EXPLOITING COARSE-GRAINED PARALLELISM IN B+ TREE SEARCHES ON APUS

Mayank Daga
AMD Research

Mark Nutter
AMD Heterogeneous System Software

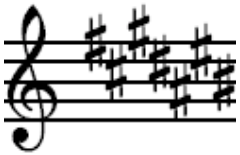
AMD

B+ TREE SEARCHES

- B+ Tree is a fundamental data structure used in
  - Relational Database Management Systems (RDBMS)

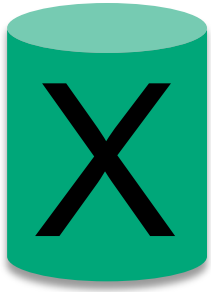
  - Key-Value Database Management Systems

- High-throughput, read-only index searches are gaining traction in



  - Audio-search
  - Video-copy detection

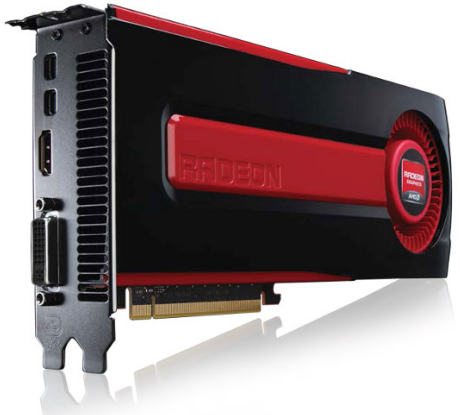
  - Online Transaction Processing (OLTP) Benchmarks
- Increase in memory capacity allows many database tables to reside in memory
  - Brings computational performance to the forefront



AMD

DATABASE PRIMITIVES ON ACCELERATORS

- Discrete graphics processing units (dGPUs) provide a compelling mix of
  - Performance per Watt
  - Performance per Dollar

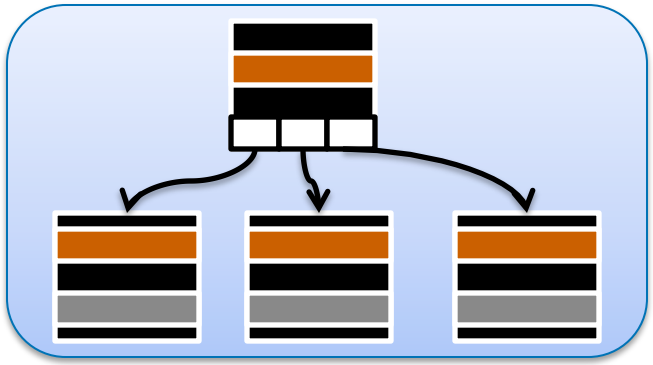


- dGPUs have been used to accelerate critical database primitives
  - scan
  - sort
  - join
  - aggregation
  - B+ Tree Searches?

AMD

B+ TREE SEARCHES ON ACCELERATORS

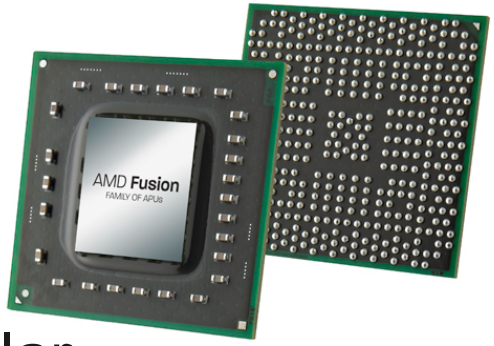
- B+ Tree searches present significant challenges
  - Irregular representation in memory
    - An artifact of malloc() and new()



  - Today's dGPUs do not have a direct mapping to the CPU virtual address space
    - Indirect links need to be converted to relative offsets
  - Requirement to copy the tree to the dGPU, which entails
    - *One is always bound by the amount of GPU device memory*

OUR SOLUTION

- Accelerated B+ Tree searches on a fused CPU+GPU processor (or APU[1])
  - Eliminates data-copies by combining x86 CPU and vector GPU cores on the same silicon die



