Tim Kaldewey, Guy Lohman, Rene Mueller, Peter Volk (*Summer Intern*)

May 21 2012

GPU Join Processing Revisited

© 2011 IBM Corporation

Why GPU?

	GPU (GTX580)	CPU (i7-2600)
Cores	16 x 32	4 (x2 HT)
Peak Compute Performance	1331 GFLOPS	109 GFLOPS
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s

Relatively easy programmable using Nvidia's Compute Unified Device Architecture (CUDA), a C API

© 2011 IBM Corporation

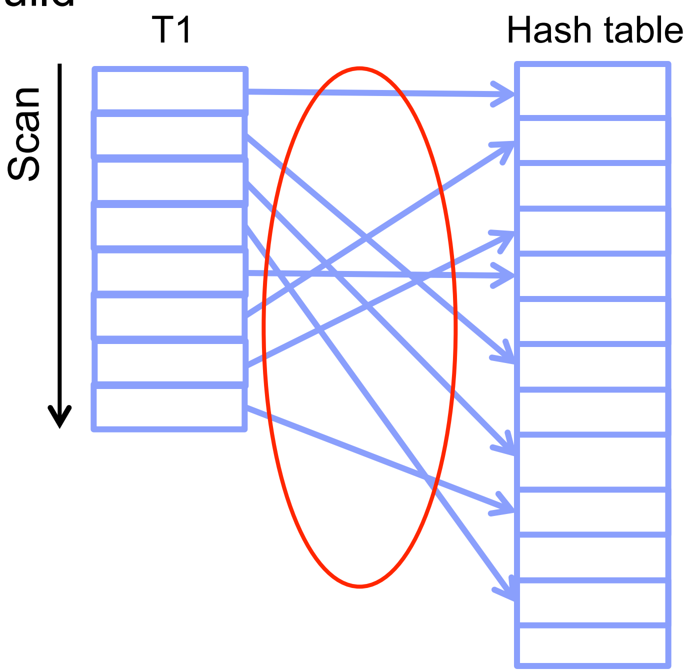
Recap Hash Join – Memory Access Pattern

Join two tables (|T1| < |T2|) in 2 steps

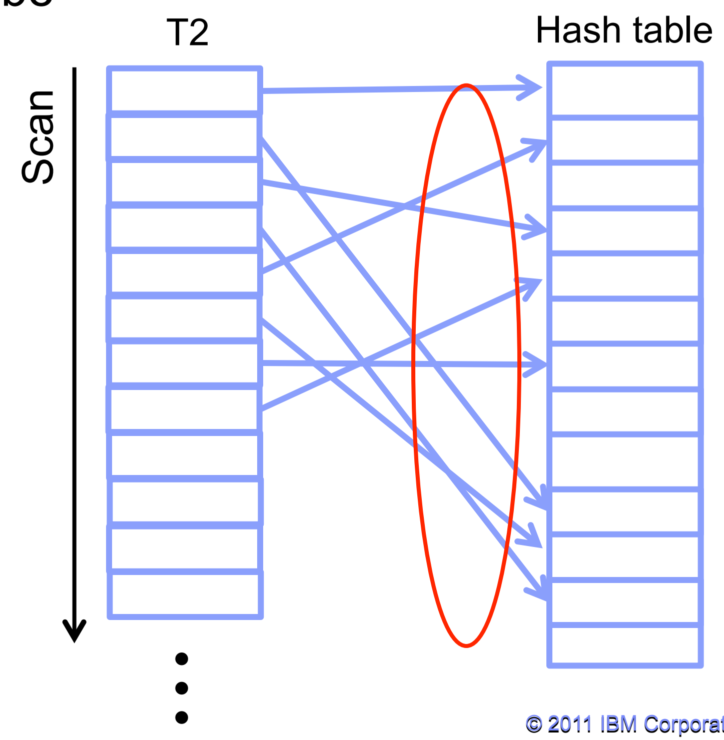
1. Scan T1 (build table) to build hash table
2. Scan T2 (probe table) and probe hash table for matching key

Build and probe both produce a random memory access pattern

1) Build



2) Probe

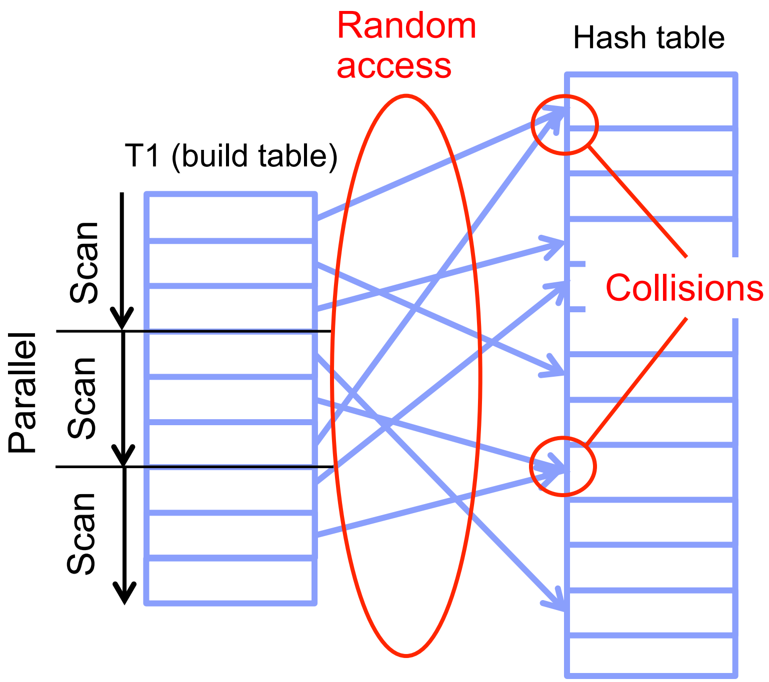


© 2011 IBM Corporation

Recap Hash Join – Parallelism

Parallel hash join requires additional consideration(s):

- Collisions during hash table build
- Multiple threads scan the build table (T1) and insert into the hash table in parallel



- Atomic “compare-and-swap” to efficiently handle collisions

Why GPU?

	GPU (GTX580)	CPU (i7-2600)
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s
Random Access Bandwidth [Measured]	7.7 GB/s	0.8 GB/s
Compare And Swap [Measured]	4.6 GB/s	0.4 GB/s

© 2011 IBM Corporation

Why GPU?

	GPU (GTX580)	CPU (i7-2600)
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s
Random Access Bandwidth [Measured]	7.7 GB/s	0.8 GB/s
Compare And Swap [Measured]	4.6 GB/s	0.4 GB/s

Probe

Build HT

© 2011 IBM Corporation

Why not GPU?

	GPU (GTX580)	CPU (i7-2600)
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s
Random Access Bandwidth [Measured]	7.7 GB/s	0.8 GB/s
Compare And Swap [Measured]	4.6 GB/s	0.4 GB/s
Memory Size	3 GB	32GB

Server hardware up to 6 GB / 1 TB

© 2011 IBM Corporation

Agenda

- Introduction
  - Impressive (raw) GPU performance on up to 6 GB of memory

- Beyond 6 GB
  - Need to transfer data to/from GPU via PCI-E
  - Overlap GPU processing with data copy from CPU to GPU (*conventional*)
  - Uniform address space (*new*)

- GPU join implementations
  - (Conventional) Hash join
  - Partitioned Hash join

- Evaluation
  - Throughput
  - Selectivity
  - Table Size
  - GPU vs. CPU

- Work in progress

© 2011 IBM Corporation

Data Transfers

All data processed by the GPU has to go through PCI-E

	GPU (GTX580)	CPU (i7-2600)
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s
Random Access Bandwidth [Measured]	7.7 GB/s	0.8 GB/s
Compare And Swap [Measured]	4.6 GB/s	0.4 GB/s
PCI-E Bandwidth [Measured]	6.2 GB/s	N/A

