Efficient Relational Database Management using Graphics Processors

Naga K. Govindaraju, Dinesh Manocha

http://gamma.cs.unc.edu/DB

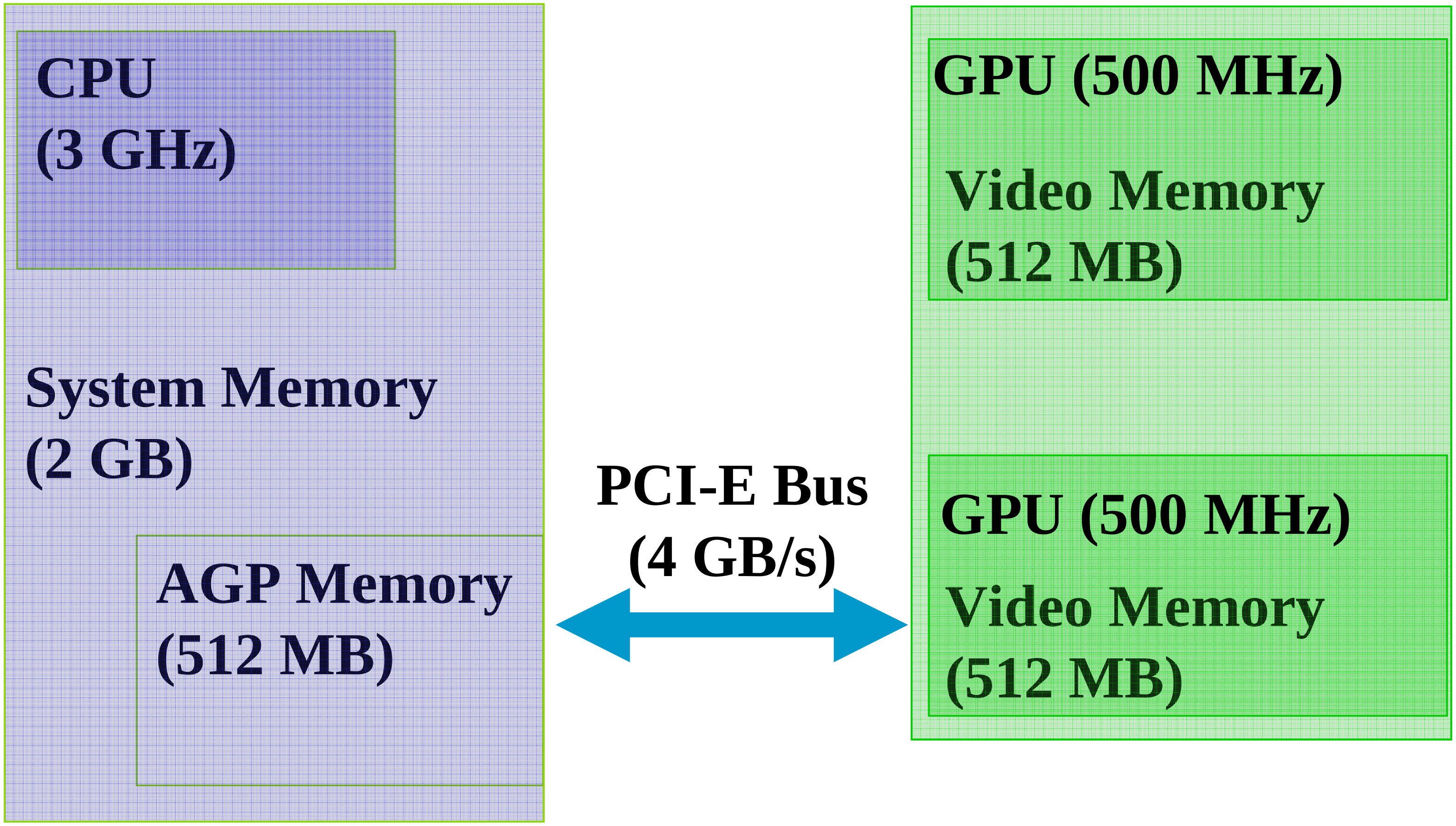
Goal

- Efficiently perform relational database operations using GPUs
    - Conjunctive selections
    - Aggregations
    - Semi-linear queries
    - Join queries
- Essential components

Motivation: Fast operations

- Increasing database sizes
- Faster processor speeds but low improvement in query execution time
  - Memory stalls
    - **50 – 90% due to cache misses [Ailamaki02]**
  - Branch mispredictions
  - Resource stalls
  - Ref: [Ailamaki99,01] [Boncz99] [Manegold00,02] [Meki00] [Shatdal94] [Rao99] [Zhou02]……

Fast Database Operations



CPU
(3 GHz)

System Memory
(2 GB)

AGP Memory
(512 MB)

PCI-E Bus
(4 GB/s)

GPU (500 MHz)

Video Memory
(512 MB)

GPU (500 MHz)

Video Memory
(512 MB)

Others

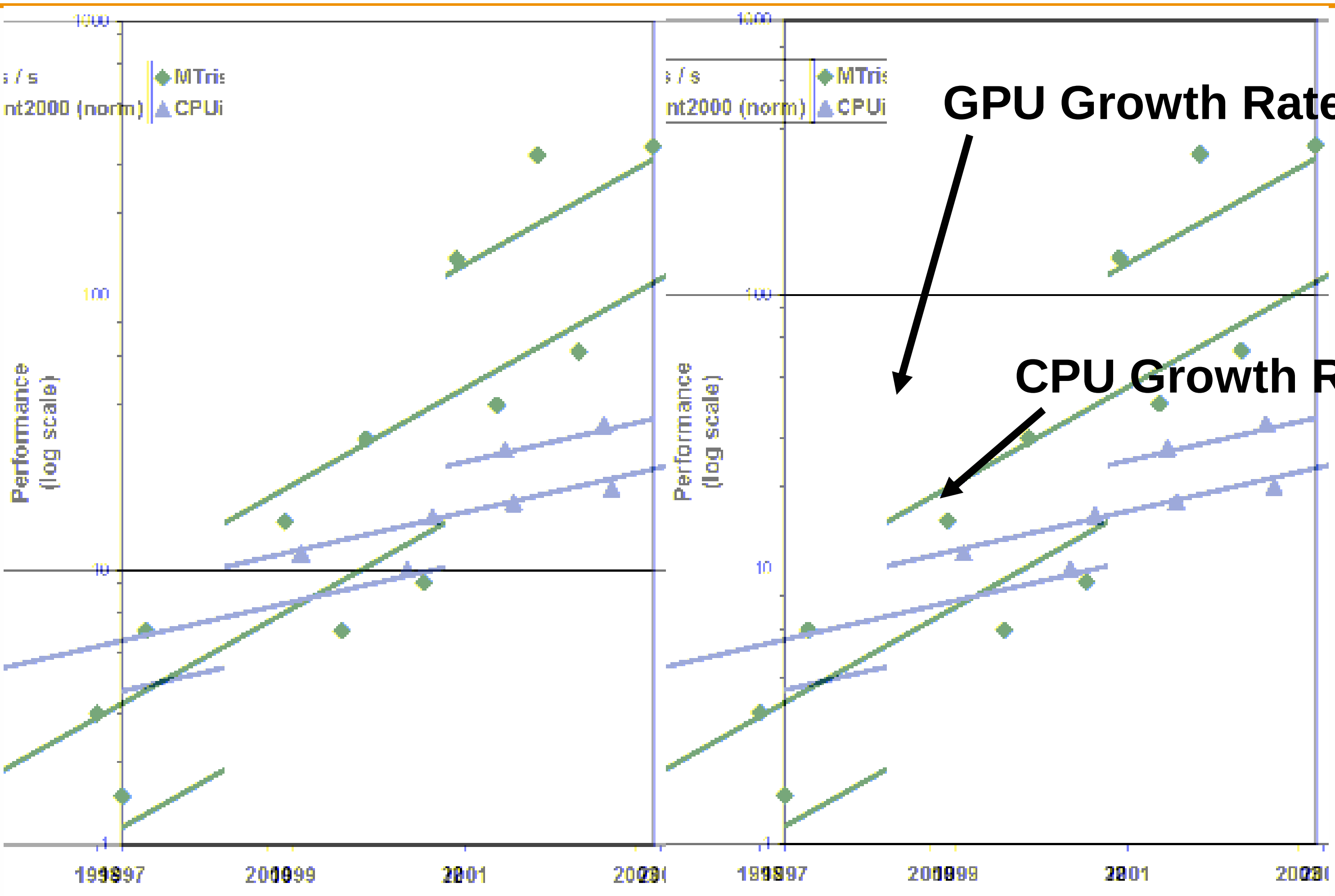
Ours

Characteristics of Database Operations

Database operations require

- High memory bandwidth to avoid stalls (35.2 GBps on GPUs)
- Efficient evaluation of comparisons for predication (64 comparisons per clock cycle)
- High computational power for aggregations and join queries (1.8 Tera Flops on Playstation 3 GPU)

