GPU Processors in Databases (1)

MOLAP based on parallel scan

Krzysztof Kaczmarski

Warsaw University of Technology, Poland

November 2011

The following presentation is based on my three papers:

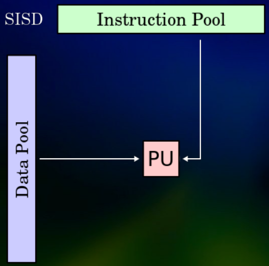
1. K. Kaczmarski. “Comparing GPU and CPU in OLAP Cubes Creation”. In: *SOFSEM*. Ed. by Ivana Cerná et al. Vol. 6543. Lecture Notes in Computer Science. Springer, 2011, pp. 308–319. ISBN: 978-3-642-18380-5
2. K. Kaczmarski and T. Rudny. “MOLAP Cube Based on Parallel Scan Algorithm”. In: *ADBIS*. Ed. by Johann Eder, Mária Bieliková, and A Min Tjoa. Vol. 6909. Lecture Notes in Computer Science. Springer, 2011, pp. 125–138. ISBN: 978-3-642-23736-2
3. K. Kaczmarski. “Experimental B+-tree for GPU”. In: *ADBIS 2011 Research Communications*. Ed. by J. Eder, M. Bielikova, and A.M. Tjoa. Österreichische Computer Gesellschaft, 2011, pp. 232–240. ISBN: 978-3-85403-285-4

Outline of the lecture

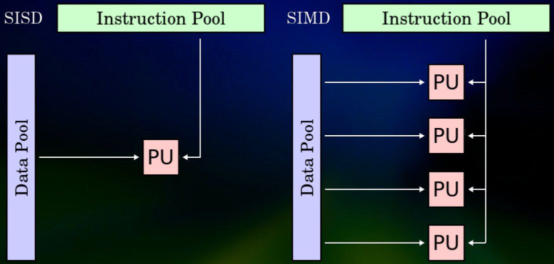
Introduction

GPU and MOLAP databases
- Scan
- MOLAP Cube based on scan primitives
- Results
- Summary

