Institute of Microelectronic Systems
Prof. Dr.-Ing. H. Blume

Leibniz
Universität
Hannover

Real-Time Semi-Global Matching Disparity Estimation on the GPU

Christian Banz, Holger Blume, Peter Pirsch

Motivation

- Stereo Matching is a frequently required task
- Applications
  - Driver Assistance Systems
  - 3D image processing for real-time feedback
  - Keyhole sugery
  - Subtitle placement

- Implementation on off-the-shelf devices for low-cost availability in desktop computing systems: GPUs

Estimation

Identifying the projection point of a 3-D real-world point in two or more images taken from distinct viewpoints

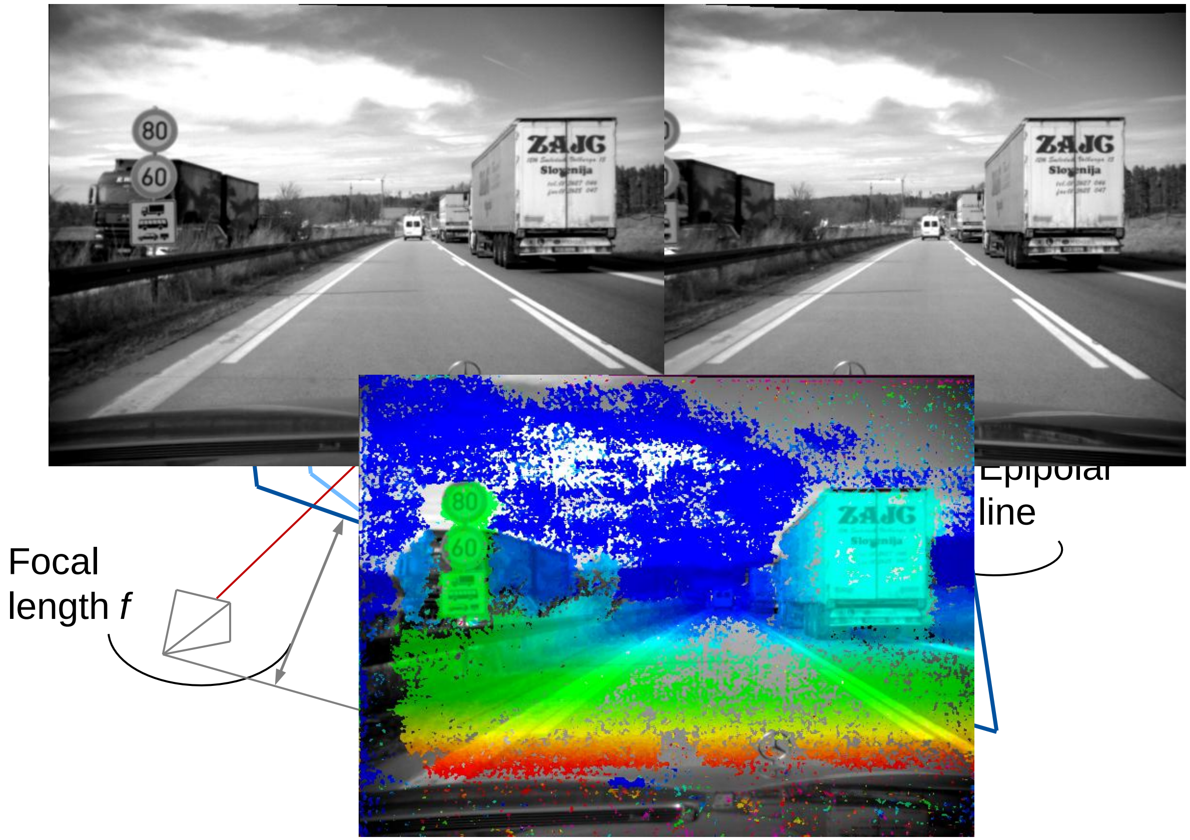
Existing implementations :

GPU [Gibson '08 / Ernst '08]

3 frames/second

CPU/SIMD [Hirschmüller '08]

0.25 frames/second



New features on current GPU architectures

- Rapid development of GPU architecture
- Fermi architecture has many new features
  - Dual warp scheduler
  - Branch prediction
  - Concurrent kernel execution
  - Improved context switching
  - Significantly more bandwidth and computational power

