The next frontier: GPUs

ANIL SHANBHAG

Intelligent Machines

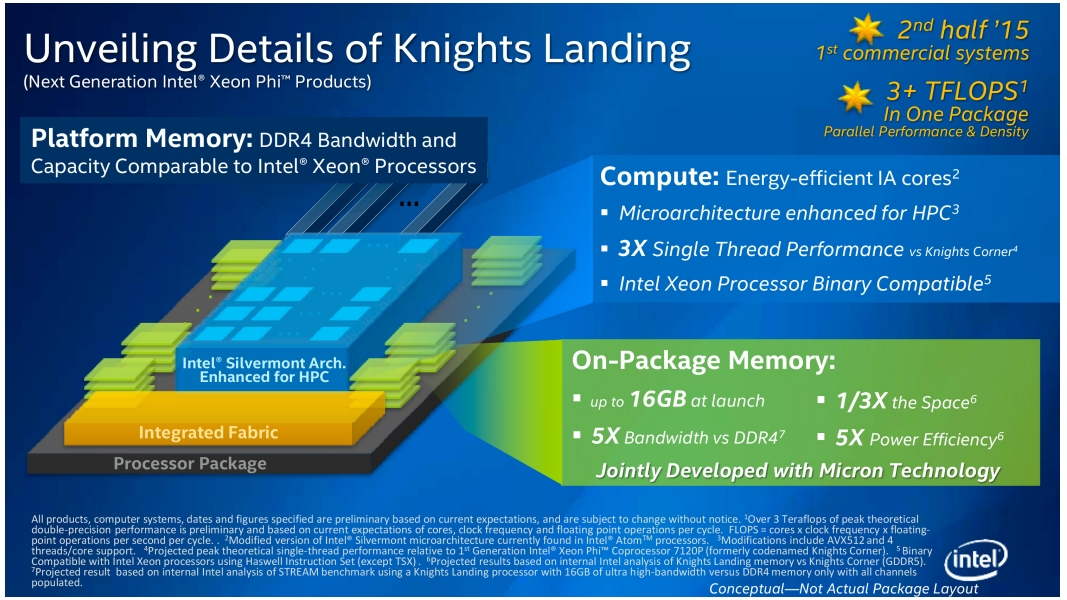
Moore's Law Is Dead. Now What?

Shrinking transistors have powered 50 years of advances in computing—but now other ways must be found to make computers more capable.

- MIT Tech Review, May 2016

Variety of hardware

Common theme: Large number of cores with SIMD/SIMT



Introduction to NVIDIA GPUs in Azure

Introducing Amazon EC2 P2 Instances, the largest GPU-Powered virtual machine in the cloud

GPUs are now available for Google Compute Engine and Cloud Machine Learning

Tuesday, February 21, 2017

You can now get VMs

With multiple Tesla K80 cards

Upto 8 K80s per machine
Total GPU memory: 192GB DDR5 !!