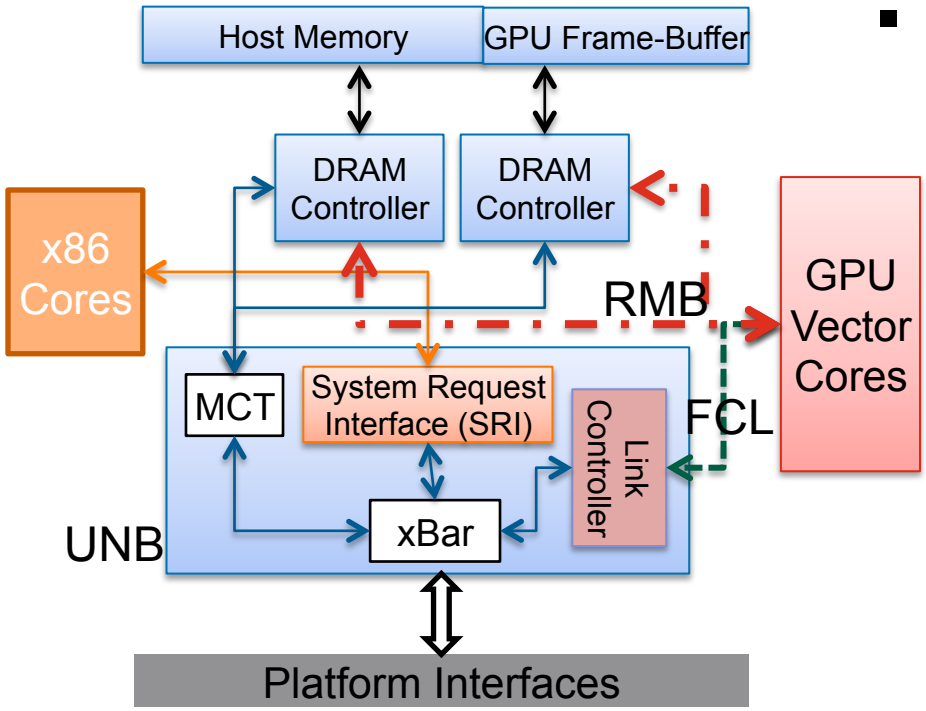
- Developed a memory allocator to form a regular representation of the tree in memory
  - Fundamental data structure is *not* altered
  - Merely parts of its layout is changed

[1] www.hsafoundation.com

OUTLINE

- Motivation and Contribution
- Background
  - AMD APU Architecture
  - B+ Trees
- Approach
  - Transforming the Memory Layout
  - Eliminating the Divergence
- Results
  - Performance
  - Analysis
- Summary and Next Steps

AMD APU ARCHITECTURE

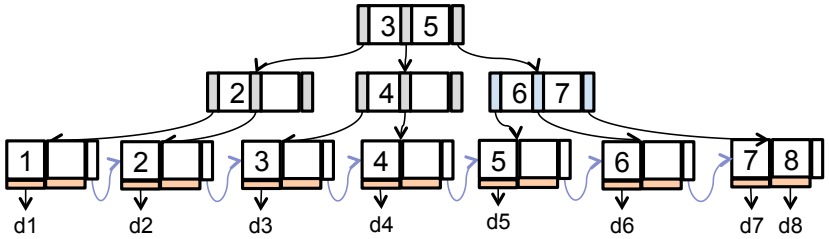


AMD 2nd Gen. A-series APU

- The APU consists of a dedicated IOMMUv2 hardware
  - Provides direct mapping between GPU and CPU virtual address (VA) space
  - Enables GPUs to access the system memory
  - Enables GPUs to track whether pages are resident in memory

UNB - Unified Northbridge, MCT - Memory Controller,
RMB - Radeon Memory Bus, FCL - Fusion Compute Link

