# FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs

N. Satish, C. Kim, J. Chhugani, A. Nguyen, V. Lee, D. Kim, P. Dubey

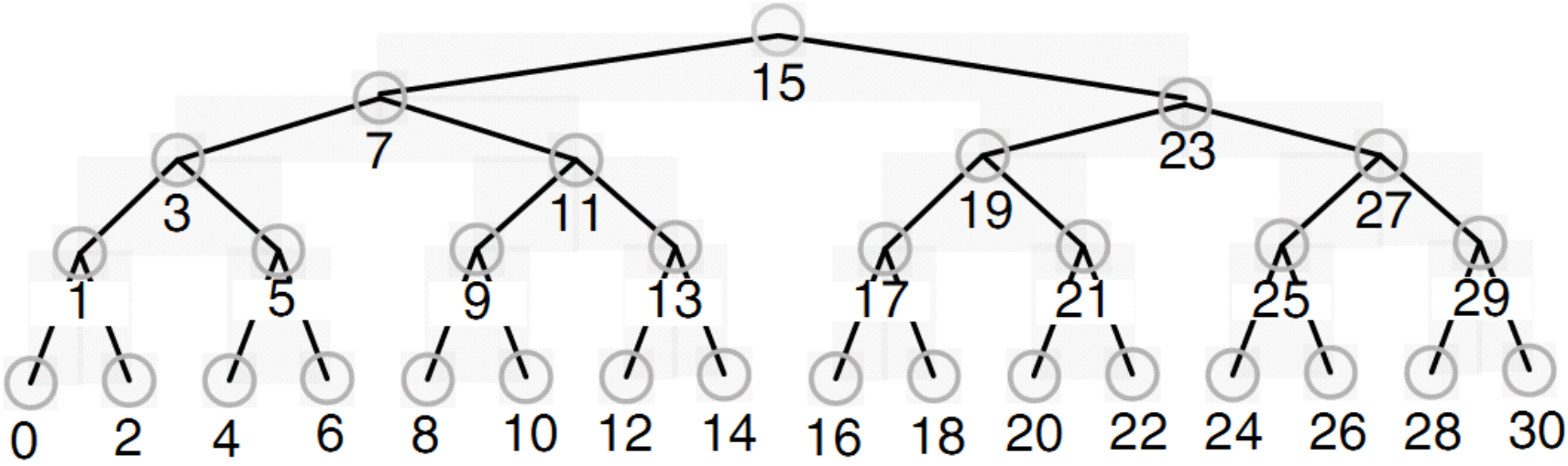
SIGMOD 2010

Presented by: Andy Hwang

# Motivation

- Index trees are not optimized for architecture
- Only one node is accessed per tree level, ineffective cache line utilization
  - Prefetch cannot be used (depends on comparison of search key to parent)
- Nodes in different pages, causing TLB misses
- Previous work optimized for page, cache, SIMD separately, not together
- Compression can be used to save memory bandwidth