© 2011 IBM Corporation

Data Transfers

All data processed by the GPU has to go through PCI-E

	GPU (GTX580)	CPU (i7-2600)
Peak Memory Bandwidth [Spec]	192 GB/s	21 GB/s
Peak Memory Bandwidth [Measured]	177 GB/S	18 GB/s
Random Access Bandwidth [Measured]	7.7 GB/s	0.8 GB/s
Compare And Swap [Measured]	4.6 GB/s	0.4 GB/s
PCI-E Bandwidth [Measured]	6.2 GB/s	N/A

Probe

Build HT

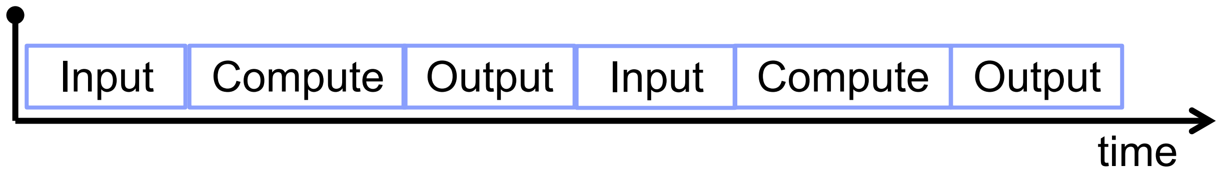
Read input tables

PCI-E sets the upper bound for GPU join performance !

© 2011 IBM Corporation

How to get data to the GPU?

- Synchronous copy

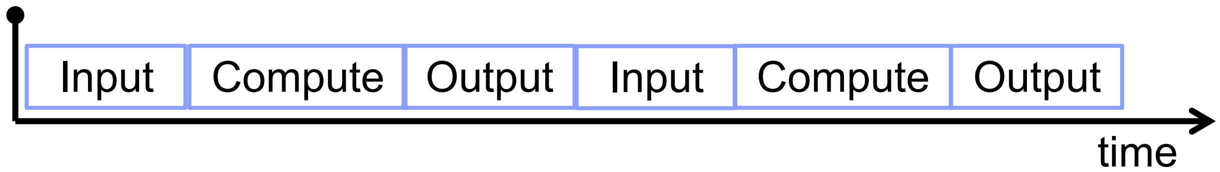


  - Memcopy through device driver
  - Memcopy “occupies” a CPU core
  - PCI-E link idle during compute

© 2011 IBM Corporation

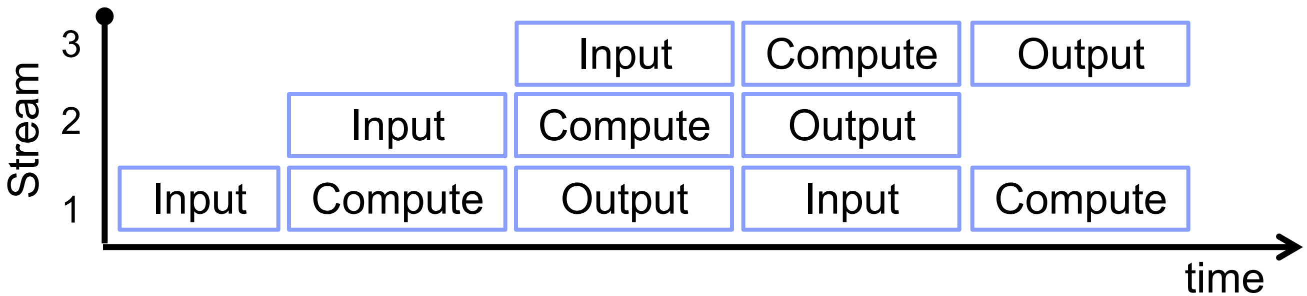
How to get data to the GPU?

- Synchronous copy



  - Memcopy through device driver
  - Memcopy "occupies" a CPU core
  - PCI-E link idle during compute

- Asynchronous copy aka streams



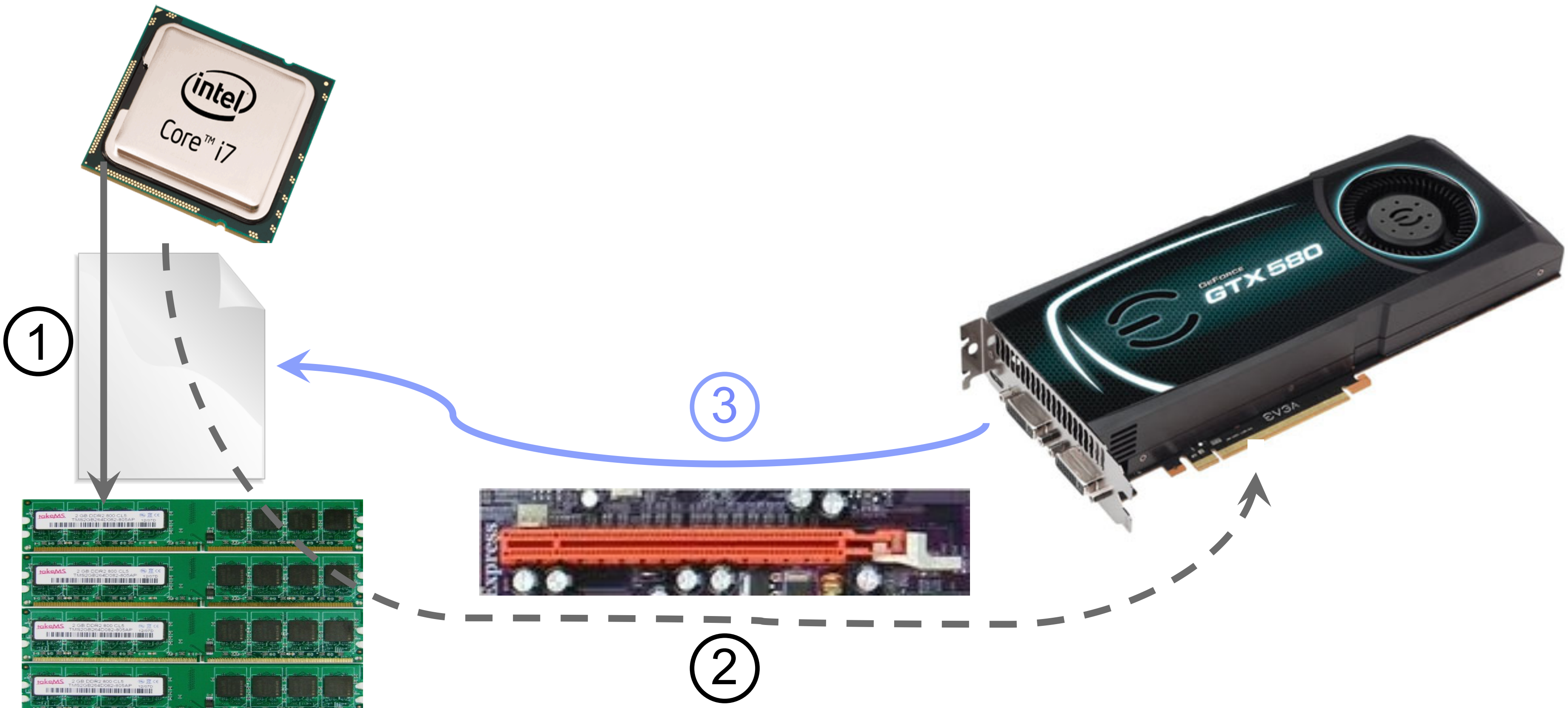
  - Can overlap memcopy and compute operations to maximize PCI-E throughput
  - Complex implementation
  - Prior work measured ~3 GB/s throughput[1]

[1] H. Pirk, S. Manegold, and M. Kersten. Accelerating foreign-key joins using asymmetric memory channels. In ADMS'11

© 2011 IBM Corporation

How to get data to the GPU?