B+ TREES

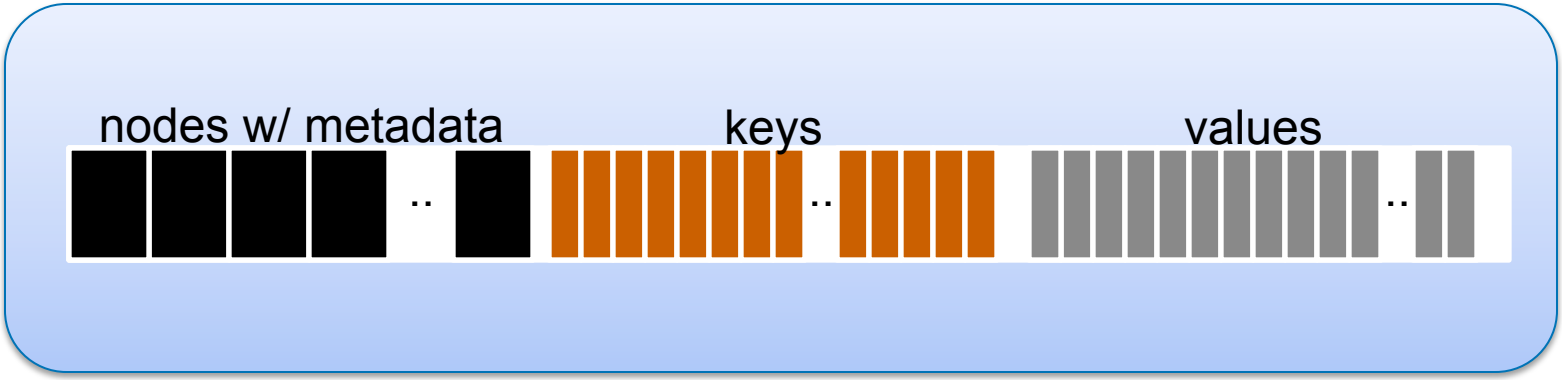


- A B+ Tree …
  - is a dynamic, multi-level index
  - Is efficient for retrieval of data, stored in a block-oriented context
  - has a high fan-out to reduce disk I/O operations
- Order (b) of a B+ Tree measures the capacity of its nodes
- Number of children (m) in an internal node is
  - [b/2] <= m <= b
  - Root node can have as few as two children
- Number of keys in an internal node = (m – 1)

APPROACH FOR PARALLELIZATION

- Fine-grained (Accelerate a single query)
  - Replace Binary search in each node with K-ary search
  - Maximum performance improvement = log(k)/log(2)
  - Results in poor occupancy of the GPU cores
- Coarse-grained (Perform many queries in parallel)
  - Enables data-parallelism
  - Increases memory bandwidth with parallel reads
  - Increases throughput (transactions per second for OLTP)

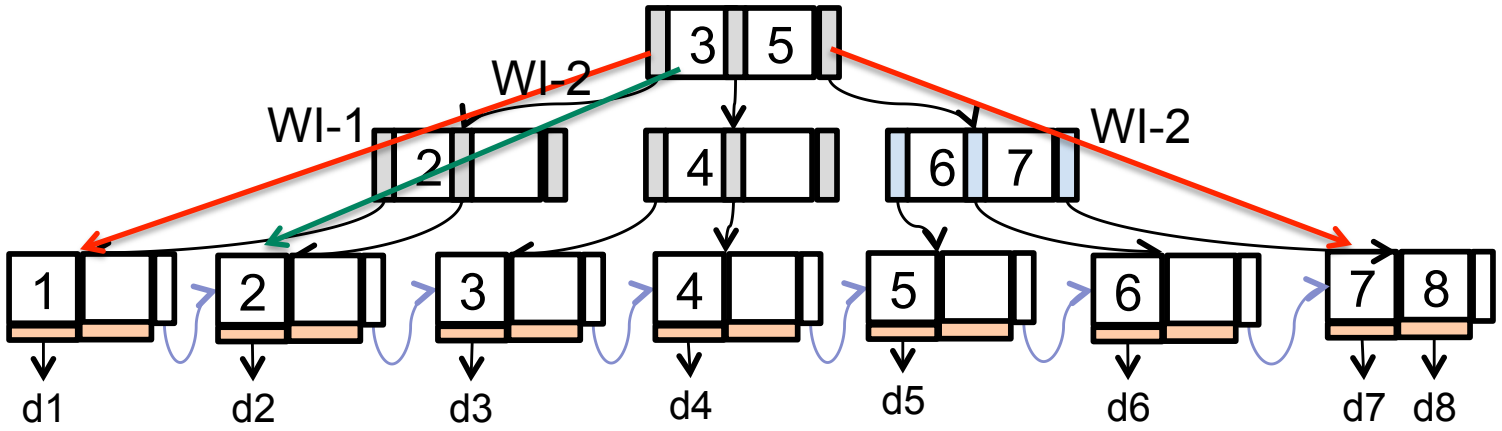
TRANSFORMING THE MEMORY LAYOUT



- Metadata
  - Number of keys in a node
  - Offset to keys/values in the buffer
  - Offset to the first child node
  - Whether a node is a leaf
- Pass a pointer to this memory buffer to the accelerator