- Investigate SGM implementations for new GPUs

Performance limiting factors

- Software:
  - Effective memory bandwidth usage (i.e. aligned access)
  - Instruction throughput
- Hardware:
  - Latency of the memory interface
  - Latency of the arithmetic pipeline
    - issue rate: 1 instruction/cycle, but latency: several ten cycles
- *Both* has significant influence on performance

- Deep understanding of physical hardware necessary
  - Further reference: Slides by Volkov

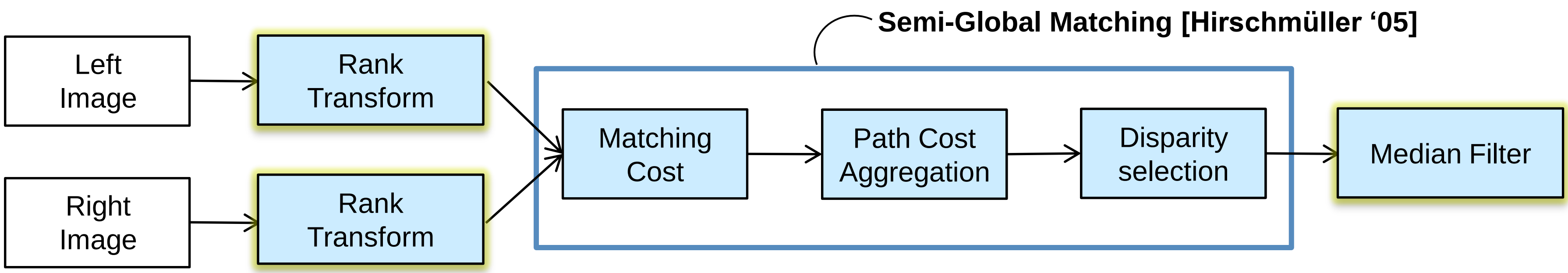
Measures for high performance

- Ensure coalesced and aligned memory access
- Maximize data reuse (very little redundant memory access)
- High ratio of computational instructions to ancillary instructions

- Transfer coherent *chunks* of data to/from memory
- Keep the hardware pipeline filled
  - serial loop with independent arithmetic on local data

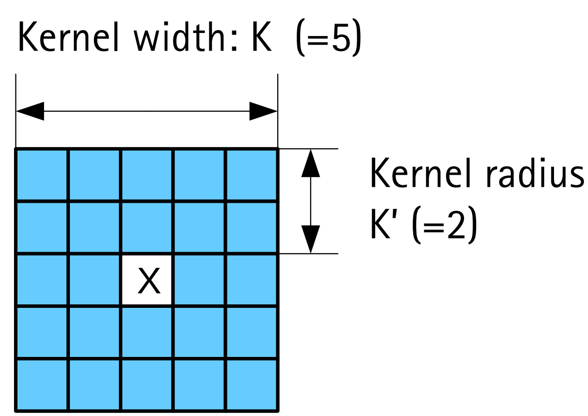
- Occupancy is not a good metric
  - Indicates the amount of ressouces occupied
  - *NOT* the effectiveness of usage

Semi-Global Matching



Rank transform und median filter

- 2D transformations
- non-separable
- requires access to local processing window



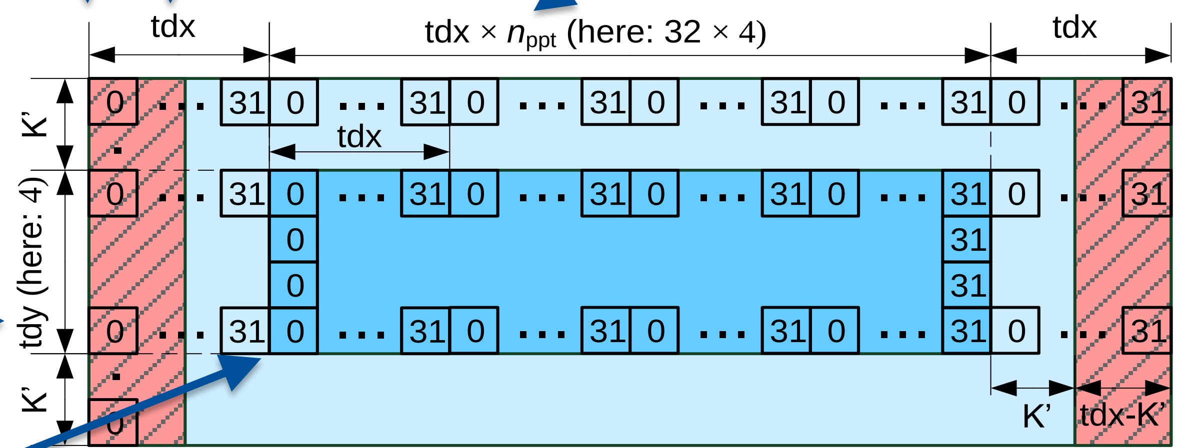
Non-separable 2D image transform
Median Filter and Rank Transform