- Unified Virtual Addressing (UVA)
  - Allows the GPU to access CPU memory “directly”, i.e. without placing a copy in GPU in device memory:
    1. CPU thread pins memory page
    2. GPU function call contains a pointer to the page
    3. GPU(CUDA) threads execute load/store instructions



© 2011 IBM Corporation

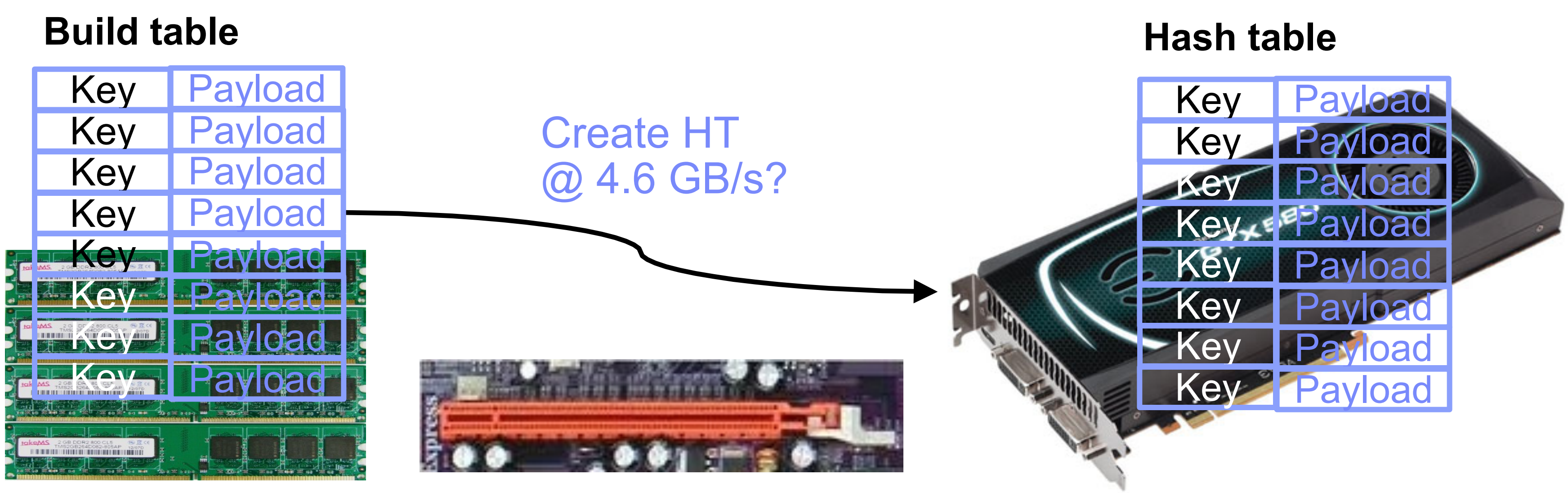
Agenda

- Introduction
  - Impressive (raw) GPU performance on up to 6 GB of memory
- Beyond 6 GB
  - Need to transfer data to/from GPU via PCI-E
  - Overlap GPU processing with data copy from CPU to GPU (*conventional*)
  - Uniform address space (*new*)
- GPU join implementation(s)
  - (Conventional) Hash join
  - Partitioned Hash join
- Evaluation
  - Throughput
  - Selectivity
  - Table Size
  - GPU vs. CPU
- Work in progress

© 2011 IBM Corporation

Hash Join Using UVA

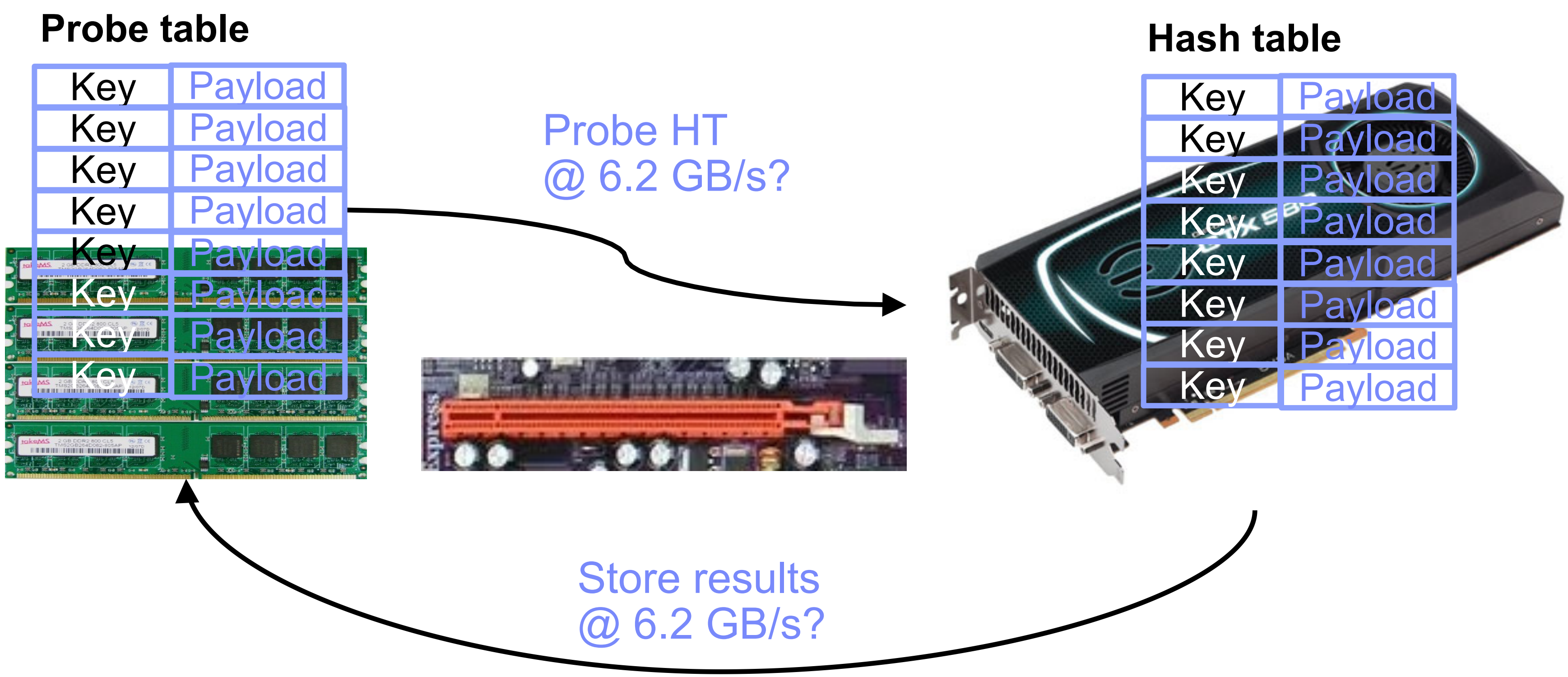
1. Simultaneously read build table from host memory
   & create hash table on device