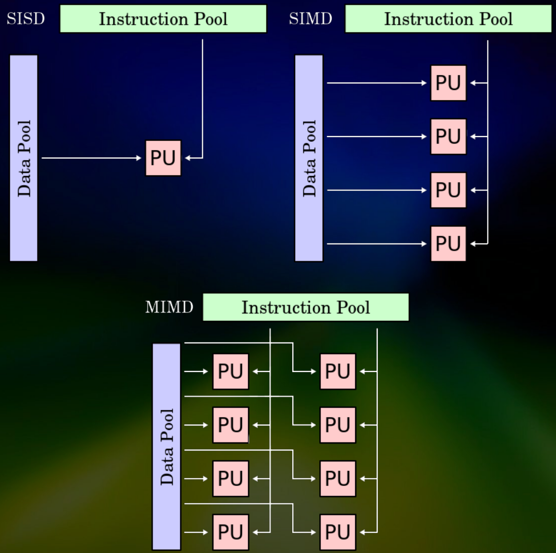
Flynn Taxonomy



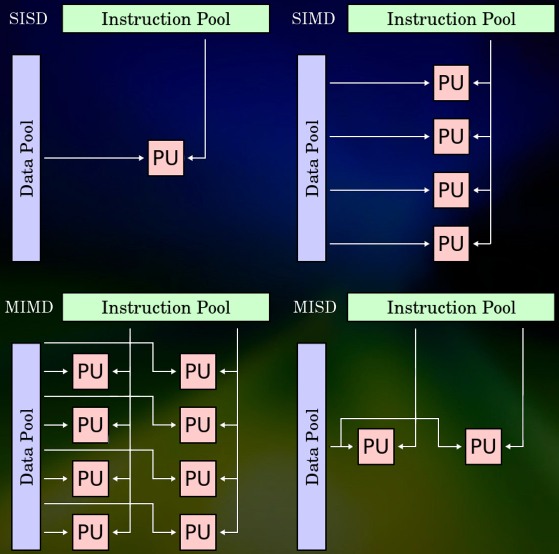
Flynn Taxonomy



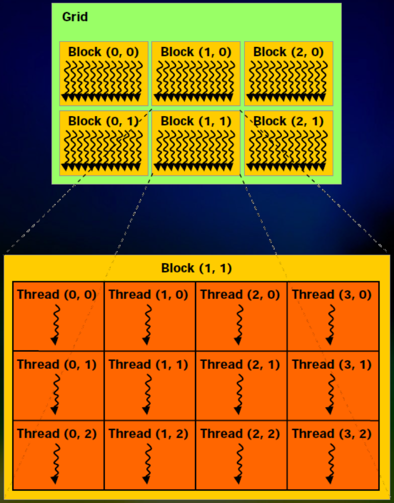
Flynn Taxonomy



Flynn Taxonomy



Grid, Block and Threads



NVIDIA. *CUDA whitepapers*. www.nvidia.com/cuda

Outline of the lecture

Introduction

GPU and MOLAP databases
- Scan
- MOLAP Cube based on scan primitives
- Results
- Summary

Motivation

- Increasing number of real time data

Motivation

- Increasing number of real time data

For Example

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second

Motivation

- Increasing number of real time data

For Example

- Network content download statistics for CDN systems
- 100 k new log entries / second
- Expected to grow exponentially very soon

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations
- Requirement to track changes of statistics in seconds

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations
- Requirement to track changes of statistics in seconds
- Limited budget for statistics gathering with increasing demands

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations
- Requirement to track changes of statistics in seconds
- Limited budget for statistics gathering with increasing demands

For Example

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations
- Requirement to track changes of statistics in seconds
- Limited budget for statistics gathering with increasing demands

For Example

  - Smaller machines favoured for big and expensive clusters

Motivation

- Increasing number of real time data

For Example

  - Network content download statistics for CDN systems
  - 100 k new log entries / second
  - Expected to grow exponentially very soon

- Often unpredictable data dimensions resulting in more expensive computations
- Requirement to track changes of statistics in seconds
- Limited budget for statistics gathering with increasing demands

For Example

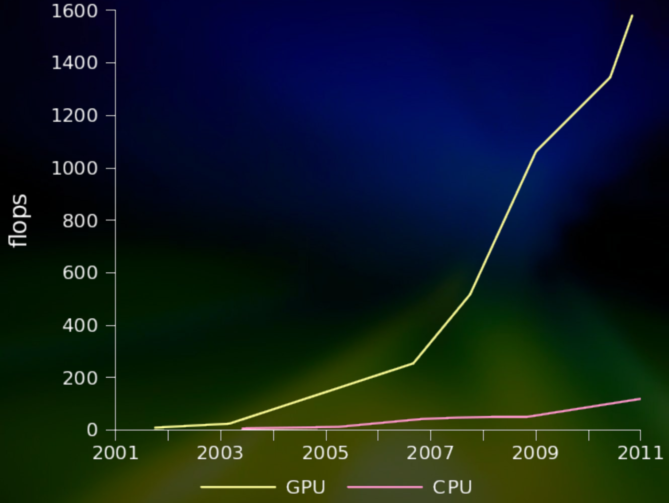
  - Smaller machines favoured for big and expensive clusters
  - Why not use GPUs ?

Why not use GPUs ?

Why not use GPUs ?

NVIDIA, *CUDA whitepapers*

Why not use GPUs ?



(Intel, NVIDIA specs.)

Why not use GPUs ?

- GPU needs dedicated programming

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers
  - Tens thousands+ of developers and counting...

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers
  - Tens thousands+ of developers and counting...
- Time consuming data copying from RAM to GPU

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers
  - Tens thousands+ of developers and counting...
- Time consuming data copying from RAM to GPU
  - Ongoing research on direct I/O operations.

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers
  - Tens thousands+ of developers and counting...
- Time consuming data copying from RAM to GPU
  - Ongoing research on direct I/O operations.
- Not all tasks may be implemented on GPU

Why not use GPUs ?