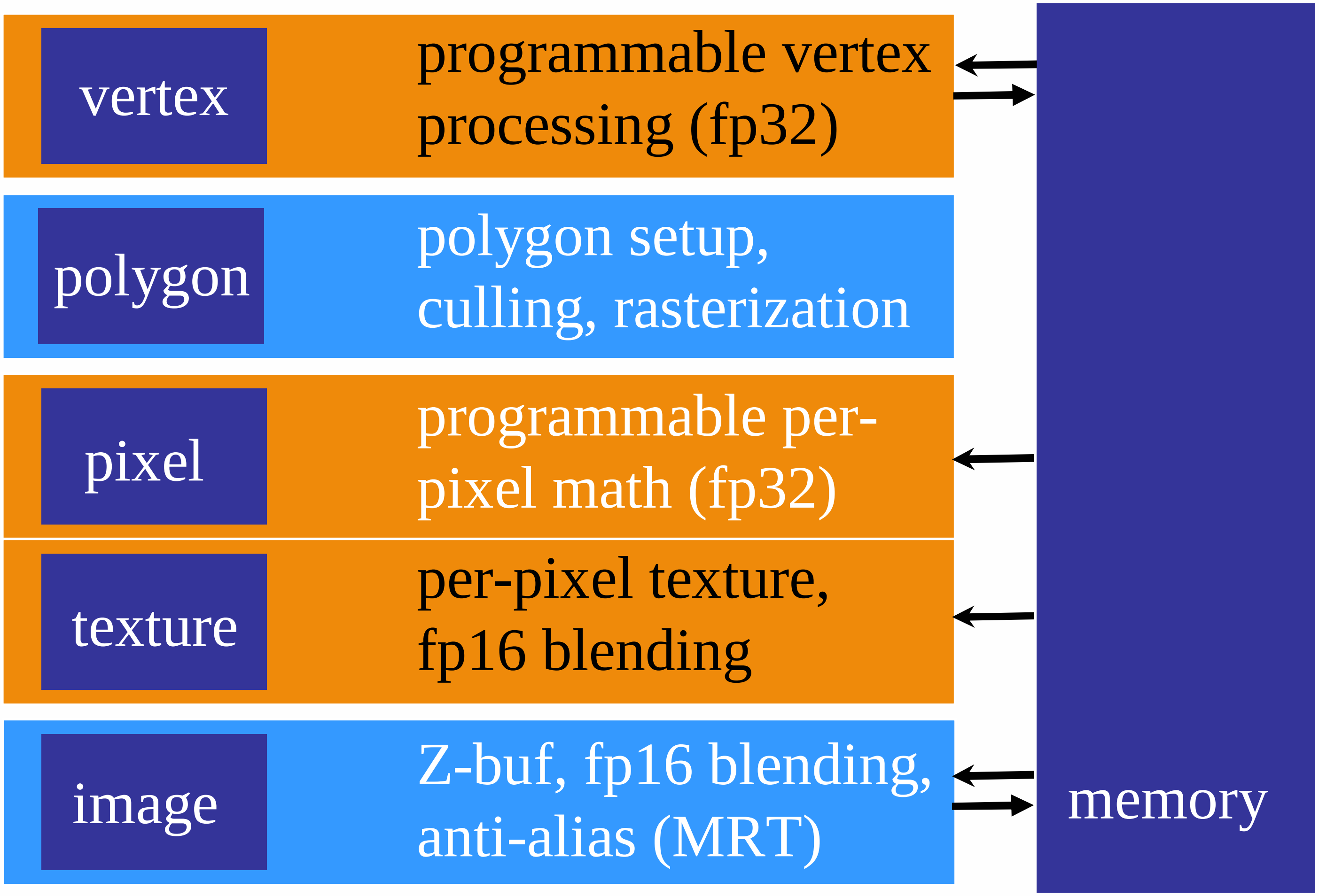
Exploiting Technology Moving Faster than Moore's Law



GPU Growth Rate

CPU Growth Rate

Graphics Pipeline



vertex – programmable vertex processing (fp32)

polygon – polygon setup, culling, rasterization

pixel – programmable per-pixel math (fp32)

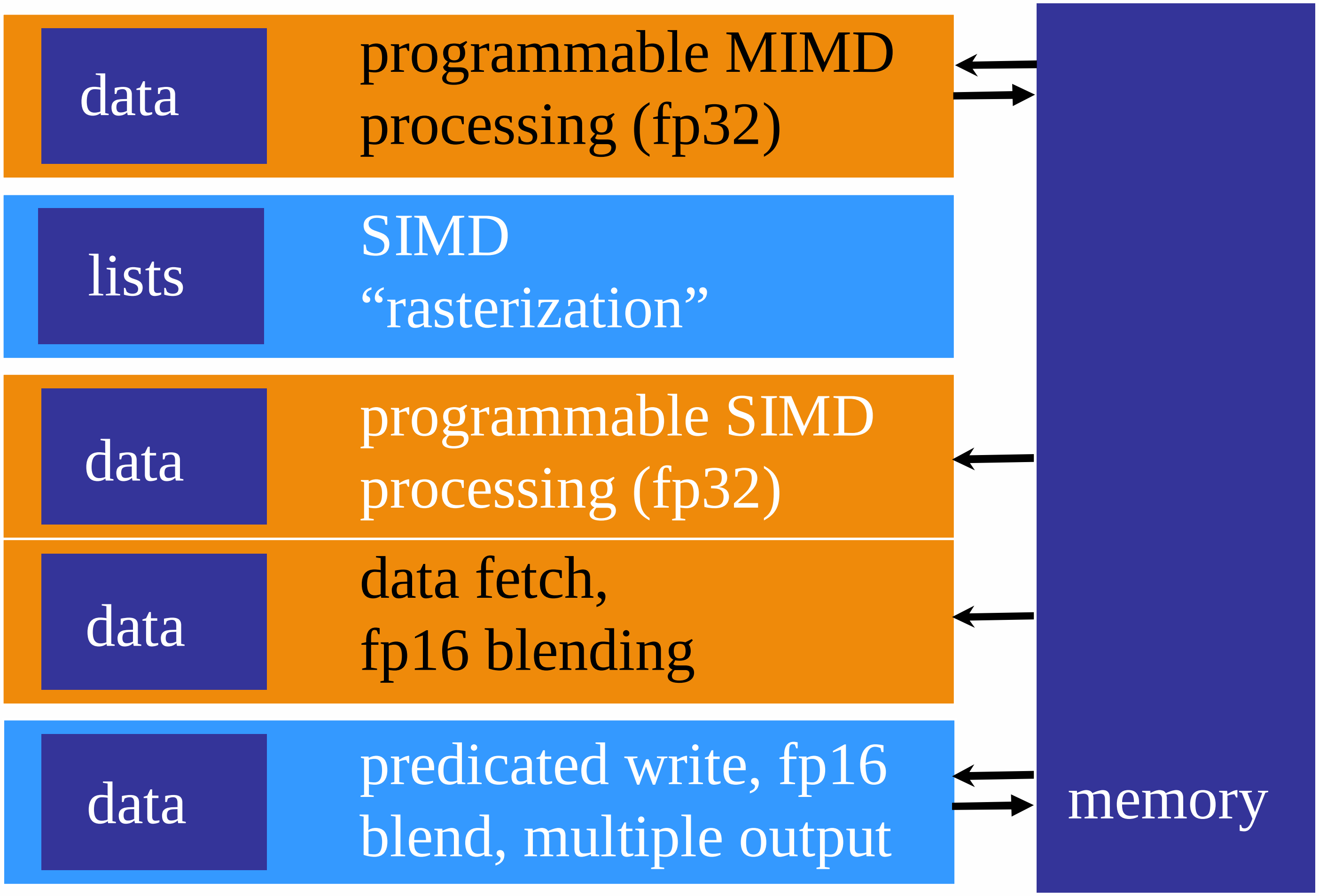
texture – per-pixel texture, fp16 blending

image – Z-buf, fp16 blending, anti-alias (MRT)

memory

NON-Graphics Pipeline

Courtesy:
David Kirk,
Chief Scientist,
NVIDIA



data

programmable MIMD processing (fp32)

lists

SIMD “rasterization”

data

programmable SIMD processing (fp32)

data

data fetch,
fp16 blending

data

predicated write, fp16 blend, multiple output

memory

Basic DB Operations

Basic SQL query

Select   A

From T

Where C

A= attributes or aggregations (SUM, COUNT, MAX etc)

T=relational table

C= Boolean Combination of Predicates (using operators AND, OR, NOT)

