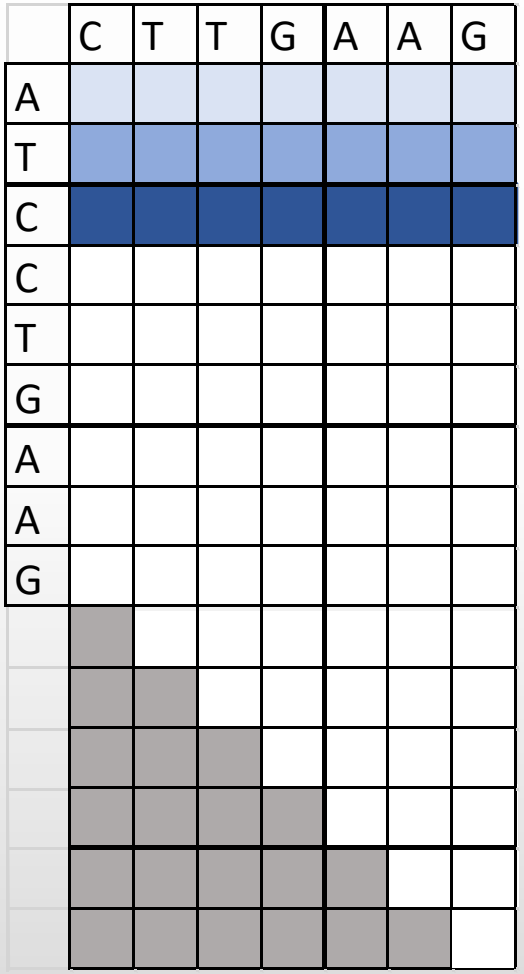
FastZ's Executor

- Executor trimming
  - DP and traceback matrices for much smaller OAL from Inspector and not larger search space – work reduction
- Small and static memory allocation of the traceback matrix
- Trimming to OAL
  - Reduction in memory footprint
  - Maximize parallelism – seed packing
- Traceback optimizations
  - Single byte traceback state
  - Collect traceback state in GPU Shared Memory and write to memory using a single cache block write

DP matrix and traceback only for OAL

FastZ's Cyclic-Use-and-Discard

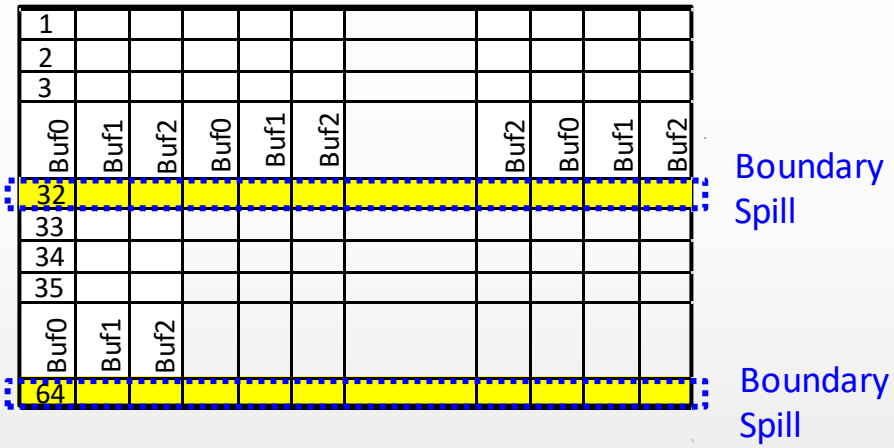
- Live state of DP matrix limited to 3 anti-diagonals
  - Multiple concurrent seeds → severe cache pollution
  - Unnecessary spills to/from memory
- Cyclic-use-and-discard buffers hold 3 anti-diagonals
  - Overwrite oldest anti-diagonal as new anti-diagonal is computed
  - Small → place in registers



