A Virtual Machine Model for Accelerating Relational Database Joins using a General Purpose GPU

Kevin Angstadt
University of Virginia
Charlottesville, Virginia
kaa2nx@virginia.edu

Ed Harcourt
St. Lawrence University
Canton, New York
edharcourt@stlawu.edu

Databases Are Used Everywhere

SQLite: Most Deployed DB

Virtual Machine-based Database Engine

- 300 million copies of Firefox
- 20 million Apple computers
- 500 million iPhones*
- 1 billion Android Devices**
- 450 million registered Skype users
- 10 million Solaris 10 installations

Adapted from: https://www.sqlite.org/mostdeployed.html
*http://onforb.es/1gpk4Fs
**http://www.androidcentral.com/android-passes-1-billion-activations

GPUs as General Purpose Processors

GPUs as General Purpose Processors

G2 instance: 65¢/hr

GPUs as General Purpose Processors

G2 instance: 65¢/hr

GPUs as General Purpose Processors

G2 instance: 65¢/hr

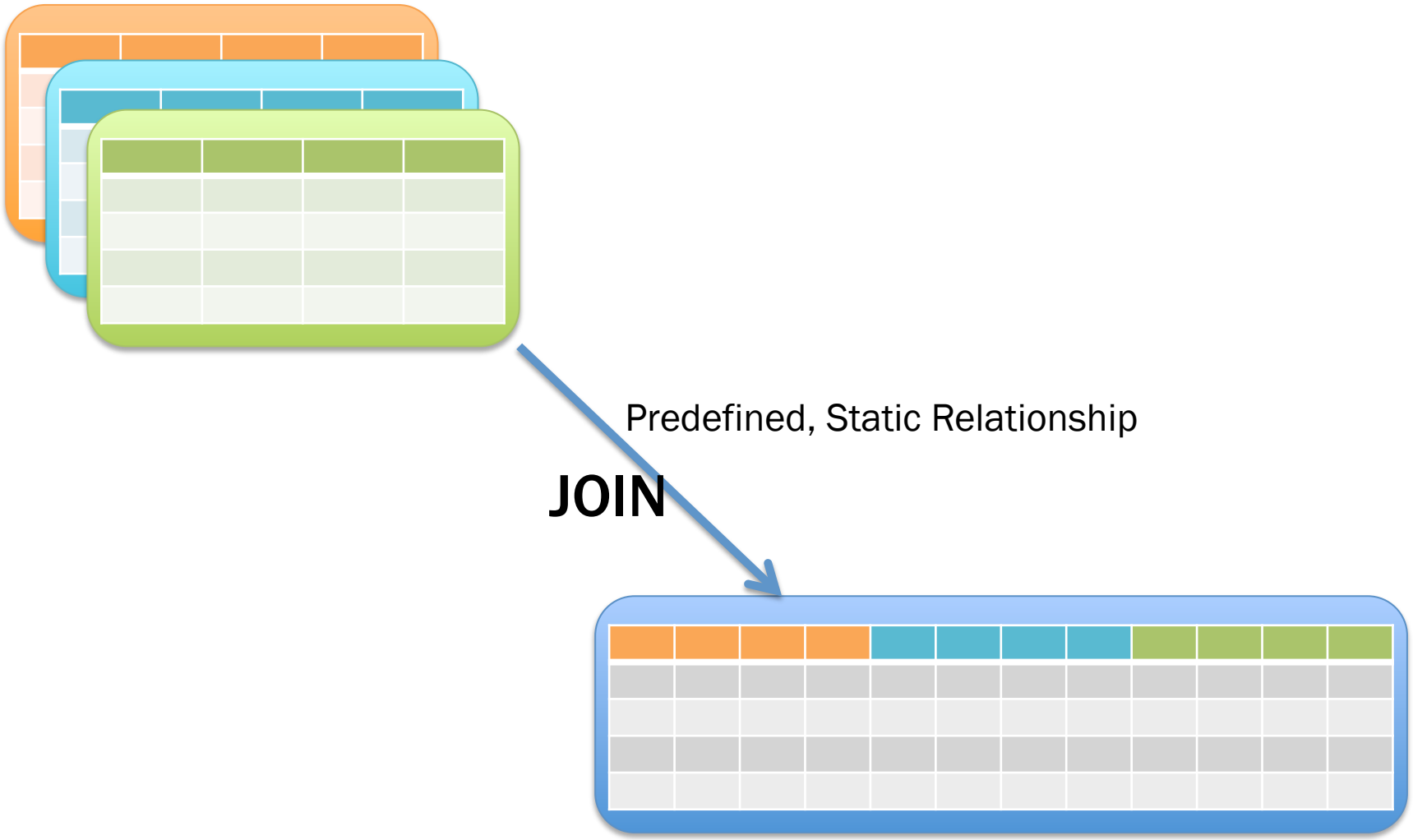
Why don’t we combine SQLite and GPUs to improve performance?

Focus on JOINs

Overview

- Review of Database JOINs
- Why GPUs a good fit?
- Virtual Machine Implementation
  - Query Workflow
  - SQL Queries as Programs
  - Memory Concerns
  - VM Design
- Experimental Results