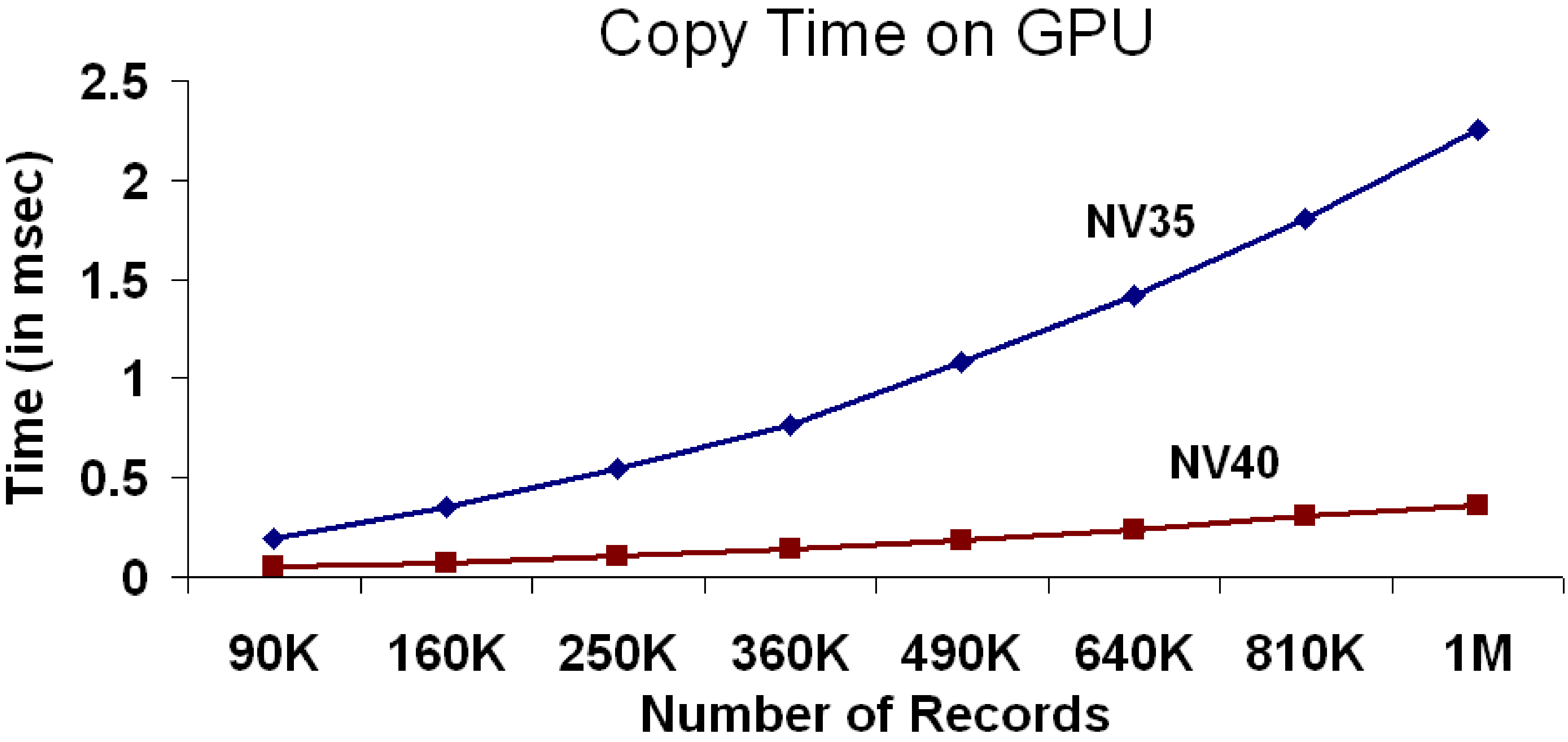
Database Operations

- Predicates
  - a_i op constant or a_i op a_j
  - op: <,>,<=,>=,!=, =, TRUE, FALSE
- Boolean combinations
  - Conjunctive Normal Form (CNF)
- Aggregations
  - COUNT, SUM, MAX, MEDIAN, AVG
- Join queries

Data Representation

- Attribute values a_i are stored in 2D textures on the GPU
- A fragment program is used to copy attributes to the depth buffer

Copy Time to the Depth Buffer



Predicate Evaluation

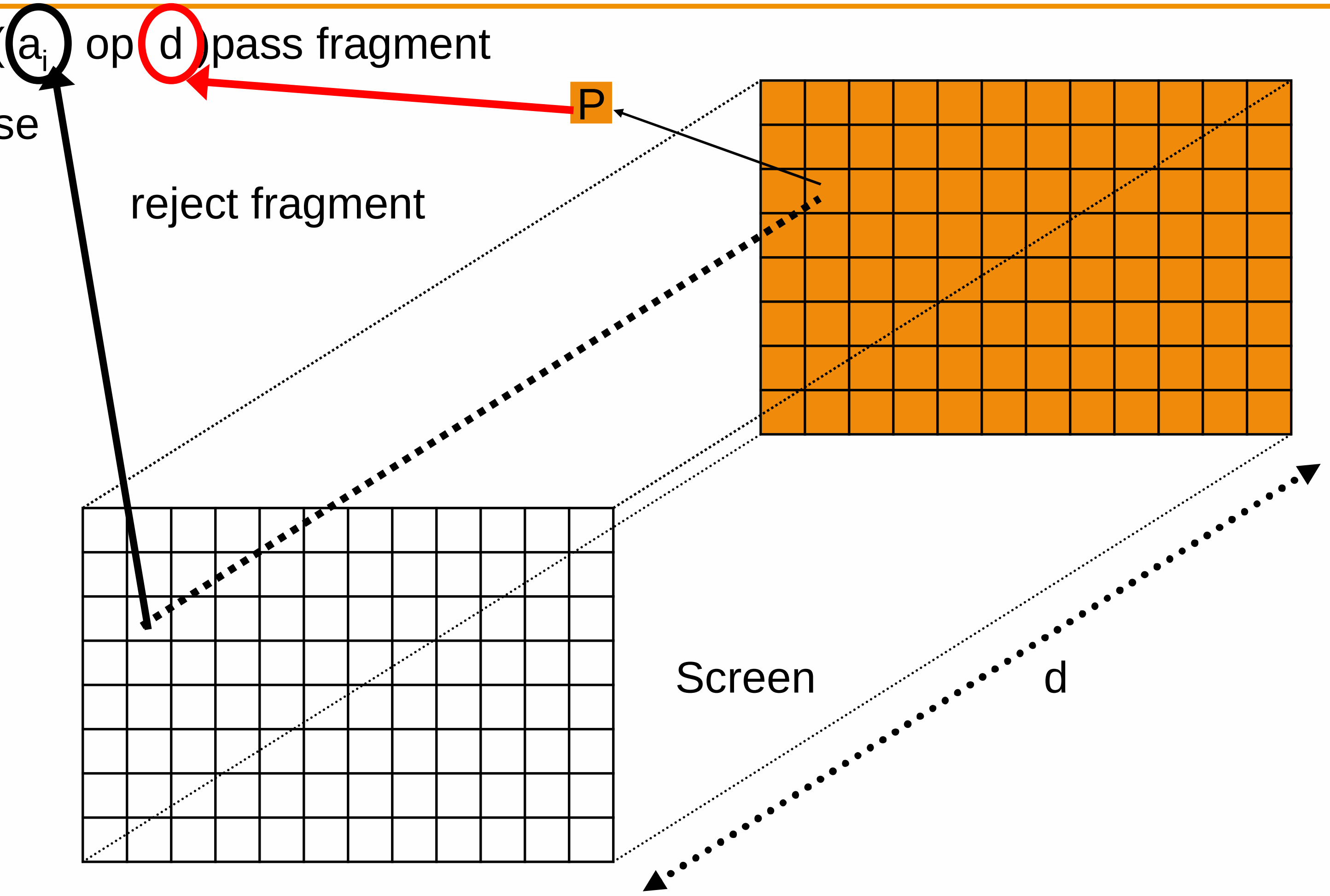
- a_i op constant (d)
  - Copy the attribute values a_i into depth buffer
  - Specify the comparison operation used in the depth test
  - Draw a screen filling quad at depth d and perform the depth test