128 byte alignment boundary
= no overhead

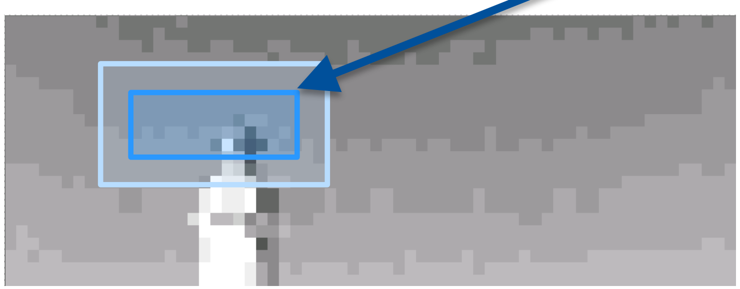
One warp = tdx in **parallel**
coalesced memory access

n_{ppt} pixel per thread **serially**
more computation per thread
= keep pipeline filled

tdy: increase
data reuse
more instructions
per ALU (cuda core)



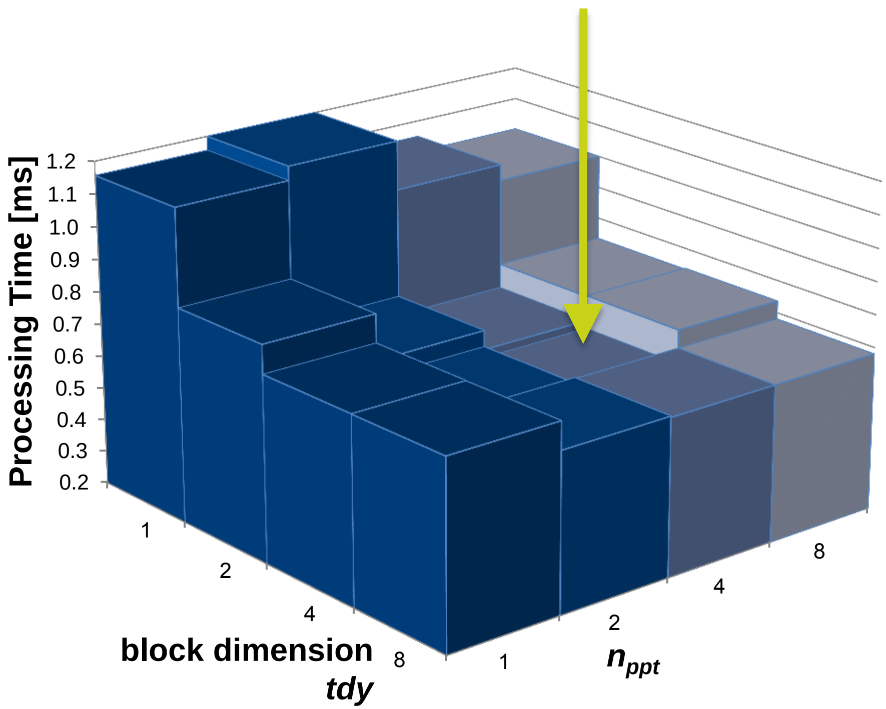
local image window in shared memory



Extension of the separable convolution [Podlozhnyuk]