Database JOIN



Cross-Join Example

- T_1 and T_2 share attribute c_2

T_1

c_1	c_2
1	w
2	z
3	z

T_2

c_2	c_3
x	5
z	6
w	7

Cross-Join Example

$\mathbf{T}_1$

c_1	c_2
1	w
2	z
3	z

$\bowtie$

$\mathbf{T}_2$

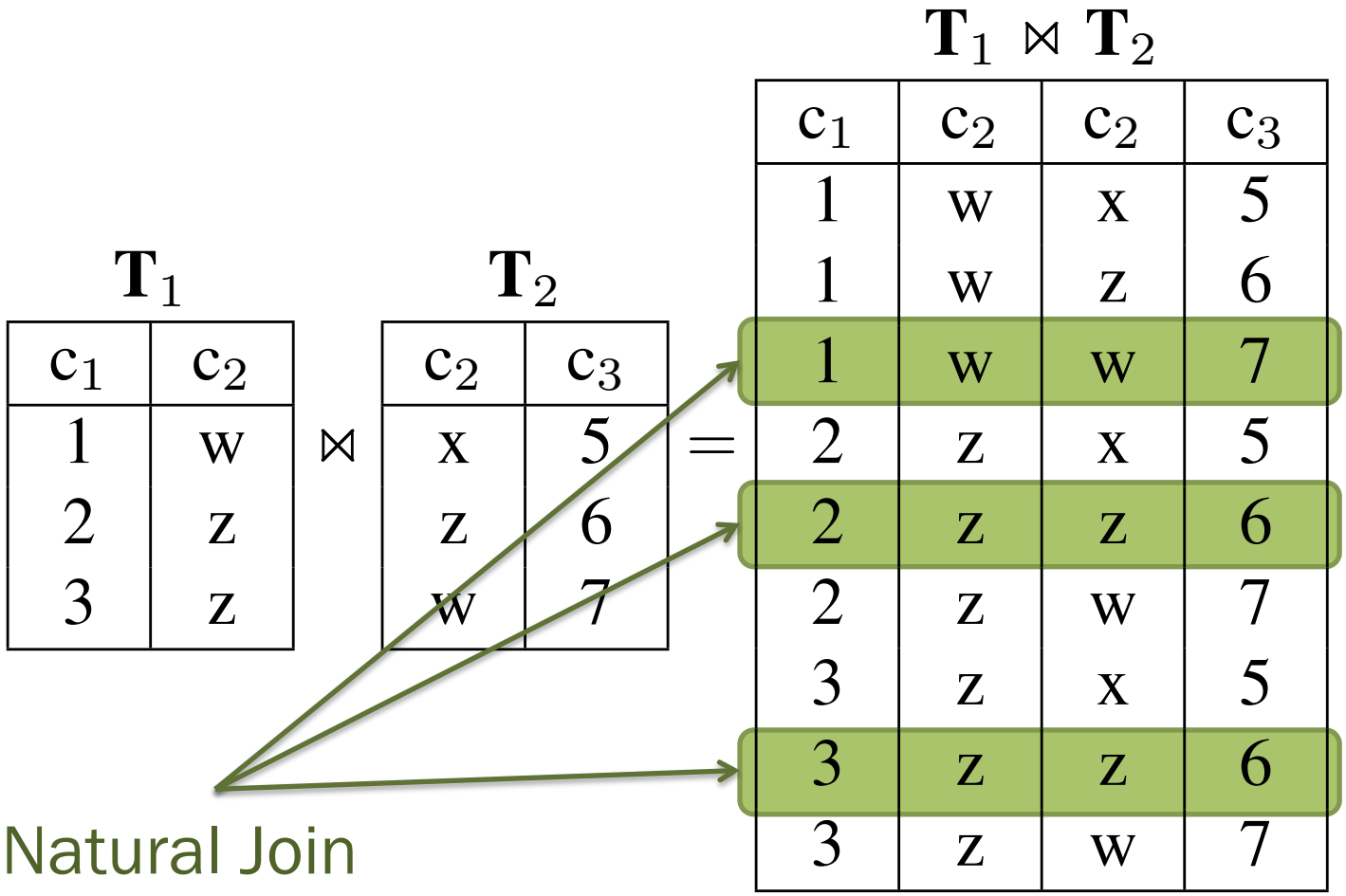
c_2	c_3
x	5
z	6
w	7

$=$

$\mathbf{T}_1 \bowtie \mathbf{T}_2$

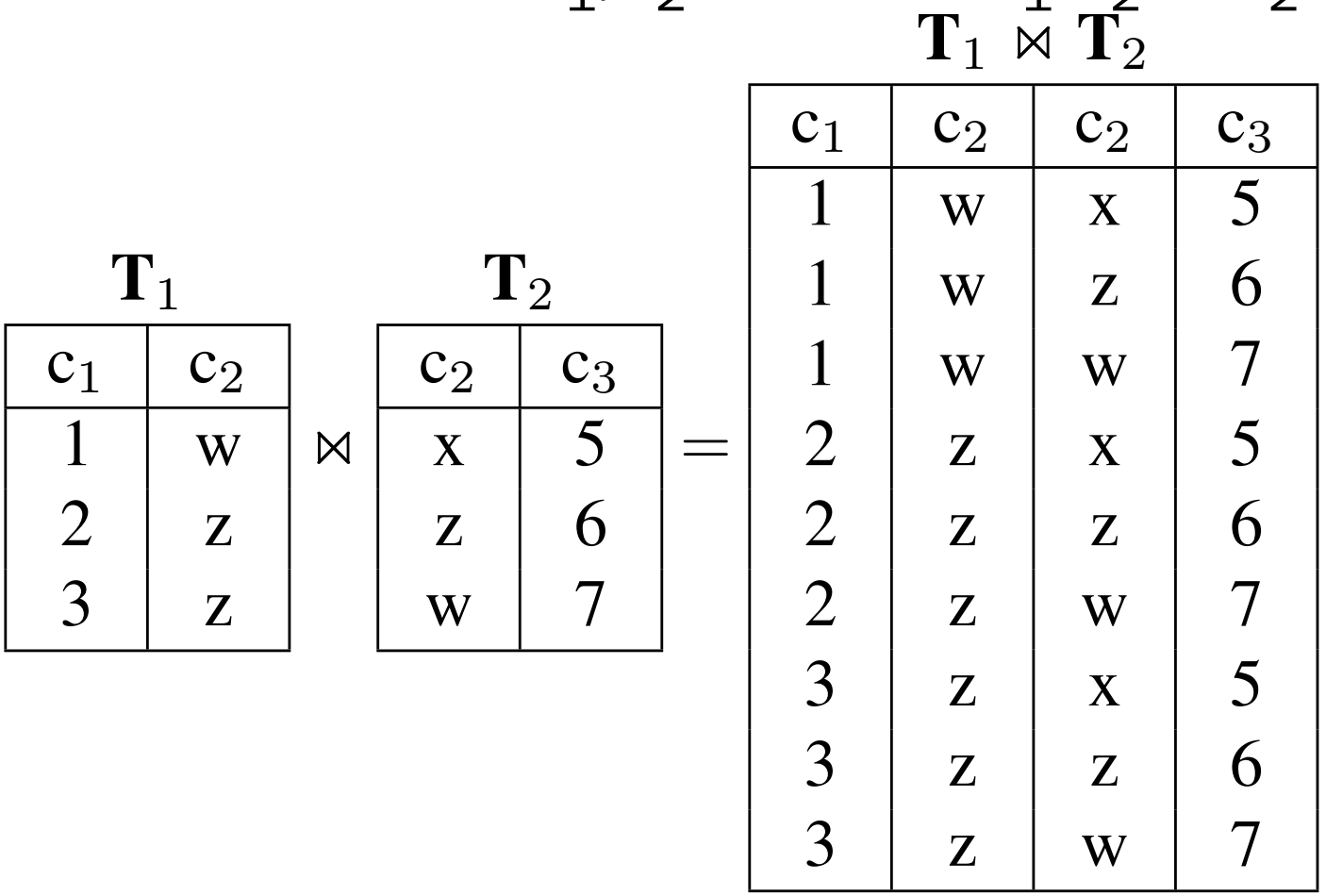
c_1	c_2	c_2	c_3
1	w	x	5
1	w	z	6
1	w	w	7
2	z	x	5
2	z	z	6
2	z	w	7
3	z	x	5
3	z	z	6
3	z	w	7