ELIMINATING THE DIVERGENCE

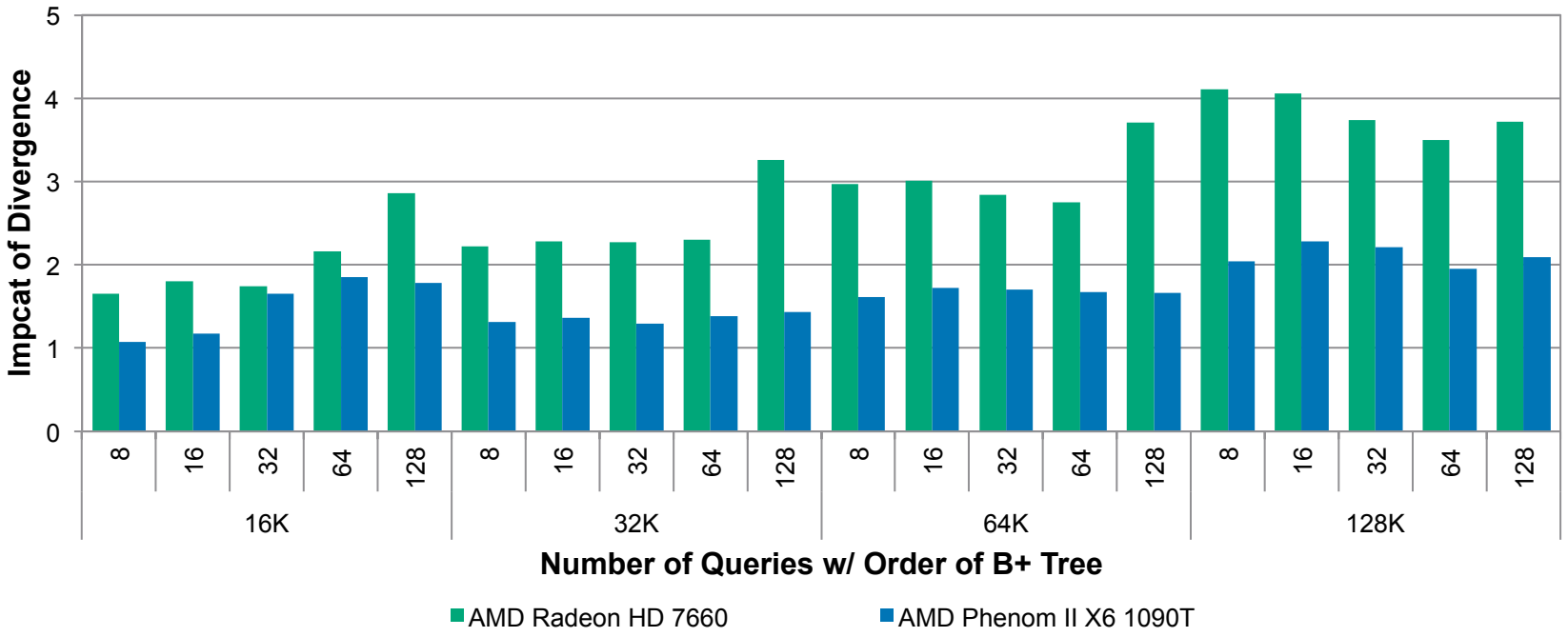
- Each work-item/thread executes a single query
- May increase divergence within a wave-front
  - Every query may follow a different path in the B+ Tree



- Sort the keys to be searched
  - Increases the chances of work-items within a wave-front to follow similar paths in the B+ Tree
  - We use Radix Sort[1] to sort the keys on the GPU

[1] D. G. Merrill and A. S. Grimshaw, "Revisiting sorting for gpgpu stream architectures," in *Proceedings of the 19th intl. conf. on Parallel architectures and compilation techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010.

IMPACT OF DIVERGENCE IN B+ TREE SEARCHES



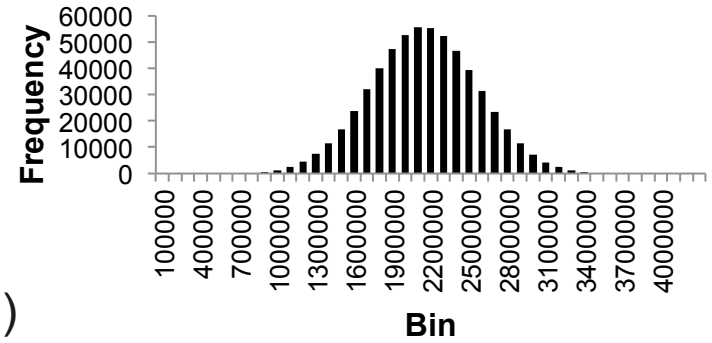
Impact of Divergence on GPU – 3.7-fold (average)
Impact of Divergence on CPU – 1.8-fold (average)