- GPU needs dedicated programming
  - CUDA is very close to C – “low learning curve”
- Very few skilled developers
  - Tens thousands+ of developers and counting...
- Time consuming data copying from RAM to GPU
  - Ongoing research on direct I/O operations.
- Not all tasks may be implemented on GPU
  - Yes, this is really hard.
    We need a good parallel, separable and efficient algorithm.

GPU Programming

- Parallel primitives are good building blocks for robust and scalable parallel algorithms.

GPU Programming

- Parallel primitives are good building blocks for robust and scalable parallel algorithms.
- They automatically use all available cores as efficiently as possible.

GPU Programming

- Parallel primitives are good building blocks for robust and scalable parallel algorithms.
- They automatically use all available cores as efficiently as possible.
- In this paper we:

GPU Programming

- Parallel primitives are good building blocks for robust and scalable parallel algorithms.
- They automatically use all available cores as efficiently as possible.
- In this paper we:
  - Describe massively parallel algorithm of MOLAP cube creation based on scan primitive

GPU Programming

- Parallel primitives are good building blocks for robust and scalable parallel algorithms.
- They automatically use all available cores as efficiently as possible.
- In this paper we:
  - Describe massively parallel algorithm of MOLAP cube creation based on scan primitive
  - Evaluate its practical application

Prefix sums

Definition
The **scan** operation takes a binary associative operator $\oplus$, and an array of n elements $[x_0, x_1, \ldots, x_{n-1}]$, and returns the array

$$[x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \cdots \oplus x_{n-1})].$$

The **prescan** operation takes a binary associative operator $\oplus$ with identity I, and an array of n elements $[x_0, x_1, \ldots, x_{n-1}]$, and returns the array

$$[I, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \cdots \oplus x_{n-2})].$$

Prefix sums

Definition
The **scan** operation takes a binary associative operator $\oplus$, and an array of n elements $[x_0, x_1, \ldots, x_{n-1}]$, and returns the array

$$[x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \cdots \oplus x_{n-1})].$$

The **prescan** operation takes a binary associative operator $\oplus$ with identity I, and an array of n elements $[x_0, x_1, \ldots, x_{n-1}]$, and returns the array

$$[I, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \cdots \oplus x_{n-2})].$$

There are efficient CUDA based implementations with:

- step complexity $O(\log n)$
- work complexity $O(n)$

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1
prescan(F)	0	0	0	0	1	2	2	2

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1
prescan(F)	0	0	0	0	1	2	2	2
pack(A,F)	8	1	2					

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1
prescan(F)	0	0	0	0	1	2	2	2
pack(A,F)	8	1	2					

There is also a version of scan for segments defined by flags:

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1
prescan(F)	0	0	0	0	1	2	2	2
pack(A,F)	8	1	2					

There is also a version of scan for segments defined by flags:

A	6	3	4	8	1	2	4	2
F	1	0	1	0	0	0	0	1

Prescan example - pack operation

A	6	3	4	8	1	2	4	2
F	0	0	0	1	1	0	0	1
prescan(F)	0	0	0	0	1	2	2	2
pack(A,F)	8	1	2					

There is also a version of scan for segments defined by flags:

A	6	3	4	8	1	2	4	2
F	1	0	1	0	0	0	0	1
seg-scan(A,F)	6	9	4	12	13	15	19	2

Applications of prefix sums primitive

- Computation of minimum, maximum, average, etc. of an array
- Lexical comparison of strings of characters
- Addition of multi-precision numbers that cannot be represented in a single machine word
- Evaluation of polynomials
- Solving of recurrence equations
- Radix sort
- Quick sort
- Solving tridiagonal linear systems
- Removal of marked elements from an array
- Dynamical allocation of processors
- Lexical analysis (parsing into tokens)
- Searching for regular expressions
- Implementation of some tree operations
- and many more...