Cross-Join Example



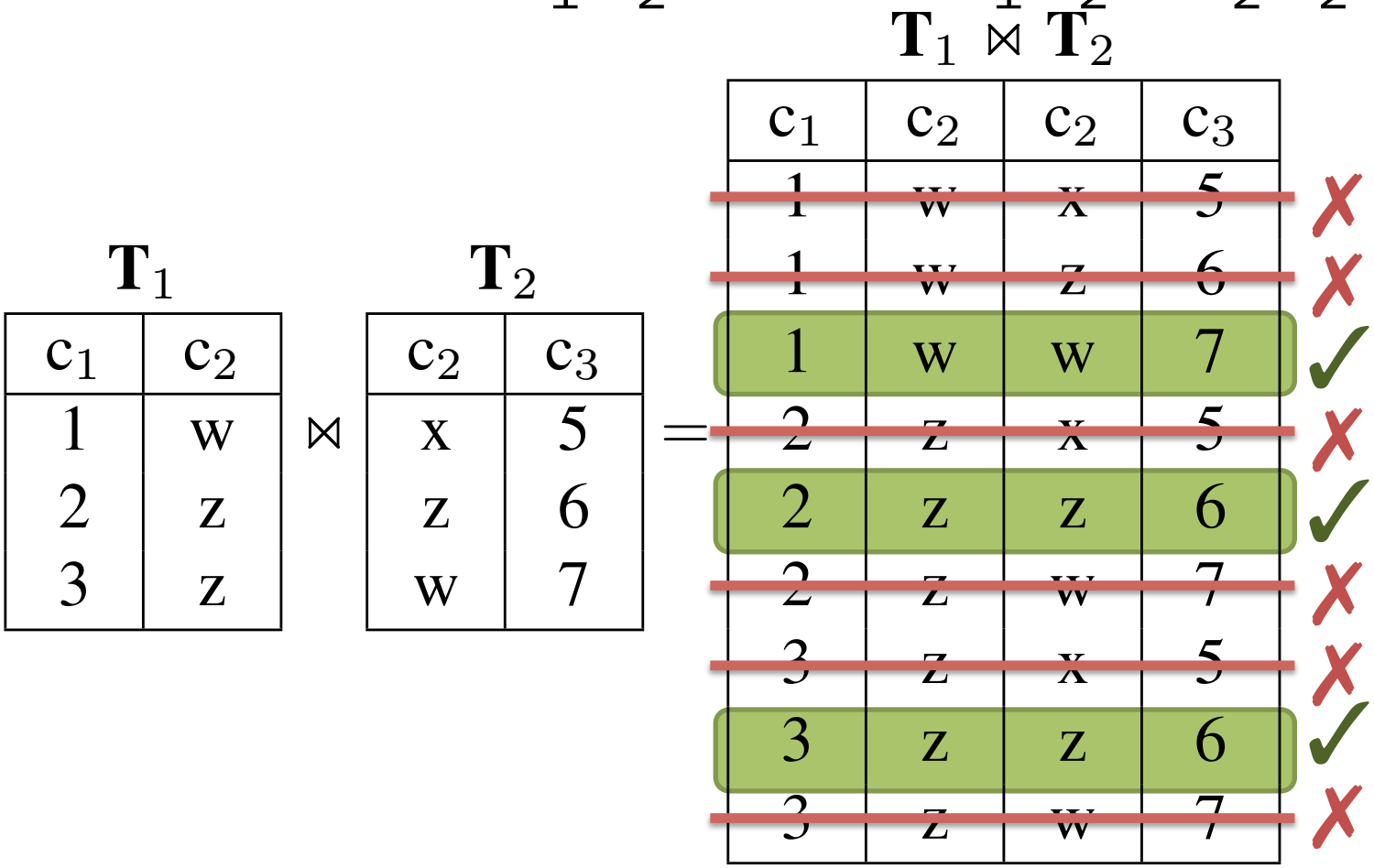
Restrict Result with Predicate

- SELECT * FROM T_1,T_2 WHERE $T_1.c_2 = T_2.c_2$



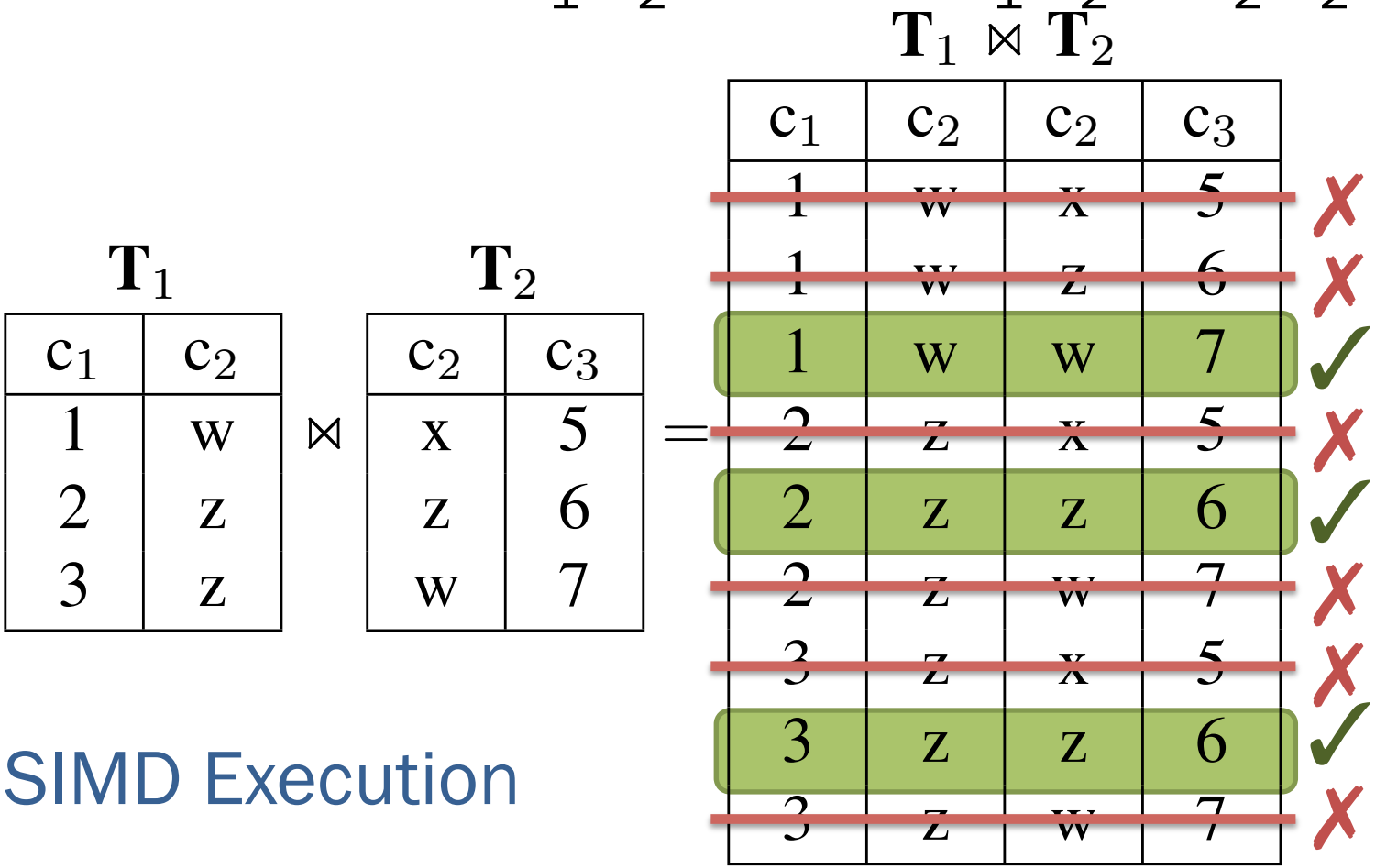
Restrict Result with Predicate

- SELECT * FROM T_1,T_2 WHERE $T_1.c_2 = T_2.c_2$



Restrict Result with Predicate

- SELECT * FROM T_1,T_2 WHERE $T_1.c_2 = T_2.c_2$



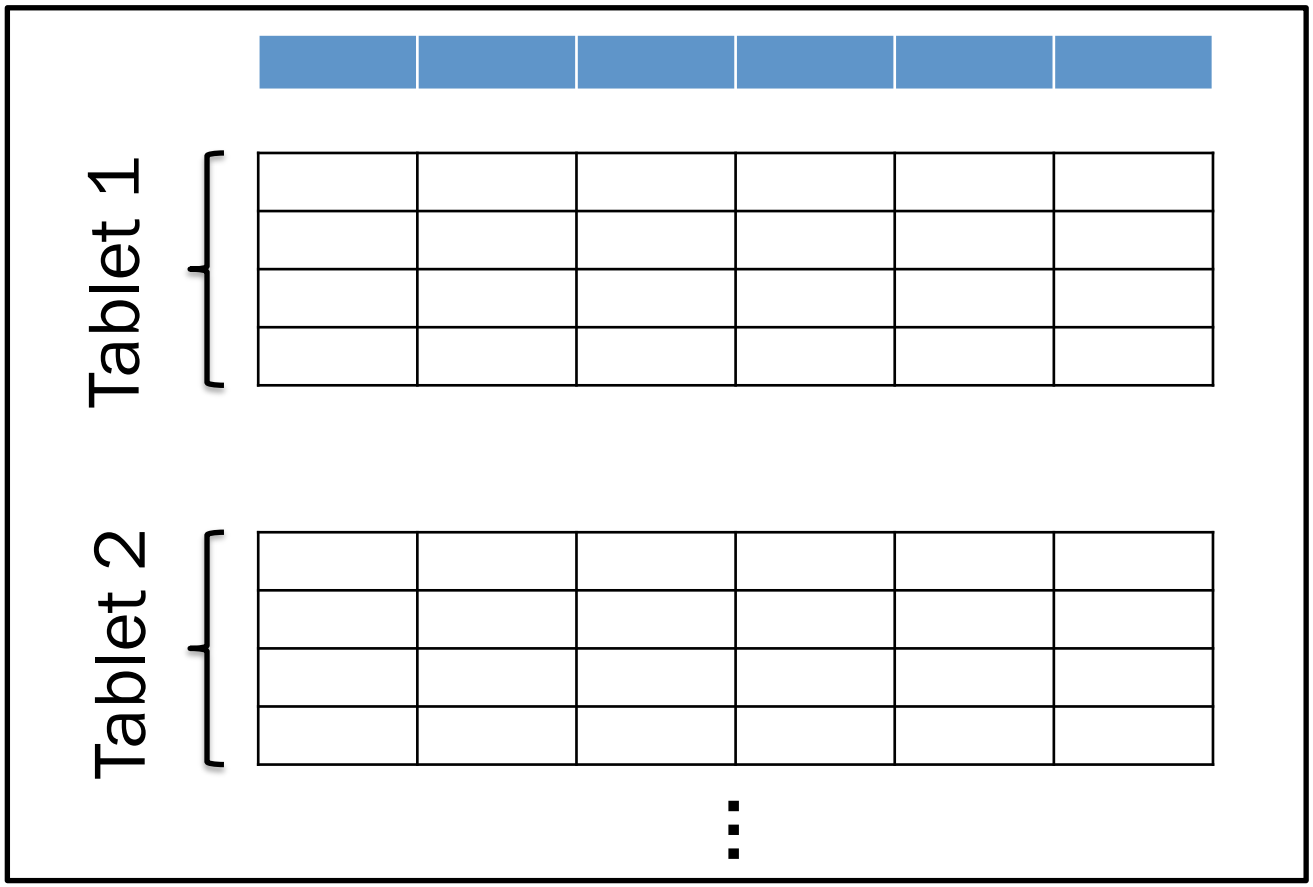
SIMD Execution

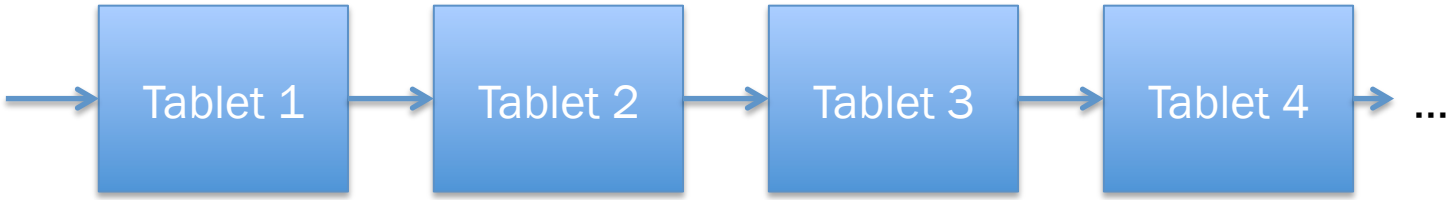
IMPLEMENTATION

Virginian Database

- Written by Peter Bakkum, NEC Labs, New Jersey
- Virtual Machine-based Implementation
  - Similar to SQLite
- Demonstrated speedup for single-table queries
- Presented “Tablet” data structure for efficient processing on CPU/GPU

Table 1





Query Workflow

Input Query

Query Workflow

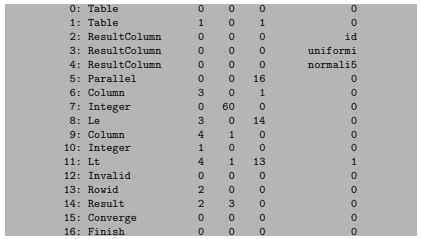
Input Query

```
  0: Table          0    0    0          0
  1: Table          1    0    1          0
  2: ResultColumn   0    0    0         id
  3: ResultColumn   0    0    0   uniformi
  4: ResultColumn   0    0    0   normali5
  5: Parallel       0    0   16          0
  6: Column         3    0    1          0
  7: Integer        0   60    0          0
  8: Le             3    0   14          0
  9: Column         4    1    0          0
 10: Integer        1    0    0          0
 11: Lt             4    1   13          1
 12: Invalid        0    0    0          0
 13: Rowid          2    0    0          0
 14: Result         2    3    0          0
 15: Converge       0    0    0          0
 16: Finish         0    0    0          0
```