Parameter Study

Best configuration: *tdy* = 4 and n_{ppt} = 4
Performance: 0.64 ms
Speed-up > 2

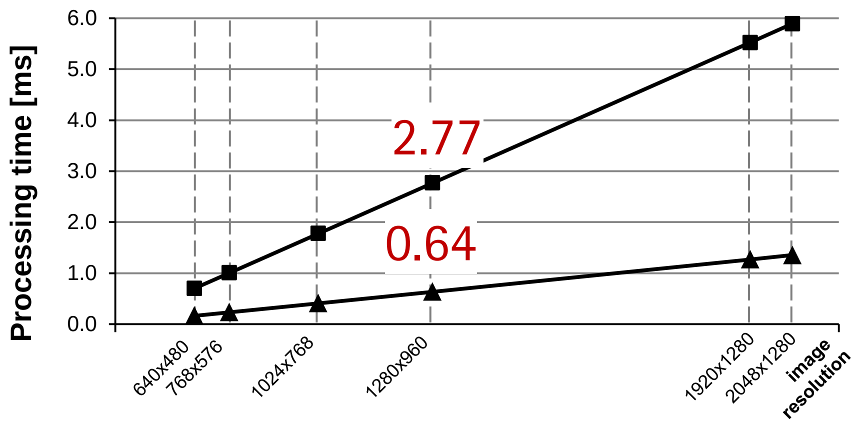


tdx = 32 (i.e. warp width)

1280 x 960 pixel image
3x3 median filter

Results (2D transform)

3x3 median filter (greyscale)

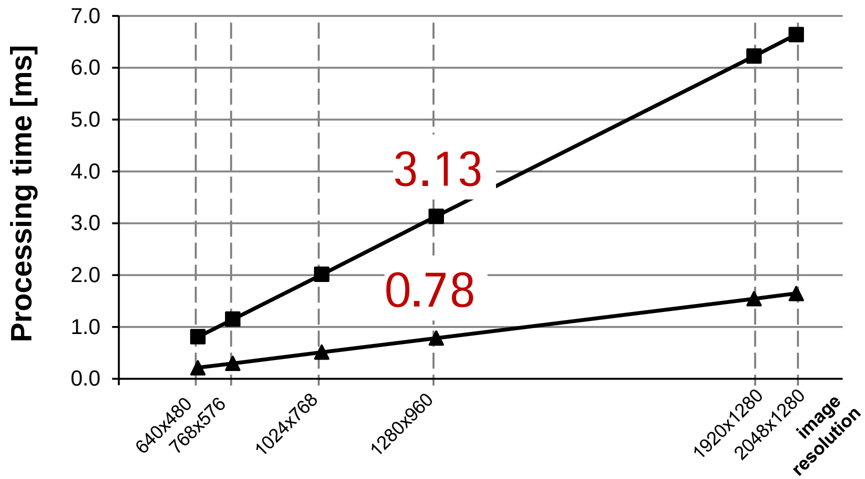


"Standard" implementation using texture memory

proposed implementation with optimal configuration set

Speed-Up: 4.3

9x9 rank transform (greyscale)