© 2011 IBM Corporation

Hash Join Using UVA

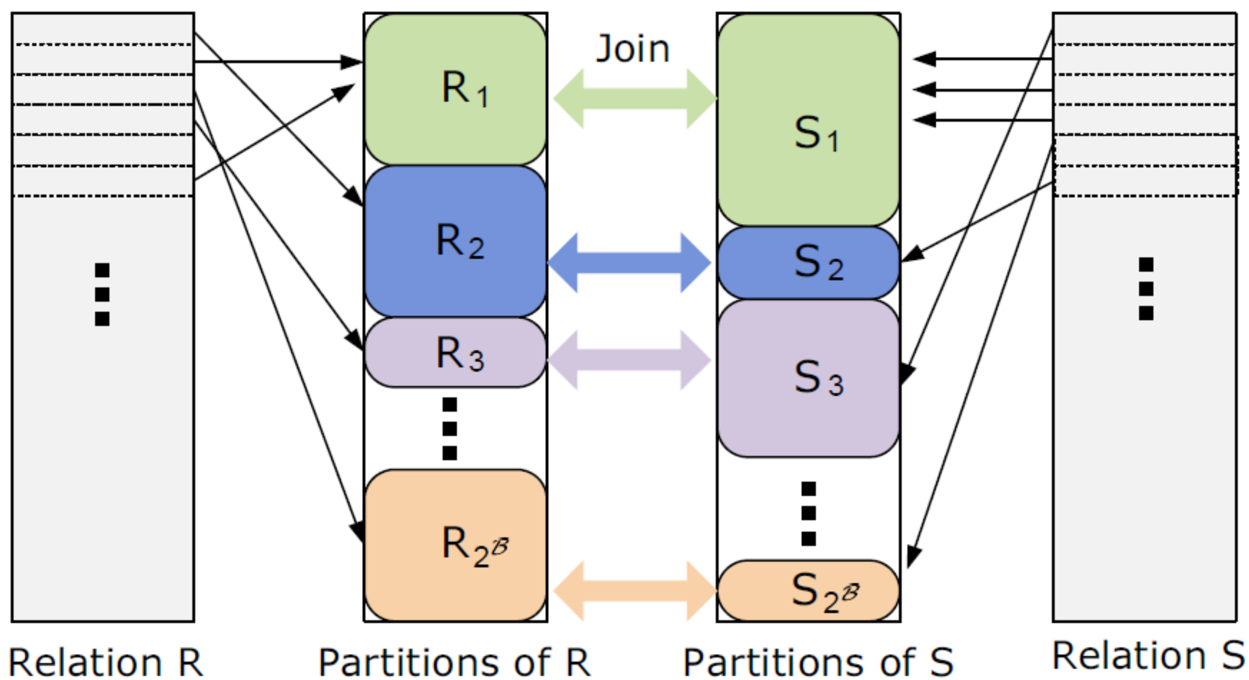
2. Simultaneously read probe table from host memory
   & probe hash table on device
   & store results in host memory



© 2011 IBM Corporation

Partitioned Hash Join

- Partition both tables based on their hash prefix [2]
  - Choose partition size that fits in cache/on-chip memory
- Need to join only sub-tables with same hash prefix
- Requires at least 3 passes over the input data
  - 2 for partitioning (1 creating a histogram, 1 to move data to partitions)
  - 1 for the actual join



[2] C. Kim, E. Sedlar, J. Chhugani, T. Kaldewey, A. Nguyen, A. Di Blas, V. Lee, N. Satish, P. Dubey. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs. VLDB 2009

© 2011 IBM Corporation

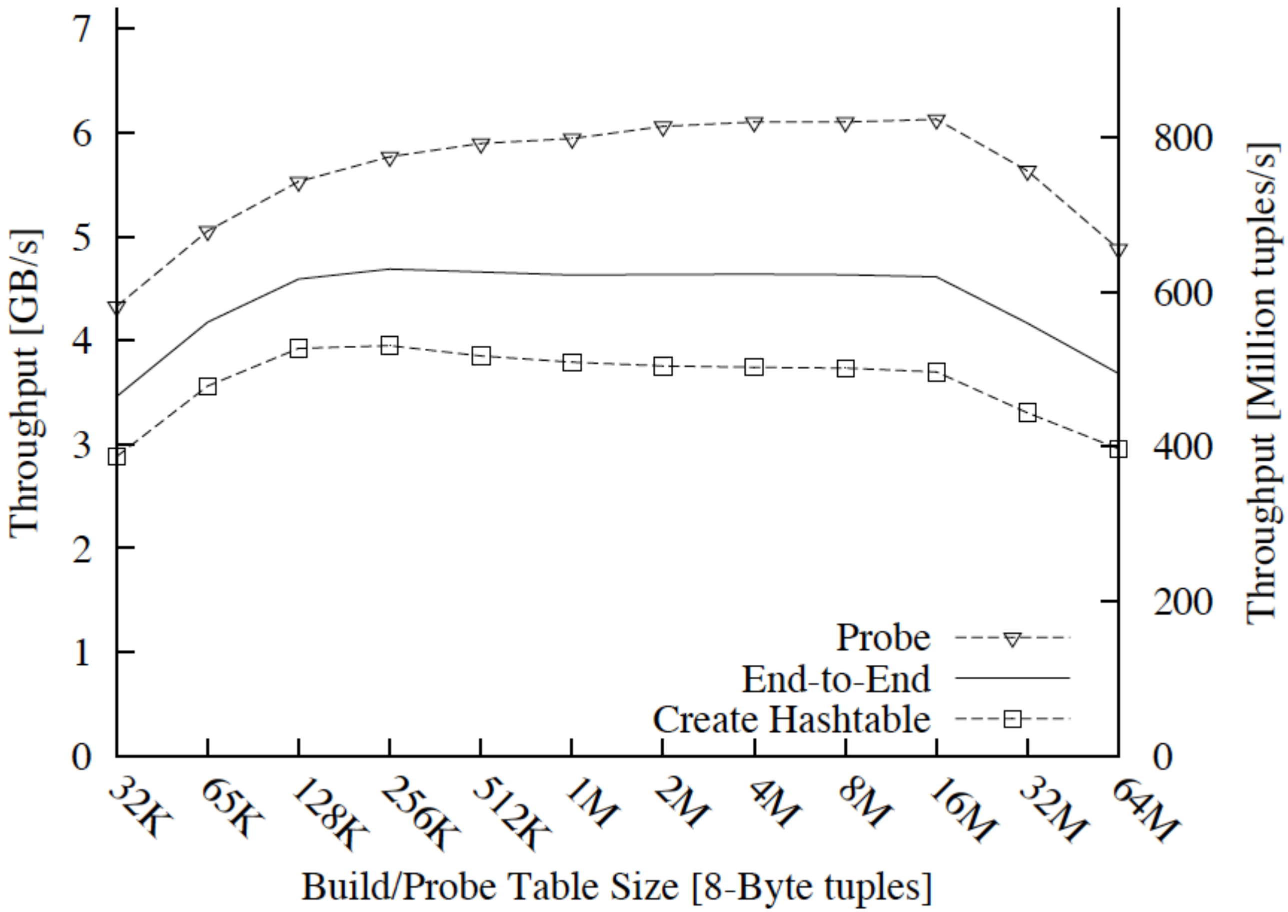
Agenda

- Introduction
  - Impressive (raw) GPU performance on 6 GB of memory
- Beyond 6 GB
  - Need to transfer data to/from GPU via PCI-E
  - Overlap GPU processing with data copy from CPU to GPU (*conventional*)
  - Uniform address space (*new*)
- GPU join implementation(s)
  - (Conventional) Hash join
  - Partitioned Hash join
- Evaluation
  - Throughput
  - Selectivity
  - Table Size
  - GPU vs. CPU
- Work in progress