Virtual Machine Program

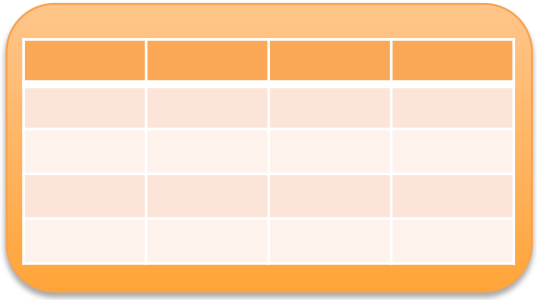
Query Workflow

Input Query



```
  0: Table          0   0   0          0
  1: Table          1   0   1          0
  2: ResultColumn   0   0   0         id
  3: ResultColumn   0   0   0   uniformi
  4: ResultColumn   0   0   0   normali5
  5: Parallel       0   0  16          0
  6: Column         3   0   1          0
  7: Integer        0  60   0          0
  8: Le             3   0  14          0
  9: Column         4   1   0          0
 10: Integer        1   0   0          0
 11: Lt             4   1  13          1
 12: Invalid        0   0   0          0
 13: Rowid          2   0   0          0
 14: Result         2   3   0          0
 15: Converge       0   0   0          0
 16: Finish         0   0   0          0
```

Virtual Machine Program

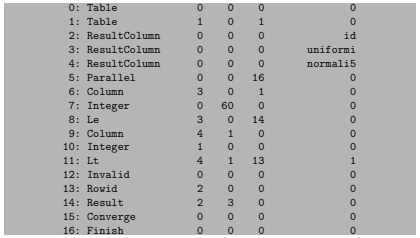


Set Up Result Table

Query Workflow

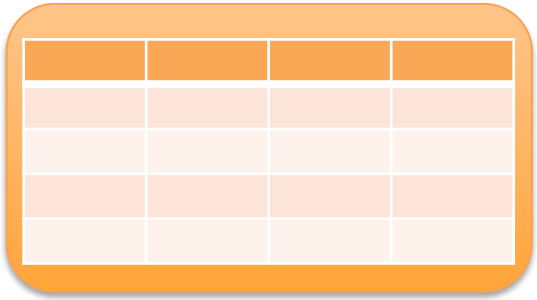
Input Query

Execute



```
  0: Table          0   0   0          0
  1: Table          1   0   1          0
  2: ResultColumn   0   0   0         id
  3: ResultColumn   0   0   0   uniformi
  4: ResultColumn   0   0   0   normali5
  5: Parallel       0   0  16          0
  6: Column         3   0   1          0
  7: Integer        0  60   0          0
  8: Le             3   0  14          0
  9: Column         4   1   0          0
 10: Integer        1   0   0          0
 11: Lt             4   1  13          1
 12: Invalid        0   0   0          0
 13: Rowid          2   0   0          0
 14: Result         2   3   0          0
 15: Converge       0   0   0          0
 16: Finish         0   0   0          0
```

Virtual Machine Program



Set Up Result Table

SQL → OPCODE Program

```
SELECT test.id, test1.uniformi, test.normali5 FROM test,test1
                                    WHERE test1.uniformi > 60 AND test.normali5 < 0;
```

SQL → OPCODE Program

SELECT test.id, test1.uniformi, test.normali5 FROM test,test1
WHERE test1.uniformi > 60 AND test.normali5 < 0;

```
 0: Table             0    0    0             0
 1: Table             1    0    1             0
 2: ResultColumn      0    0    0            id
 3: ResultColumn      0    0    0      uniformi
 4: ResultColumn      0    0    0      normali5
 5: Parallel          0    0   16             0
 6: Column            3    0    1             0
 7: Integer           0   60    0             0
 8: Le                3    0   14             0
 9: Column            4    1    0             0
10: Integer           1    0    0             0
11: Lt                4    1   13             1
12: Invalid           0    0    0             0
13: Rowid             2    0    0             0
14: Result            2    3    0             0
15: Converge          0    0    0             0
16: Finish            0    0    0             0
```

SQL → OPCODE Program

SELECT test.id, test1.uniformi, test.normali5 FROM test,test1
WHERE test1.uniformi > 60 AND test.normali5 < 0;

```
 0: Table          0    0    0          0
 1: Table          1    0    1          0
 2: ResultColumn   0    0    0         id
 3: ResultColumn   0    0    0   uniformi
 4: ResultColumn   0    0    0   normali5
 5: Parallel       0    0   16          0
 6: Column         3    0    1          0
 7: Integer        0   60    0          0
 8: Le             3    0   14          0
 9: Column         4    1    0          0
10: Integer        1    0    0          0
11: Lt             4    1   13          1
12: Invalid        0    0    0          0
13: Rowid          2    0    0          0
14: Result         2    3    0          0
15: Converge       0    0    0          0
16: Finish         0    0    0          0
```

SQL → OPCODE Program

```
SELECT test.id, test1.uniformi, test.normali5 FROM test,test1
                                WHERE test1.uniformi > 60 AND test.normali5 < 0;
```

```
 0: Table          0     0    0          0
 1: Table          1     0    1          0
 2: ResultColumn   0     0    0         id
 3: ResultColumn   0     0    0   uniformi
 4: ResultColumn   0     0    0   normali5
 5: Parallel       0     0    16         0
 6: Column         3     0    1          0
 7: Integer        0     60   0          0
 8: Le             3     0    14         0
 9: Column         4     1    0          0
10: Integer        1     0    0          0
11: Lt             4     1    13         1
12: Invalid        0     0    0          0
13: Rowid          2     0    0          0
14: Result         2     3    0          0
15: Converge       0     0    0          0
16: Finish         0     0    0          0
```

Compute for each row

Allocating Result Table

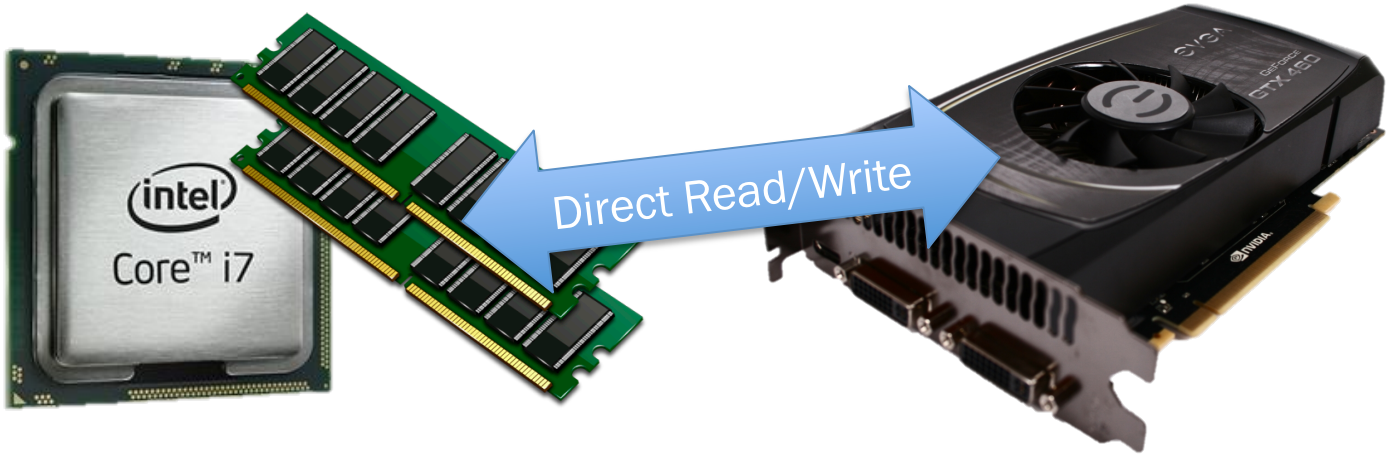
- How much memory must be allocated?

$$||TableA|| \cdot ||TableB||$$

- If each table has 3500 rows, then we have...

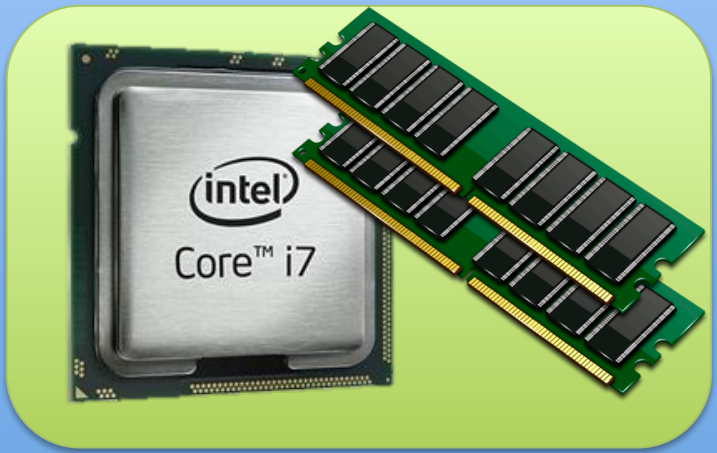
$$12\ 250\ 000\ \text{rows}$$

- Memory limits of GPU: use mapped memory!