a_i op d

If (a_i op d) pass fragment

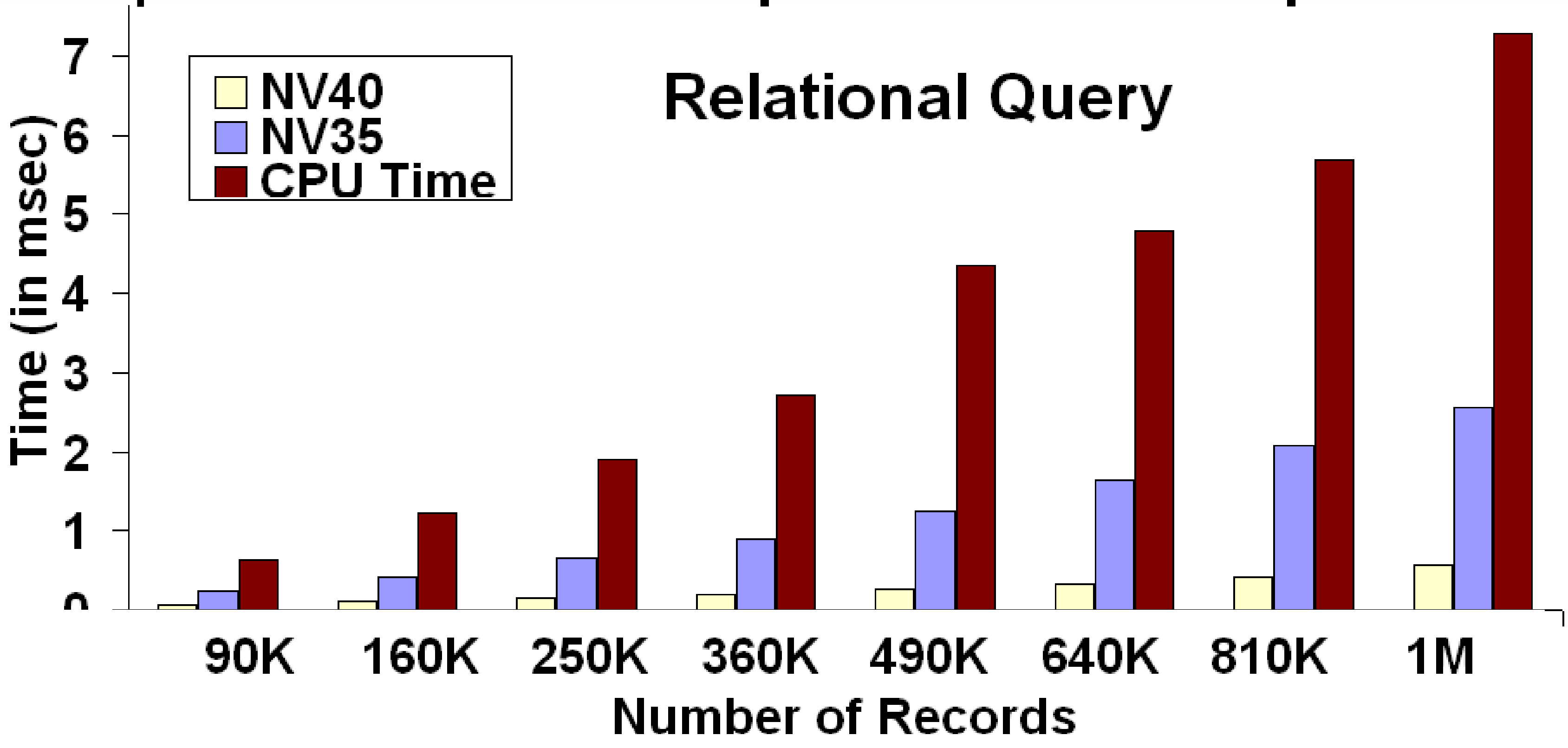
Else

reject fragment



Predicate Evaluation

CPU implementation — Intel compiler 7.1 with SIMD optimizations



GPU is nearly 20 times faster than 2.8 GHz Xeon

Predicate Evaluation

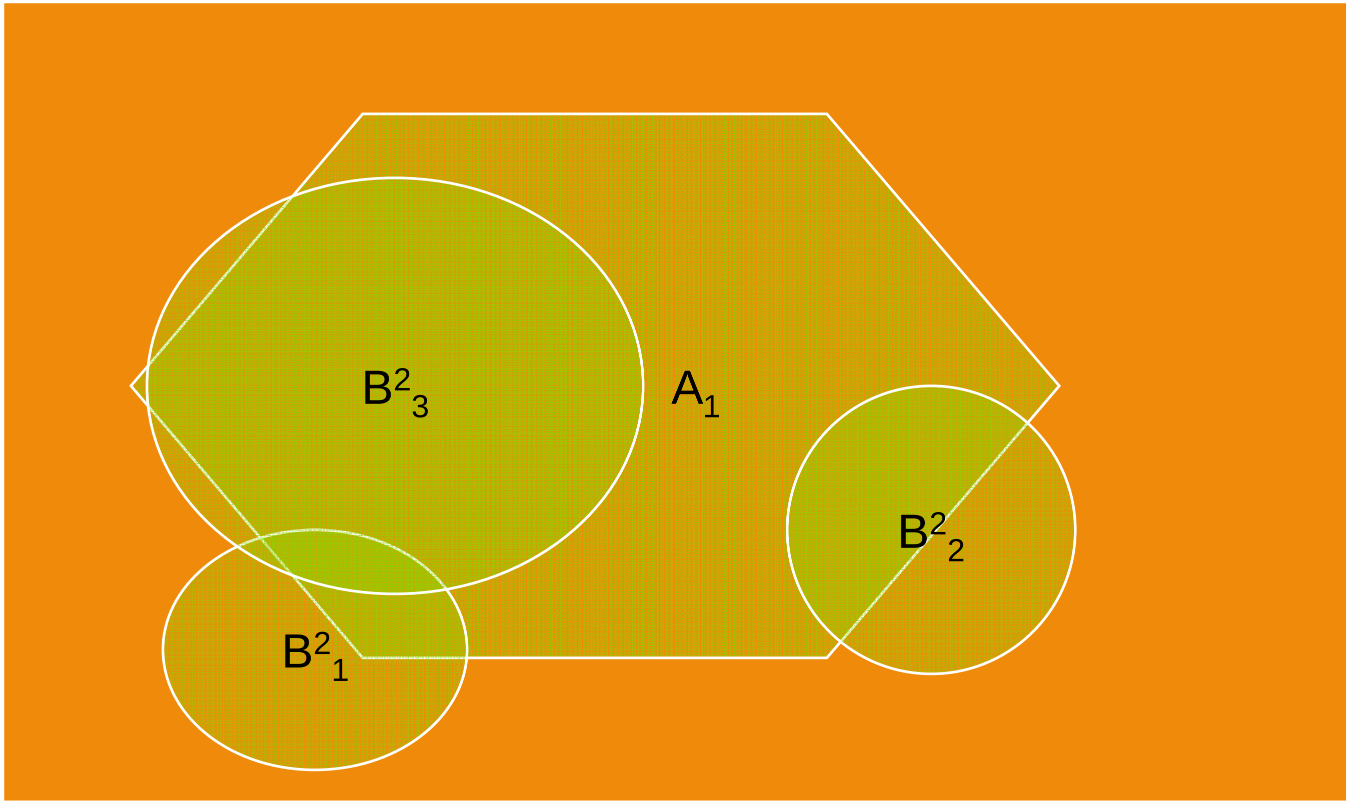
- a_i op a_j
  - Equivalent to ($a_i - a_j$) op 0

Boolean Combination

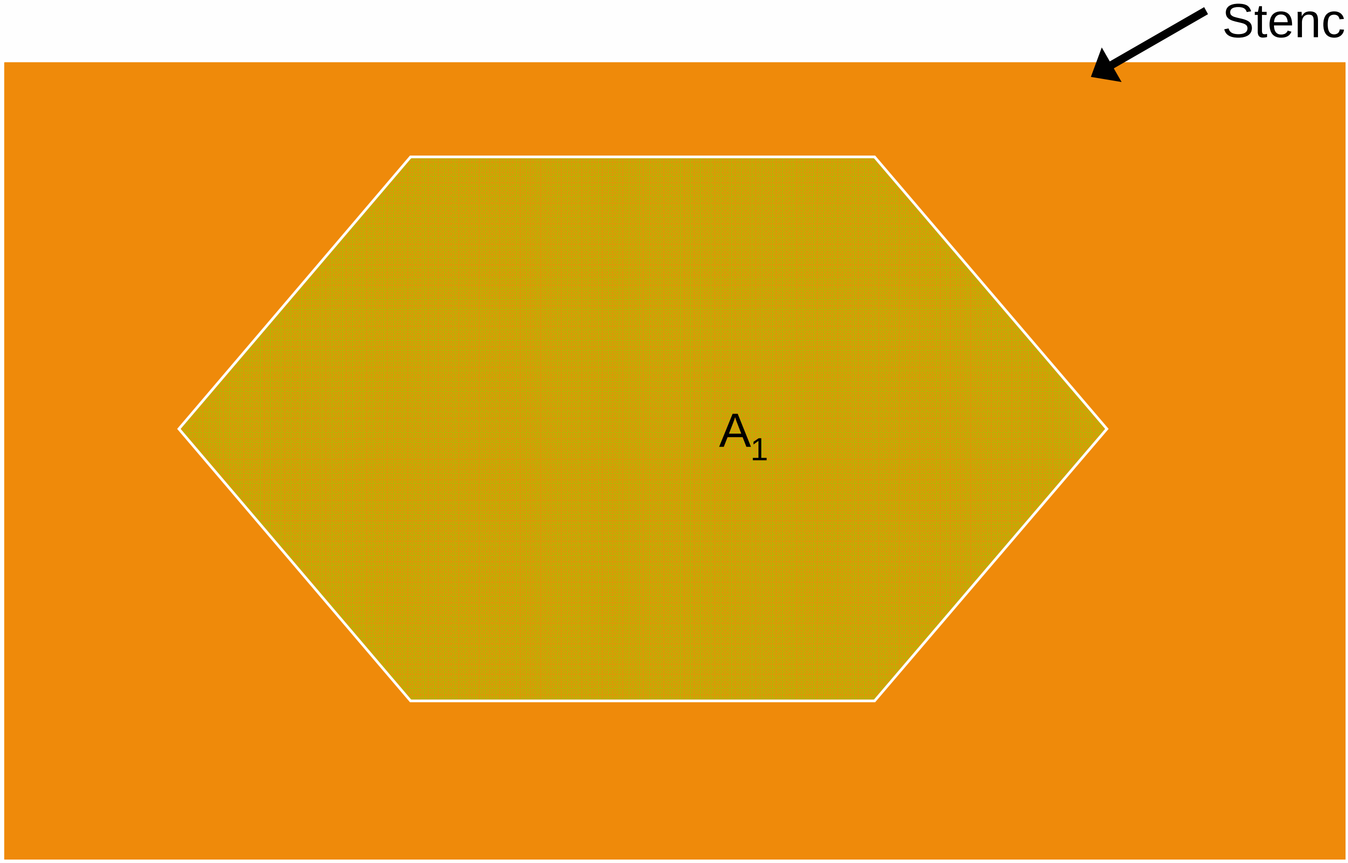
- CNF:
  - $(A_1 \text{ AND } A_2 \text{ AND } \ldots \text{ AND } A_k)$ where $A_i = (B^i_1 \text{ OR } B^i_2 \text{ OR } \ldots \text{ OR } B^i_{mi})$
- Performed using stencil test recursively
  - $C_1 = (\text{TRUE AND } A_1) = A_1$
  - $C_i = (A_1 \text{ AND } A_2 \text{ AND } \ldots \text{ AND } A_i) = (C_{i-1} \text{ AND } A_i)$
- Different stencil values are used to code the outcome of C_i
  - Positive stencil values — pass predicate evaluation
  - Zero — fail predicate evaluation