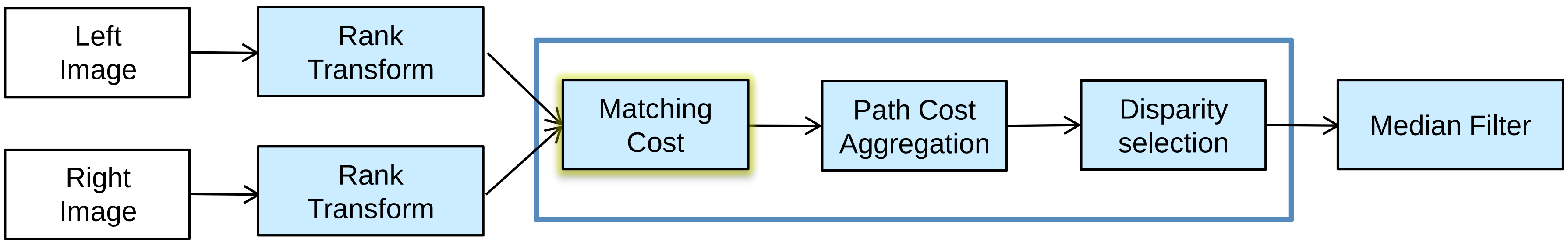


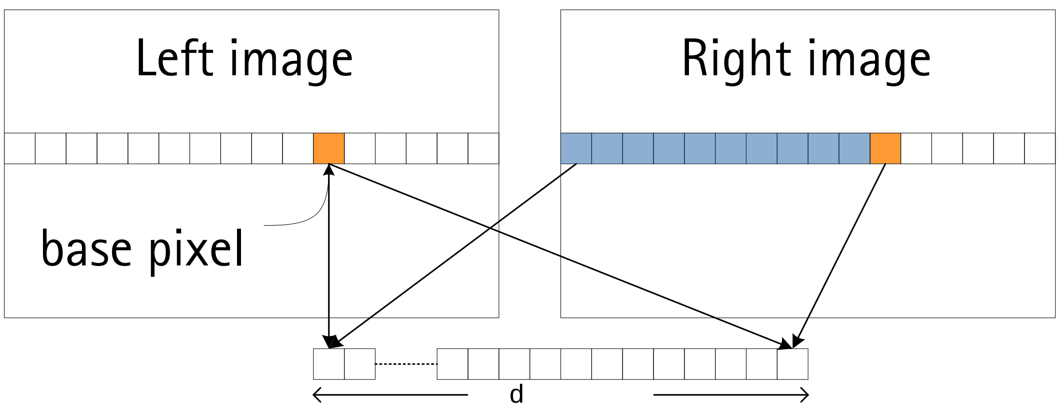
"Standard" implementation using texture memory

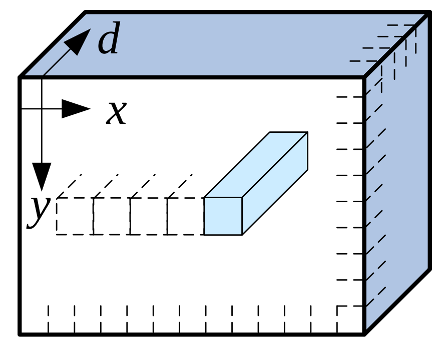
proposed implementation with optimal configuration set

Speed-Up: 4.0

Matching Cost Calculation







$C(x, y, d)$

```
for (d=0;d<128;d++) {
  C[x,d] = |left[x] - right[x-d]|;
}
```

Matching Cost (MC) Implementation Options

Kernel	Time [ms]	Bandwidth / Max. Bandw.	Performance bound
(1) MC Unaligned	16.32	48.6 GB/s/ 144.0 GB/s	Memory Access Scheme

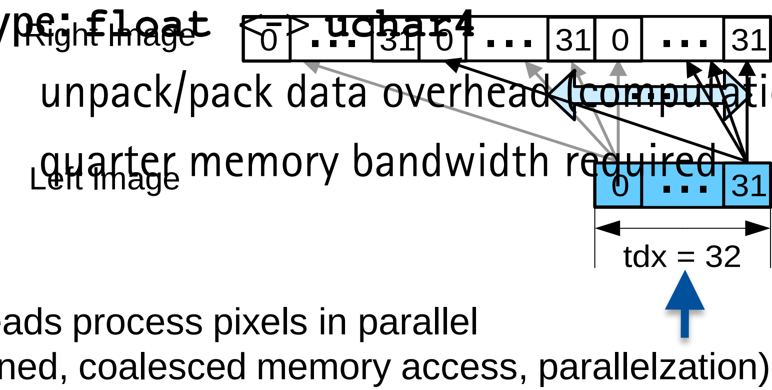
Unaligned: Each thread processes one point of C(x,y,d)

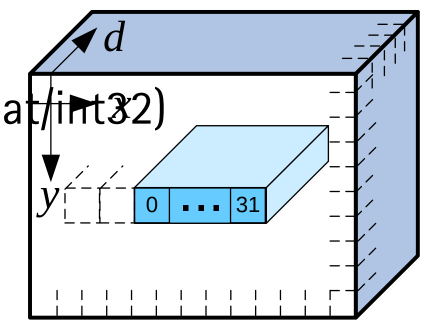
- Only `|left[x] - right[x-d]|` → not compute bound!

Proposed: Combination of parallel and serial with aligned memory access

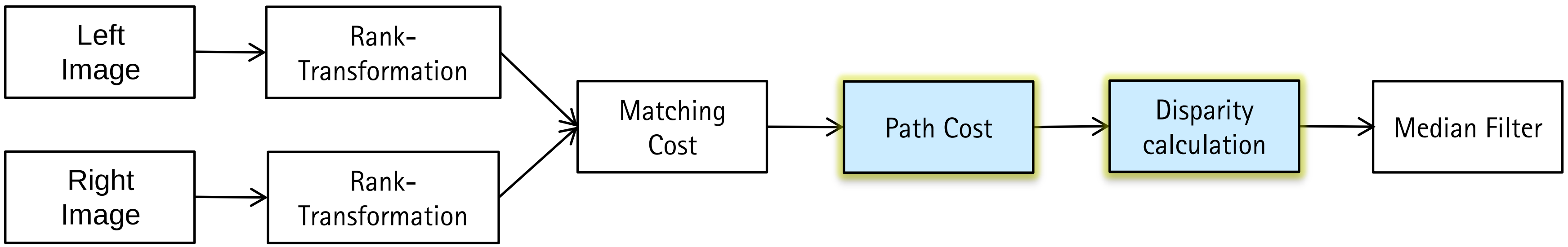
Datatype: `float` <-> `uchar4`

- unpack/pack data overhead (computation only efficient in float/int32)
- quarter memory bandwidth required