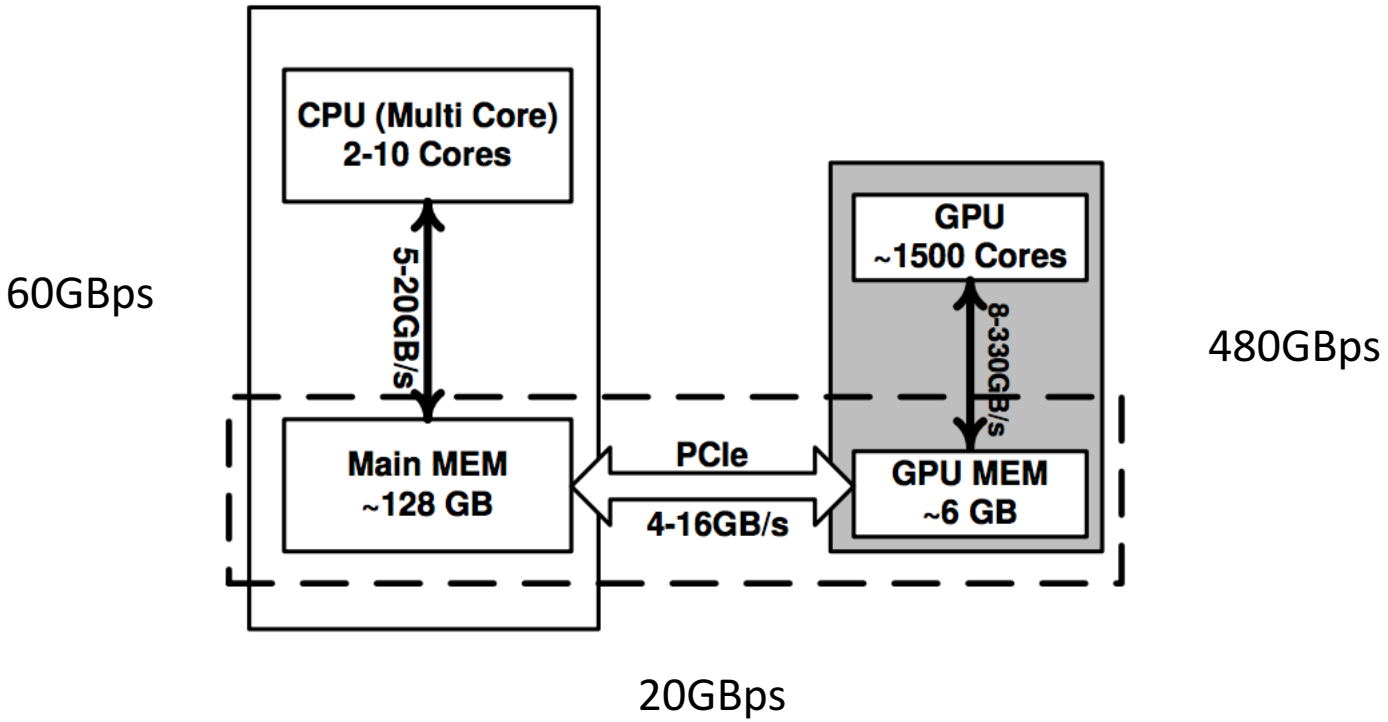
Outline

GPU Architecture

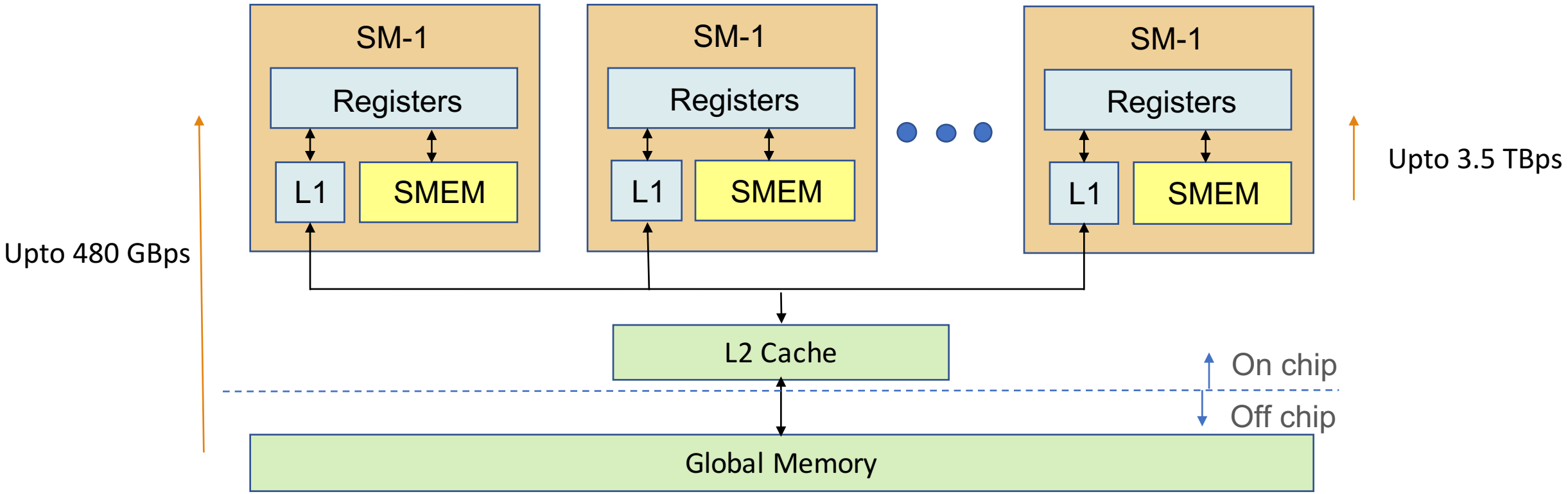
Doing Top-K on the GPU

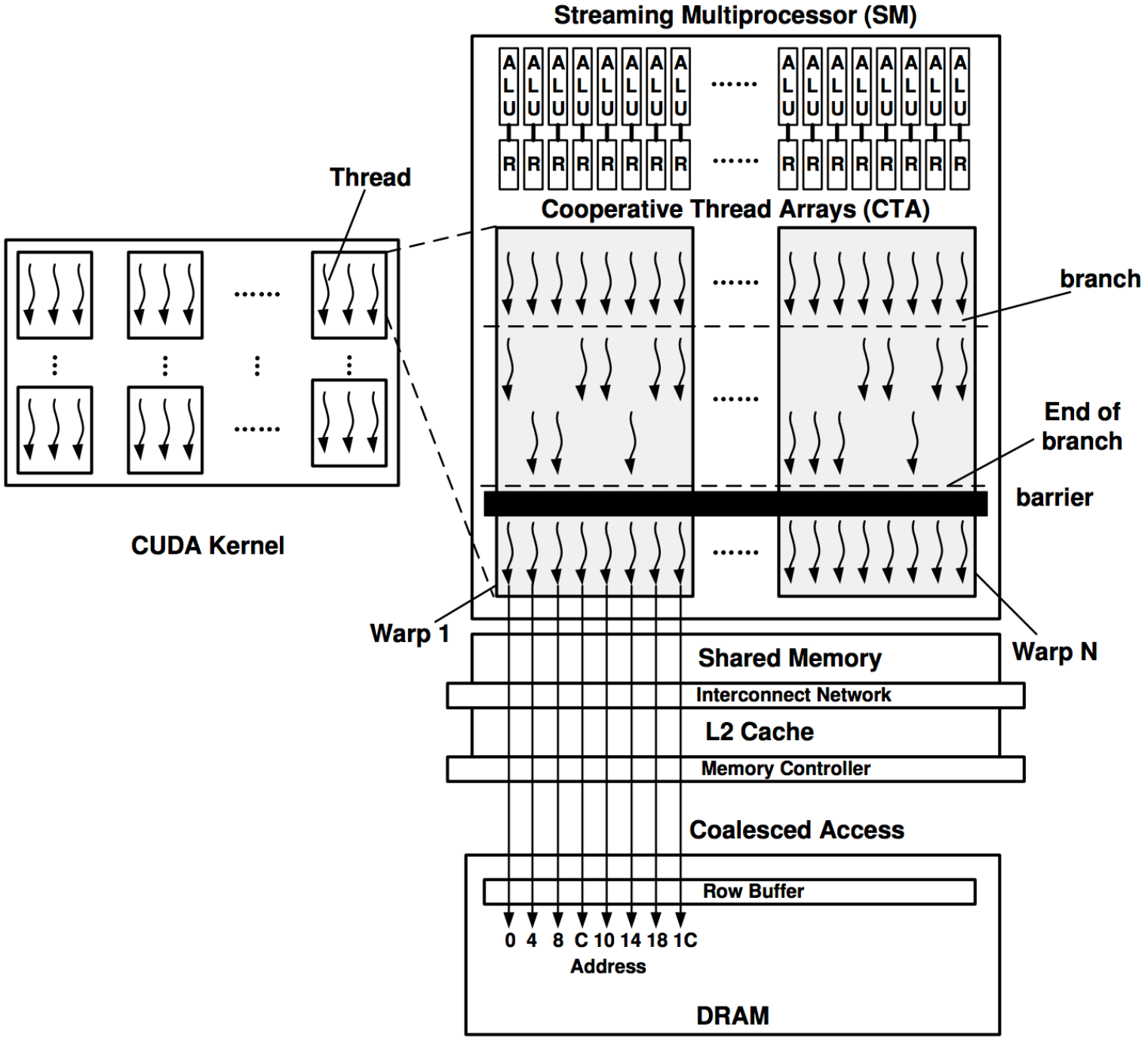
Some recent work on CPU-GPU co-processing

CPU-GPU


60GBps
CPU (Multi Core)
2-10 Cores
5-20GB/s
Main MEM
~128 GB
PCIe
4-16GB/s
GPU
~1500 Cores
8-330GB/s
GPU MEM
~6 GB
480GBps
20GBps

Inside a GPU


SM-1
Registers
L1
SMEM
SM-1
Registers
L1
SMEM
SM-1
Registers
L1
SMEM
Upto 3.5 TBps
Upto 480 GBps
L2 Cache
On chip
Off chip
Global Memory



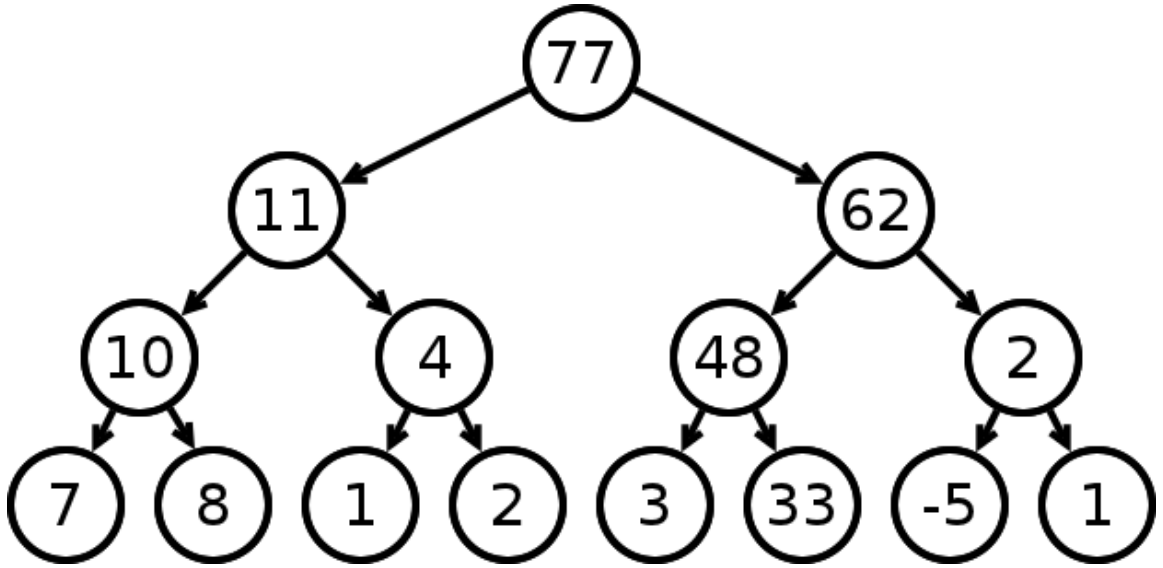
GPU Execution Model

Top-K

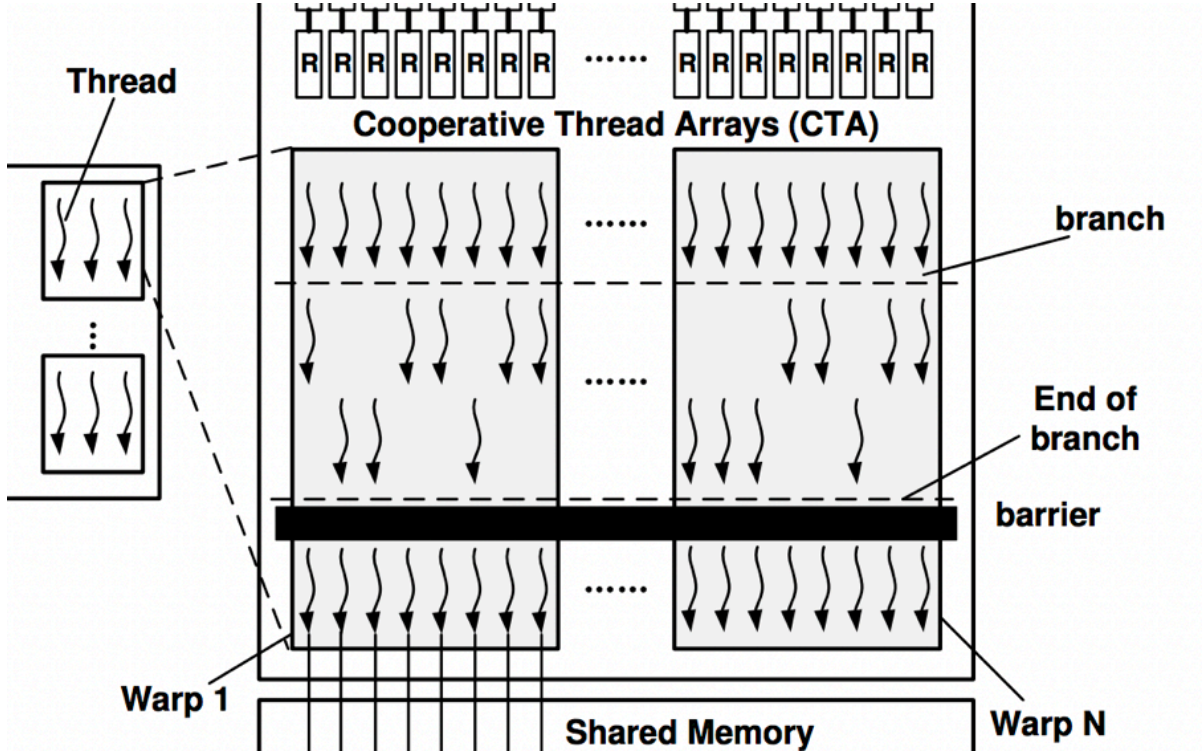
From a list on n elements – find the highest k elements

Use a min-heap to maintain the top-k ---------->

On multi-core machines,
each core can maintain its local top-k
find the global top-k among local top-k's



Problems !!

