Multi-Level Parallel Query Execution Framework for CPU and GPU

Hannes Rauhe[1,3] Jonathan Dees[2,3] Kai-Uwe Sattler[1] Franz Faerber[3]

(1) Ilmenau University of Technology

(2) Karlsruhe Institute of Technology

(3) SAP AG

September 3, 2013

1. Motivation: GPUs for Query Execution

2. Execution of OLAP Queries with OpenCL
   - GPU architecture
   - Compute/Accumulate

3. Results GPU vs. CPU

4. Related Work and Conclusion

Our Motivation is Skepticism

GPUs are good at/with:

- Graphic processing
- Floating point numbers
- SIMD style processing
- Compute intensive algorithms (minor data transfer)

general query processing in a DBMS:

- (often) String processing
- Integers/decimals, strings
- Branching / pipeline breakers
- Data intensive algorithms (mostly touching records once)

- **Can it really be faster? What are the limits?**
- compare the best approach on the CPU with the best one on the GPU

Disclaimer

the best approach and the best implementation for TPC-H: [1]

- hand-written C-Code (Hint: can be generated)
- read-only data structures
- every trick allowed by the TPC-H rules

==> Not a fair comparison to a general DBMS

Q	SF100				SF300			
	We [s]	Top tpch[1]	𝔽	𝕊	We [s]	Top tpch[2]	𝔽	𝕊
1	0.20	1.7	9	5	0.60	3.3	6	6
2	0.01	0.3	30	16	0.01	1.0	100	105
3	0.10	0.2	2	1	0.25	0.8	3	3
4	0.01	0.1	10	5	0.01	0.3	30	32
5	0.05	0.7	14	8	0.41	1.7	4	4
6	0.03	0.1	3	2	0.11	0.3	3	3
7	0.10	0.7	7	4	0.39	1.6	4	4
8	0.02	0.9	45	25	0.02	2.5	125	132
9	0.18	4.6	33	18	0.50	12.2	24	26
10	0.06	2.1	35	19	0.15	7.6	51	53
11	0.03	0.4	13	7	0.08	1.5	19	20

1 Motivation: GPUs for Query Execution

2 Execution of OLAP Queries with OpenCL
- GPU architecture
- Compute/Accumulate

3 Results GPU vs. CPU

4 Related Work and Conclusion

GPU Architecture – external card

- massively parallel -> SIMD, limited possibility of independent threads
- high latency/high bandwidth
- ==> Synchronization is very(!) expensive