Path Cost Calculation

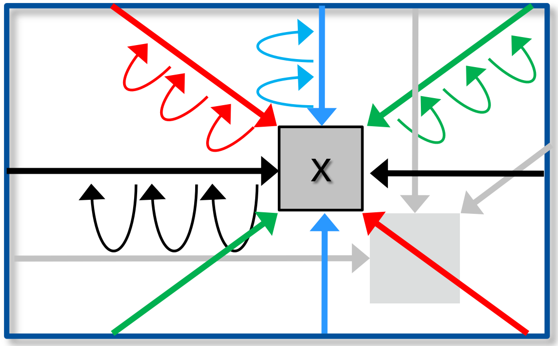


Main challenges imposed by Path Cost Calculation

- Main calculation is *recursive* and *not scan-aligned*
- Values off all 8 paths must be kept
  - for the *next pixel along the path*
  - and until *all paths to a pixel are known*

→ **Parallelization concept which adheres to principles for fast implementation on the GPU**

$$
\begin{aligned}
L_{\mathbf{r}}(\mathbf{p},d) = C(\mathbf{p},d) + {} & \min\{\, L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d), \\
& L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1, \\
& L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1, \\
& \min_i \{L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)\} + P_2 \} - \\
& \min_i L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)
\end{aligned}
$$



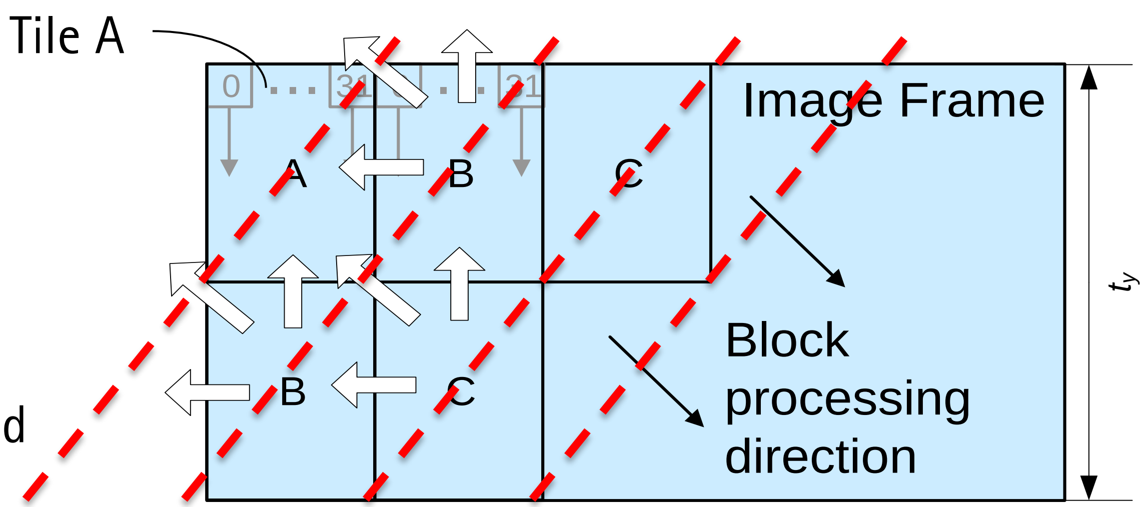
Path cost calculation

- Same parallelization scheme as for matching cost calculation
  - Major difference: Follow path direction serially