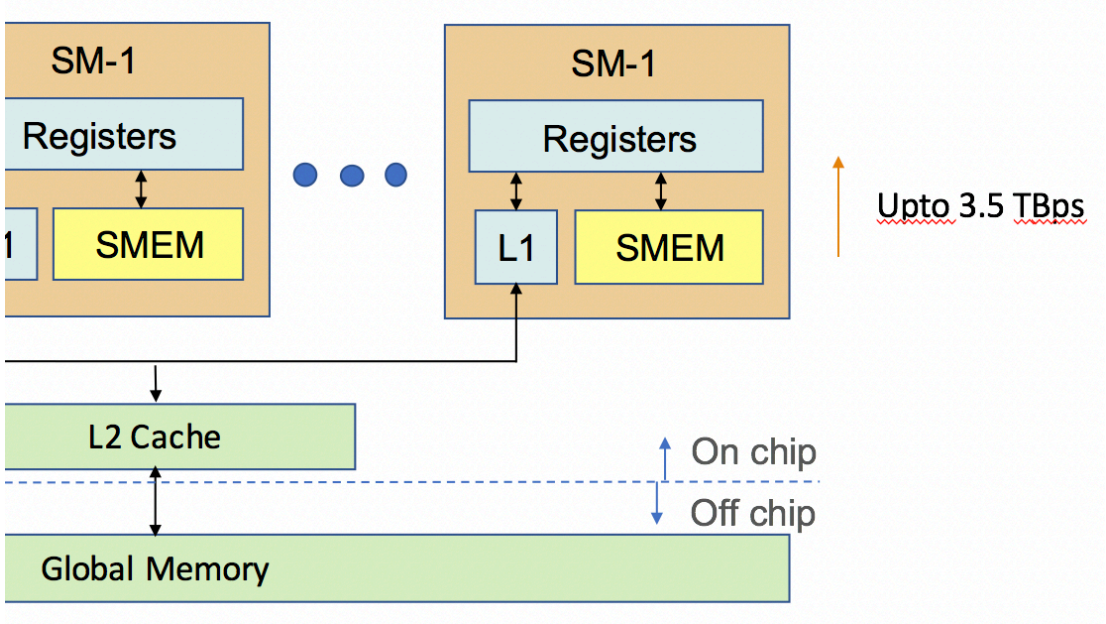


1) Thread Divergence

A warp (32 threads) execute in lock steps.

So if one thread wants to update its heap, everyone runs the same code path ☹

Problems !!



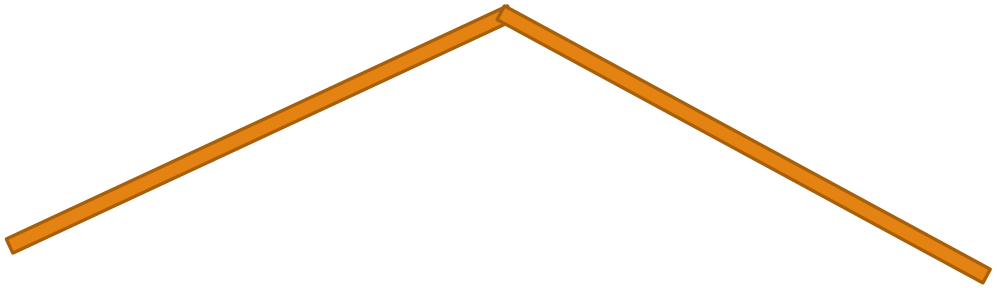
2) Occupancy

Shared memory is a precious resource.

Each thread needs ***k*** entries in shared memory.
Grows linearly with ***k***.

After a point affects occupancy.

Bitonic Top-K



Sorting a Bitonic Sequence

- Let $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ be a bitonic sequence such that
  - $a_0 \leq a_1 \leq \cdots \leq a_{n/2-1}$, and
  - $a_{n/2} \geq a_{n/2+1} \geq \cdots \geq a_{n-1}$

- Consider the following subsequences of s

$$s_1 = \langle \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots, \min(a_{n/2-1}, a_{n-1}) \rangle$$

$$s_2 = \langle \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots, \max(a_{n/2-1}, a_{n-1}) \rangle$$

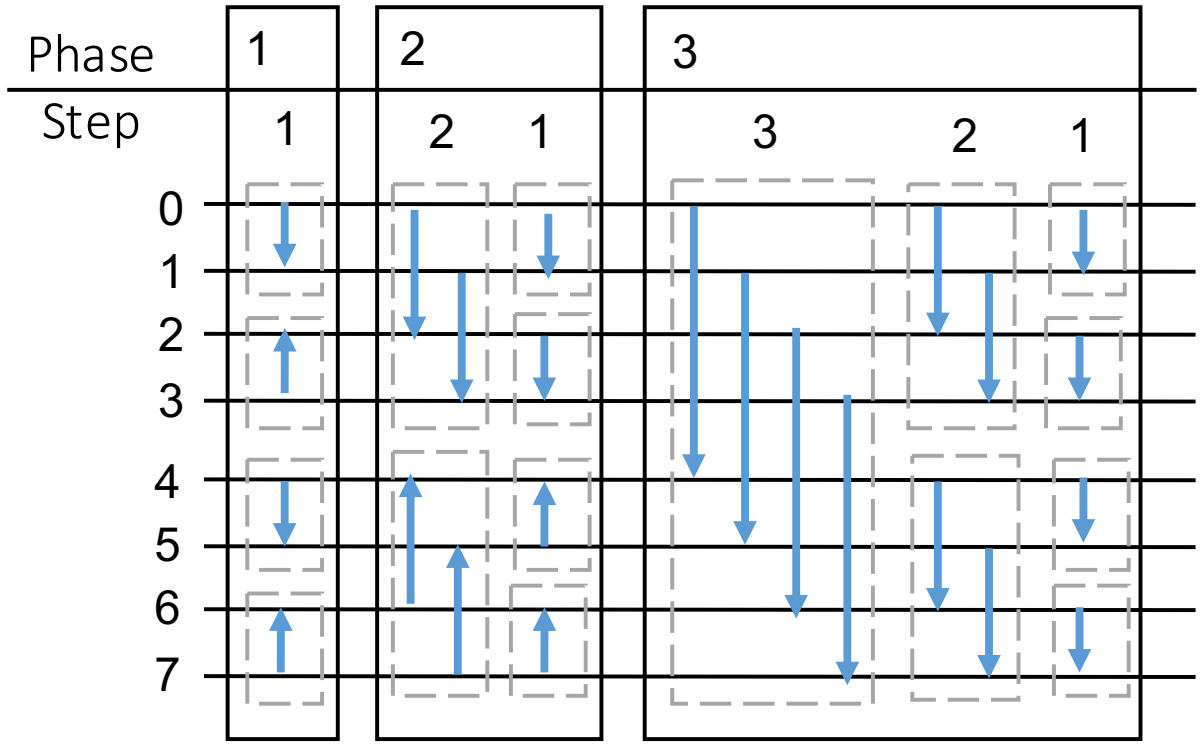
- Sequence properties
  - s_1 and s_2 are both bitonic
  - $\forall_x \forall_y \; x \in s_1, y \in s_2 , x < y$
- Apply recursively on s_1 and s_2 to produce a sorted sequence
- Works for any bitonic sequence, even if $|s_1| \neq |s_2|$

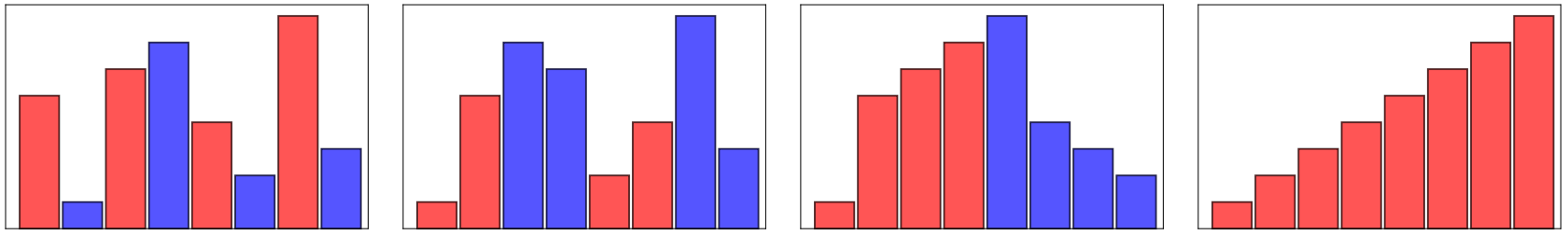
Bitonic Sort

- List of length 1 sorted by definition
- List of length 2 bitonic by definition

- Sort length 2 sequences ascending /descending using previous
- Neighboring sequences form bitonic sequences of length 4

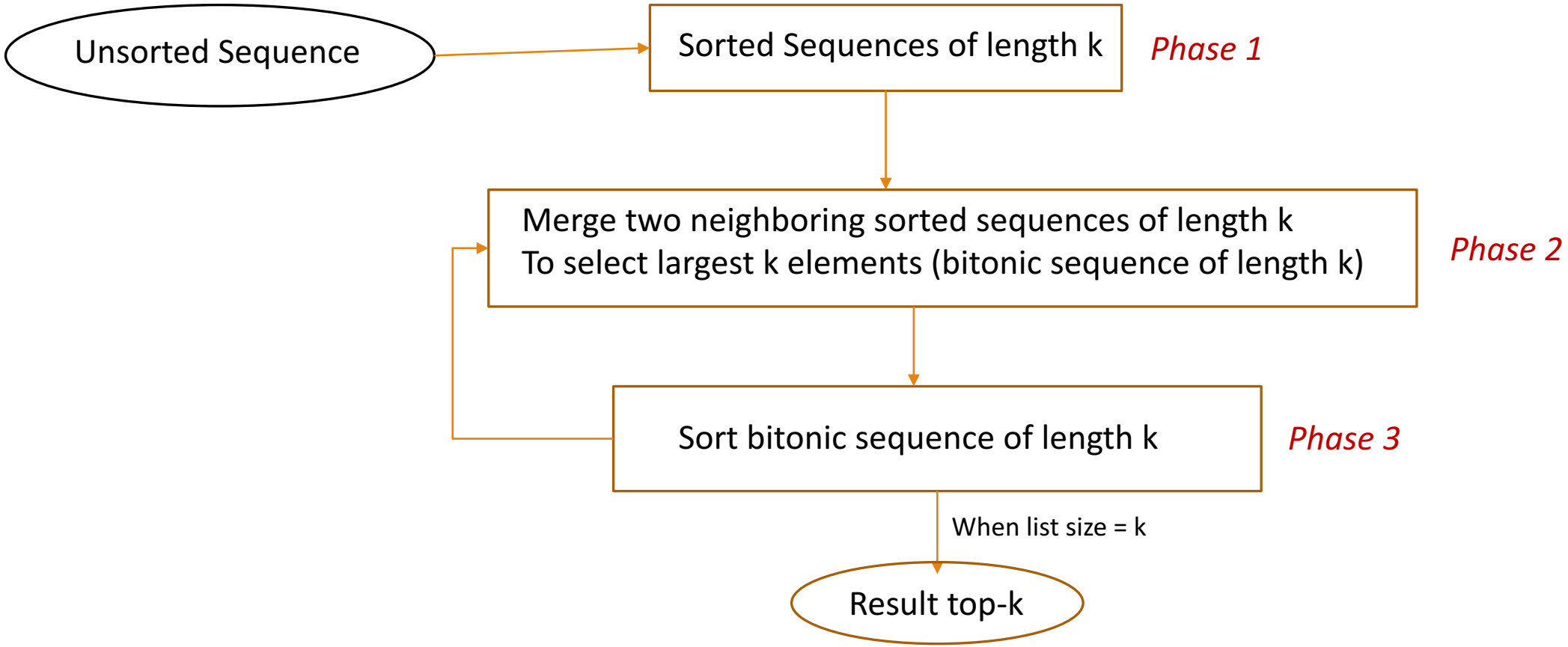
- Sort them ascending descending to create length 8 sequence
- …..
- Sort bitonic sequence of length n to form sorted sequence of length n