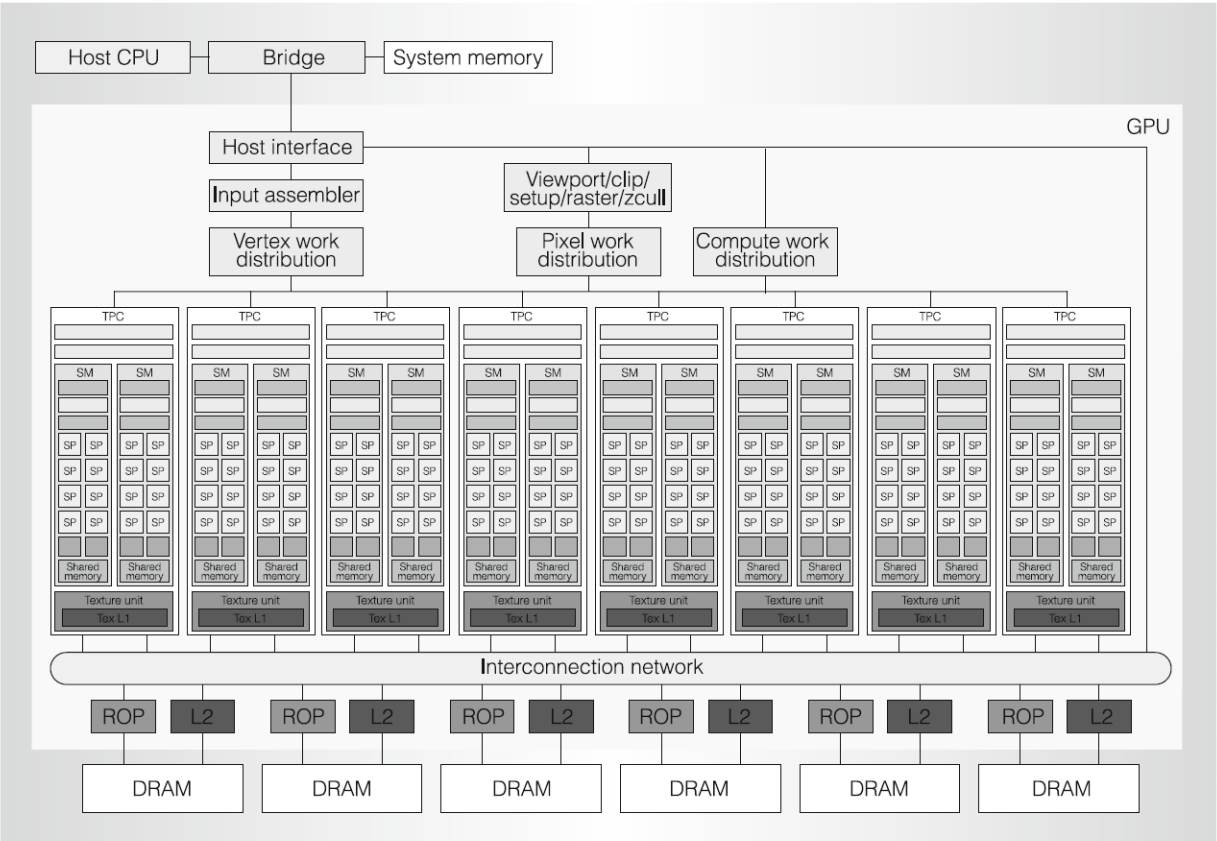
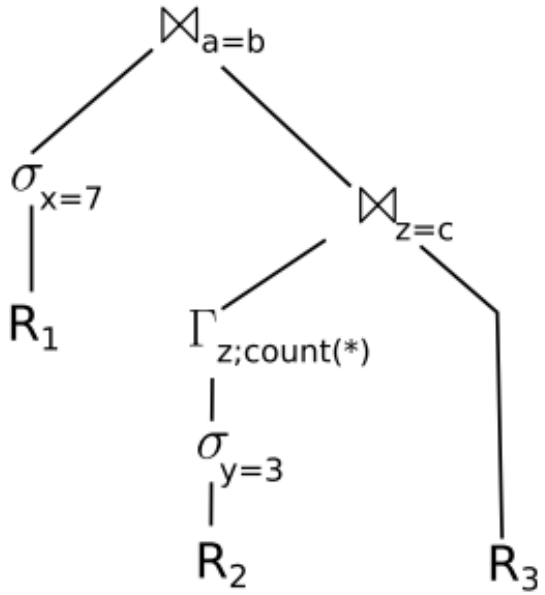


Fig. from Lindholm et al, 2008 [2]

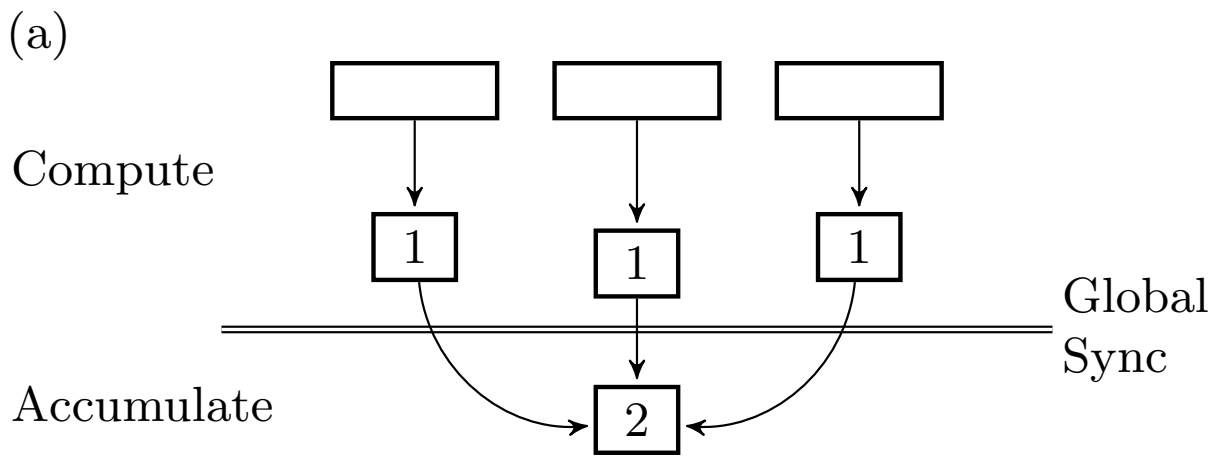
Problem Iterator Model



- tuple-wise: high overhead for synchronization, function calls
- column/vector-wise: high overhead materialization, synchronization
- similar problems on the CPU if I/O is not the bottleneck (Neumann, VLDB 2012 [3]):
- approach: generate/compile executable code for each query