Fast register buffers cut 97% inspector and executor traffic

FastZ's Cyclic-Use-and-Discard

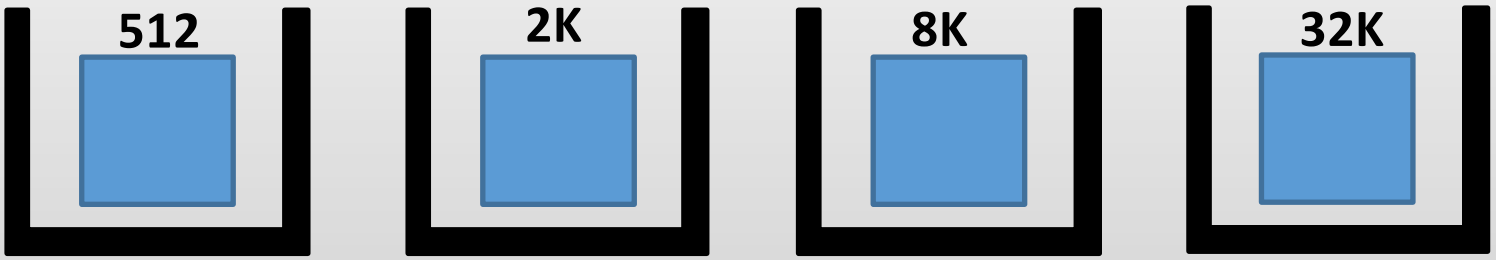
- Live state of DP matrix limited to 3 anti-diagonals
  - Multiple concurrent seeds → severe cache pollution
  - Unnecessary spills to/from memory
- Cyclic-use-and-discard buffers hold 3 anti-diagonals
  - Overwrite oldest anti-diagonal as new anti-diagonal is computed
  - Small → place in registers



Fast register buffers cut 97% inspector and executor traffic

FastZ's Load Balancing

- GPU's bulk synchronous approach
  - Kernel terminates only when all blocks across SMs terminate
  - Long and short alignments within a kernel → load imbalance
- Inspector: no traceback → low imbalance → CUDA streams suffice
- Executor – OAL from the inspector
  - Bin seeds by alignment length
  - Four bins



Outline

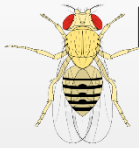
- Introduction
- Background
- FastZ
- Methodology
- Results
- Conclusion

Genomes and Pairwise alignment

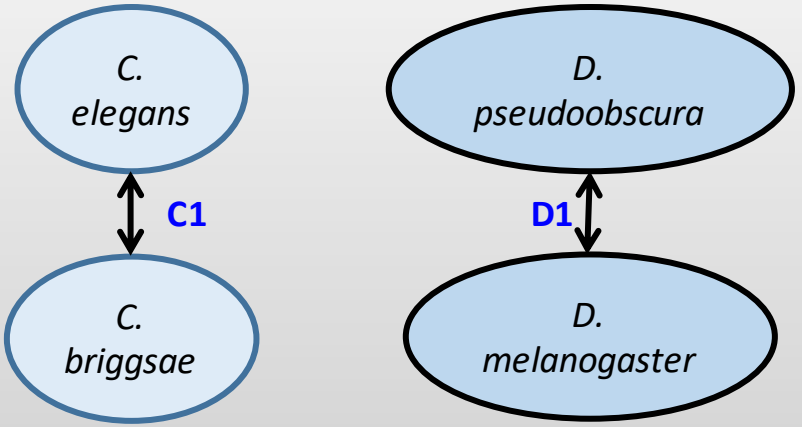
Common Name	Species	Basepairs
Nematodes	*C. elegans* (*chr1*)	15,072,434
	C. briggsae (*chr1*)	15,455,979
	C. elegans (*chr2*)	15,279,421
	C. briggsae (*chr2*)	16,627,154
	C. elegans (*chr3*)	13,783,801
	C. briggsae (*chr3*)	14,578,851
	C. elegans (*chr4*)	17,493,829
	C. briggsae (*chr4*)	17,485,439
	C. elegans (*chr5*)	20,924,180
	C. briggsae (*chr5*)	19,495,157
Fruit flies	*D. melanogaster* (*chr2R*)	25,286,936
	D. pseudoobscura (*chr2*)	30,794,189
Mosquitoes	*A. albimanus* (*chrX*)	12,318,379
	A. atroparvus (*chrX*)	17,503,697
	A. gambiae (*chrX*)	24,393,108

- Organisms
  - Nematodes
  - Fruit Flies
  - Mosquitoes
- Pairwise WGA – similar organisms