© 2011 IBM Corporation

Throughput

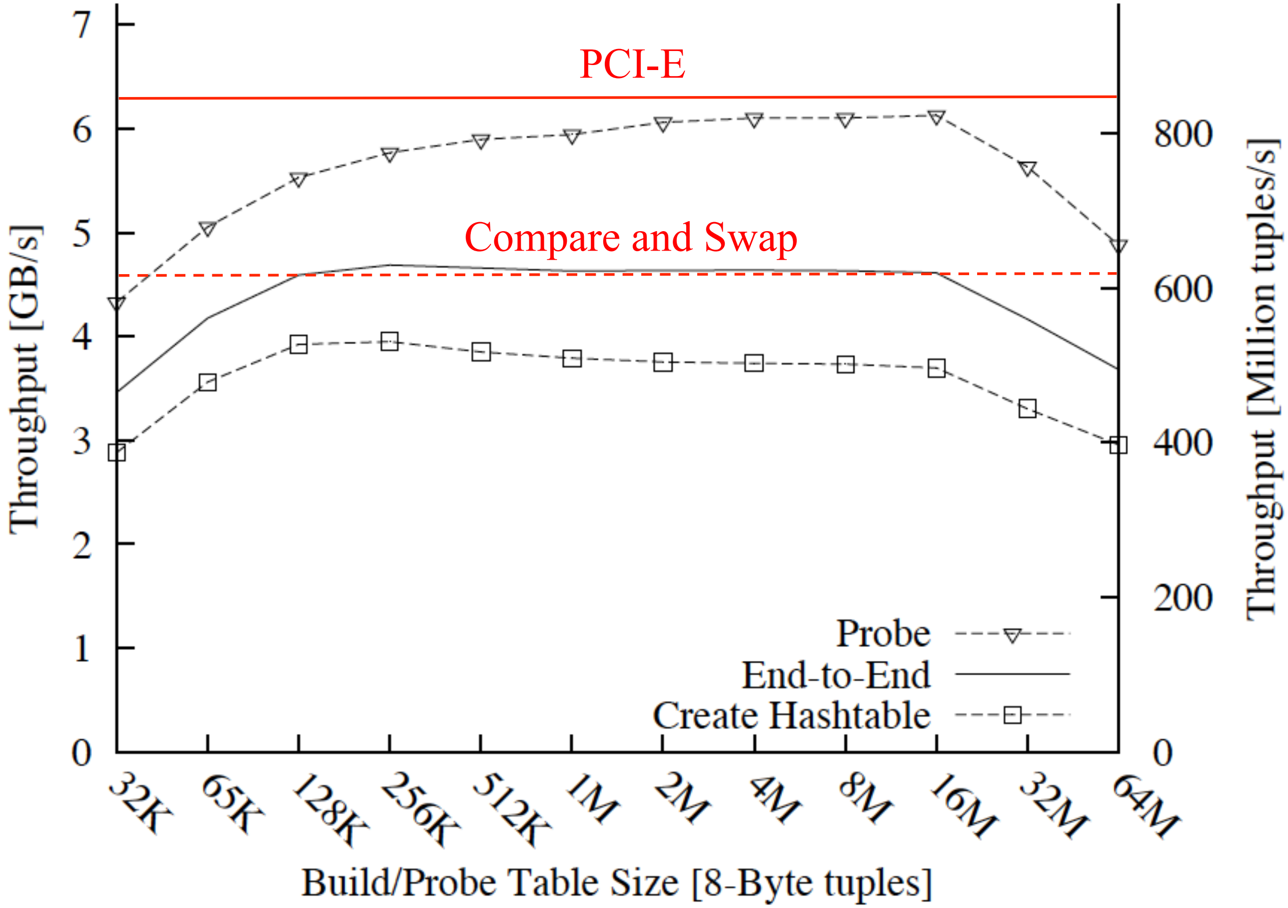
- Joining 2 equal size tables of 32-bit <key,row-ID> pairs (4 + 4 Byte)
  - Uniformly distributed randomly generated keys
  - 3% of the keys in the probe table have a match in the build table
  - Measuring End-to-End throughput, i.e. input tables & results in CPU memory



© 2011 IBM Corporation

Throughput

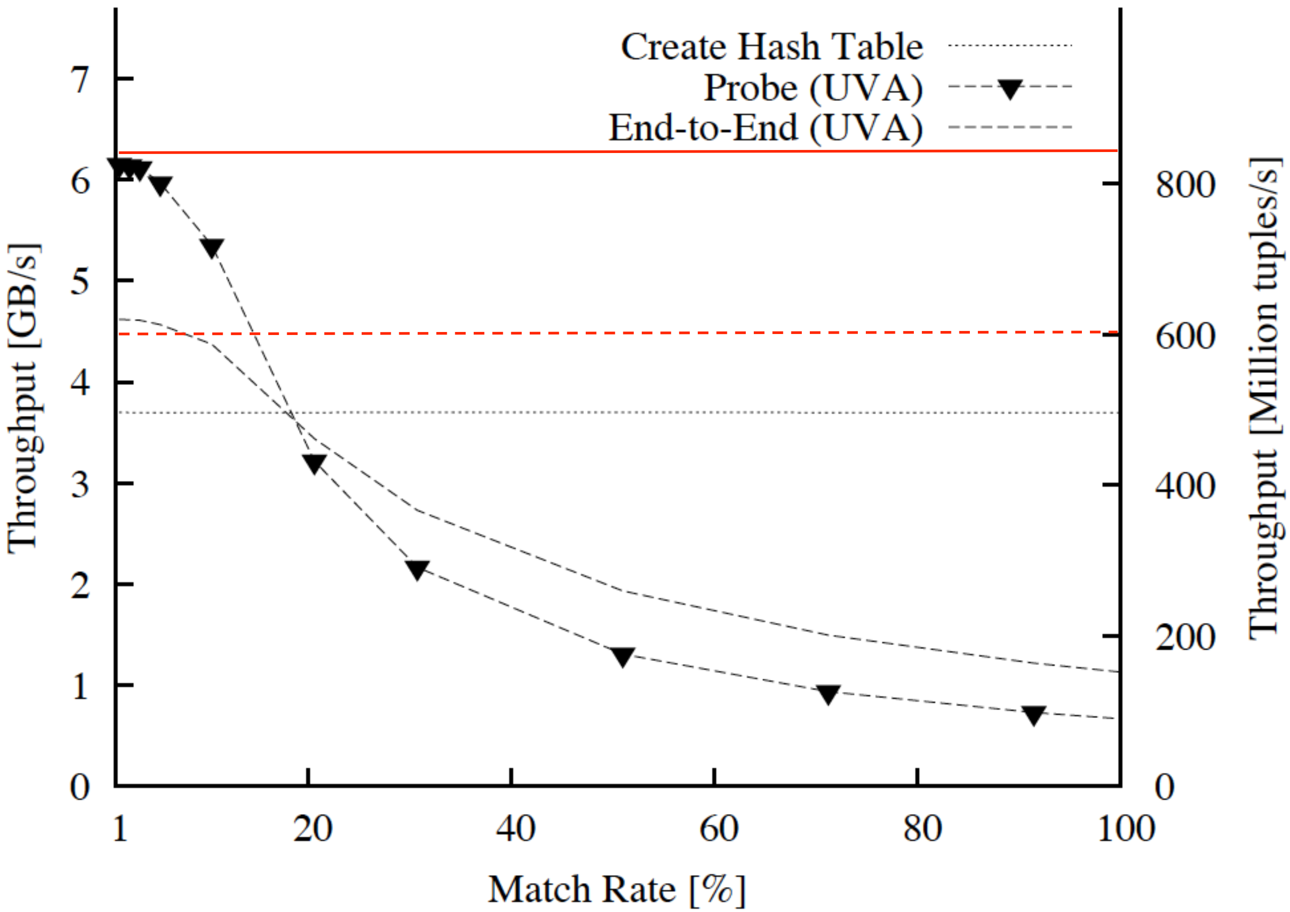
- Joining 2 equal size tables of 32-bit <key,row-ID> pairs (4 + 4 Byte)
  - Uniformly distributed randomly generated keys
  - 3% of the keys in the probe table have a match in the build table
  - Measuring End-to-End throughput, i.e. input tables & results in CPU memory