A_1 AND A_2

$$A_2 = (B^2{}_1 \text{ OR } B^2{}_2 \text{ OR } B^2{}_3\ )$$

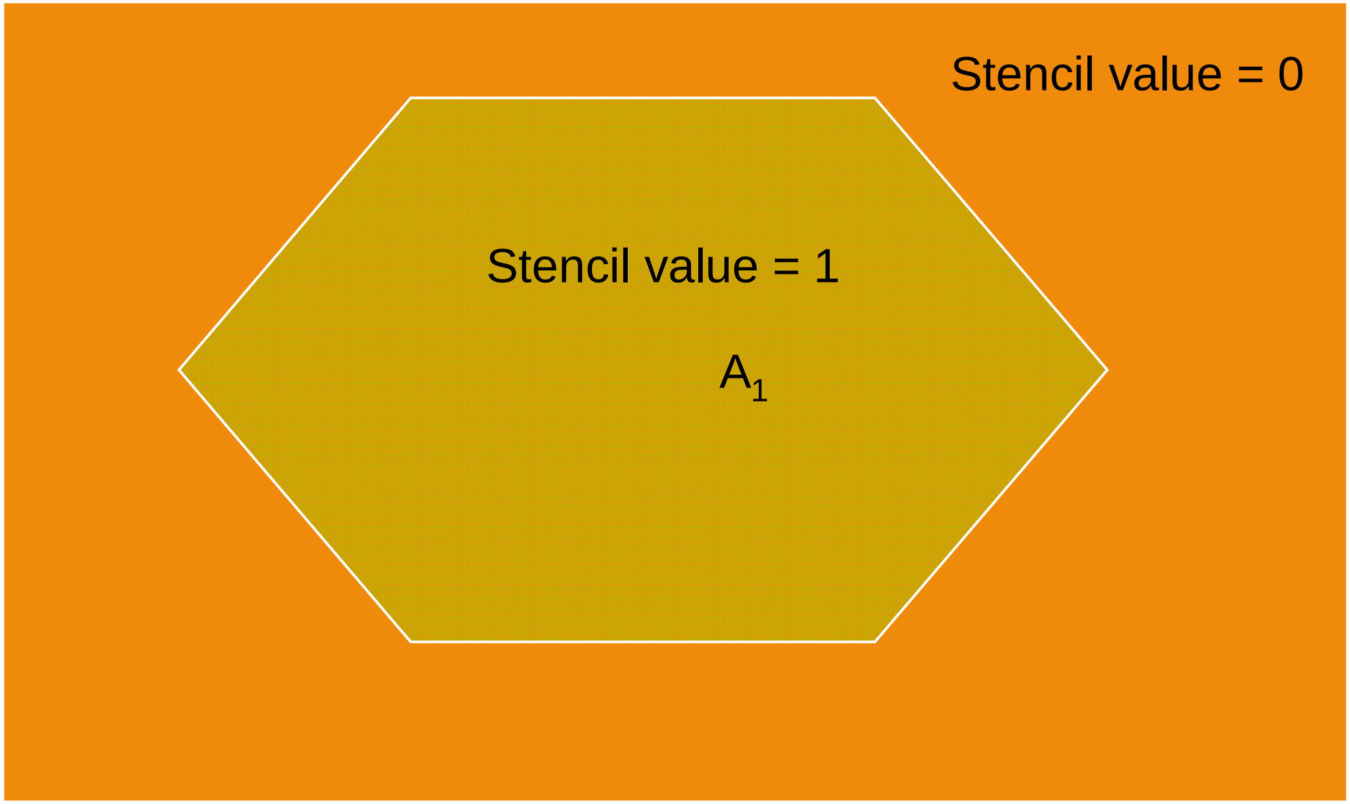


A_1 AND A_2

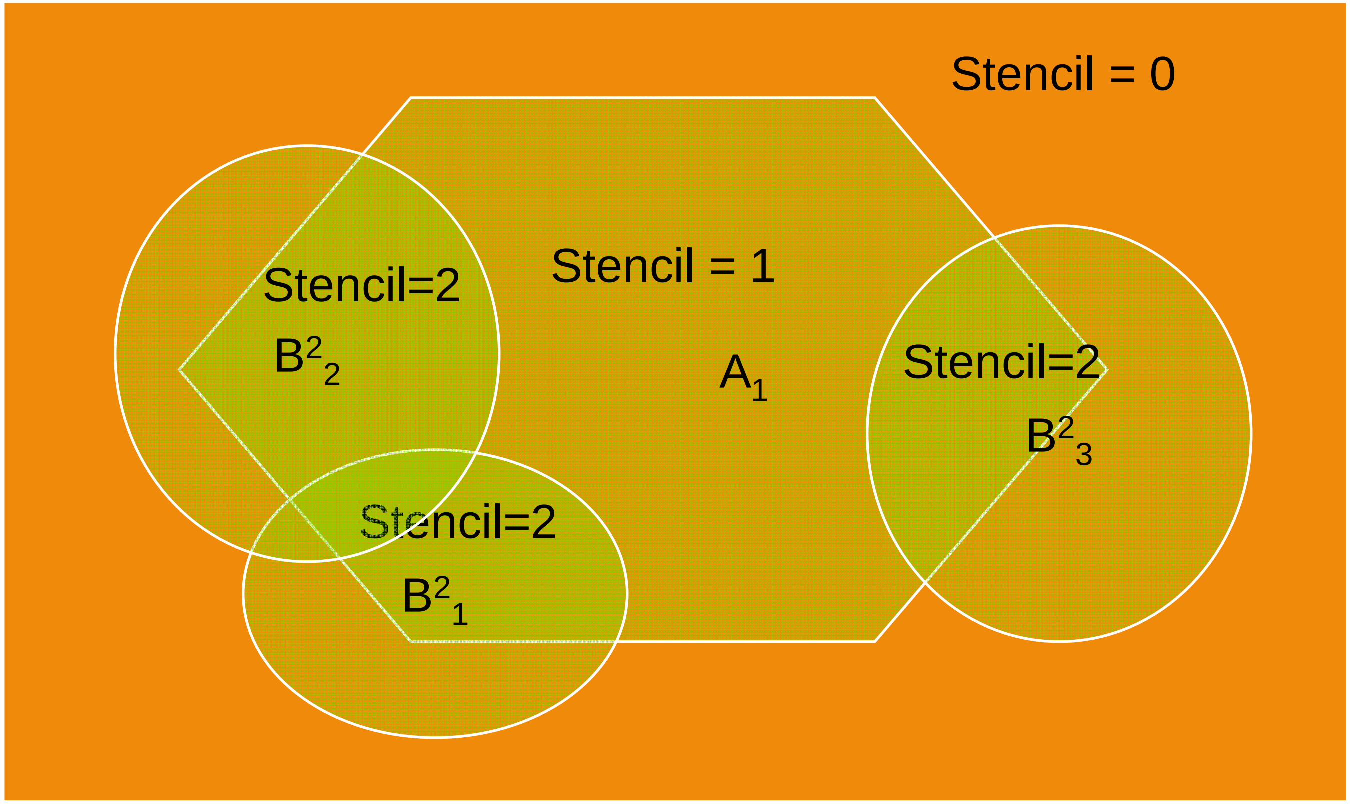


A_1 AND A_2

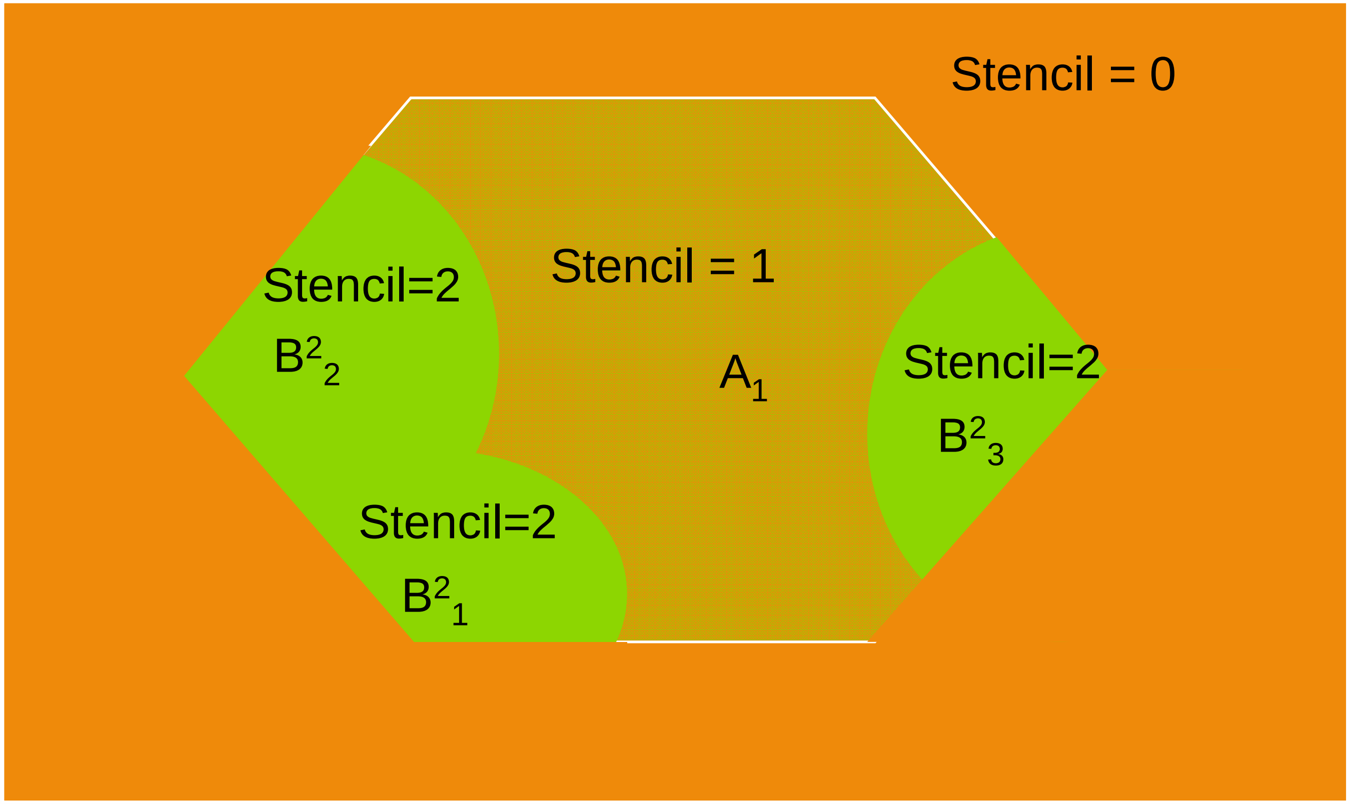
TRUE AND A_1



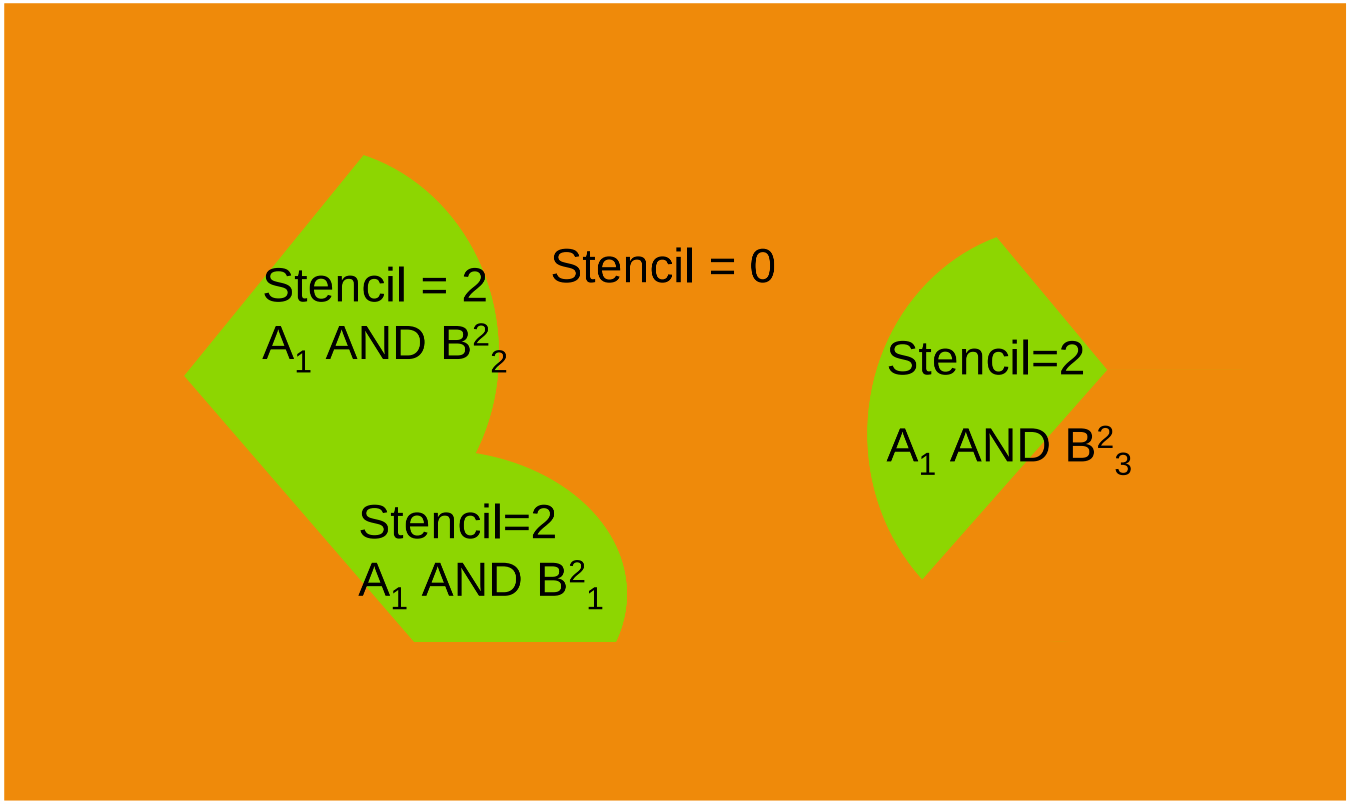
A_1 AND A_2


Stencil = 0
Stencil=2
B^2_2
Stencil = 1
A_1
Stencil=2
B^2_3
Stencil=2
B^2_1

A_1 AND A_2


Stencil = 0
Stencil = 1
A_1
Stencil=2
B^2_2
Stencil=2
B^2_3
Stencil=2
B^2_1

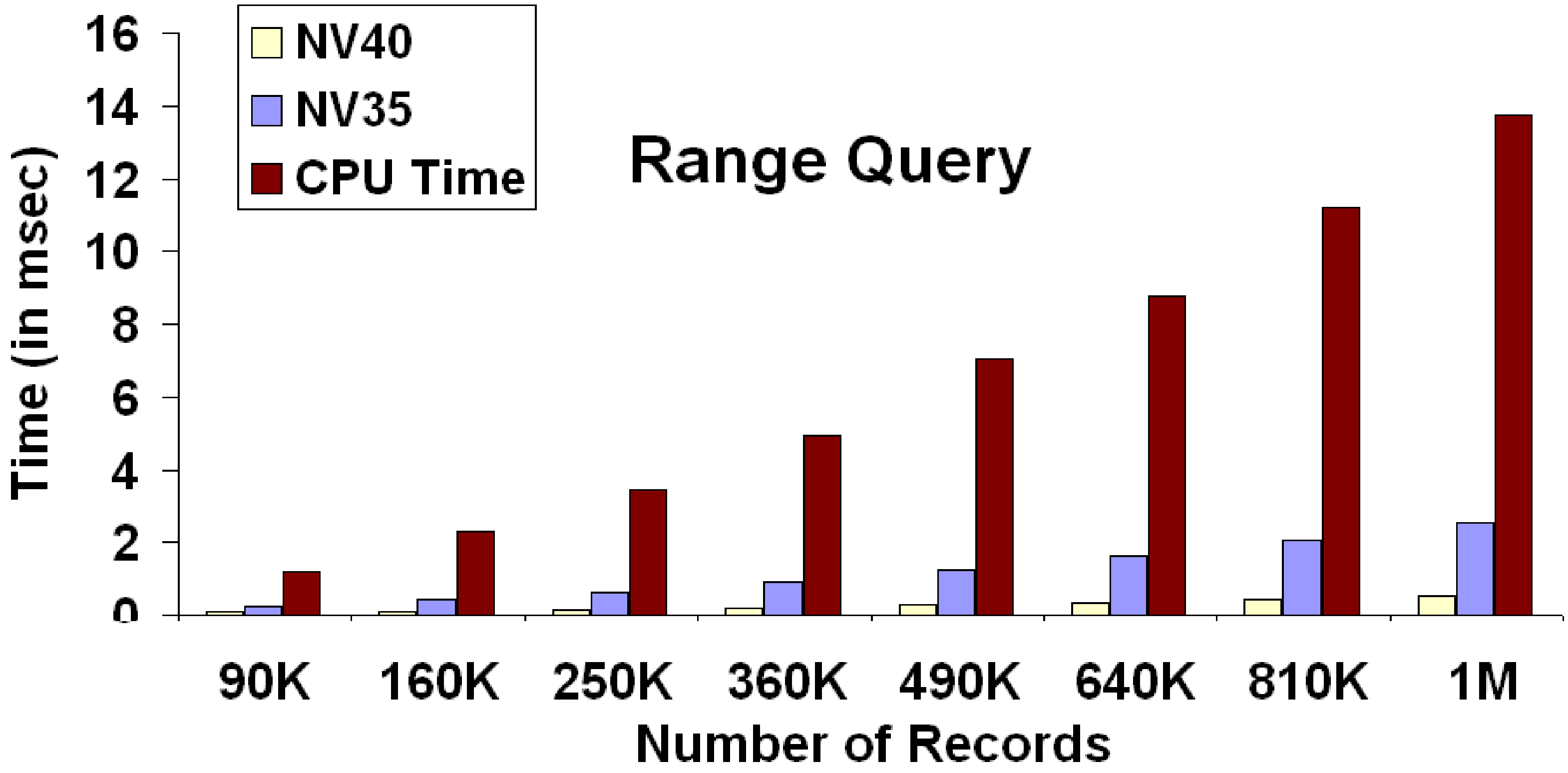
A_1 AND A_2


Stencil = 2
A_1 AND $B^2{}_2$
Stencil = 0
Stencil=2
A_1 AND $B^2{}_3$
Stencil=2
A_1 AND $B^2{}_1$

Range Query

- Compute a_i within [low, high]
  - Evaluated as (a_i >= low) AND (a_i <= high)
- Use NVIDIA depth bounds test to evaluate both conditionals in a single clock cycle

Range Query



GPU is nearly 20 times faster than 2.8 GHz Xeon

Aggregations

- COUNT, MAX, MIN, SUM, AVG

COUNT

- Use occlusion queries to get the number of pixels passing the tests
- Syntax:
  - Begin occlusion query
  - Perform database operation
  - End occlusion query
  - Get count of number of attributes that passed database operation
- Involves no additional overhead!