Dense representation GPU optimised

	d_0	d_1	m_0
r_0	2008	10	23
r_1	2008	12	5
r_2	2008	12	43
r_3	2008	15	8
r_4	2009	15	90
r_5	2009	17	21
r_6	2009	19	3
r_7	2009	19	3

Dense representation GPU optimised

	d_0	d_1	m_0
r_0	2008	10	23
r_1	2008	12	5
r_2	2008	12	43
r_3	2008	15	8
r_4	2009	15	90
r_5	2009	17	21
r_6	2009	19	3
r_7	2009	19	3

	2008	2009
10	23	0
11	0	0
12	48	0
13	0	0
14	0	0
15	8	90
16	0	0
17	0	21
18	0	0
19	0	6

Dense representation GPU optimised

	d_0	d_1	m_0
r_0	2008	10	23
r_1	2008	12	5
r_2	2008	12	43
r_3	2008	15	8
r_4	2009	15	90
r_5	2009	17	21
r_6	2009	19	3
r_7	2009	19	3

	2008	2009
10	23	0
11	0	0
12	48	0
13	0	0
14	0	0
15	8	90
16	0	0
17	0	21
18	0	0
19	0	6

h	c
0	23
2	48
5	8
15	90
17	21
19	6

Creation Algorithm Idea

	d_0	d_1	d_2	d_3	d_4	m_0
r_0	2008	10	04	190	6	23
r_1	2008	10	04	190	6	5
r_2	2008	10	04	190	6	43
r_3	2008	10	04	190	6	8
r_4	2008	10	04	190	8	90
r_5	2008	10	04	190	8	21
r_6	2008	10	05	164	4	3
r_7	2008	10	05	164	4	3

Creation Algorithm Idea

	d_0	d_1	d_2	d_3	d_4	m_0	f
r_0	2008	10	04	190	6	23	1
r_1	2008	10	04	190	6	5	0
r_2	2008	10	04	190	6	43	0
r_3	2008	10	04	190	6	8	0
r_4	2008	10	04	190	8	90	1
r_5	2008	10	04	190	8	21	0
r_6	2008	10	05	164	4	3	1
r_7	2008	10	05	164	4	3	0

Creation Algorithm Idea

	d_0	d_1	d_2	d_3	d_4	m_0	f	ps_f
r_0	2008	10	04	190	6	23	1	0
r_1	2008	10	04	190	6	5	0	1
r_2	2008	10	04	190	6	43	0	1
r_3	2008	10	04	190	6	8	0	1
r_4	2008	10	04	190	8	90	1	1
r_5	2008	10	04	190	8	21	0	2
r_6	2008	10	05	164	4	3	1	2
r_7	2008	10	05	164	4	3	0	3

Creation Algorithm Idea

	$d0$	$d1$	$d2$	$d3$	$d4$	$m0$	f	ps_f	iss_fm0
$r0$	2008	10	04	190	6	23	1	0	79
$r1$	2008	10	04	190	6	5	0	1	56
$r2$	2008	10	04	190	6	43	0	1	51
$r3$	2008	10	04	190	6	8	0	1	8
$r4$	2008	10	04	190	8	90	1	1	111
$r5$	2008	10	04	190	8	21	0	2	21
$r6$	2008	10	05	164	4	3	1	2	6
$r7$	2008	10	05	164	4	3	0	3	3

Creation Algorithm Idea

	d_0	d_1	d_2	d_3	d_4	m_0
r_0	2008	10	04	190	6	23
r_1	2008	10	04	190	6	5
r_2	2008	10	04	190	6	43
r_3	2008	10	04	190	6	8
r_4	2008	10	04	190	8	90
r_5	2008	10	04	190	8	21
r_6	2008	10	05	164	4	3
r_7	2008	10	05	164	4	3

f
1
0
0
0
1
0
1
0

ps_f
0
1
1
1
1
2
2
3

iss_fm_0
79
56
51
8
111
21
6
3

h	c	
132	79	c_0
134	111	c_1
135	6	c_2

Brute Querying Algorithm Idea (max)

	h	c
$c0$	132	79
$c1$	134	111
$c2$	135	6
$c3$	137	15
$c4$	190	98
$c5$	196	4

Brute Querying Algorithm Idea (max)

	h	c	f
$c0$	132	79	1
$c1$	134	111	0
$c2$	135	6	1
$c3$	137	15	0
$c4$	190	98	1
$c5$	196	4	0