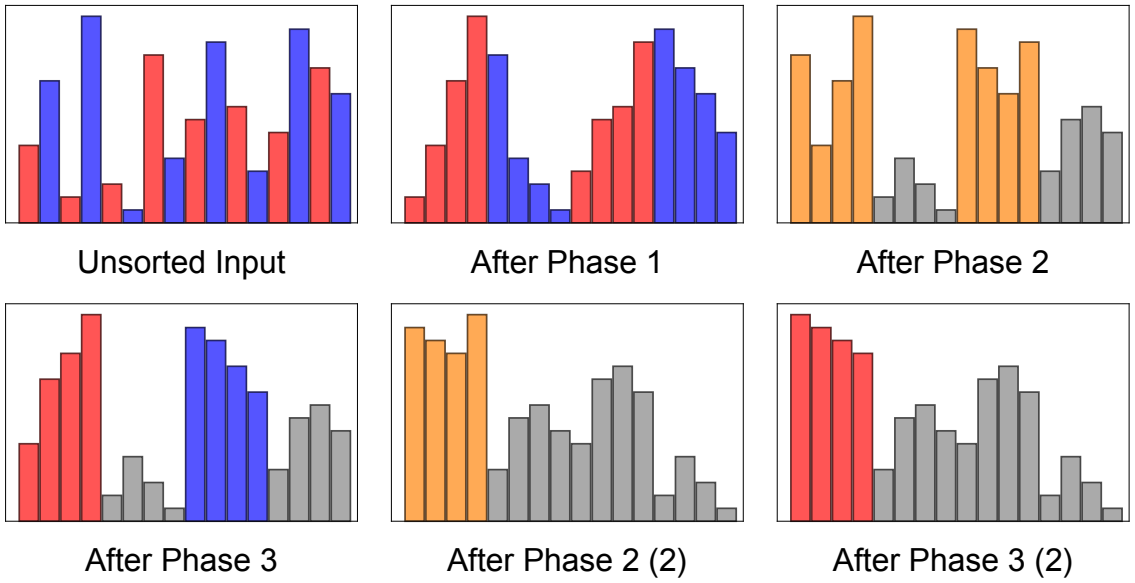


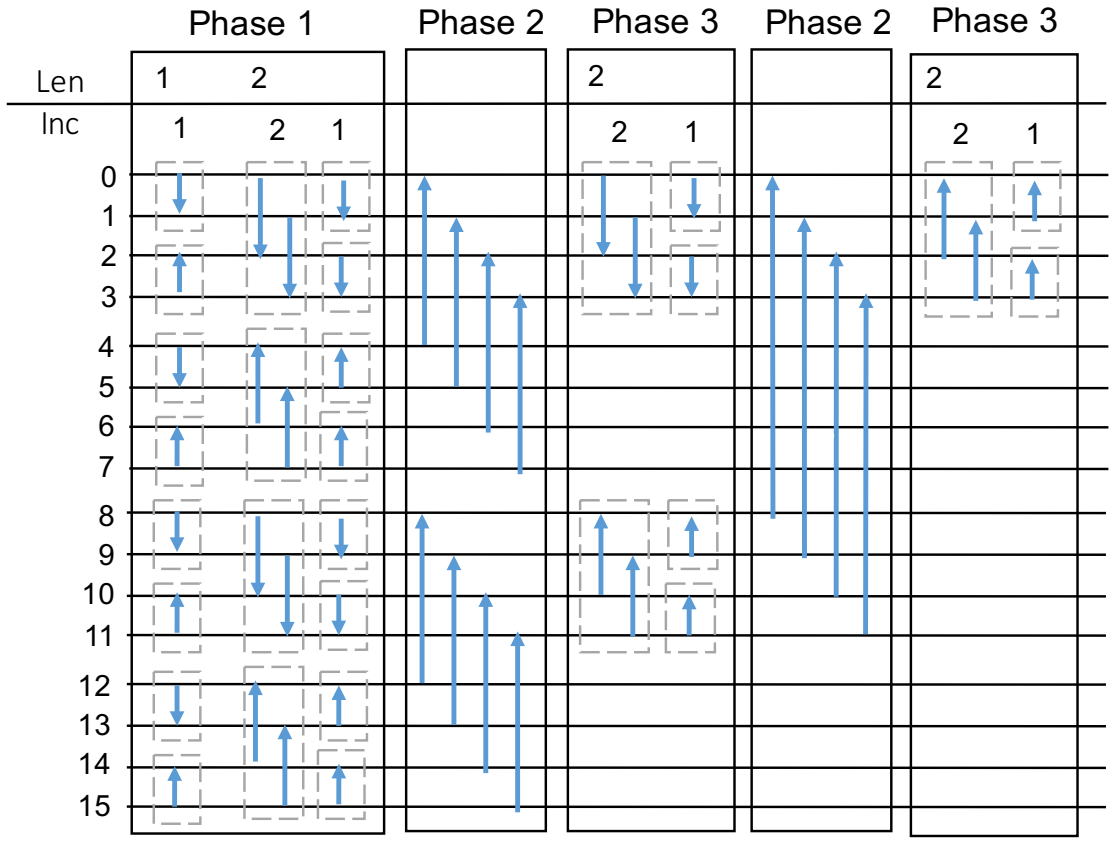
Unsorted Input | After Phase 1 | After Phase 2 | After Phase 3