- Efficient selectivity computation

MAX, MIN, MEDIAN

- Kth-largest number
- Traditional algorithms require data rearrangements
- We perform
  - no data rearrangements
  - no frame buffer readbacks

K-th Largest Number

- Given a set S of values
  - c(m) —number of values ¤ m
  - v_k — the k-th largest number
- We have
  - If c(m) > k-1, then m ⭘ v_k
  - If c(m) ⭘ k-1, then m > v_k
- Evaluate one bit at a time

2nd Largest in 9 Values

0011	1011	1101
0111	0101	0001
0111	1010	0010

$m = 0000$

$v_2 = 1011$

Draw a Quad at Depth 8
Compute c(1000)

0011	1011	1101
0111	0101	0001
0111	1010	0010

$m = 1000$
$v_2 = 1011$

1st bit = 1

0011	1011	1101
0111	0101	0001
0111	1010	0010

m = 1000
v_2 = 1011

c(m) = 3

Draw a Quad at Depth 12
Compute c(1100)

0011	1011	1101
0111	0101	0001
0111	1010	0010

m = 1100

v_2 = 1011

2^{nd} bit = 0

0011	1011	1101
0111	0101	0001
0111	1010	0010

m = 1100

v_2 = 1011

c(m) = 1

Draw a Quad at Depth 10
Compute c(1010)

0011	1011	1101
0111	0101	0001
0111	1010	0010

m = 1010

v_2 = 1011

3rd bit = 1

0011	1011	1101
0111	0101	0001
0111	1010	0010

$m = 1010$

$v_2 = 1011$

$c(m) = 3$

Draw a Quad at Depth 11
Compute c(1011)