# Motivation: Index Tree Layout




Bad for traversal

Motivation

Hierarchical Blocking

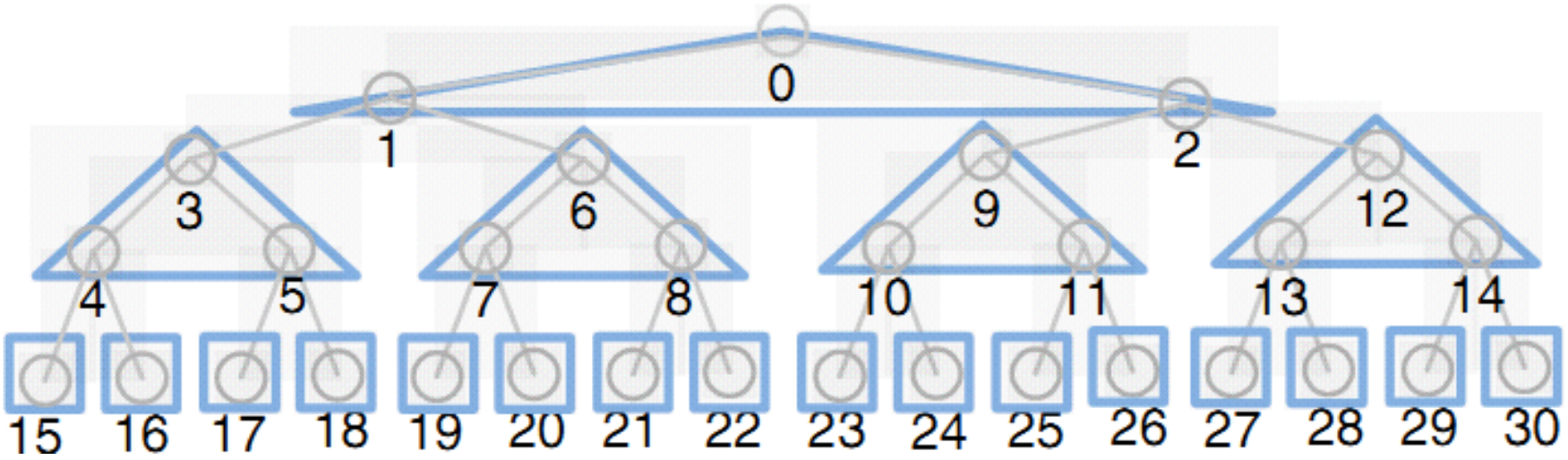
CPU/GPU Implementation

Compression

Throughput/Response Time

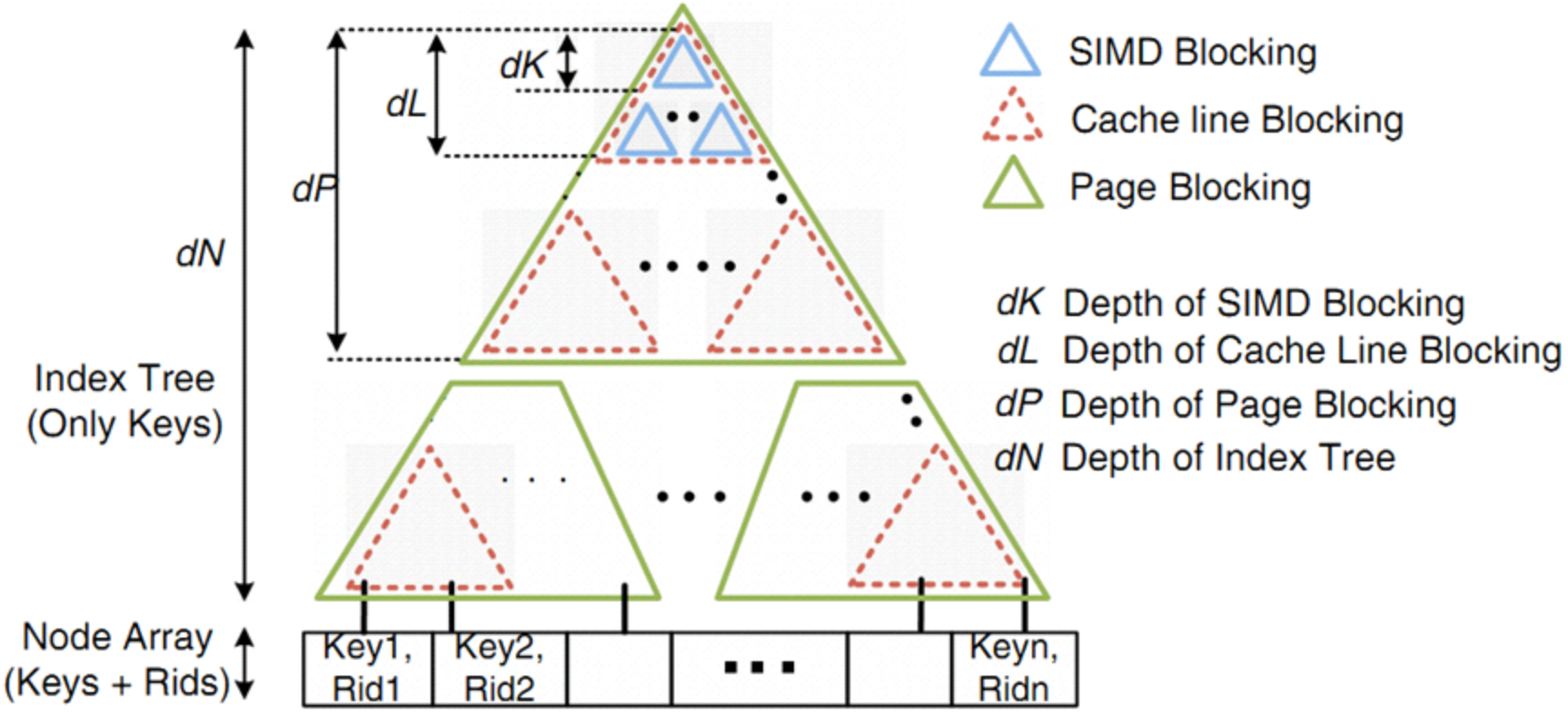
Summary/Discussion

# Hierarchical Blocking




Optimize for accesses (SIMD/cache/memory)

# Hierarchical Blocking



Motivation

Hierarchical Blocking

CPU/GPU Implementation

Compression

Throughput/Response Time

Summary/Discussion

# Tree Construction

- Assuming 4-byte keys (32-bits)
- Block size depends on SIMD instruction width, cache line size, and page size
- Use one SIMD instruction to calculate multiple indices
- Parallelize output amongst CPU cores / GPU shared multiprocessors