Virtual Machine

```
     0: Table            0    0    0           0
     1: Table            1    0    1           0
     2: ResultColumn     0    0    0          id
     3: ResultColumn     0    0    0    uniformi
     4: ResultColumn     0    0    0    normali5
     5: Parallel         0    0   16           0
     6: Column           3    0    1           0
     7: Integer          0   60    0           0
     8: Le               3    0   14           0
     9: Column           4    1    0           0
    10: Integer          1    0    0           0
    11: Lt               4    1   13           1
    12: Invalid          0    0    0           0
    13: Rowid            2    0    0           0
    14: Result           2    3    0           0
    15: Converge         0    0    0           0
    16: Finish           0    0    0           0
```



Virtual Machine

0: Table	0	0	0	0
1: Table	1	0	1	0
2: ResultColumn	0	0	0	id
3: ResultColumn	0	0	0	uniformi
4: ResultColumn	0	0	0	normali5
5: Parallel	0	0	16	0
6: Column	3	0	1	0
7: Integer	0	60	0	0
8: Parallel	0	0	16	0
9: Column	4	1	0	0
10: Integer	1	0	0	0
11: Lt	4	1	13	1
12: Invalid	0	0	0	0
13: Rowid	2	0	0	0
14: Result	2	3	0	0
15: Converge	0	0	0	0
16: Finish	0	0	0	0

Parallel/Converge

0: Table	0	0	0	0
1: Table	1	0	1	0
2: ResultColumn	0	0	0	id
3: ResultColumn	0	0	0	uniformi
4: ResultColumn	0	0	0	normali5
5: Parallel	0	0	16	0
6: Column	3	0	1	0
7: Integer	0	60	0	0
8: Parallel	0	0	16	0
9: Column	4	1	0	0
10: Integer	1	0	0	0
11: Lt	4	1	13	1
12: Invalid	0	0	0	0
13: Rowid	2	0	0	0
14: Result	2	3	0	0
15: Converge	0	0	0	0
16: Finish	0	0	0	0

Parallel/Converge

Virtual Machine


```
 0: Table          0    0    0          0
 1: Table          1    0    1          0
 2: ResultColumn   0    0    0         id
 3: ResultColumn   0    0    0   uniformi
 4: ResultColumn   0    0    0   normali5
 5: Parallel       0    0   16          0
 6: Column         3    0    1          0
 7: Integer        0   60    0          0
 8: Le             3    0   14          0
 9: Column         4    1    0          0
10: Integer        1    0    0          0
11: Lt             4    1   13          1
12: Invalid        0    0    0          0
13: Rowid          2    0    0          0
14: Result         2    3    0          0
15: Converge       0    0    0          0
16: Finish         0    0    0          0
```

Query Processing

- CPU: Nested Loop Join (NLJ) a la SQLite
- GPU: exploit 3D topological structure of CUDA threads
  - Assign source table to each thread dimension
  - Each thread represents a single entry in the cross-product
  - Write entries satisfying predicate to result table

Thread Mapping Scheme

$\mathbf{T}_1$

c_1	c_2
1	w
2	z
3	z

$\bowtie$

$\mathbf{T}_2$

c_2	c_3
x	5
z	6
w	7

$=$

$\mathbf{T}_1 \bowtie \mathbf{T}_2$

(1,w,x,5)	(1,w,z,6)	(1,w,w,7)
(2,z,x,5)	(2,z,z,6)	(2,z,w,7)
(3,z,x,5)	(3,z,z,6)	(3,z,w,7)

T_1

T_2

Thread Mapping Scheme

$\mathbf{T}_1$

c_1	c_2
1	w
2	z
3	z

$\bowtie$

$\mathbf{T}_2$

c_2	c_3
x	5
z	6
w	7

$=$

Single Kernel Thread

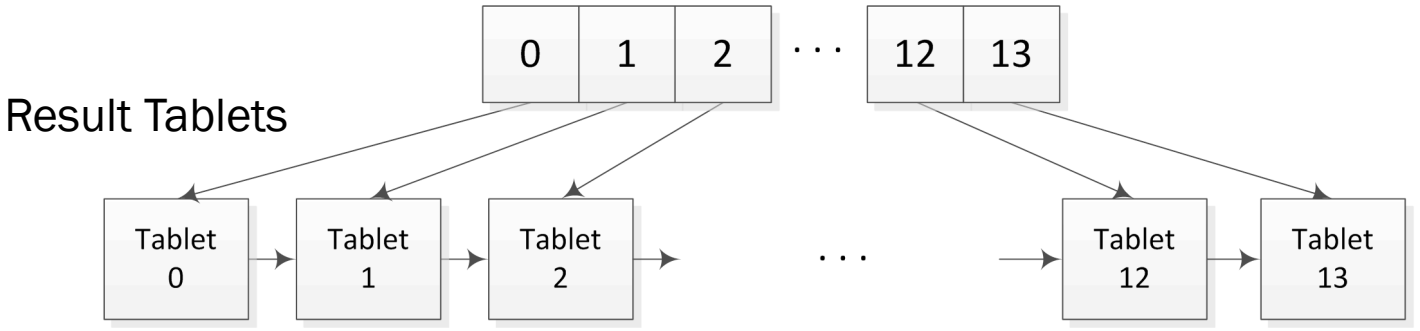
$\mathbf{T}_1 \bowtie \mathbf{T}_2$

(1,w,x,5)	(1,w,z,6)	(1,w,w,7)
(2,z,x,5)	(2,z,z,6)	(2,z,w,7)
(3,z,x,5)	(3,z,z,6)	(3,z,w,7)