0011	1011	1101
0111	0101	0001
0111	1010	0010

$m = 1011$

$v_2 = 1011$

4th bit = 1

0011	1011	1101
0111	0101	0001
0111	1010	0010

$m = 10111$

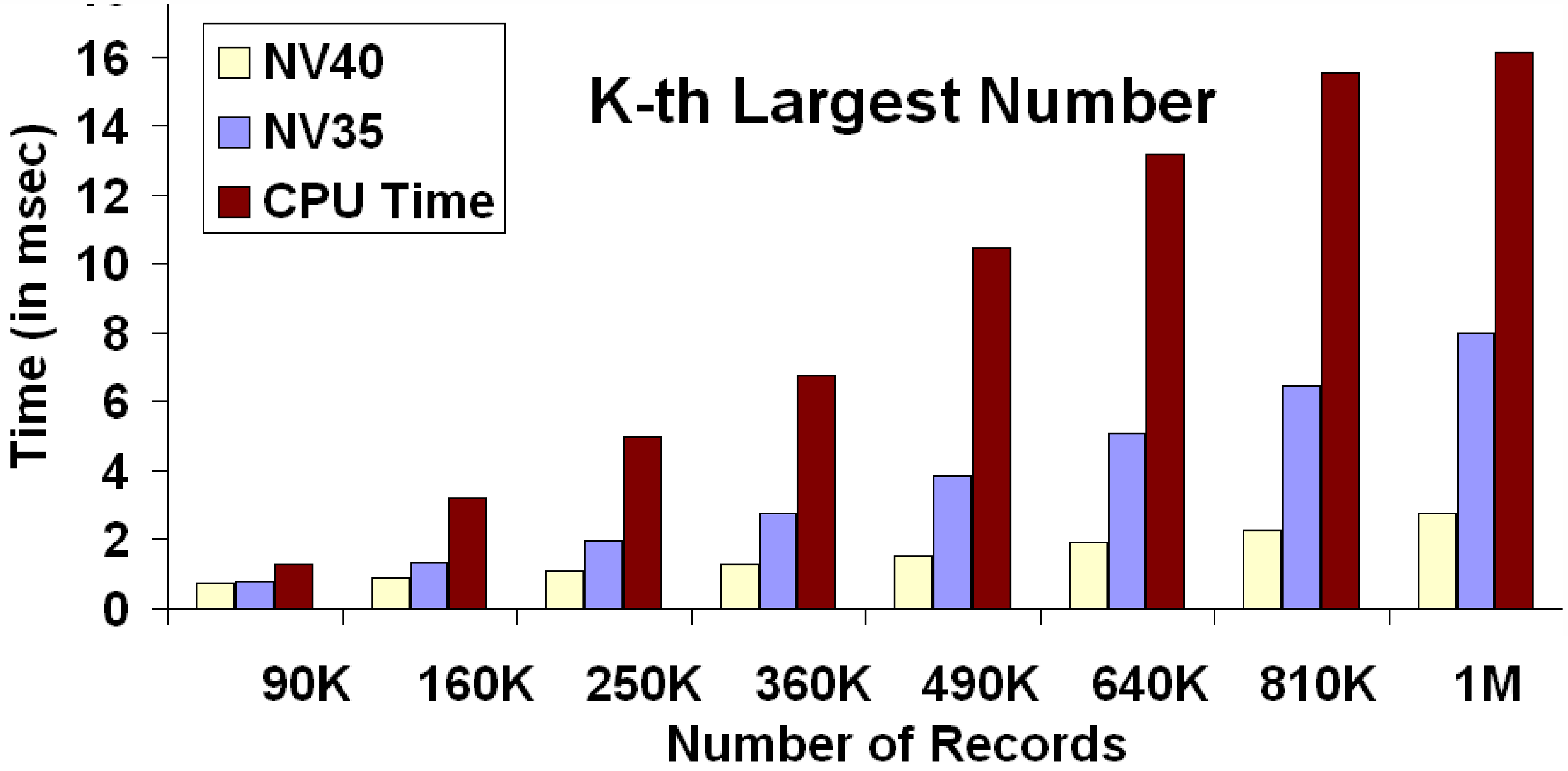
$v_2 = 1011$

$c(m) = 2$

Our algorithm

- Initialize m to 0
- Start with the MSB and scan all bits till LSB
- At each bit, put 1 in the corresponding bit-position of m
- If c(m) < k, make that bit 0
- Proceed to the next bit

Median



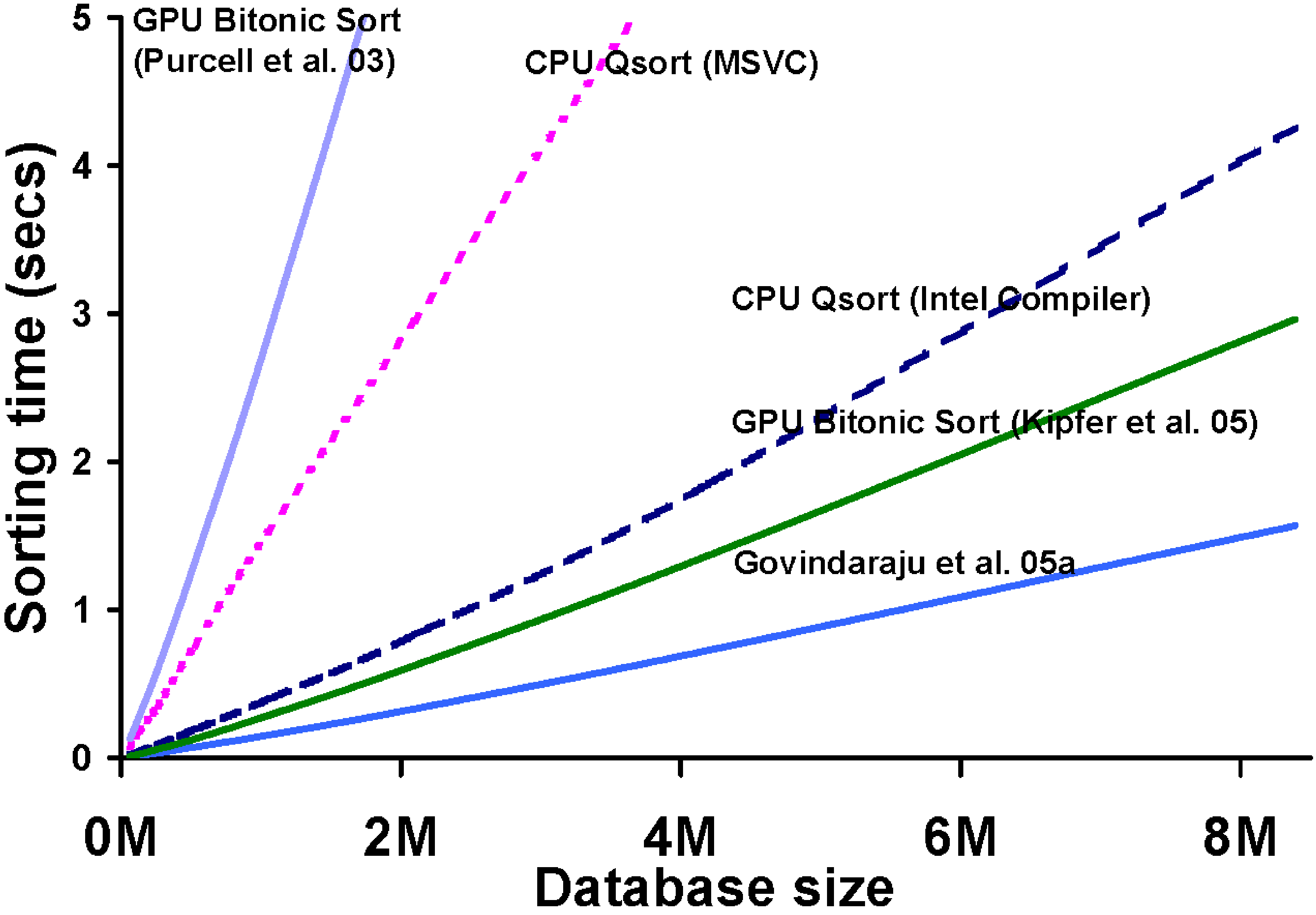
GPU is nearly 6 times faster than 2.8 GHz Xeon!

Join Queries

- Accelerated using sorting operations on the join key
- The join keys of the relations are sorted
- A bit vector representing active records are computed using binary search on the sorted keys
- Sorting is computationally intensive!

Sorting

- Quicksort on CPUs
  - Incoherent data accesses lead to performance loss
  - Instruction dependencies due to conditionals
- Periodic Balanced Sorting Network on GPUs [Govindaraju et al. 2005]
  - Implemented using blending and texture mapping functionality
  - Exploits the high parallelism and memory bandwidth


GPU Bitonic Sort (Purcell et al. 03)
CPU Qsort (MSVC)
CPU Qsort (Intel Compiler)
GPU Bitonic Sort (Kipfer et al. 05)
Govindaraju et al. 05a
Sorting time (secs)
0
1
2
3
4
5
0M
2M
4M
6M
8M
Database size