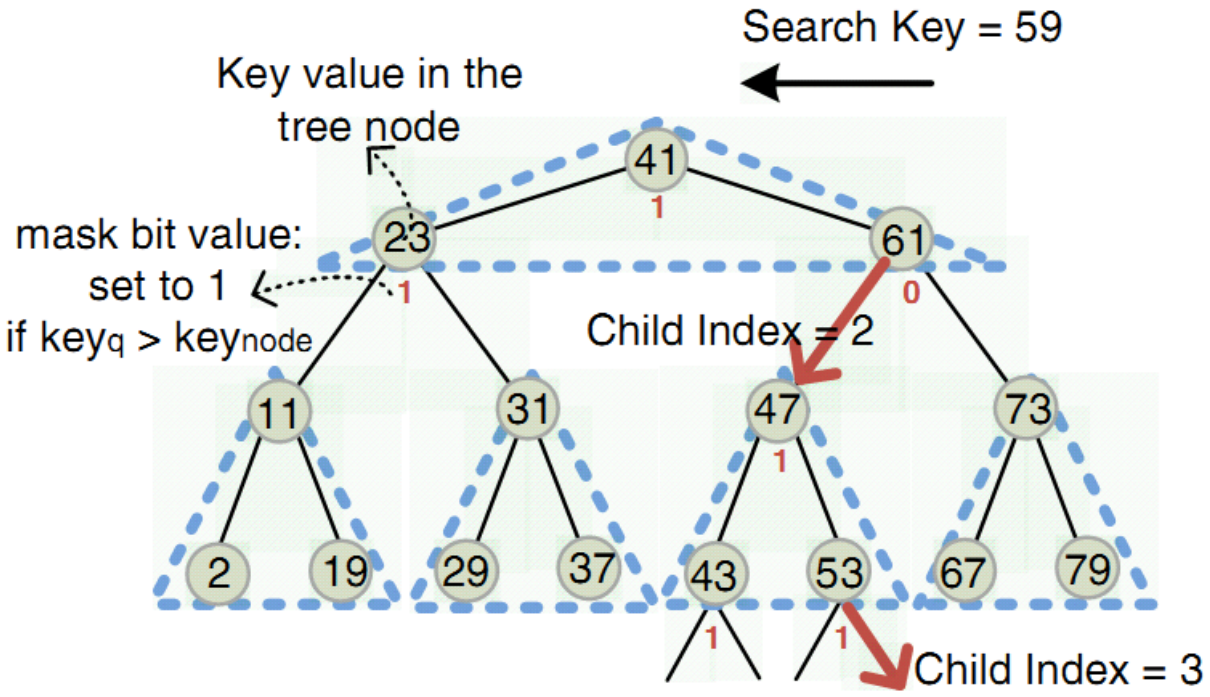
# Tree Construction: CPU

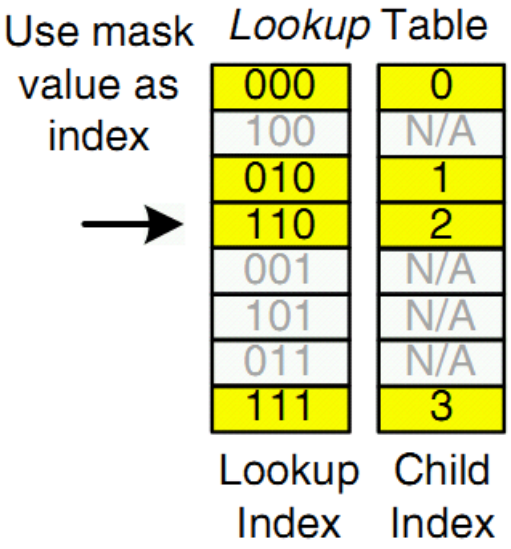
- 128-bit SIMD = max 4 nodes at once
  - SIMD block = 2 tree levels (3 nodes)
- 64-byte cache line = max 16 nodes
  - Cache line block = 4 levels (15 nodes)
- 2MB page size
  - Page block = 19 levels
  - 4KB page = 10 levels

# Tree Construction: GPU

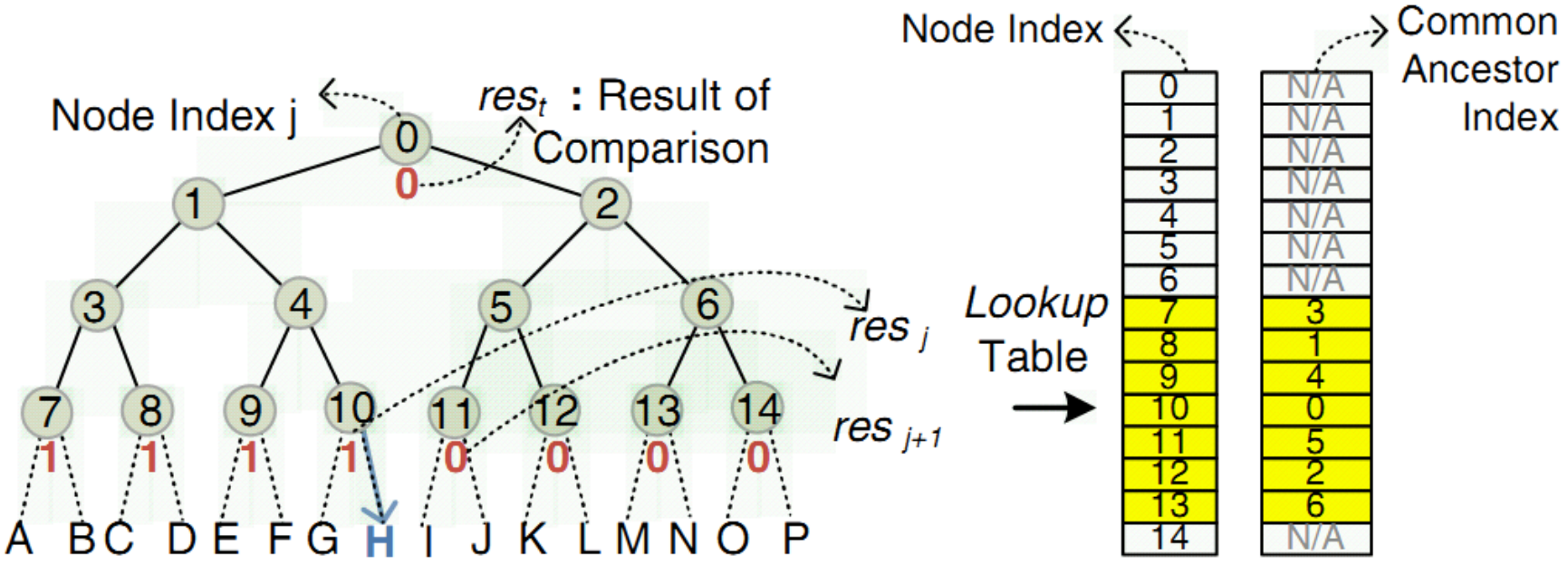
- 32 data elements (thread warp)
  - Various SIMD block sizes possible (up to 32)
  - Set depth to 4 to make use of instruction granularity at half-warp
- No cache exposed – cache line block size set equal to SIMD block size