© 2011 IBM Corporation

Selectivity

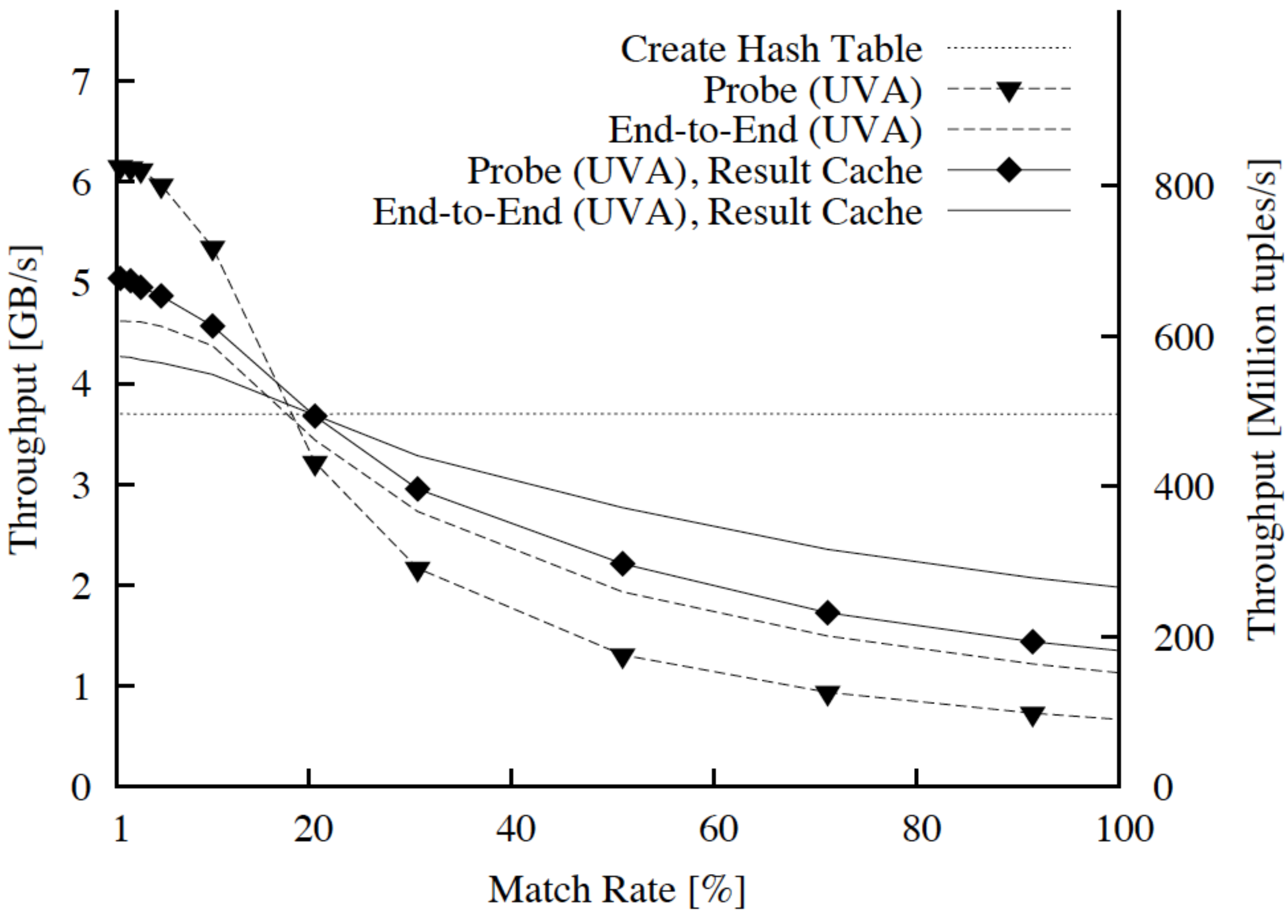
- Joining 2 tables containing 32-bit <key,row-ID> pairs with 16 million rows each
- Uniformly distributed randomly generated keys
- Varying the amount of keys that find a match from 1% to 100%



© 2011 IBM Corporation

Selectivity

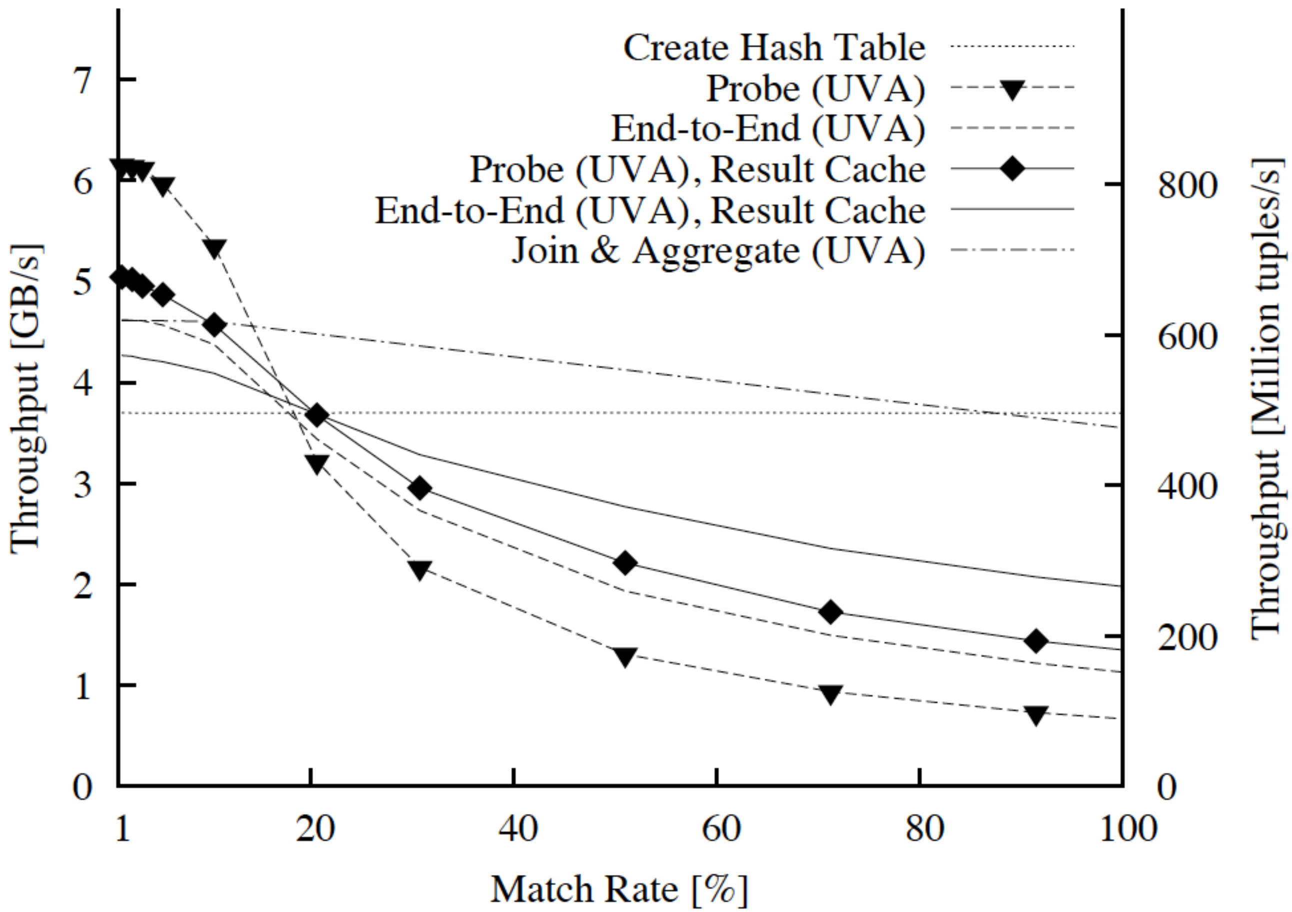
- Joining 2 tables containing 32-bit <key,row-ID> pairs with 16 million rows each
- Uniformly distributed randomly generated keys
- Varying the amount of keys that find a match from 1% to 100%
- Cache results in on-chip memory before writing them back to host (CPU) memory