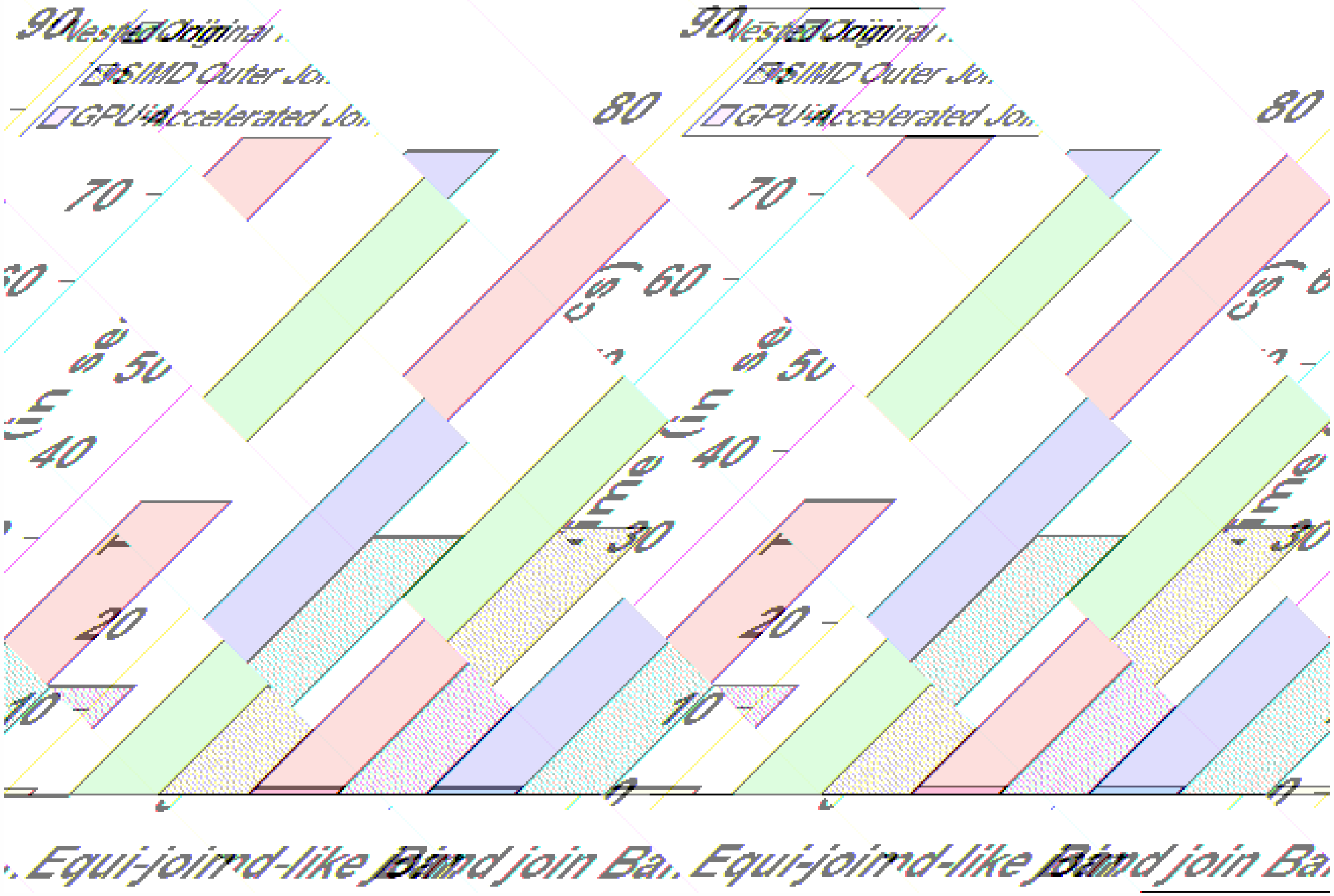
Sorting on GPUs

Talk at SIGMOD 2005, June 16, 10:00 am – 11:30 am

“Fast and Approximate Stream Mining of Quantiles and Frequencies Using Graphics Processors “

Research Session 18, Harborview II
Software announcement

Join Performance



Advantages

- Algorithms progress at GPU growth rate
- Offload CPU work
  - Streaming processor parallel to CPU
- Fast
  - Massive parallelism on GPUs
  - High memory bandwidth
  - No branch mispredictions
- Commodity hardware!

Conclusions

- Novel algorithms to perform database operations on GPUs
  - Evaluation of predicates, boolean combinations of predicates, aggregations and join queries
- Algorithms take into account GPU limitations
  - No data rearrangements
  - No frame buffer readbacks

Conclusions

- Preliminary comparisons with optimized CPU implementations is promising
- GPU as a useful co-processor

Further Details

Fast Database Operations using Graphics Processors

N. Govindaraju, B. Lloyd, W. Wang, M. Lin, D. Manocha

Proc. of ACM SIGMOD 2004

Ongoing Work: Caching

- Performance evaluation of GPU-based algorithms
- GPU-based sorting algorithms suffer from cache misses [Govindaraju et al. 05b]
- Design cache-efficient algorithms based on GPU memory access model

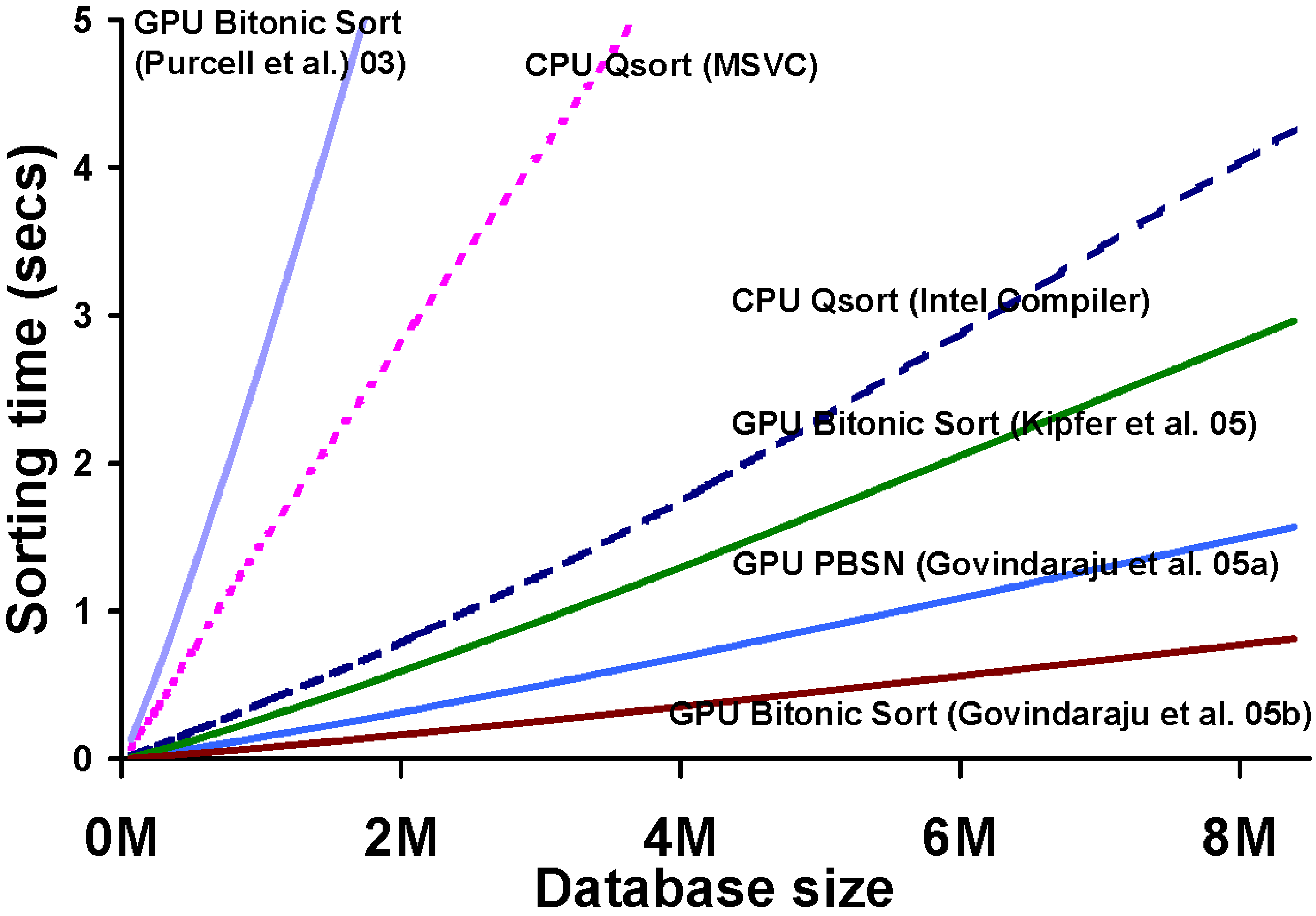
Ongoing Work: Caching

Cache studies on GPUs are very important

- Pentium IV EE has 178 million transistors and 80% is cache! (http://www.lostcircuits.com/cpu/intel_p4ee/)
- GeForce 6800 Ultra has 220 million transistors and majority is logic transistors – efficient cache usage is crucial for performance

Ongoing Work: Caching

- Caching optimizations on GPUs are different than CPU-based algorithms
  - No information on GPU cache sizes
  - Memory access model is different from CPUs
  - Due to SIMD nature, computational model is also different
- Cache-efficient algorithms can improve performance by more than 30% [Govindaraju et al. 05b]


5
4
3
2
1
0
Sorting time (secs)
GPU Bitonic Sort
(Purcell et al.) 03)
CPU Qsort (MSVC)
CPU Qsort (Intel Compiler)
GPU Bitonic Sort (Kipfer et al. 05)
GPU PBSN (Govindaraju et al. 05a)
GPU Bitonic Sort (Govindaraju et al. 05b)
0M
2M
4M
6M
8M
Database size

Ongoing Work: Classification

- More data mining operations such as classification and clustering
- Queries on spatial and temporal databases
  - k nearest neighbor computations
  - R-trees for spatial and temporal intersections

Acknowledgements

- Army Research Office
- National Science Foundation
- Office of Naval Research
- Intel Corporation
- NVIDIA Corporation
- Jasleen Sahni, Don Smith, UNC
- Collaborators: Michael Henson, Ming Lin, Brandon Lloyd, Nikunj Raghuvanshi, Wei Wang
- DaMoN organizers
- UNC GAMMA Group

Thank You

- Questions or Comments?

naga@cs.unc.edu

http://gamma.cs.unc.edu/DB

http://gamma.cs.unc.edu/STREAMING