# Tree Traversal: CPU



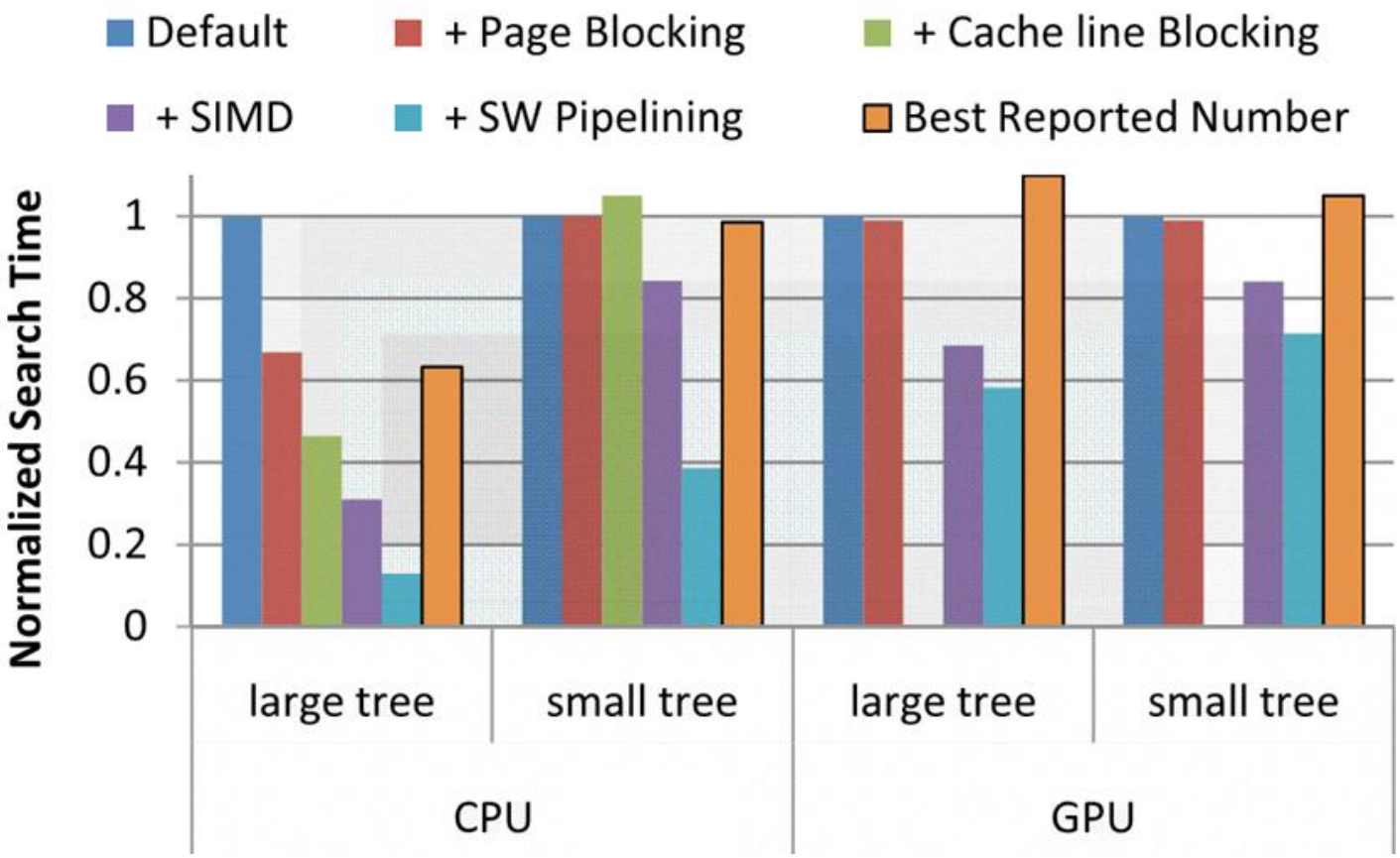
# Tree Traversal: GPU



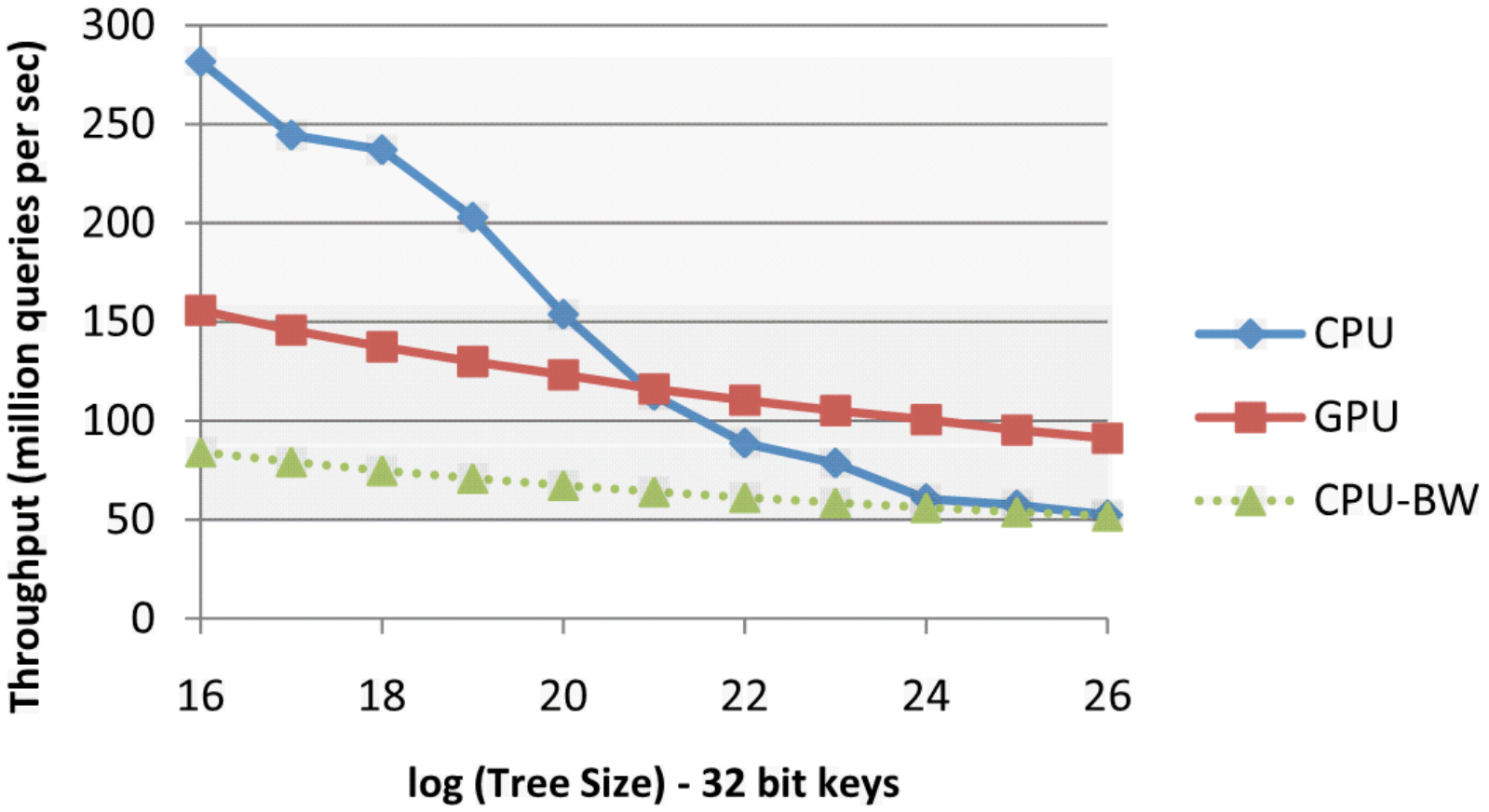
# Simultaneous Queries

- Issue queries in parallel on the hardware
- Software pipelining used to hide cache/TLB miss or GPU memory latency
- CPU: 8 concurrent queries per thread, 64 total
- GPU: 2 concurrent queries per thread warp, 960 total

# Optimization Speedup



# CPU vs GPU Search Throughput



# Tree Traversal: MICA

- Intel Many-Core Architecture Platform
  - Intel GPGPU effort
  - 32KB L1, 256KB L2 (partitioned)
  - 4 threads/core
  - Traversal code similar to CPU
- 16-wide SIMD
  - SIMD block depth = 4 (15 nodes at once)

# Tree Traversal: MICA

| | Throughput (million queries / sec) | |
|---|---|---|
| | Small Tree (64K keys) | Large Tree (16M keys) |
| CPU | 280 | 60 |
| GPU | 150 | 100 |
| MICA | 667 | 183 |

Benefits of both CPU and GPU!