[2]*


[3]*


[4]*

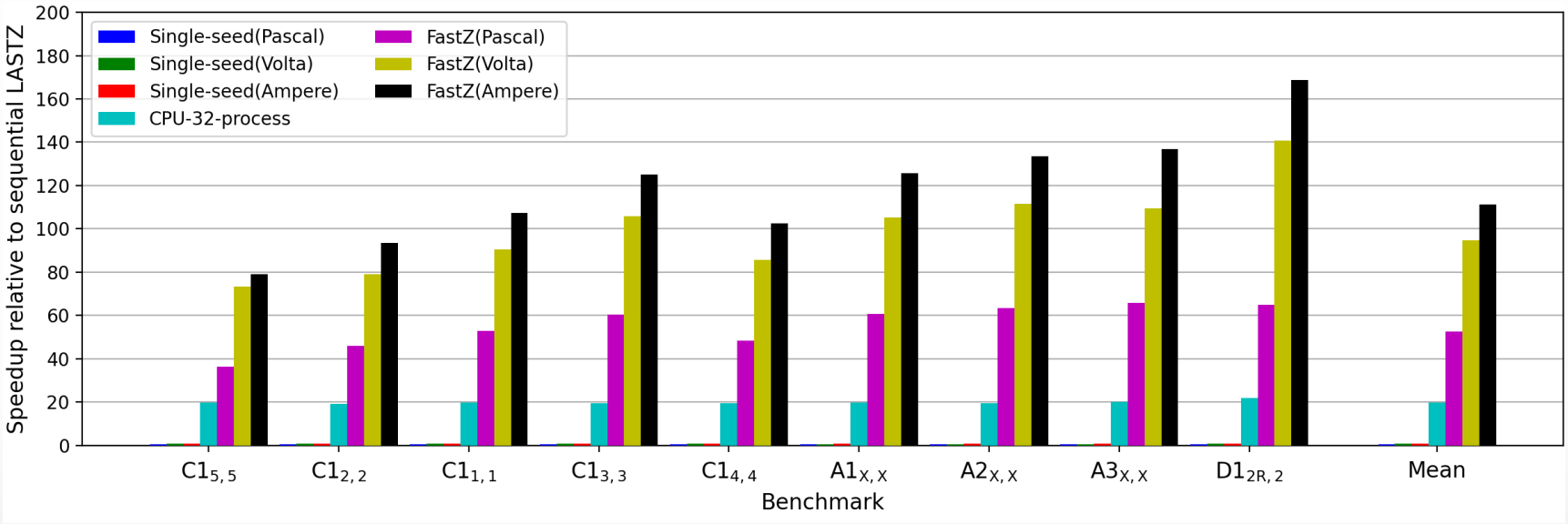


Methodology

- Comparisons
  - Sequential LASTZ
  - Multiprocess LASTZ
  - Single-seed Smith Waterman GPU implementation [ICPADS'09]
- Performance measured as execution time of 1 million seeds
  - Execution time : Inspector + Executor (including all traceback)
- Evaluation Platforms:

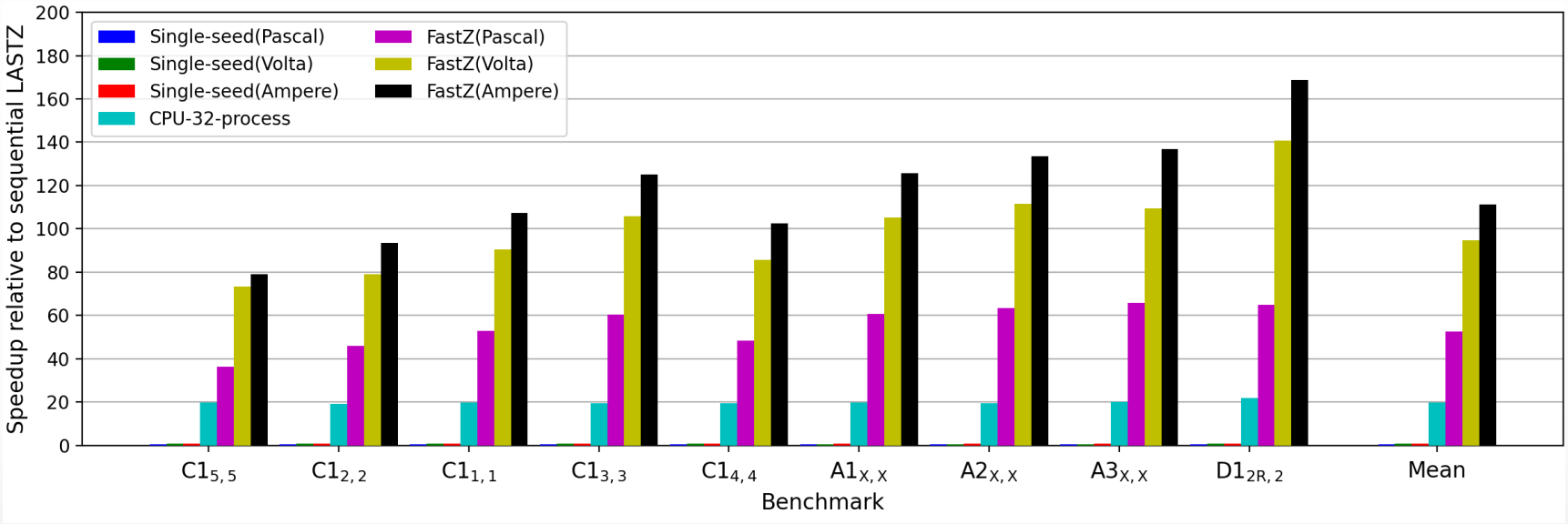
CPU	High Performance Cores	Memory (GB)
AMD Ryzen 3950x	16	32
GPU (NVIDIA)	**Streaming Multiprocessors**	**Memory (GB)**
Titan X (Pascal)	28	12
V100 (Volta)	80	32
RTX3080 (Ampere)	68	10