OUTLINE

- Motivation and Contribution
- Background
  - AMD APU Architecture
  - B+ Trees
- Approach
  - Transforming the Memory Layout
  - Eliminating the Divergence
- Results
  - Performance
  - Analysis
- Summary and Next Steps

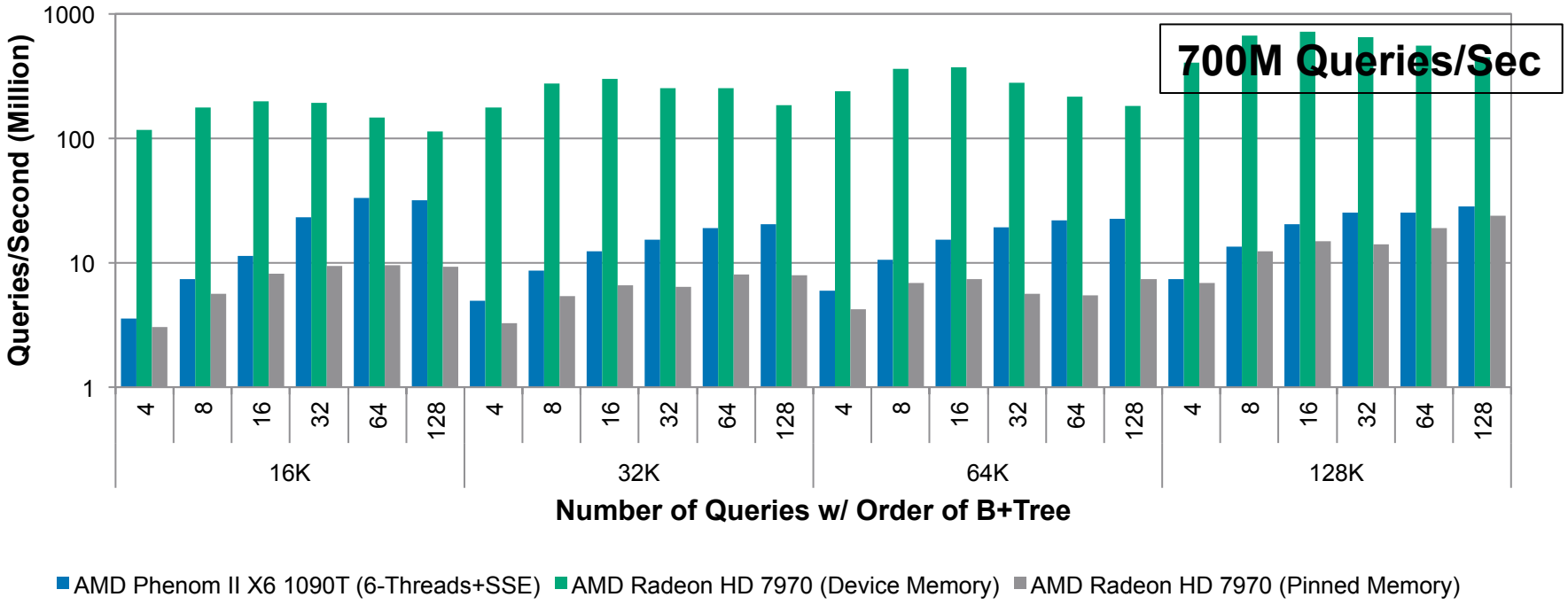
EXPERIMENTAL SETUP

- Software
  - A B+ Tree w/ 4M records is used
  - Search queries are created using
    - `normal_distribution()` (C++-11 feature)
    - The queries have been sorted
  - CPU Implementation from
    - http://www.amittai.com/prose/bplustree.html
  - Driver: AMD Catalyst™ v12.8
  - Programming Model: OpenCL™
- Hardware
  - AMD Radeon HD 7660 APU (Trinity)
    - 4 cores w/ 6GB DDR3, 6 CUs w/ 2GB DDR3
  - AMD Phenom II X6 1090T + AMD Radeon HD 7970 (Tahiti)
    - 6 cores w/ 8GB DDR3, 32 CUs w/ 3GB GDDR5
  - Device Memory does *not* include data-copy time