Motivation

Hierarchical Blocking

CPU/GPU Implementation

Compression

Throughput/Response Time

Summary/Discussion

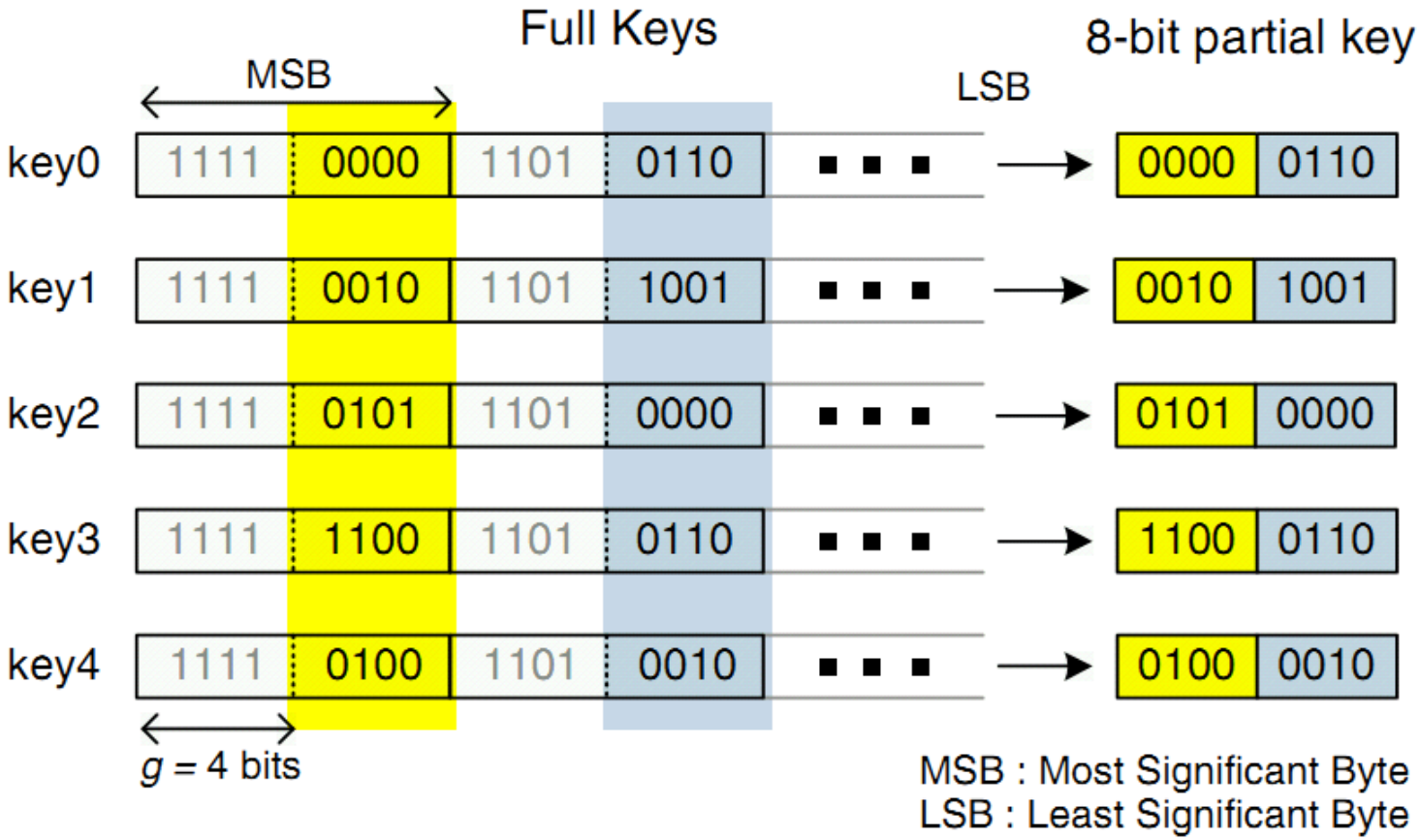
# Compression

- Key sizes are different in practice
  - Impact cache line and page usage
- Non-Contiguous Common Prefix
  - Hashing keys based on their difference (partial keys)
- 4-bit blocks as unit of compression
  - SIMD instruction to find similarity and compress