Bitonic Top-K

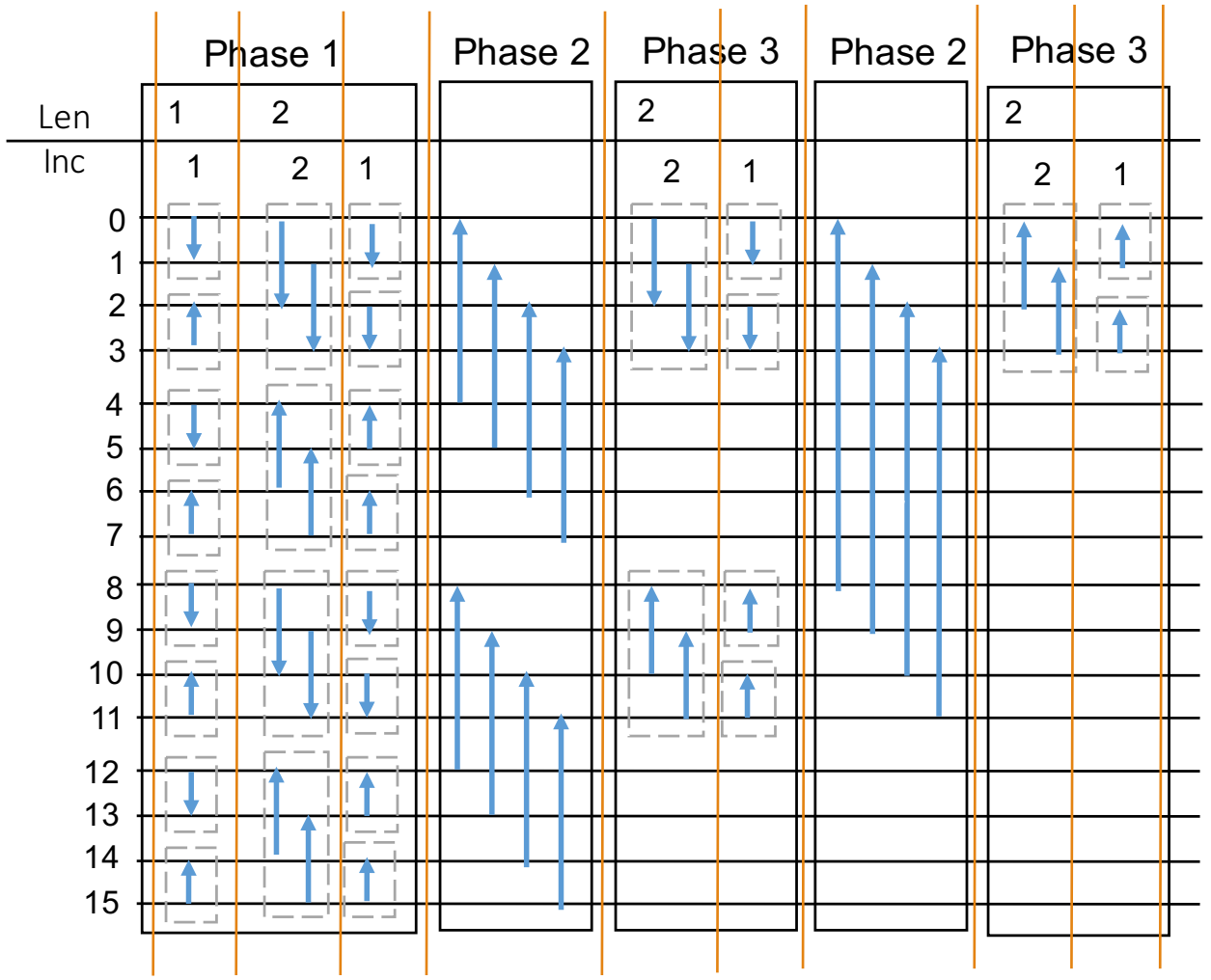


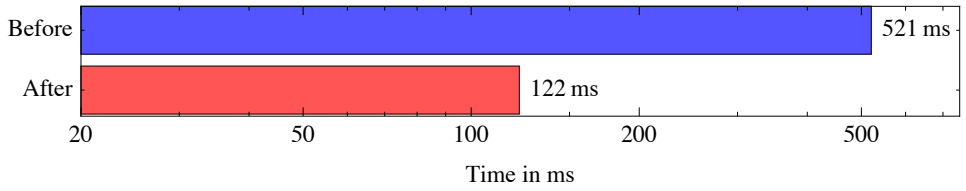
Bitonic Top-K Visualized



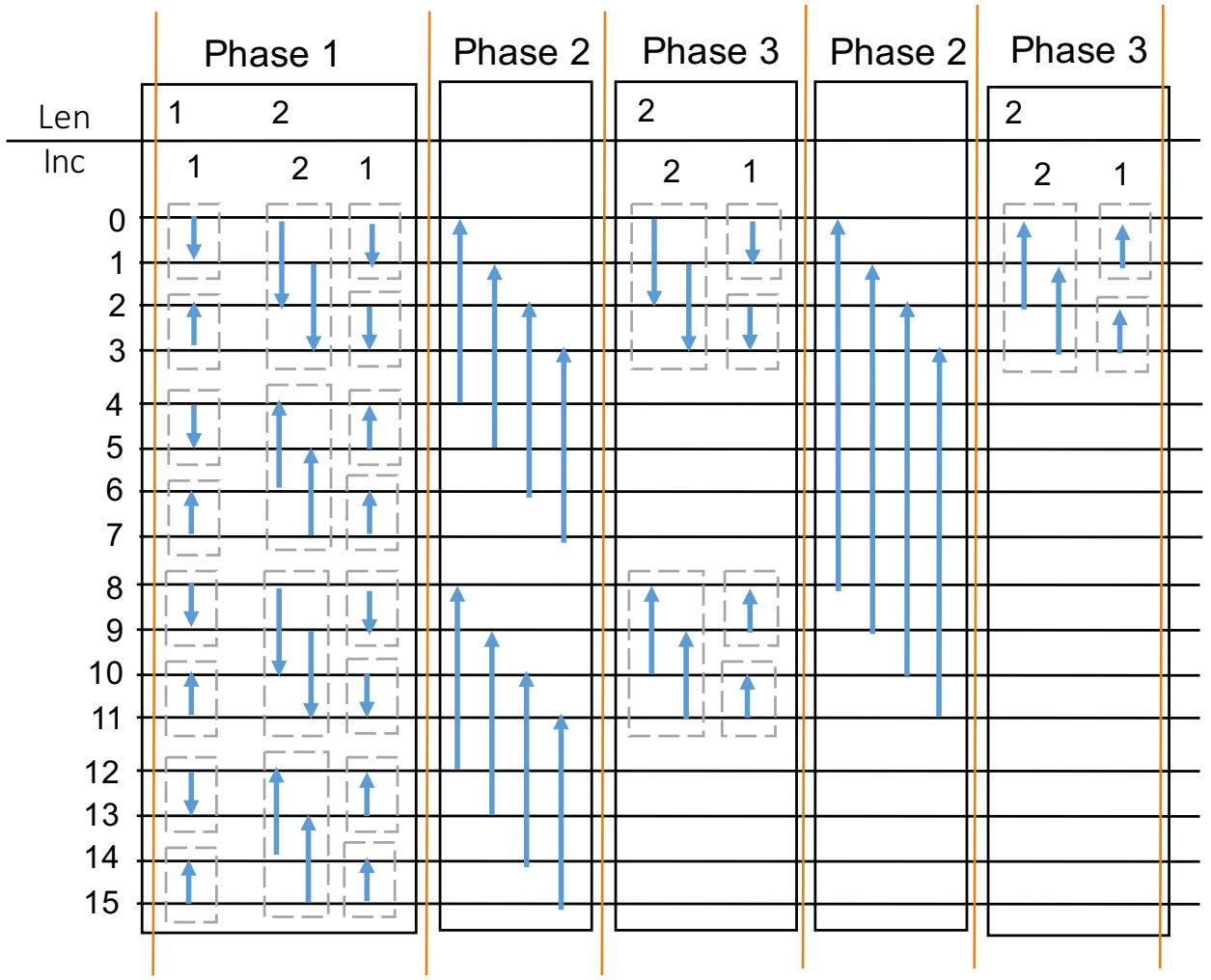


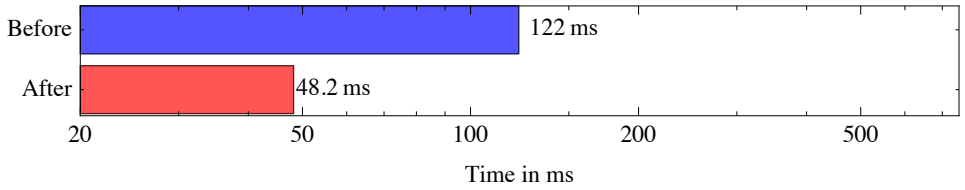
Optimization 1:
Using Shared Memory



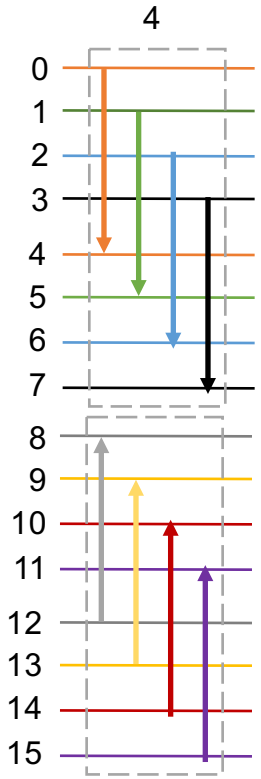


Optimization 2:
Combining Phases

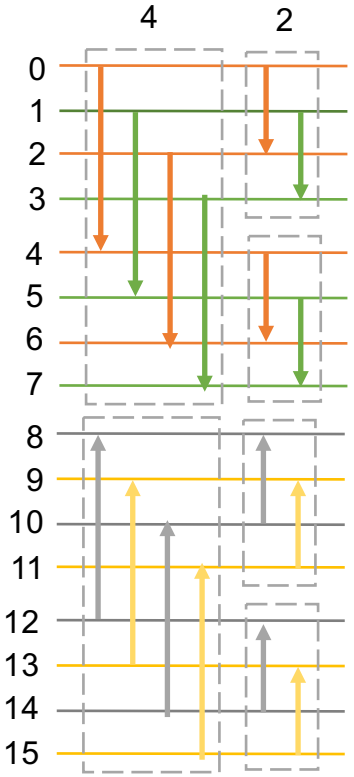




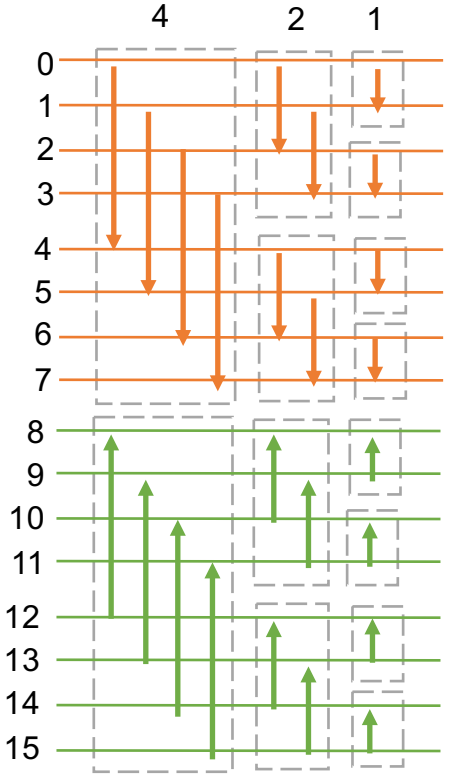
Optimization 3:
Combining Steps



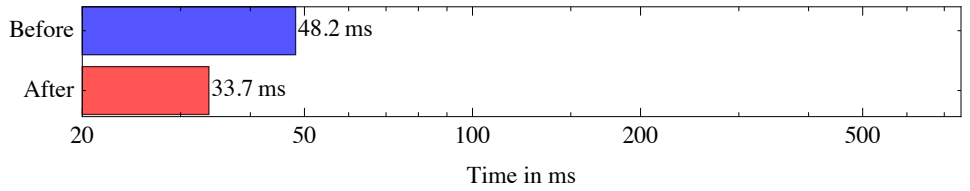
One step at a time



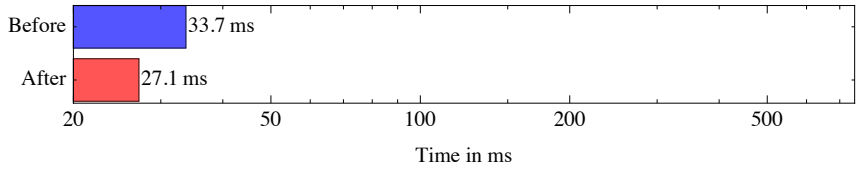
Two steps at a time



Three steps at a time



Optimization 4:
Read from Global > Process > Write to Shared

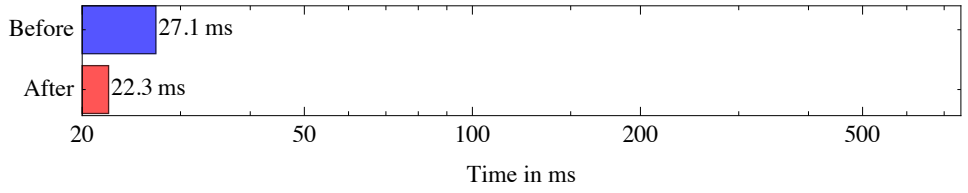


Optimization 5:
Padding

Reshape array of size n
(32 * n/32)
To
(33 * n/32)