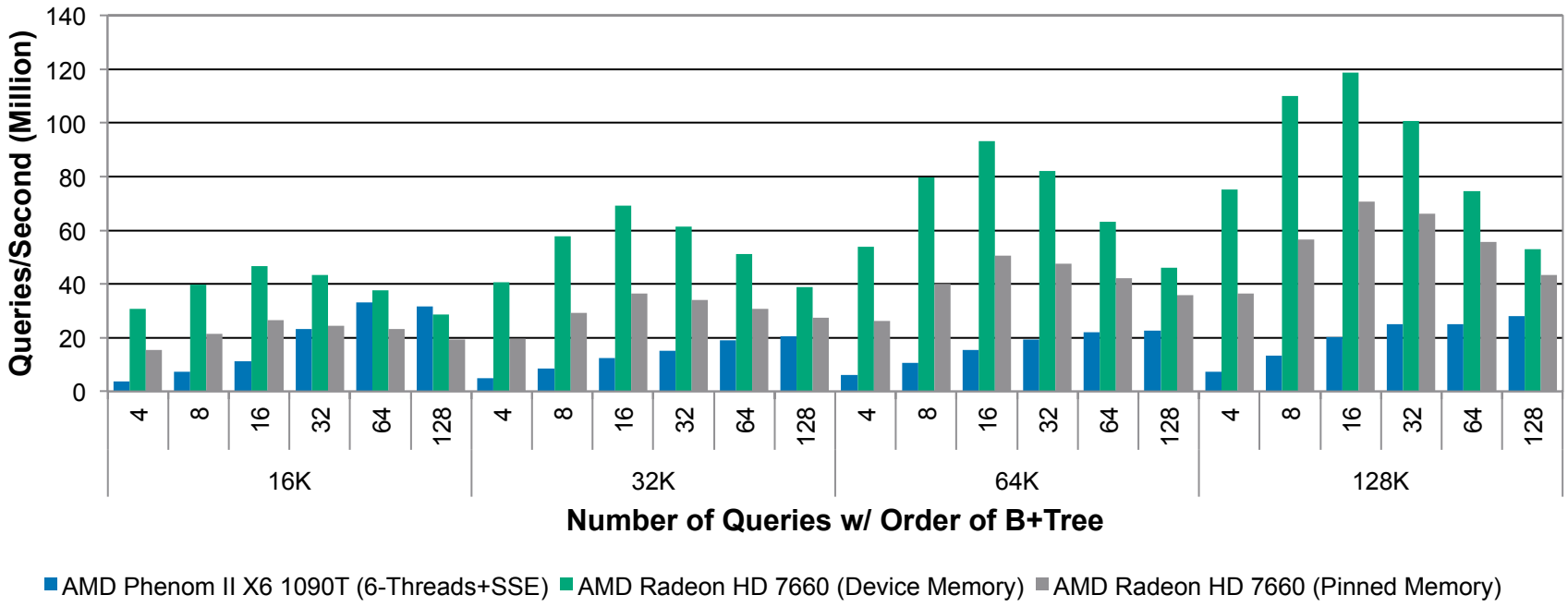
E ID (PKey)	Age
0000001	34
4 million entries	
4194304	50

RESULTS – QUERIES PER SECOND



dGPU (device memory)	~350M Queries/Sec. (avg.)
dGPU (pinned memory)	~9M Queries/Sec. (avg.)
Phenom CPU	~18M Queries/Sec. (avg.)