Brute Querying Algorithm Idea (max)

	h	c
$c0$	132	79
$c1$	134	111
$c2$	135	6
$c3$	137	15
$c4$	190	98
$c5$	196	4

f
1
0
1
0
1
0

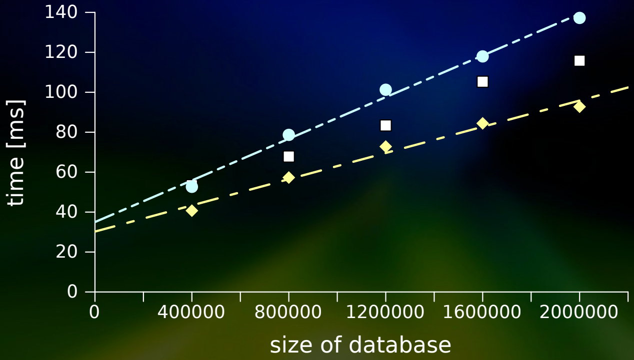
p
79
6
98

Brute Querying Algorithm Idea (max)

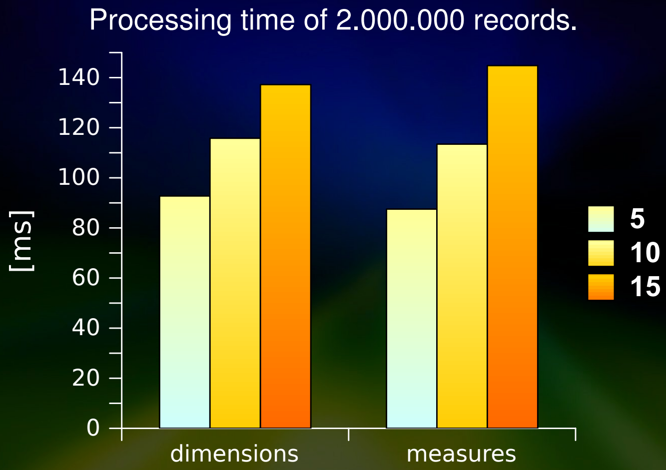
	h	c	f	p	r
$c0$	132	79	1	79	98
$c1$	134	111	0	6	
$c2$	135	6	1	98	
$c3$	137	15	0		
$c4$	190	98	1		
$c5$	196	4	0		

Results – Cube Creation Time

Processing time for 5, 10 and 15 dimensions.



Results – Good scalability



Conclusions

We achieved:

- Ultra-fast cube creation and querying.

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.
- Very general solution due to common scan implementation.

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.
- Very general solution due to common scan implementation.
- Ability to incrementally modify existing cubes.

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.
- Very general solution due to common scan implementation.
- Ability to incrementally modify existing cubes.

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.
- Very general solution due to common scan implementation.
- Ability to incrementally modify existing cubes.

Problems:

- Memory consumption

Conclusions

We achieved:

- Ultra-fast cube creation and querying.
- Scalability much better than in classical CPU based implementations.
- Very general solution due to common scan implementation.
- Ability to incrementally modify existing cubes.

Problems:

- Memory consumption
- **cudpp** library still under development (may speed up)

Future steps

- Reduce instead of scan $\rightarrow$ further speed-up

Future steps

- Reduce instead of scan $\rightarrow$ further speed-up
- Query Engine Improvement

Future steps

- Reduce instead of scan $\rightarrow$ further speed-up
- Query Engine Improvement
  - A new query language designed for vector processing

Future steps

- Reduce instead of scan $\rightarrow$ further speed-up
- Query Engine Improvement
  - A new query language designed for vector processing
- Multiple GPU device implementation $\rightarrow$ horizontal data distribution

Thank you.