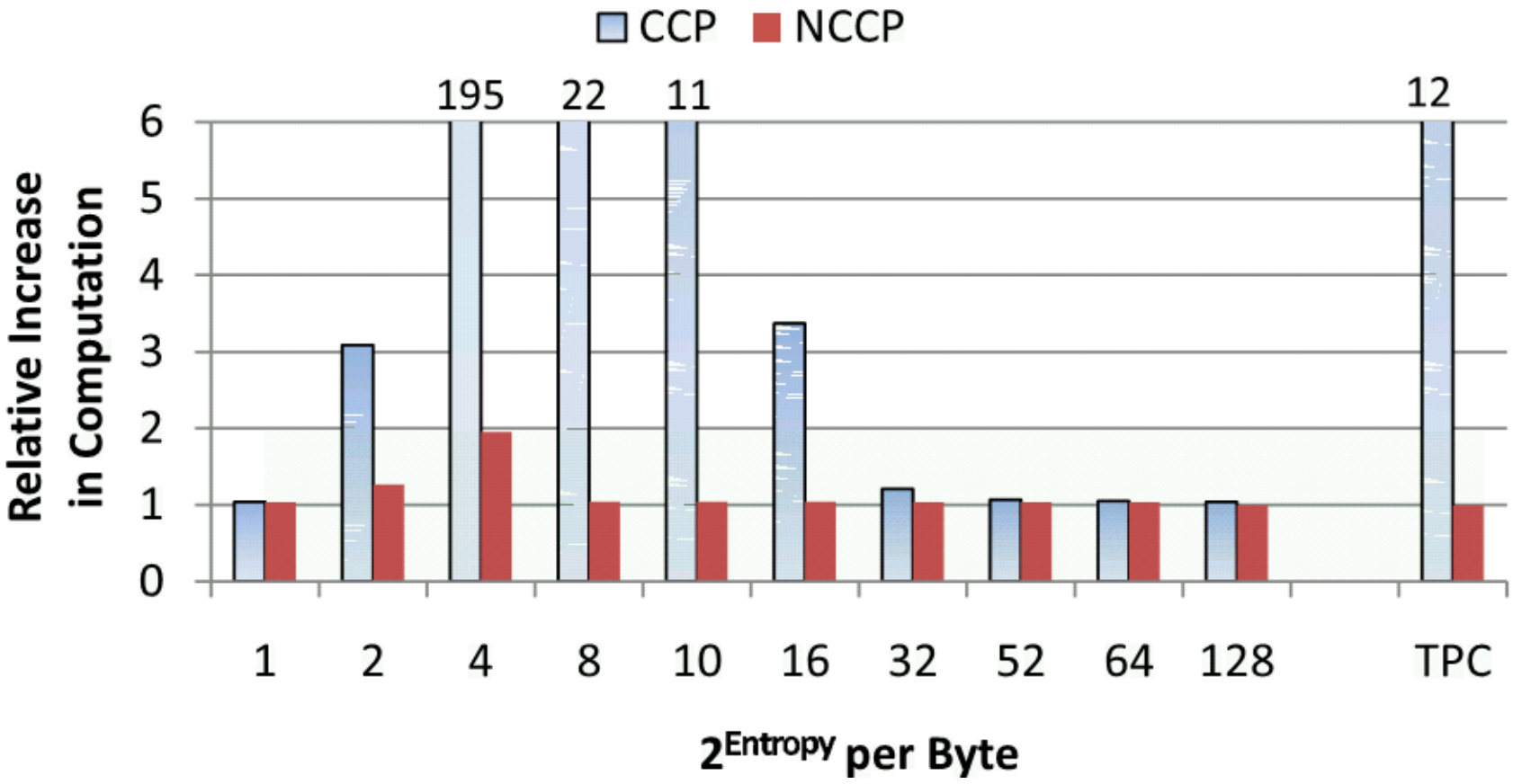
# Compression

- First page partial key size is larger (128 bits) to reduce false positives
  - Subsequent pages have partial key size 32
- Construction overhead increased
  - +75% for variable size keys, +30% integer keys
- During traversal, the query key is compressed