RESULTS – QUERIES PER SECOND

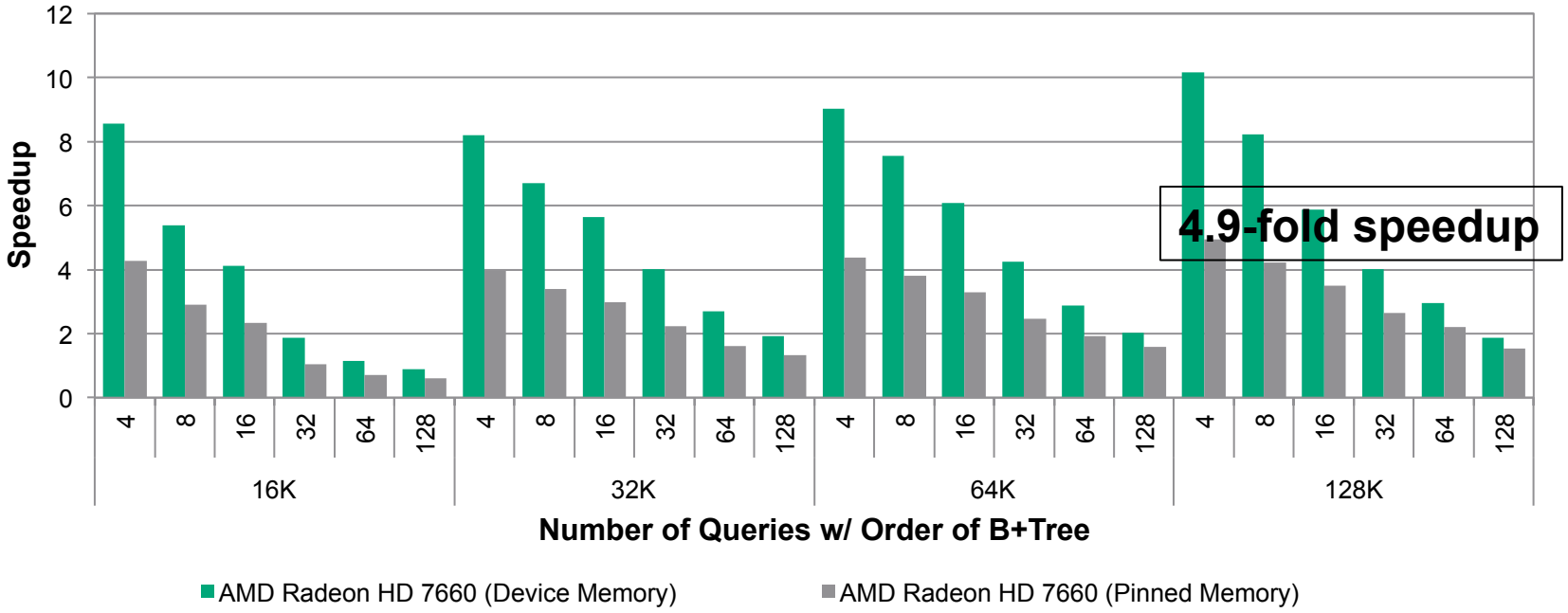


APU (device memory) ~66M Queries/Sec. (avg.)

APU (pinned memory) ~40M Queries/Sec. (avg.)

APU (pinned memory) is faster than the CPU implementation

RESULTS - SPEEDUP



Baseline: six-threaded, hand-tuned, SSE-optimized CPU implementation.

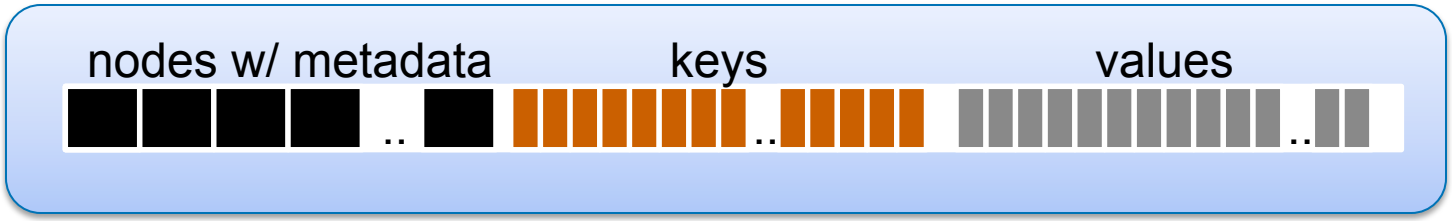
Average Speedup – 4.3-fold (Device Memory), 2.5-fold (Pinned Memory)

- Efficacy of IOMMUv2 + HSA on the APU

Platform	Size of the B+ Tree		
	< 1.5GB	1.5GB – 2.7GB	> 2.7GB
Discrete GPU *(memory size = 3GB)*	✓	✓	✗
APU *(prototype software)*	✓	✓	✓

ANALYSIS

- The accelerators and the CPU yield best performance for different orders of the B+ Tree
  - CPU → order = 64
    - Ability of CPUs to prefetch data is beneficial for higher orders



  - APU and dGPU → order = 16
    - GPUs do not have a prefetcher → cache line should be most efficiently utilized
    - GPUs have a cache-line size of 64 bytes
      - Order = 16 is most beneficial (16 * 4 bytes)