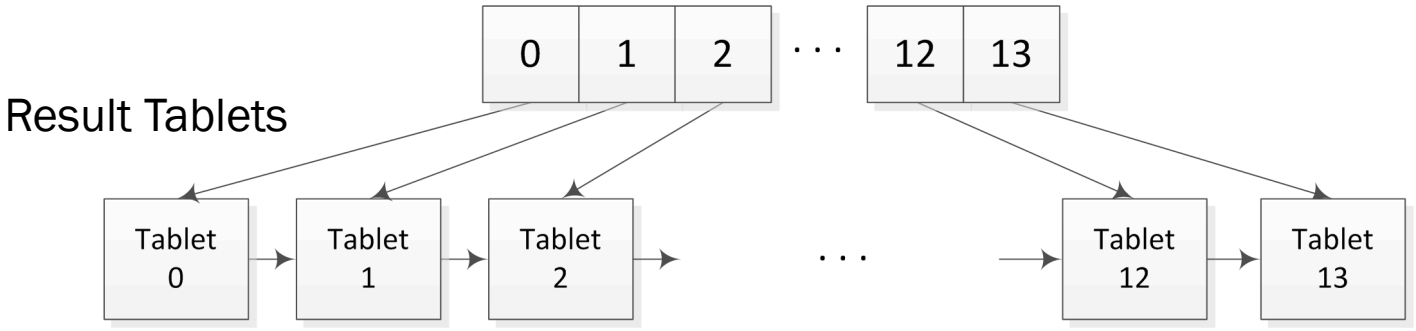
T_1

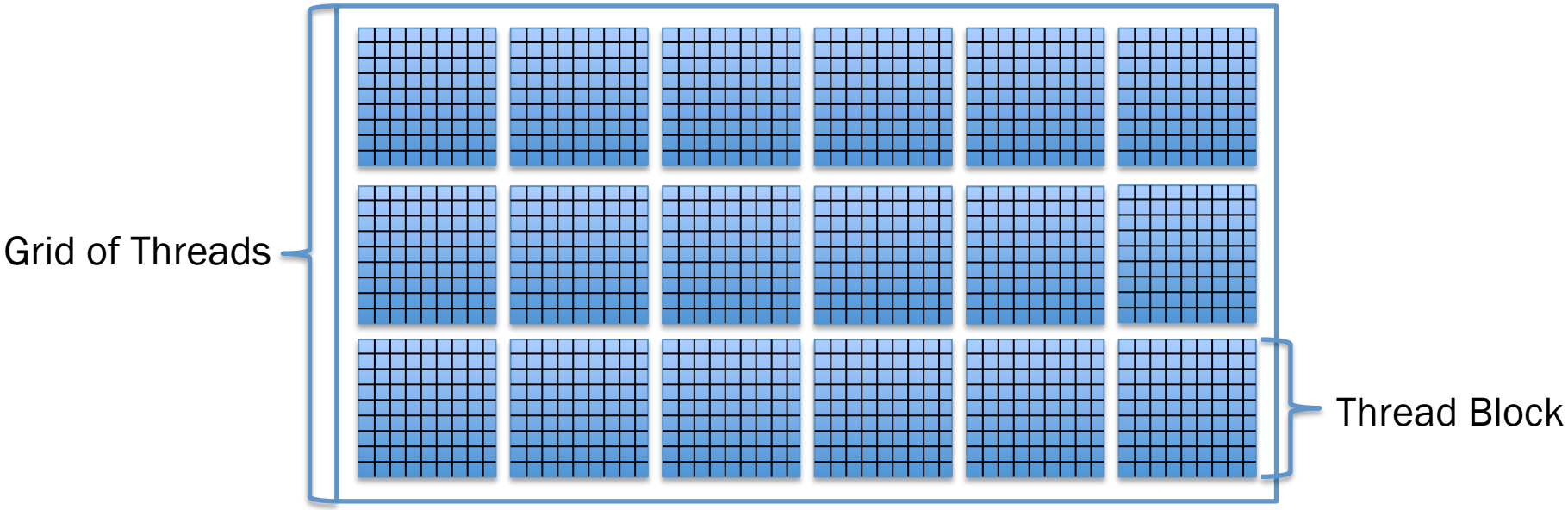
T_2

Writing Results

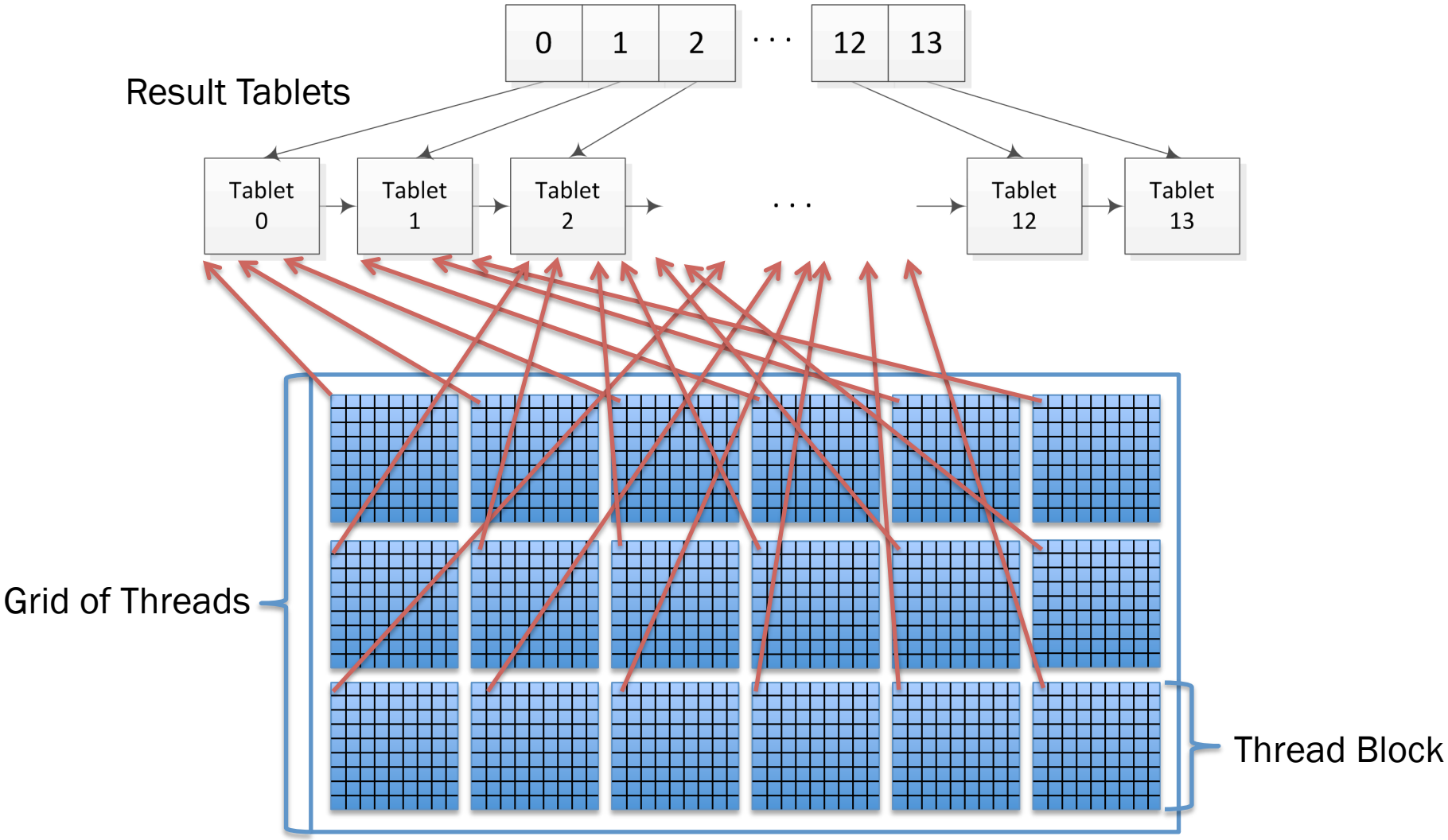


Writing Results

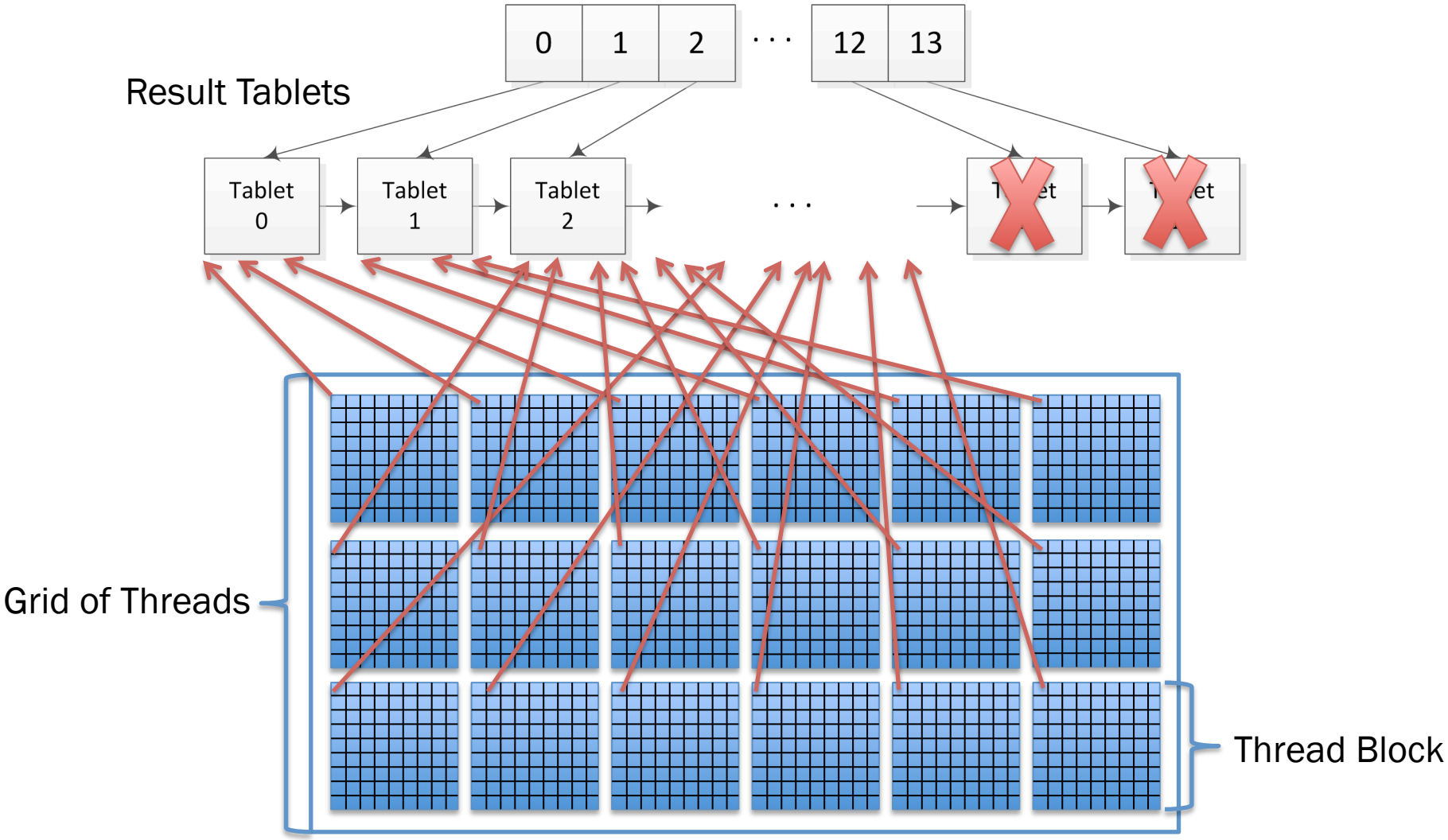




Writing Results



Writing Results



EXPERIMENTAL RESULTS

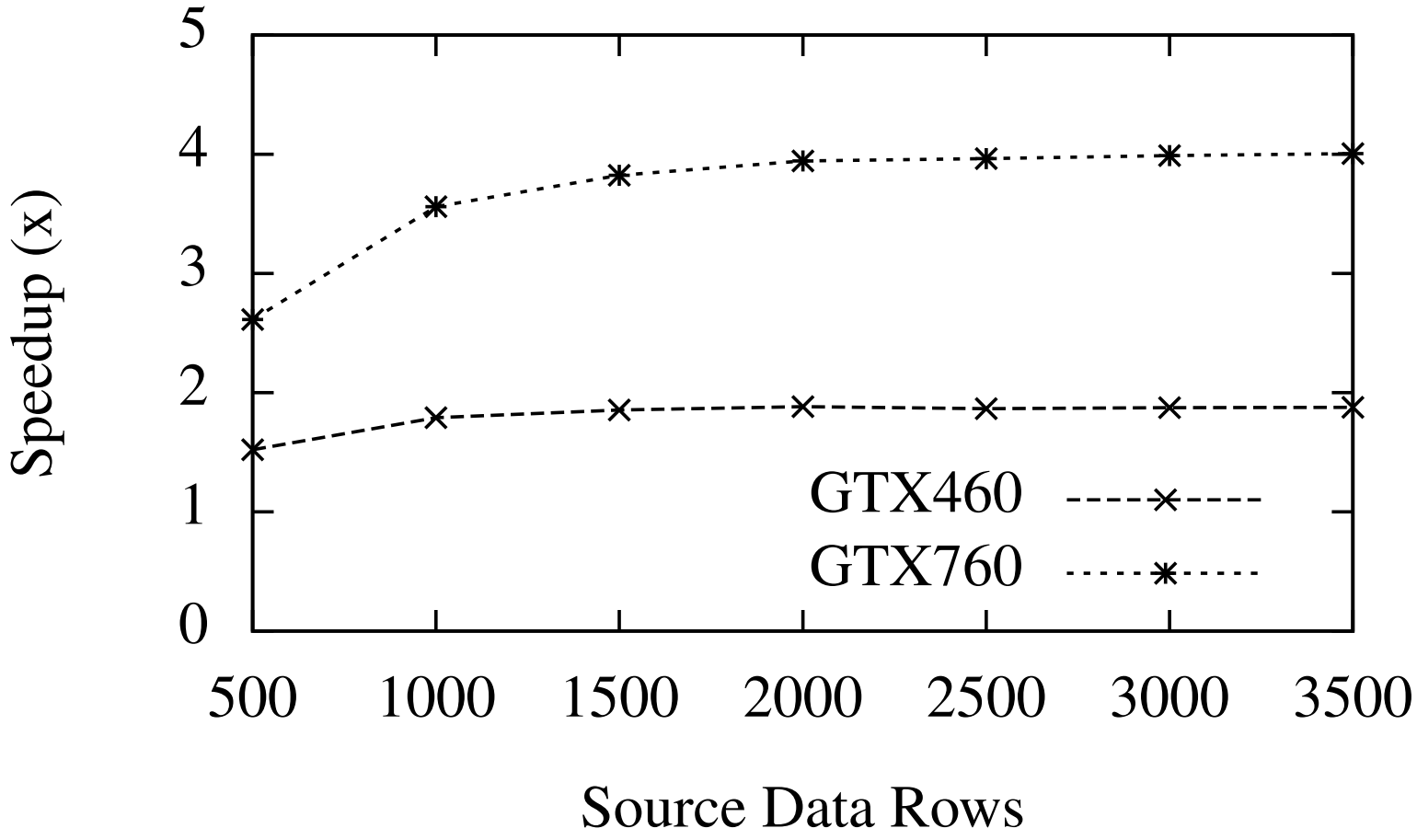
Test Machine

- Intel Core® i7 920 CPU @ 2.66 GHz
- NVIDIA GTX460 GPU
  - Fermi Microarchitecture