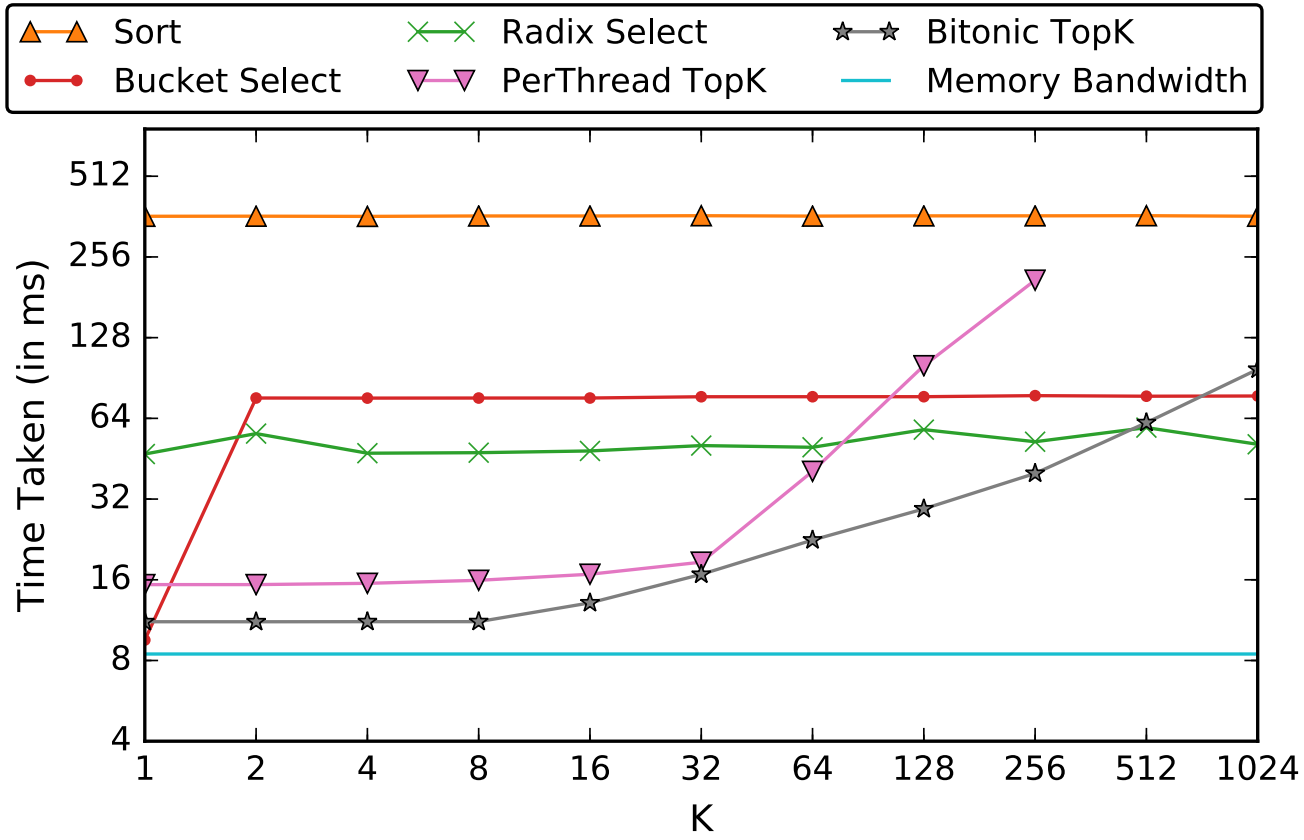
0								1								2								15							
4								5								6								16							
8								9								10								17							
12								13								14								18							
16								17								18								19							
20								21								22								23							
24								25								26								27							
28								29								30								31							

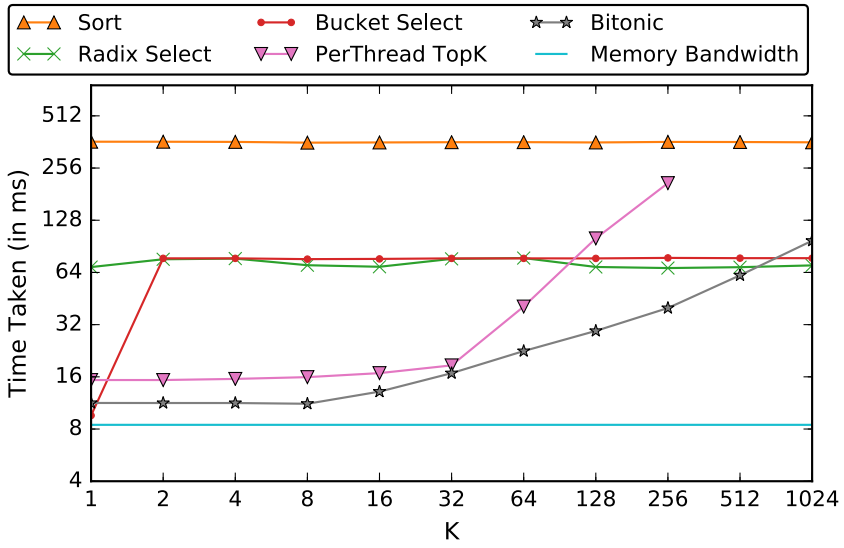
0								1								2								15							
X	4								5								6								16						
	X	8								9								10								17					
		X	12								13								14								18				
			X	16								17								18								19			
				X	20								21								22								23		
					X	24								25								26								27	
						X	28								29								30								31



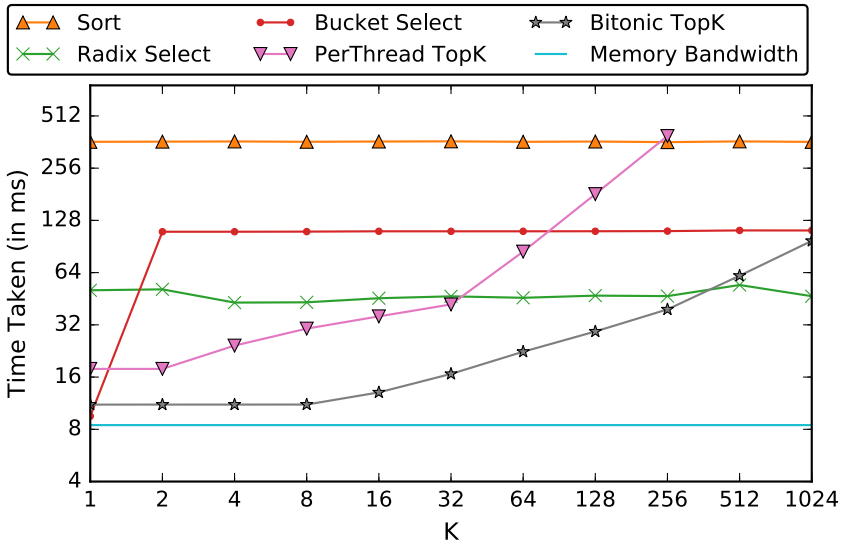
Experiments

For 2^29 (1/2 billion) floats from U(0,1)
On Nvidia Titan X Maxwell



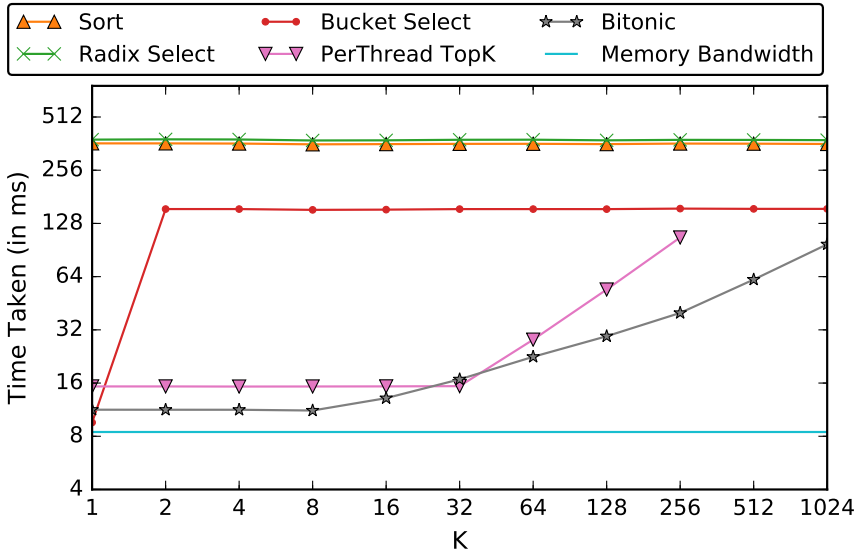


a) Large Floats U(0, 1million)