© 2011 IBM Corporation

Selectivity

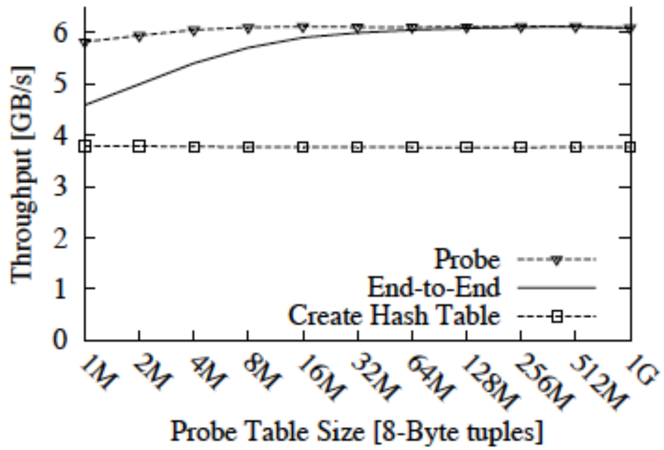
- Joining 2 tables containing 32-bit <key,row-ID> pairs with 16 million rows each
- Uniformly distributed randomly generated keys
- Varying the amount of keys that find a match from 1% to 100%
- Pipeline the results of the join into a a group-by (and aggregate) operator



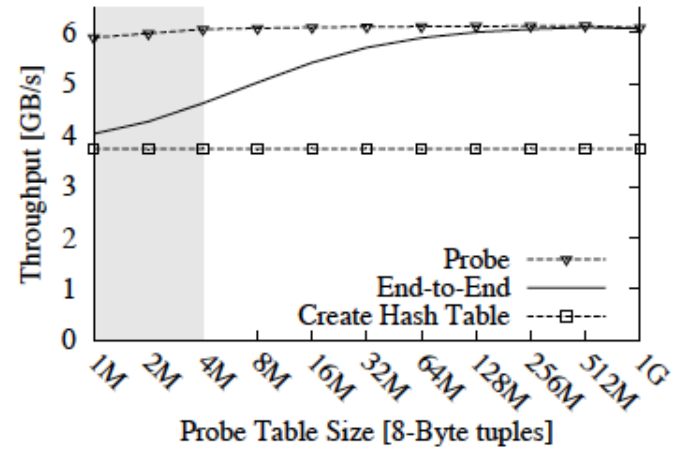
© 2011 IBM Corporation

Table Size

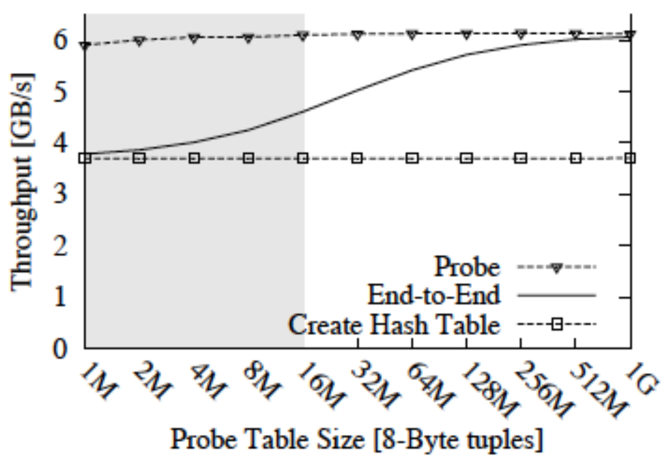
- Joining 2 tables of 32-bit <key,row-ID> pairs, for increasing table sizes
- 3% of the keys have a match in the build table



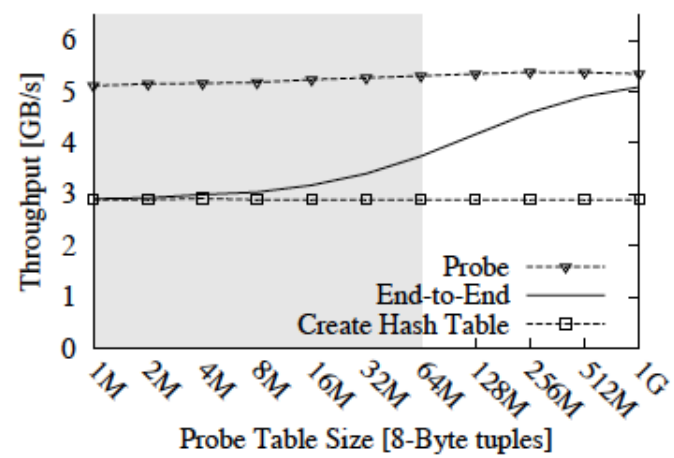
(a) 1M tuples build table



(b) 4M tuples build table



(c) 16M tuples build table

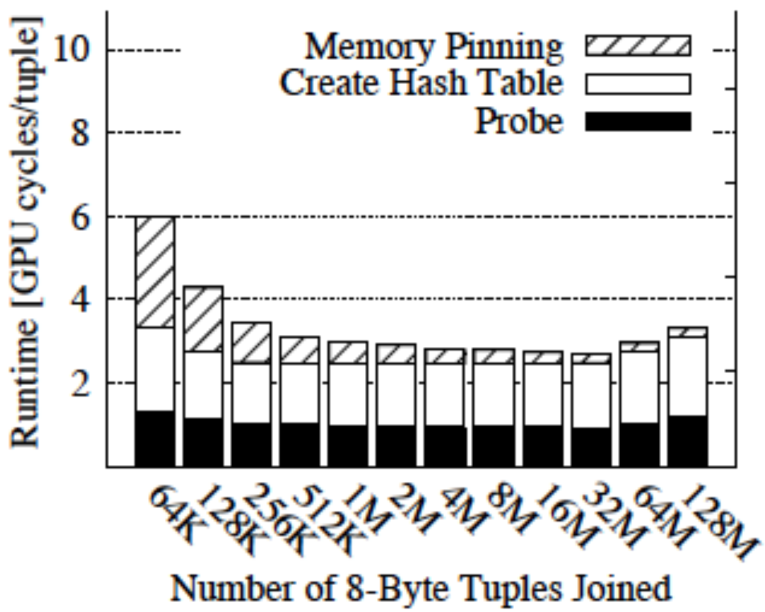


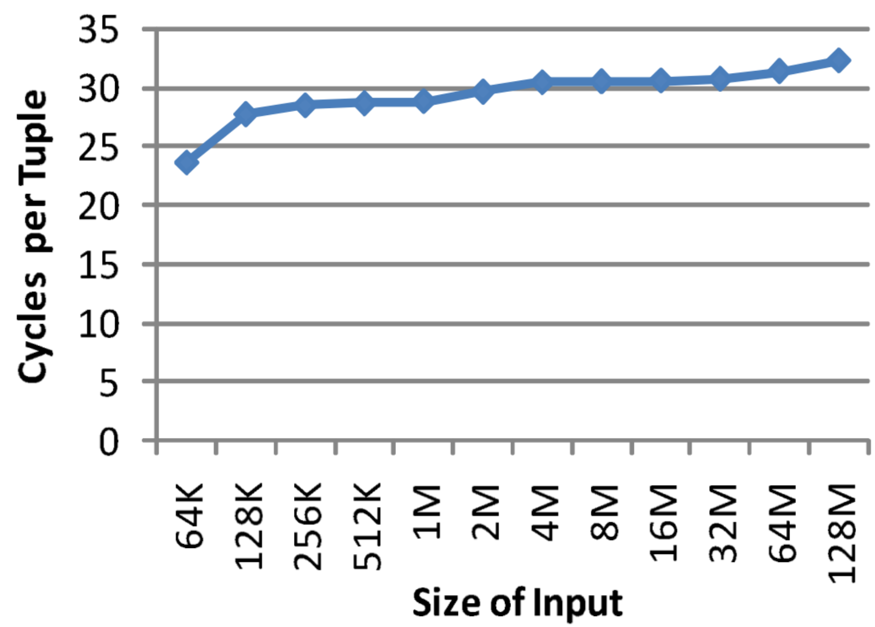
(d) 64M tuples build table

© 2011 IBM Corporation

GPU vs. CPU

- Joining 2 equal size tables of 32-bit <key,row-ID> pairs (4 + 4 Byte)
- Uniformly distributed randomly generated keys
- 3% of the keys have a match in the build
- CPU implementation[2] does not materialize results