  - 336 CUDA Cores
  - 1 GB Memory
- NVIDIA GTX760 GPU
  - Kepler Microarchitecture
  - 1152 CUDA Cores
  - 2 GB Memory
- Linux 3.13.0-39-generic kernel
- CUDA 6.5, NVIDIA 340.29 Driver

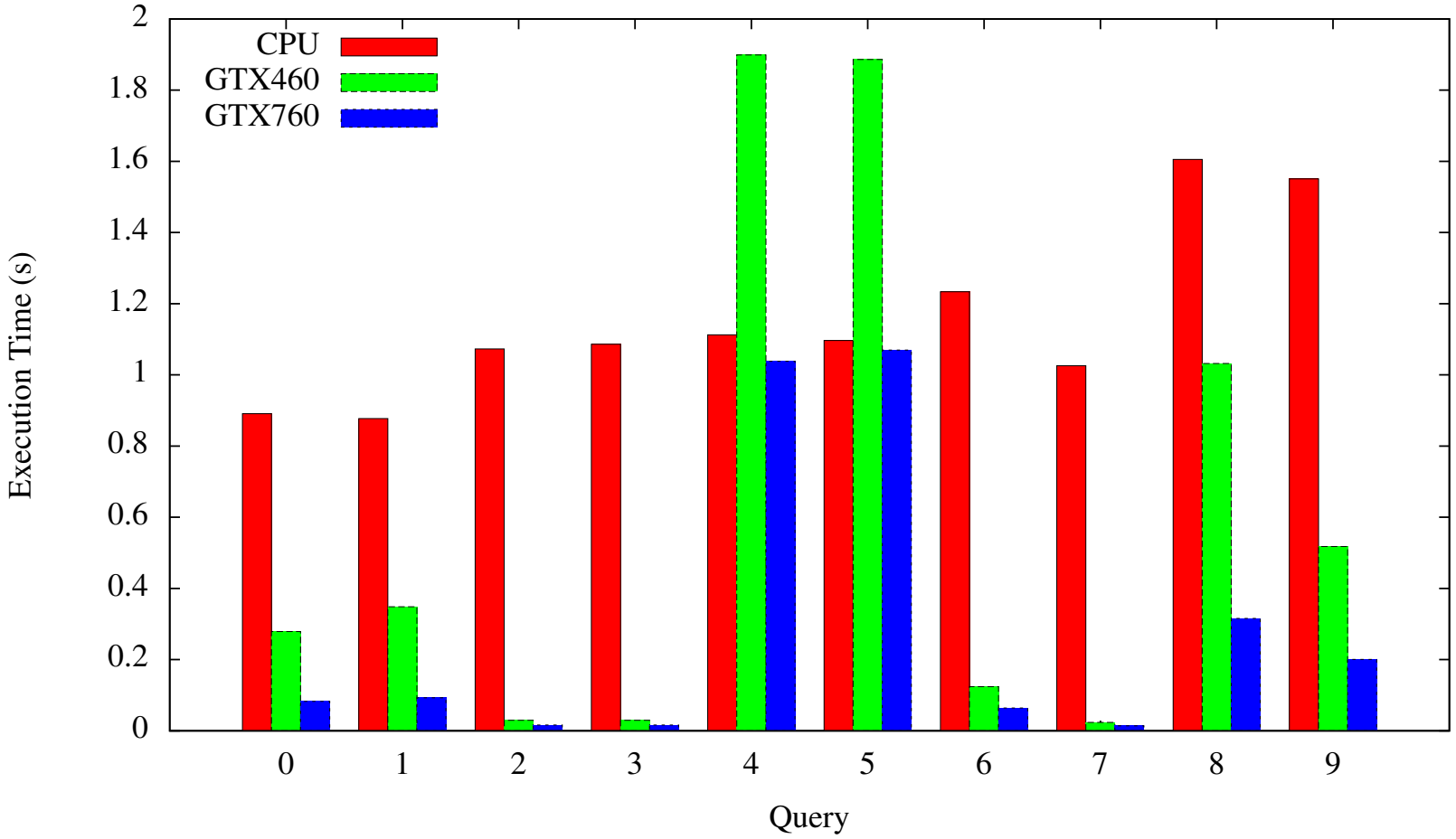
Query Test Suite

- 10 Queries
  - 5 32-bit Integer / 5 32-bit Floating Point
- Random data (Both uniformly, normally distributed)
- All results based of 10 executions