FastZ Performance (1 of 2)



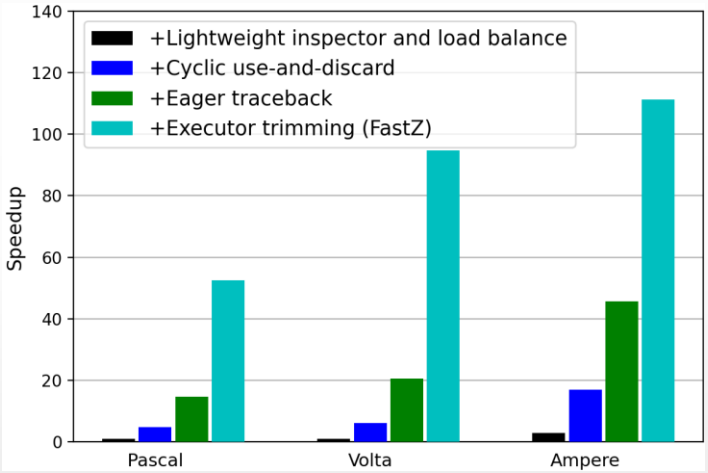
- Single-seed slower than sequential LASTZ
  - Inadequate parallelism
  - Costly inter-SM synchronization
- Multiprocess achieves 20x speedup with 32 processes
  - Does not scale linearly due to memory bandwidth limits

FastZ Performance (2 of 2)



- FastZ achieves mean speedups of
  - Pascal – 43x – 28 SMs (oldest)
  - Volta – 93x – 80 SMs
  - Ampere – 111x – 68 SMs (newest)
- FastZ's speedups vary across benchmarks due to OALs and number of long/short alignments

Isolating FastZ's Optimizations



- Contributions added in succession
  - Lightweight Inspector + load balancing
  - Cyclic-use-and-discard
  - Eager traceback
  - Executor trimming
- Later optimizations seeming better due to optical illusion
  - 50x to 100x looks larger than 2x to 4x

Each optimization contributes to the speedup significantly

Conclusion

- SWDP's challenges
  - Memory bandwidth bound
  - Alignment lengths highly varied and unknown apriori
    - Dynamic memory allocation slow in GPUs
  - Larger search space explored to find OAL
- FastZ
  - Lightweight inspector – finds OAL without traceback
  - Eager traceback in inspector for extremely short OAL
  - Executor trimmed to compute only for OAL
  - Cyclic-use-and-discard to reduce memory traffic
  - Load balancing by binning on OAL
- FastZ is 111x faster than sequential LASTZ
- Fast and highly-sensitive gapped WGA on commodity GPUs

Image References

- [1] https://www.nature.com/articles/s41586-020-2876-6
- [2] Bob Goldstein, UNC Chapel Hill http://bio.unc.edu/people/faculty/goldstein
- [3] https://elifesciences.org/digests/53237/divergent-paths
- [4] Aedes aegypti. Emil August Goeldi (1859-1917) via Wikimedia Commons.