Compute/Accumulate – CPU

- Combine operators until only two are left – process these in parallel internally
- One synchronization point
- logic partitions of biggest table



Example - Q4

Short: how many orders per priority had a commit date smaller than receipt date in the given period?

```
select o_orderpriority, count(*) from orders
where o_orderdate >= [date]
and o_orderdate < [date + 3 months]
and exists (select * from lineitem
 where l_orderkey = o_orderkey
 and l_commitdate < l_receiptdate)
group by o_orderpriority
order by o_orderpriority
```

Example - Q4 - CPU

```
// order is sorted by o_orderdate
PForeach order i with [date] ⩽ o_orderdate(i) < [date]
+3 months do
    for j in o_to_l_orderdate.range(i) do
        if l_commitdate(j) < l_receipdate(j) then
            res[o_orderpriority(i)] += 1
            break inner loop;
```

Procedure Q4 ([date])

- See Dees et al, ICDE 2013 [1]
- How to parallelize on GPUs?

GPU Architecture again

- massively parallel -> SIMD, limited possibility of independent threads
- high latency/high bandwidth
- ==> Synchronization is very(!) expensive

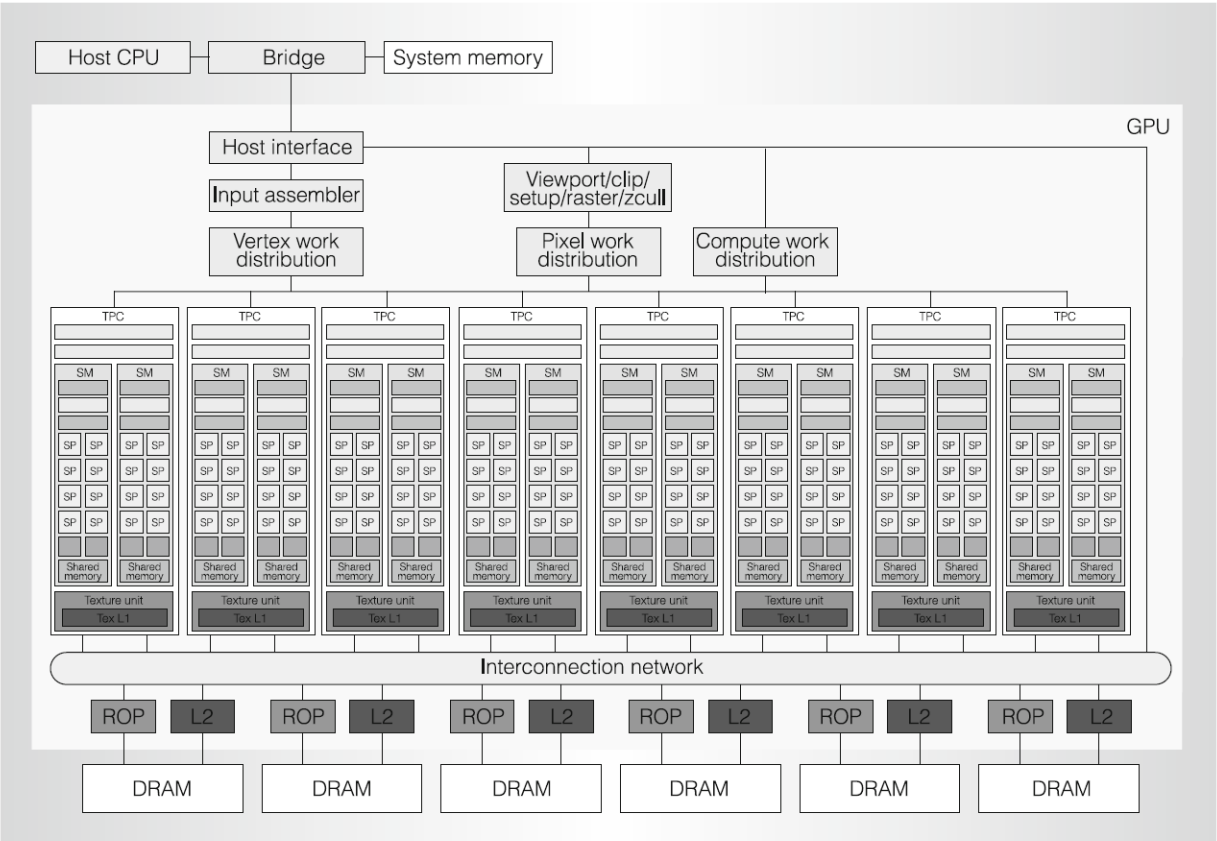
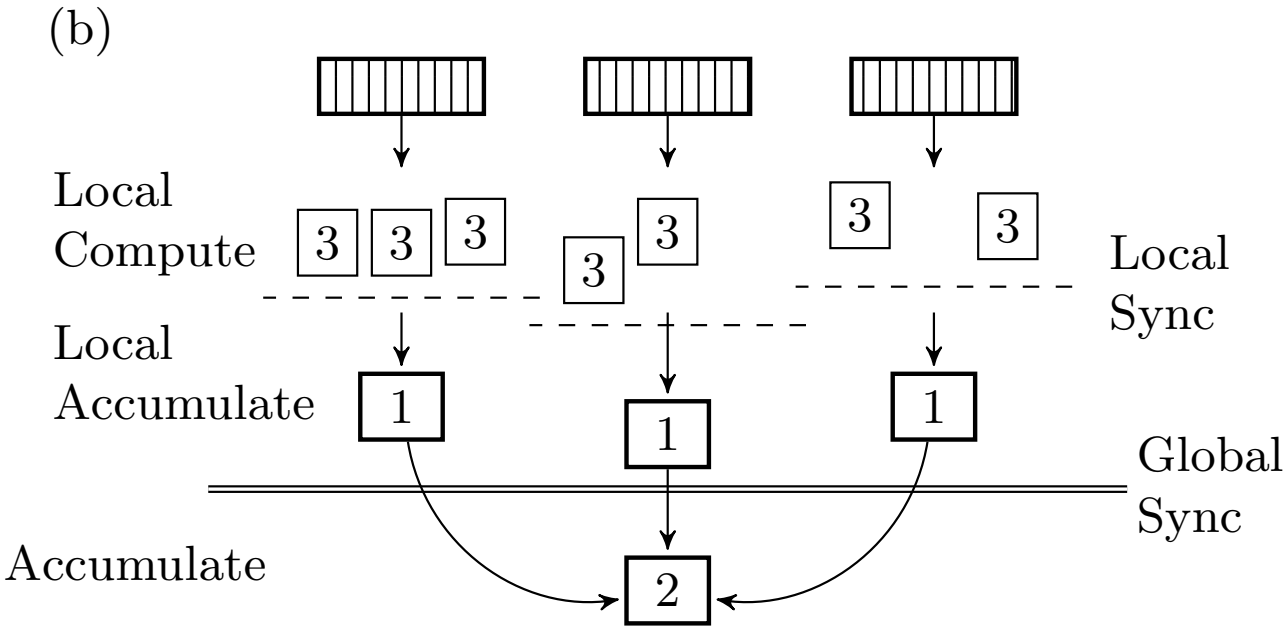


Fig. from Lindholm et al, 2008 [2]

Compute/Accumulate – extended for the GPU

add second level of parallelism (workgroups –> threads)