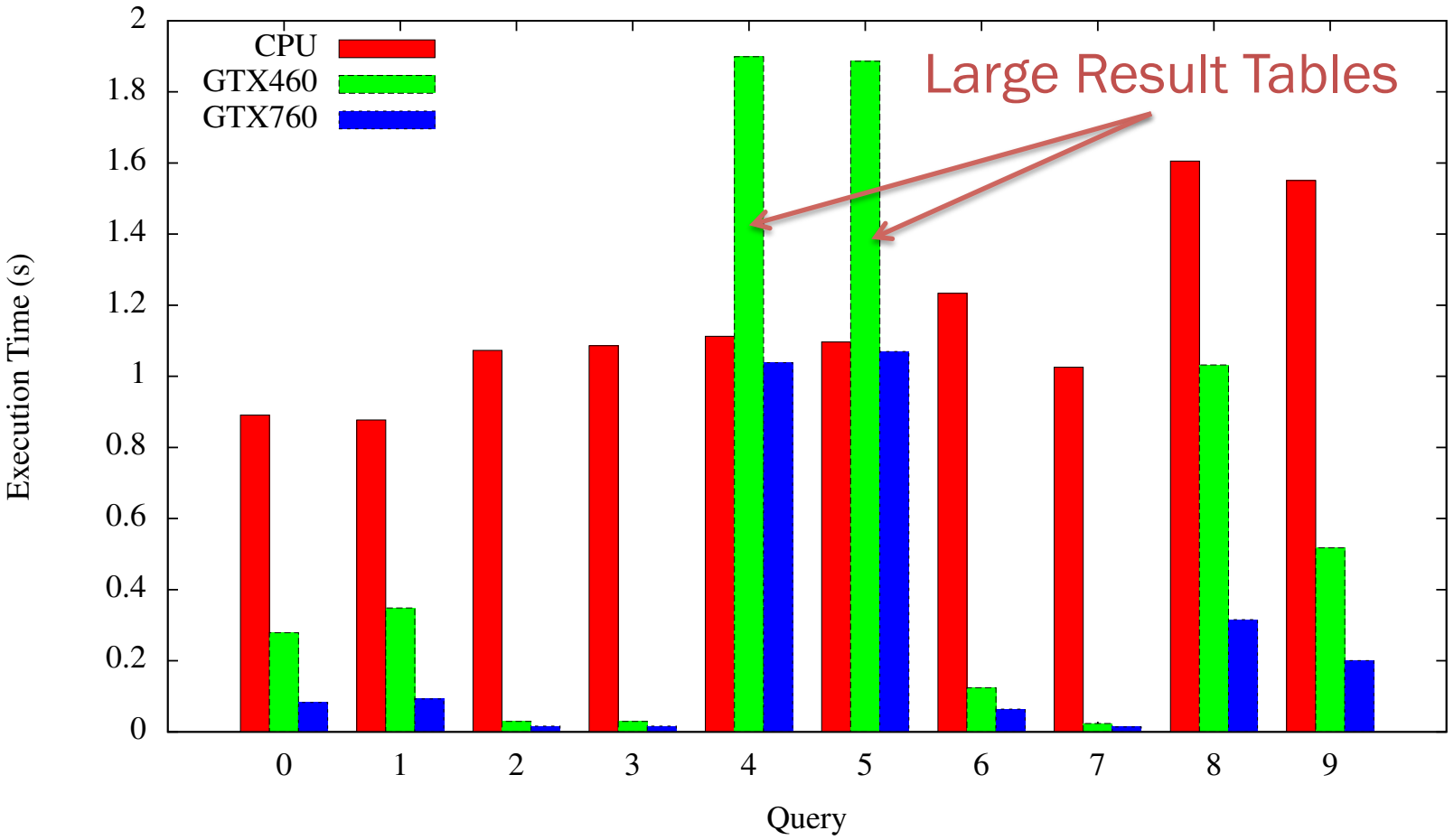
Speedup for Increasing Table Size



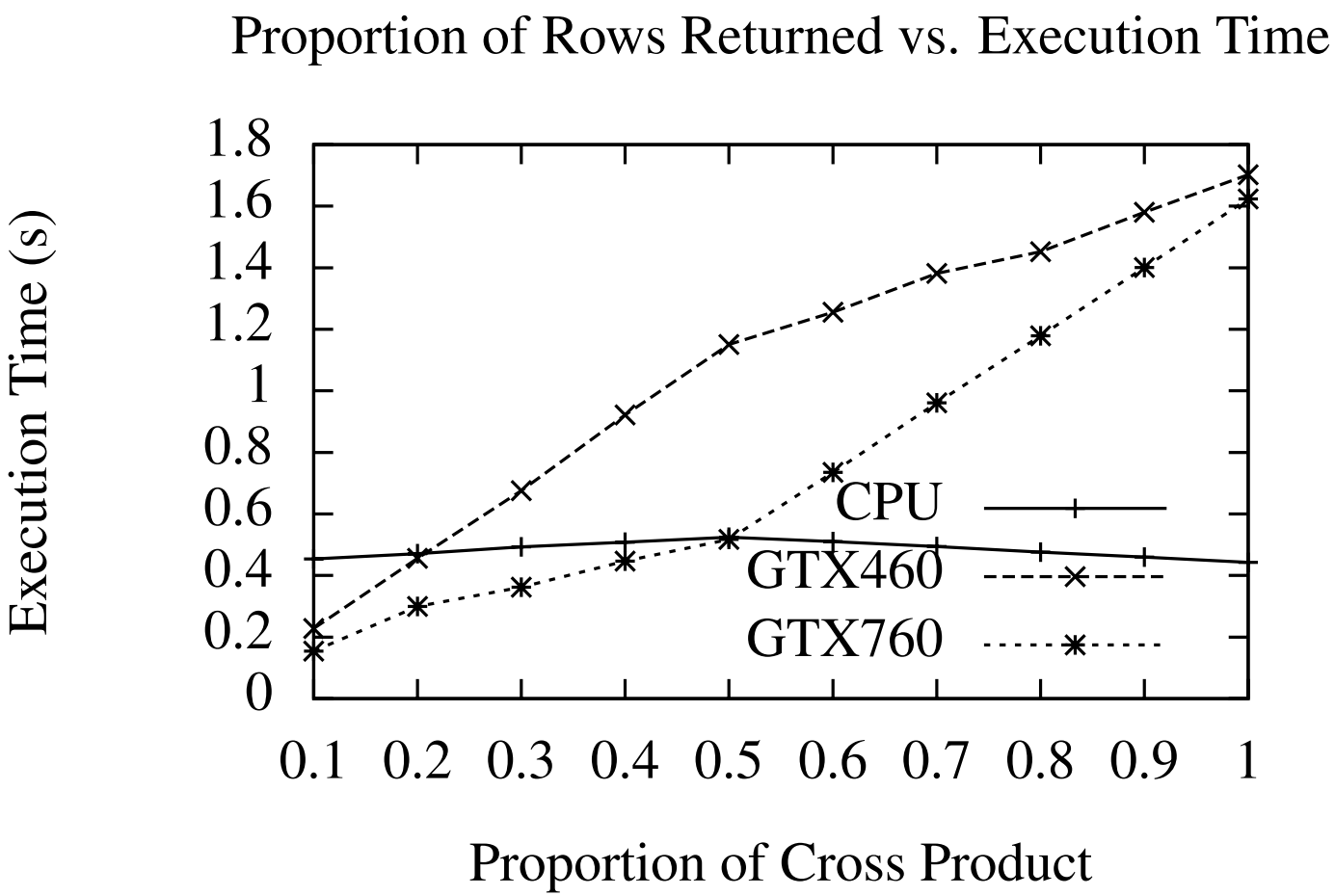
Query Execution Time



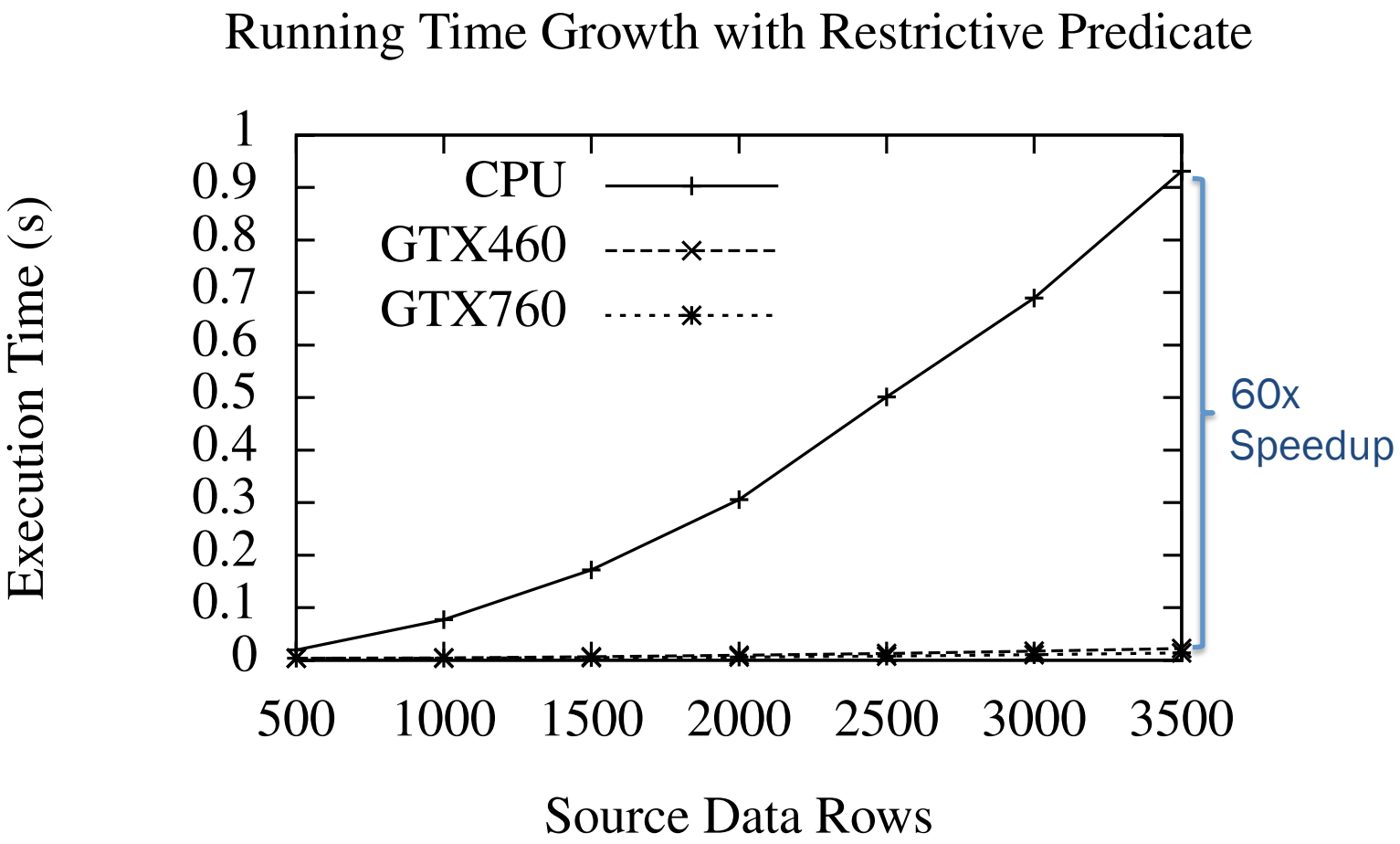
Query Execution Time



PCIe Bottleneck



Natural Join



Conclusions / Future Directions

- GPU implementation of VM-based query processor can be used to accelerate relational database joins
  - 2x-4x on average; 20x-60x in many common cases
- Scalability of a VM-based approach?
- Multiple GPUs to process queries
- Dynamically choose between CPU and GPU

Thank You!

Questions