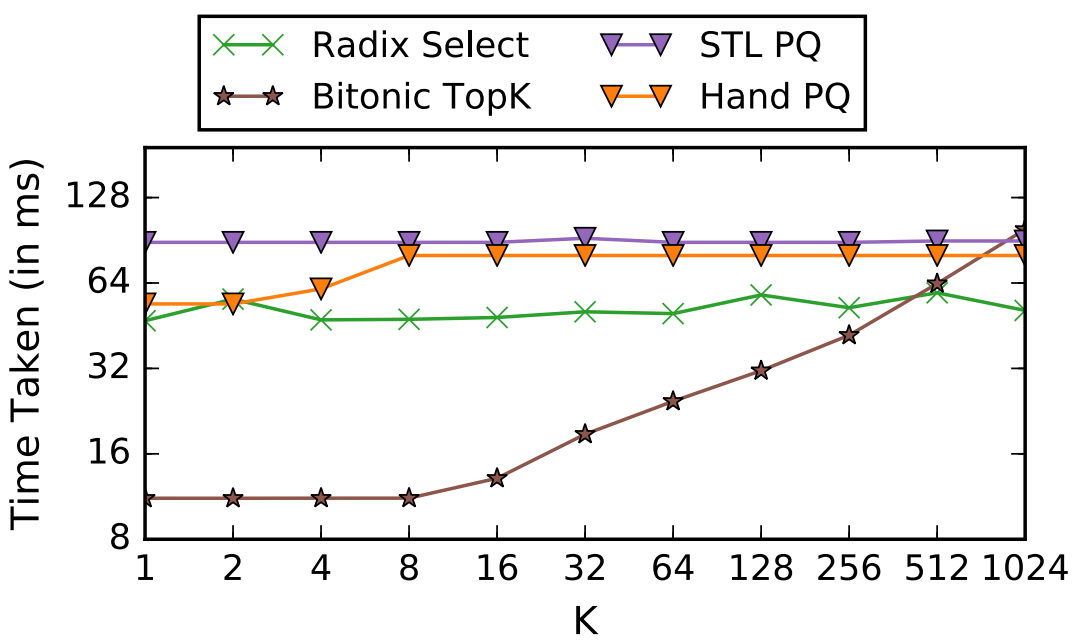


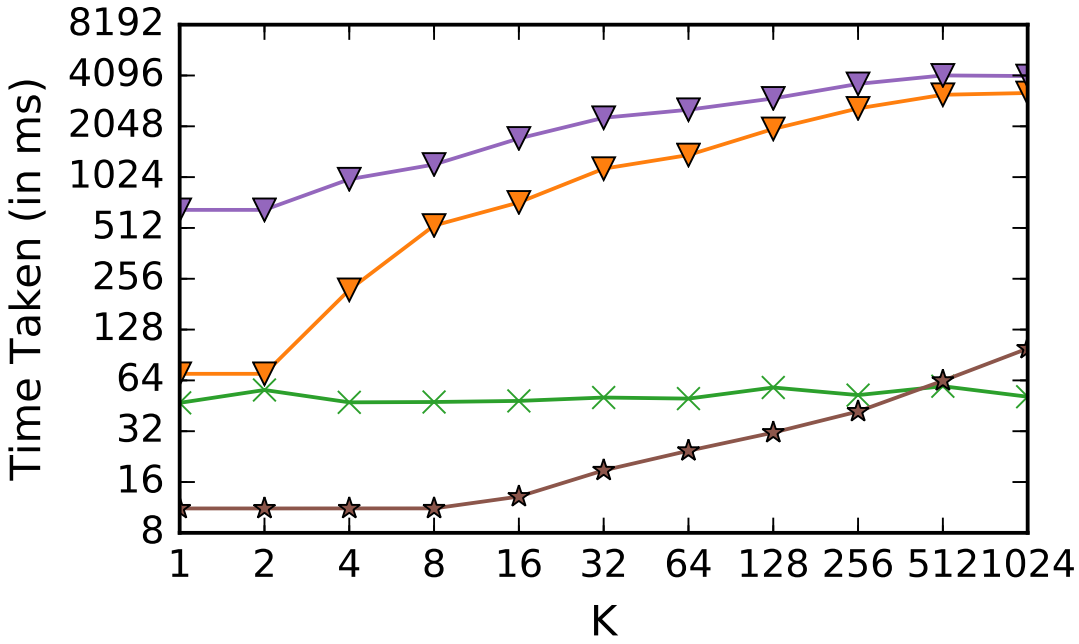
b) Increasing

With
Different Distributions



c) Bucket Killer





Comparing to CPU

Hybrid Databases

So-Far

Hardware-oblivious parallelism for in-memory column-stores (Ocelot)
VLDB 2013

The Yin and Yang of Processing Data Warehousing Queries on GPU
VLDB 2013

Concurrent Analytical Query Processing with GPUs
VLDB 2014

HippogriffDB: Balancing I/O and GPU Bandwidth
VLDB 2016

All use old hardware: The latest of these uses K20 – with 5GB DDR5 GPU memory ☹
Single GPU – used to run everything – CPU ~ idle
All of them compare against **MonetDB** ……

The Bottleneck: PCIe

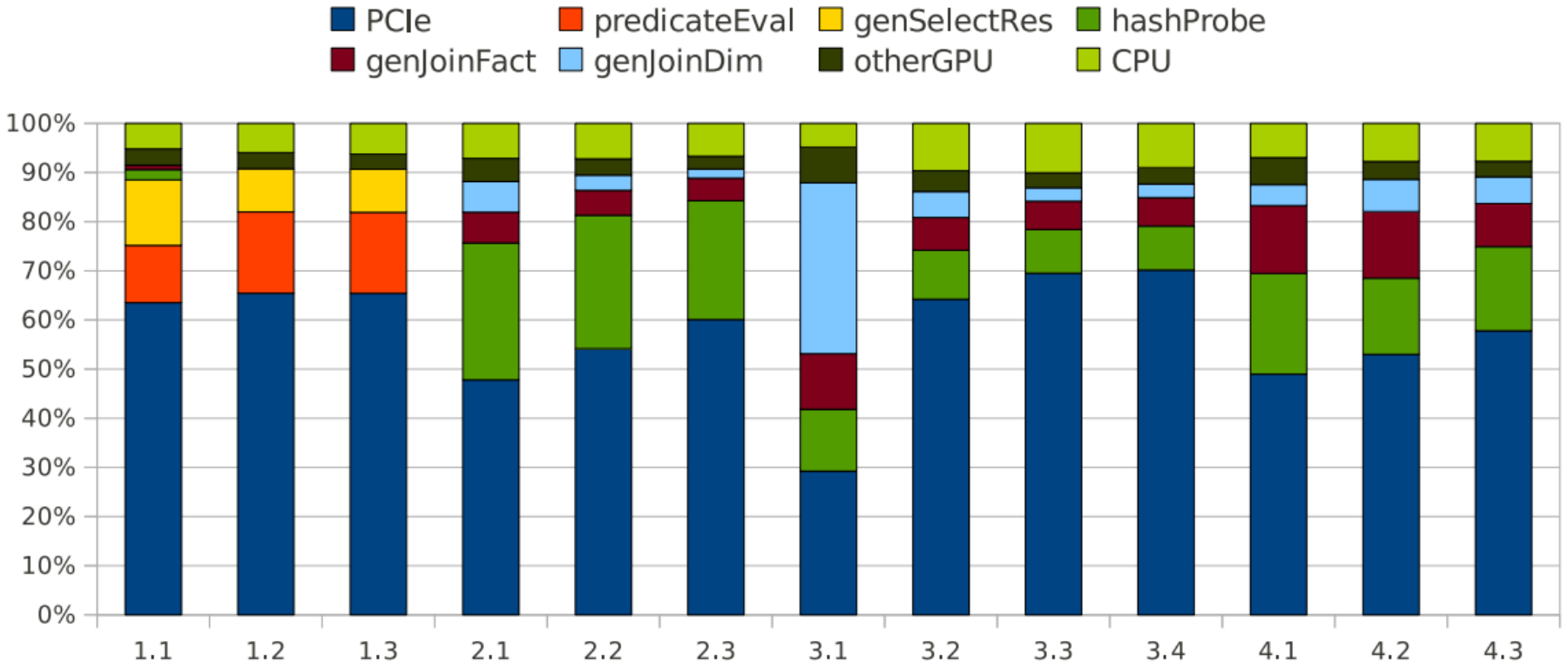
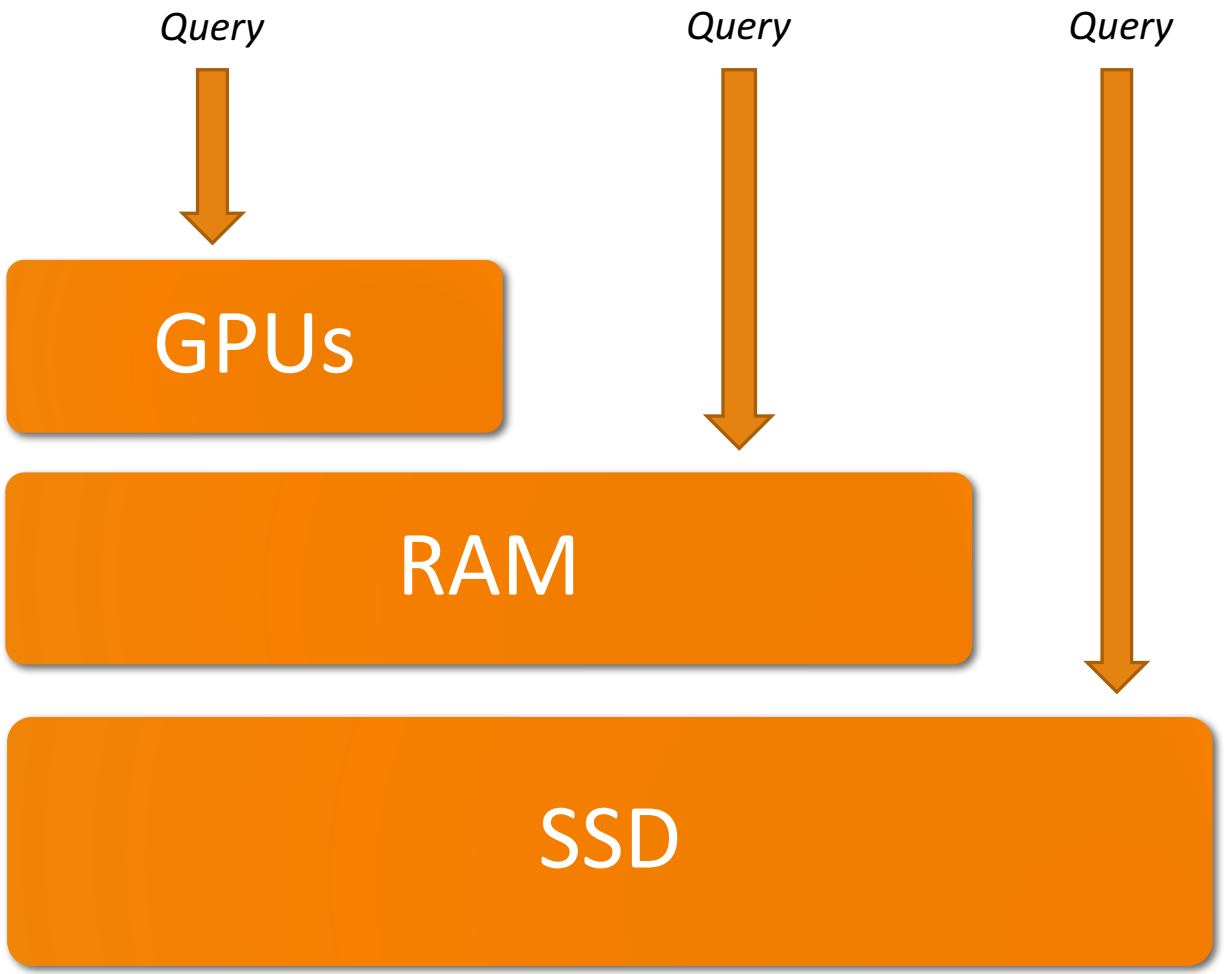