1 Motivation: GPUs for Query Execution

2 Execution of OLAP Queries with OpenCL
- GPU architecture
- Compute/Accumulate

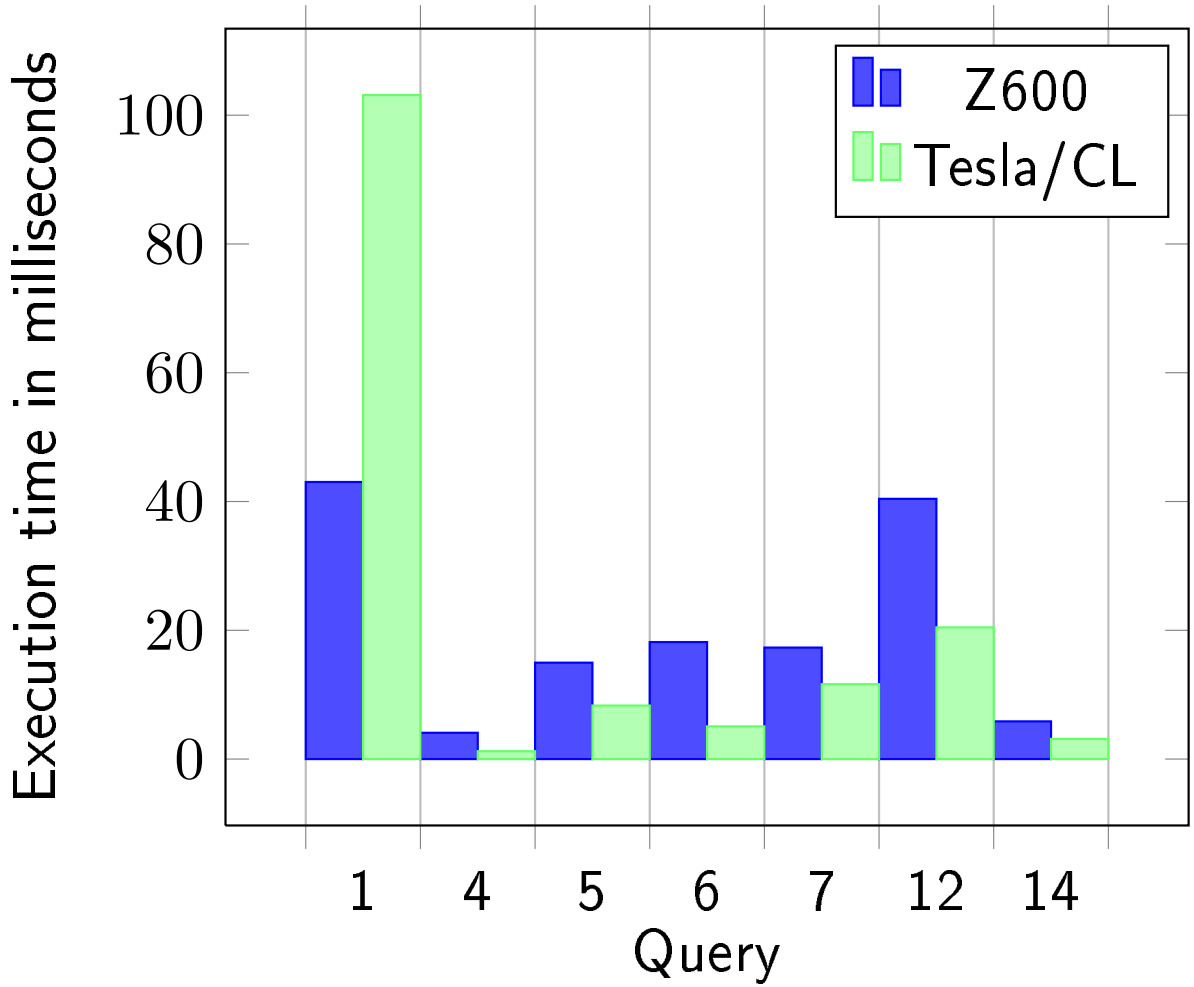
3 Results GPU vs. CPU

4 Related Work and Conclusion

Intermezzo: Comparable Hardware?