Example

Both *B*s can be processed in parallel

Data dependencies:
C can only be processed after *A* and both *B*s

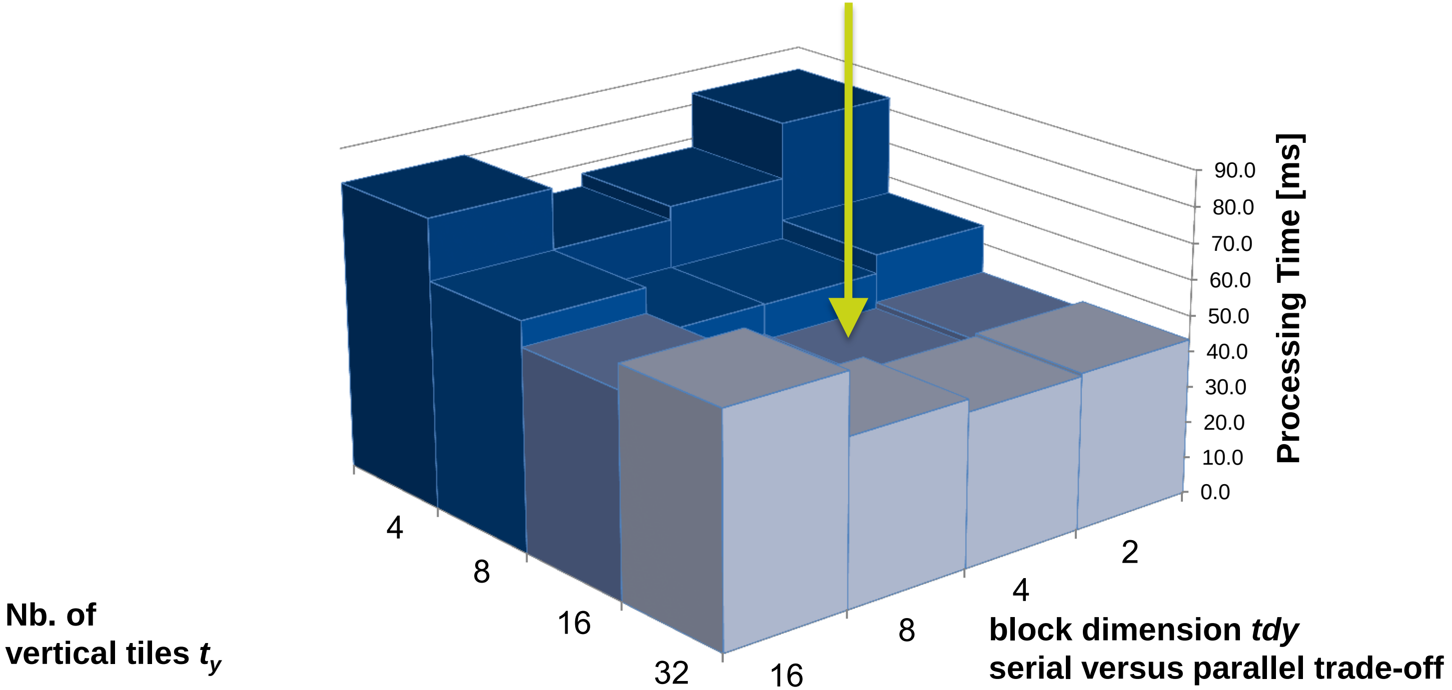


Path Cost Calculation: Parameter Study

Best configuration: t_y = 16 and *tdy* = 4

Performance: 39.81 ms

Speed-Up: > 2



Path Cost Calculation: Results

- 1 path direction = 1 kernel
  - Not all streaming multiprocessors used
- Use **concurrent kernel execution**
  - Available with Fermi architectures
  - compensates inherently serial computation of the PC kernels

Kernel	Time [ms]	Bandwidth [GB/s]	Performance bound
Matching cost + Path cost all 8 path directions **sequentially**	75.68	20.9	Latency (pipeline is not always filled)

Summary of Results

Entire disparity estimation using semi-global matching

Implementation	Architecture	Com. Cap.	Image Size	Disparity depth	Processing time [ms]
Proposed here	Tesla C2050	2.0	640x480	128	16
Proposed here	Tesla C2050	2.0	1024x768	128	36

Observations

- Very fast implementation is sometimes a *black art*
  - A single code line can ruin performance
- Very deep knowledge of hardware behavior required
  - ALU and pipeline behavior (stalls, scheduler, etc.)
  - Physical memory properties and memory cache

- Hardware-oriented programming style necessary
- Nvidia should openly communicate this to programmers and provide relevant information

Conclusion

- Efficient parallelization schemes for SGM on the GPU
- **Key enablers**
  - Combination of parallel and serial implementation (parametrizable)
  - Serial implementation must be **local and independent**
  - Efficient, aligned, **block based** memory access
- Can serve as paradigm for programming styles (hardware oriented programming)