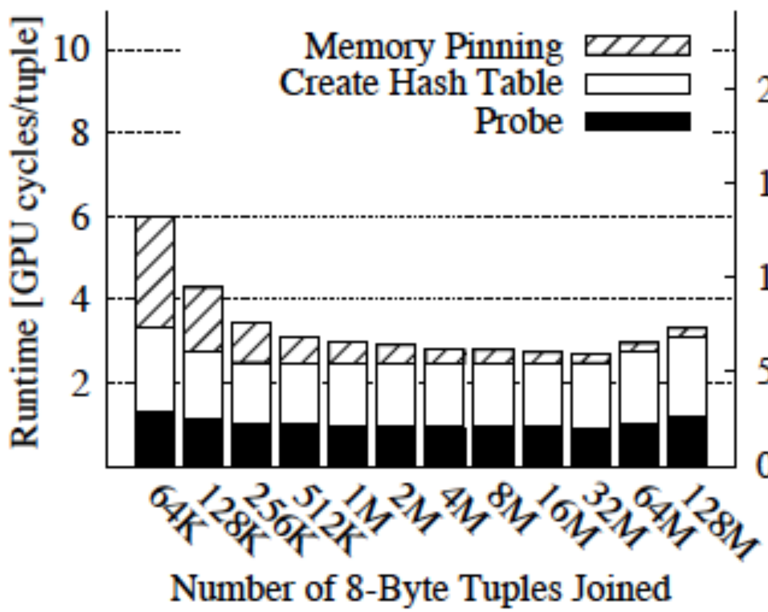


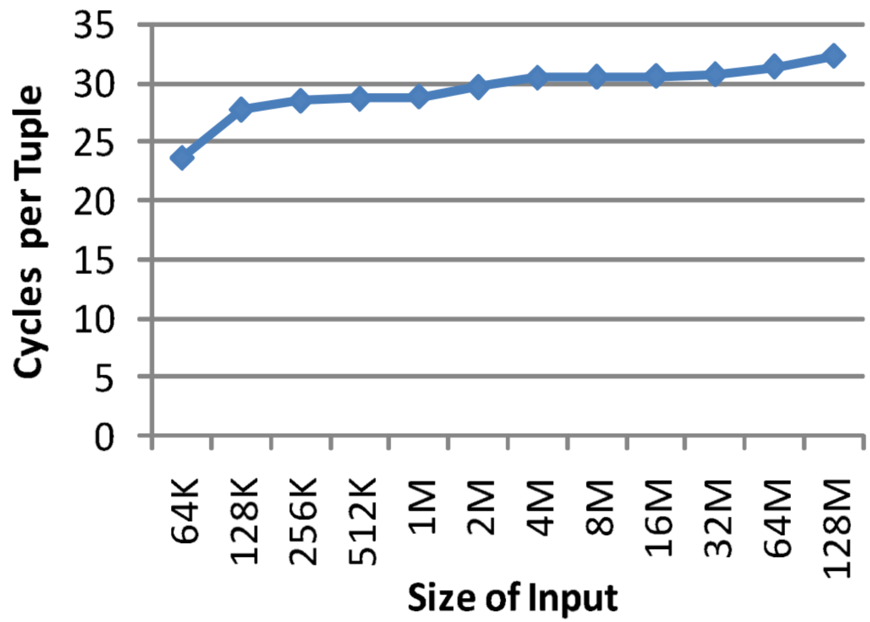
[2] C. Kim, E. Sedlar, J. Chhugani, T. Kaldewey, A. Nguyen, A. Di Blas, V. Lee, N. Satish, P. Dubey. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs. VLDB 2009

© 2011 IBM Corporation

GPU vs. CPU

- Joining 2 equal size tables of 32-bit <key,row-ID> pairs (4 + 4 Byte)
- Uniformly distributed randomly generated keys
- 3% of the keys have a match in the build
- CPU implementation[2] does not materialize results
- Cycles/tuple not a meaningful metric
  - depends on processor frequency, tuple size, …





[2] C. Kim, E. Sedlar, J. Chhugani, T. Kaldewey, A. Nguyen, A. Di Blas, V. Lee, N. Satish, P. Dubey. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs. VLDB 2009

© 2011 IBM Corporation

Conclusion

- Conventional HJ is very efficient on the GPU
- Using up to 96 % of available PCI-E bandwidth
- On average ½ order of magnitude faster than CPU join

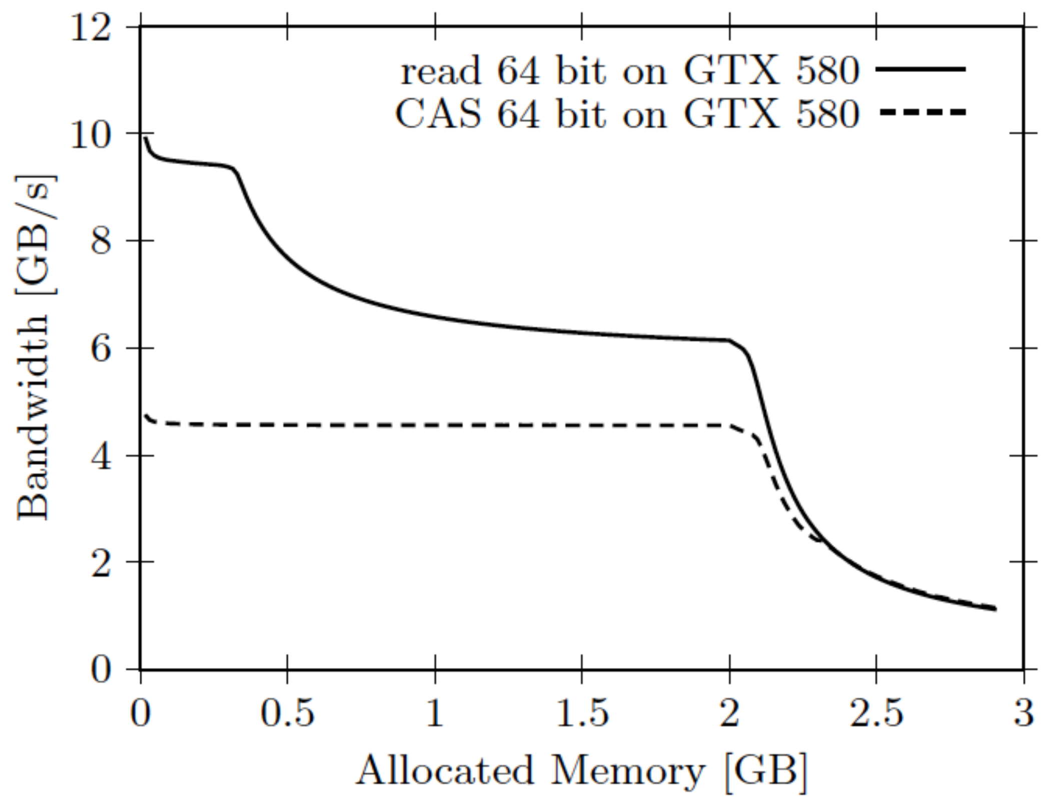
© 2011 IBM Corporation

Conclusion

- Conventional HJ is very efficient on the GPU
- Using up to 96 % of available PCI-E bandwidth
- On average ½ order of magnitude faster than CPU join

Work in Progress

- Hash tables larger than 2GB
- Hash tables larger than 6GB
- Handling data on external storage
- Macro Benchmarks



© 2011 IBM Corporation

Questions ?

© 2011 IBM Corporation