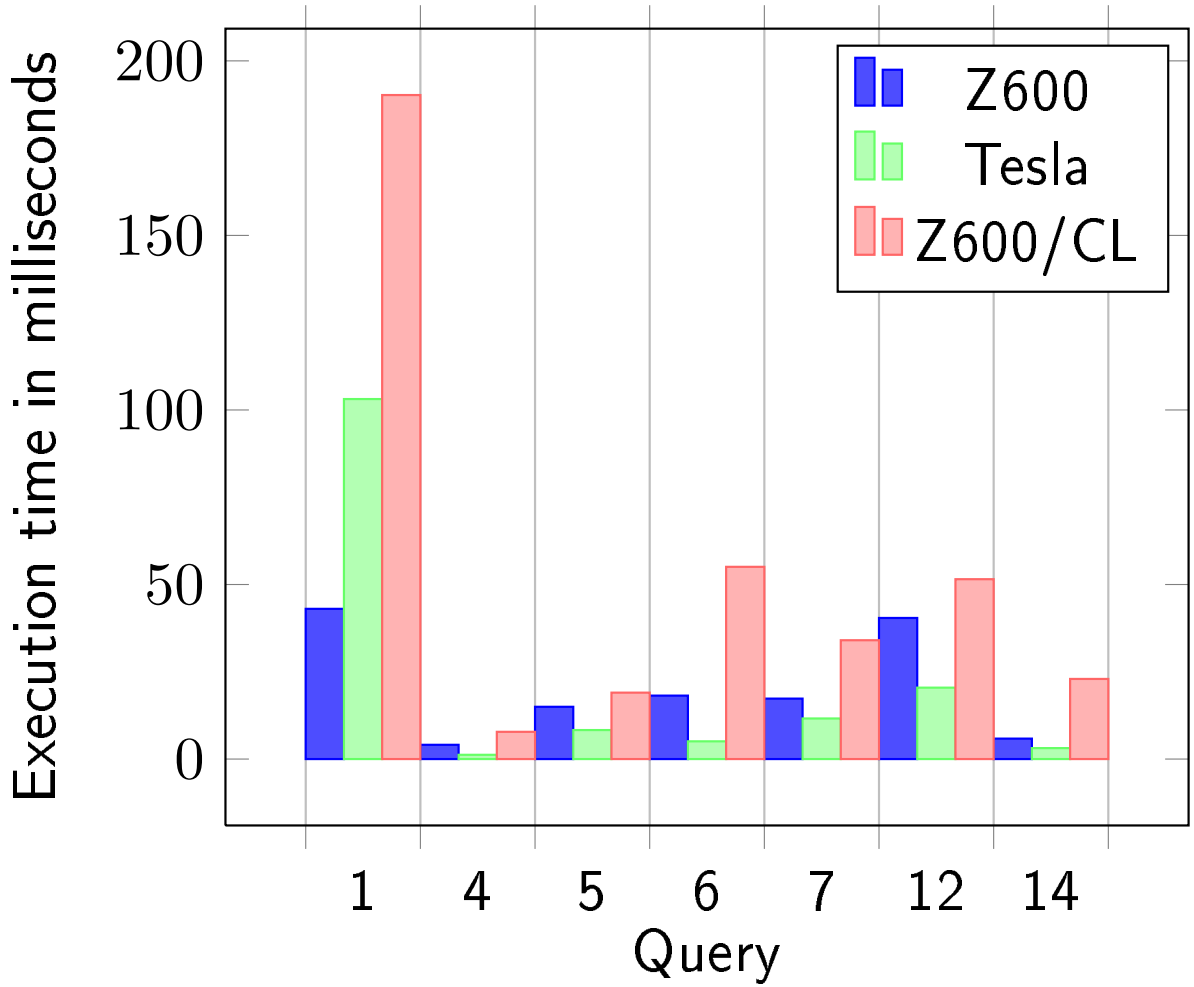
Results

- TPC-H SF10 no Transfer! (**Z600: 2x6-Core Xeon** vs. **Tesla C2050**)



Results

- TPC-H SF10 no Transfer! (**Z600: 2x6-Core Xeon** vs. **Tesla C2050**)



1 Motivation: GPUs for Query Execution

2 Execution of OLAP Queries with OpenCL
- GPU architecture
- Compute/Accumulate

3 Results GPU vs. CPU

4 Related Work and Conclusion

Related Work

- Ocelot (VLDB 2013)
  - MonetDB operators ported to OpenCL
  - every(?) query MonetDB can handle - full TPC-H
  - result: good on hot cache (no data transfer)
  - Query 1 is slower on the GPU than on the CPU
- gpudb (VLDB 2013)
  - SSBM queries (comparable to TPC-H)
  - always faster (with data transfer) compared to MonetDB and OpenCL on the CPU

Conclusion/Future

Can the GPU be faster than the CPU at query execution?

- Yes, but not necessarily (in our set: Q1 was slower)
- Transferring to the GPU is slower than processing
- there are some limits: see paper

Other findings:

- architectures are not that different
  –> OpenCL can be executed on GPU and CPU (One code to rule them all –> the Ocelot approach!)
- the hand-written code can be generated:
- C code Generator already available as a prototype
  –> modify Generator for OpenCL-Code

A little provocation: Can you be faster?

Challenge accepted!

Contact information:

Hannes Rauhe
hannes.rauhe@sap.com – `http://blog.notapaper.de` (Code available)


J. Dees and P. Sanders.
Efficient Many-Core Query Execution in Main Memory Column-Stores.
Proc. of ICDE 2013, 2013.


E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym.
Nvidia tesla: A unified graphics and computing architecture.
Micro, IEEE, 28(2):39–55, 2008.


T. Neumann.
Efficiently compiling efficient query plans for modern hardware.
Proceedings of the VLDB Endowment, 4(9):539–550, 2011.