ANALYSIS

- Minimum batch size to match the CPU performance

	Order = 64	**Order = 16**
dGPU (device memory)	4K queries	2K queries
dGPU (pinned memory	N.A.	N.A.
APU (device memory)	10K queries	4K queries
APU (pinned memory	20K queries	16K queries

- *reuse_factor* - amortizing the cost of data-copies to the GPU

$$
\begin{aligned}
Time_{accel} &= T_{copy} + (T_{acclExec} * reuse\_factor) \\
Time_{cpu} &= T_{cpuExec} * reuse\_factor \\
\text{or } reuse\_factor &<= T_{copy}/(T_{cpuExec} - T_{acclExec})
\end{aligned}
$$

	90% Queries	**100% Queries**
dGPU	15	54
APU	100	N.A.

PROGRAMMABILITY

CPU-SSE

```
int i = 0, j;
node * c = root;
__m128i vkey = _mm_set1_epi32(key);
__m128i vnodekey, *vptr;
short int mask;
/* find the leaf node */
while( !c->is_leaf ){
  for(i = 0; i < (c->num_keys-3); i+=4){
    vptr = (__m128i *)&(c->keys[i]);
    vnodekey = _mm_load_si128(vptr);
    mask =
_mm_movemask_ps(_mm_cvtepi32_ps( _mm_cmplt_epi32(vkey,
vnodekey)));
    if((mask) & 8) break;
  }
  for(j = i; j < c->num_keys; j++){
    if(key < c->keys[j]) break;
  }
  c = (node *)c->pointers[j];
  }
/* match the key in the leaf node */
for (i = 0; i < c->num_keys; i++)
  if (c->keys[i] == key) break;
/* retrieve the record */
if (i != c->num_keys)
  return (record *)c->pointers[i];
```

GPU

```
typedef global unsigned int g_uint;
typedef global mynode g_mynode;
int tid = get_global_id(0);
int i = 0;
g_mynode *c = (g_mynode *)root;
/* find the leaf node */
while(!c->is_leaf){
    while (i < c->num_keys){
        if(keys[tid] >= ((g_uint *)((intptr_t)root + c->keys))
[i])
            i++;
        else break;
    }
    c = (g_mynode *)((intptr_t)root + c->ptr +
i*sizeof(mynode));
}
/* match the key in the leaf node */
for(i=0; i<c->num_keys; i++){
    if(((g_uint *)((intptr_t)root + c->keys))[i] ==
keys[tid]) break;
}
/* retrieve the record */
if(i != c->num_keys)
    records[tid] = ((g_uint *)((intptr_t)root + c->is_leaf +
i*sizeof(g_uint)))[0];
```

RELATED WORK

- J. Fix, A. Wilkes, and K. Skadron, "*Accelerating Braided B+ Tree Searches on a GPU with CUDA*." In *Proceedings of the 2nd Workshop on Applications for Multi and Many Core Processors: Analysis, Implementation, and Performance*, in conjunction with ISCA, 2011
  - Authors report ~10-fold speedup over single-thread-non-SSE CPU implementation, using a discrete NVIDIA GTX 480 GPU (*do not take data-copies into account*)
- C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, P. Dubey, "*FAST: fast architecture sensitive tree search on modern CPUs and GPUs*", SIGMOD Conference, 2010
  - Authors report ~100M queries per second using a discrete NVIDIA GTX 280 GPU (*do not take data-copies into account*)
- J. Sewall, J. Chhugani, C. Kim, N. Satish, P. Dubey, "*PALM: Parallel, Architecture-Friendly, Latch-Free Modifications to B+ Trees on Multi-Core Processors*", Proceedings of VLDB Endowment, (VLDB 2011)
  - Applicable for B+ Tree modifications on the GPU

SUMMARY

- B+ Tree is the fundamental data structure in many RDBMS
  - Accelerating B+ Tree searches is critical
    - Presents significant challenges on discrete GPUs
- We have accelerated B+ Tree searches by exploiting coarse-grained parallelism on a APU
  - 2.5-fold (avg.) speedup over 6-threads+SSE CPU implementation
- Possible Next Steps
  - HSA + IOMMUv2 would alleviate the issue of modifying B+ Tree representation
    - Investigate CPU-GPU co-scheduling
  - Investigate modifications on the B+ Tree

AMD

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.