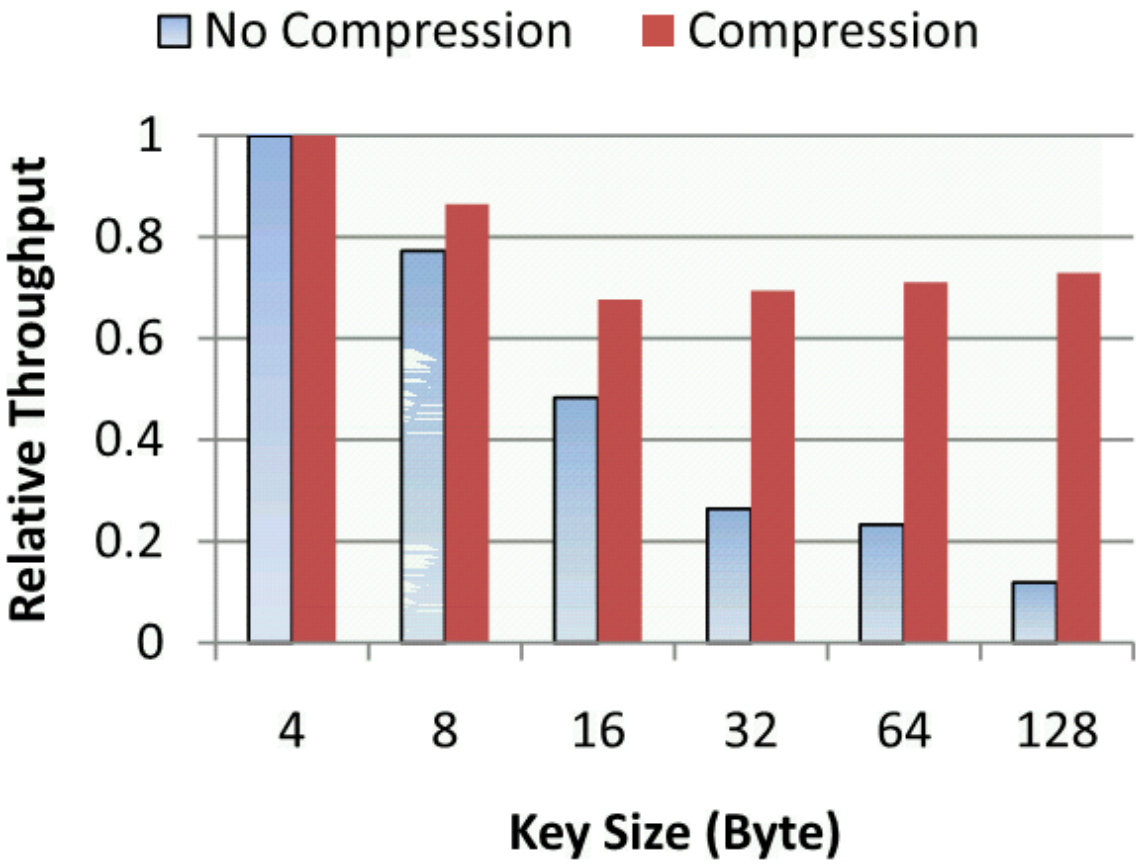
# Compression



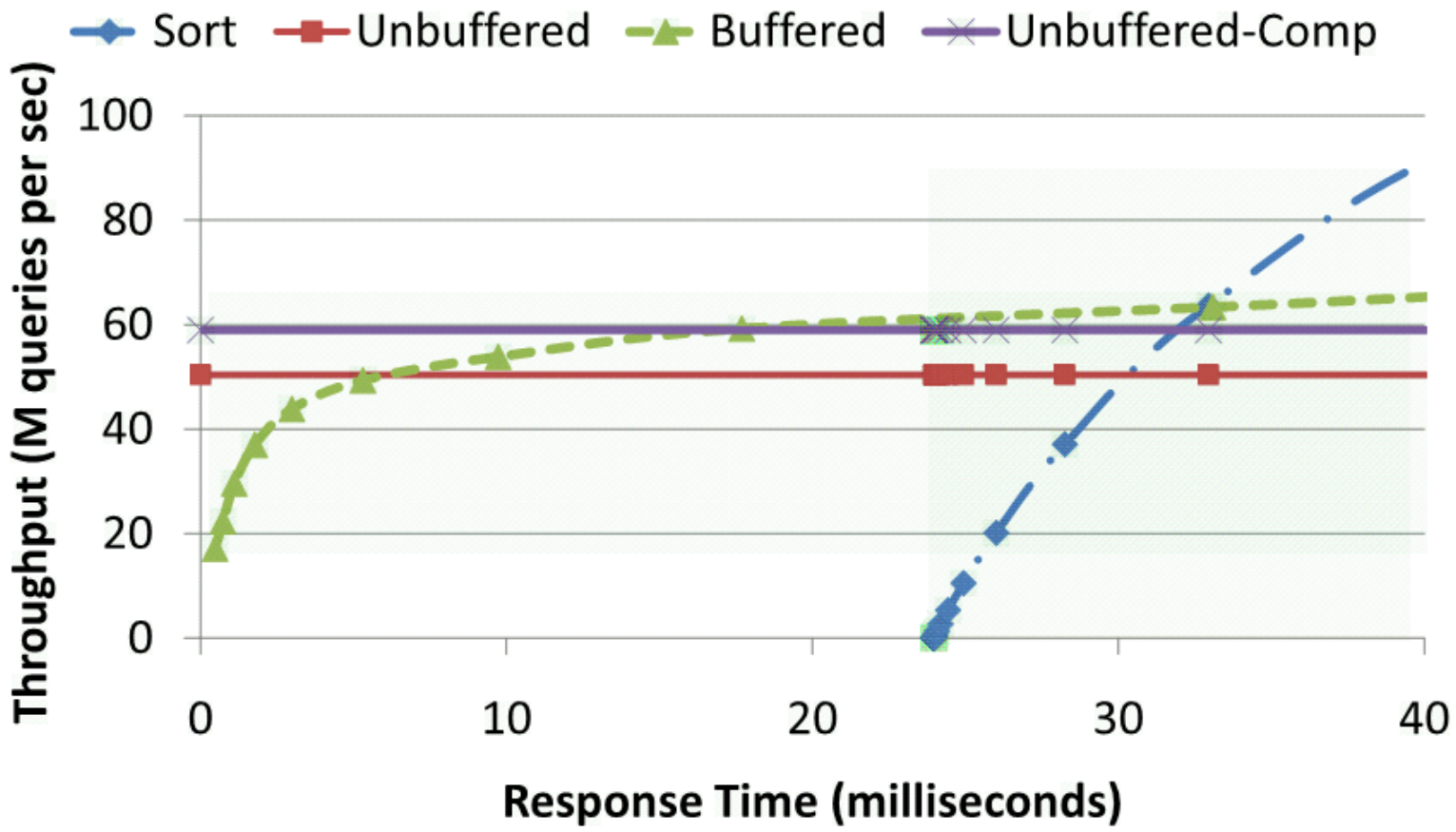
# Compression: Alphabet Size



# Compression: Throughput



# Query Batching/Buffering



# Summary

- Hierarchical blocking to optimize search tree for page, cache, SIMD instructions
  - Architectural-aware block depths
- CPU/GPU/MICA implementations
  - Fast construction, search, and parallel queries for varying tree sizes
- Hide memory latency wherever possible
- NCCP compression for integer and variable length keys
- Throughput/Response time for different query batching schemes

# Discussion

- Focus on throughput
  - Assumes large number of queries
  - Not much info on latency
- Updates
  - Full reconstruction? Flushed from cache?
- Synthetic workloads
- Deployment