Figure 5: SSBM execution time breakdown

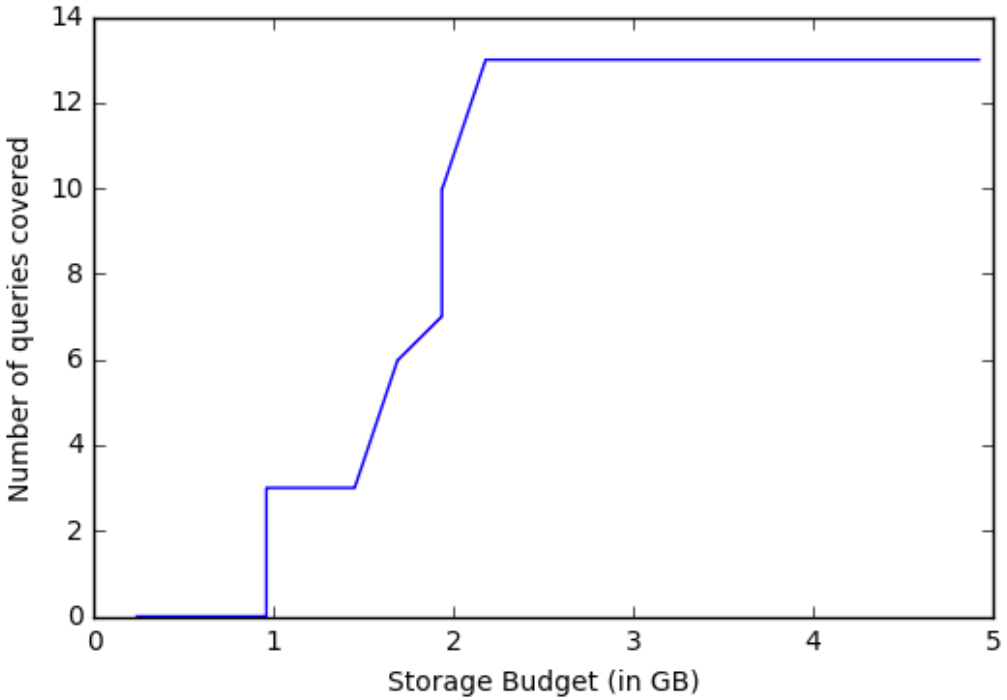
From the YinYang paper, VLDB 13



Maintain hot columns in GPU

Assuming column-orientated storage

All the data is in SSD
Most of the data is in RAM
Most frequently accessed columns are in GPU



For Star Schema Benchmark (SF=10)

GPU / CPU Co-Processing System

Based on cost models, we classify query into 3 categories

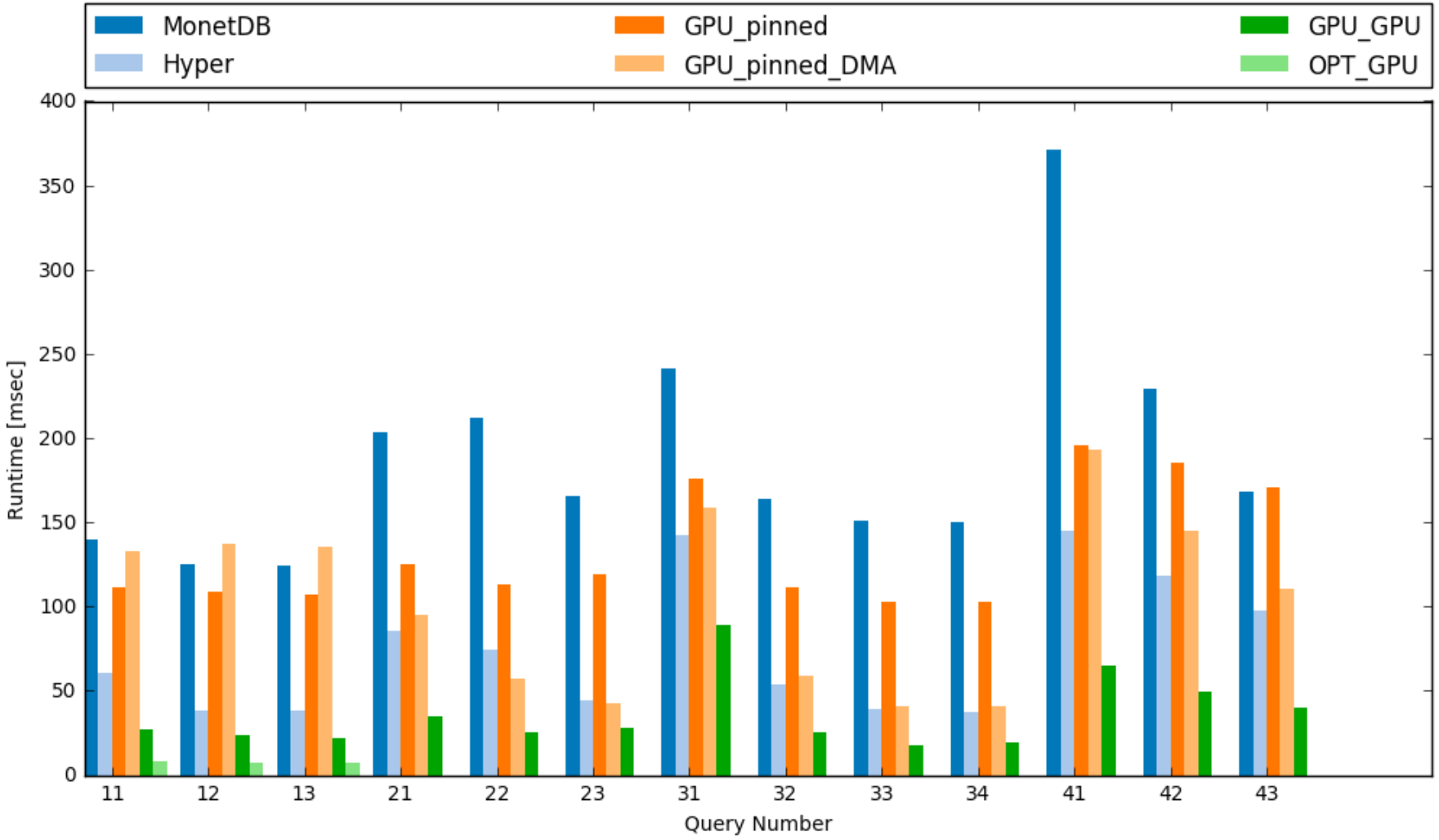
1) Run Xx faster on the GPU
2) Run < Xx faster on the GPU
3) Can’t run on the GPU

Using these, we build 3 queues:

1) GPU Queue
2) GPU or CPU Queue
3) CPU Queue

We can now use a work-stealing based approach to run the system. Each processing unit (PU) runs its own queue before stealing from other queues

Preliminary Results



Conclusion

GPU memory bandwidth makes it attractive for analytics as well

Moving data across PCI is slow

Using GPUs as a column cache is probably the way to go

Allows us to use both CPU and GPU resources

Expect 3-5x performance improvement over running on the CPU