JÜLICH
FORSCHUNGSZENTRUM

Heterogeneous Computing
with OpenCL

Wolfram Schenck | Faculty of Eng. and Math., Bielefeld University of Applied Sciences |

OpenCL Course, 22.11.2017

Overview of the Lecture

1. Multi–Device: Data Partitioning
2. Multi–Device: Load–Balancing
3. Exercise

Multi–Device: Data Partitioning

Multi–Device Programming

Misc. Scenarios

- Multi–GPU (also from differing manufacturers)
- Multi–CPU
- CPU(s) with GPU(s) (APU; „Heterogeneous Computing“)

Implementation with OpenCL

- Devices from same manufacturer (→ same platform): Single shared context possible
- Devices from differing manufacturers (→ multiple platforms): Separate contexts necessary
  - . . . and separate program objects and kernels for each context
- In any case: A separate queue for each device
  - Synchronization between queues with events

Multi–Device Programming

Basic Considerations

Important factors

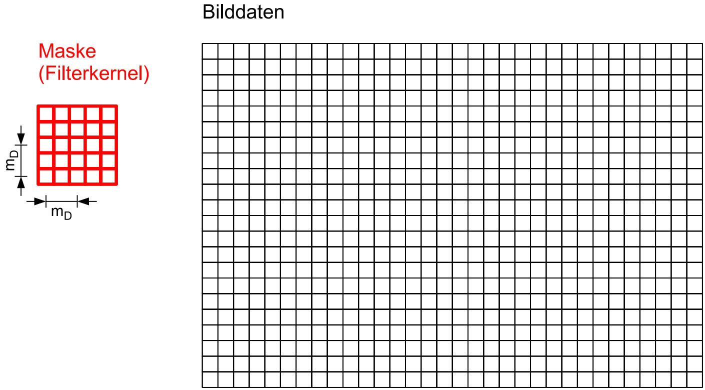
- Scheduling overhead:
  - Startup time of each device?
- Data location:
  - On which device is the data located?
- Task and data structure:
  - How should the problem be partitioned?
  - How is the relation between data parallel and task parallel parts of the algorithm?
  - How is the ratio between startup time and time for the main calculations?
- Relative performance of each device:
  - What is the best work distribution?

  → Load balancing

In the following: Example for data partitioning

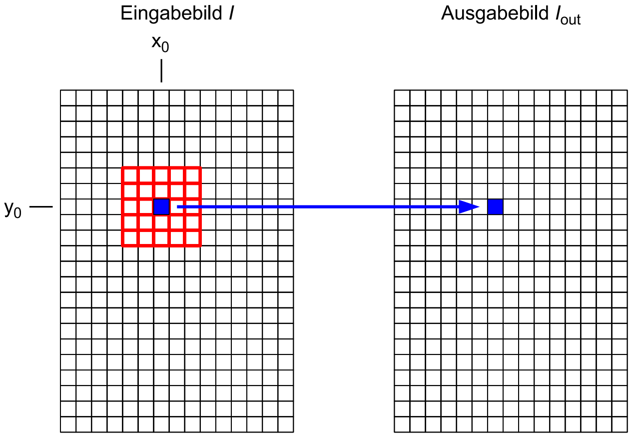
Example: Convolution

Filtering an Image with a Convolution Kernel

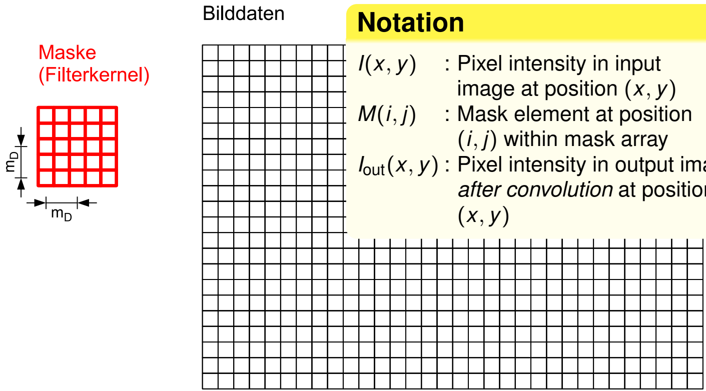


m_D : radius of the convolution kernel (“mask”)

Example: Convolution (cont.)



Example: Convolution (cont.)



Notation

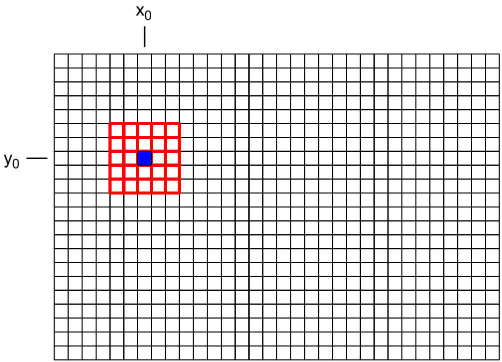
$I(x, y)$: Pixel intensity in input image at position (x, y)

$M(i, j)$: Mask element at position (i, j) within mask array

$I_{out}(x, y)$: Pixel intensity in output image *after convolution* at position (x, y)

Mask: Edge length = $(2m_D + 1)$ pixels in each dimension

Example: Convolution (cont.)

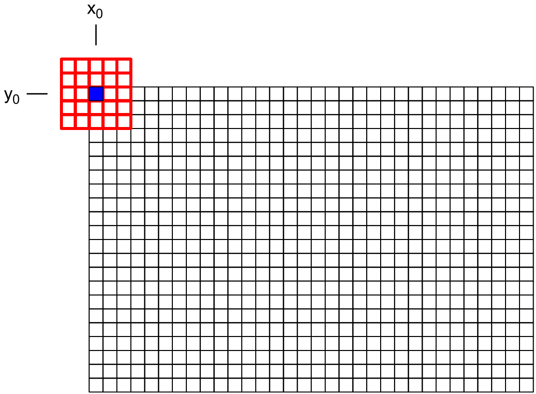


Calculation of the intensity at the blue position (x_0, y_0):

$$I_{\text{out}}(x_0, y_0) = \sum_{i=0}^{2m_D} \sum_{j=0}^{2m_D} M(i,j) I(x_0 - m_D + i, y_0 - m_D + j)$$

Convolution

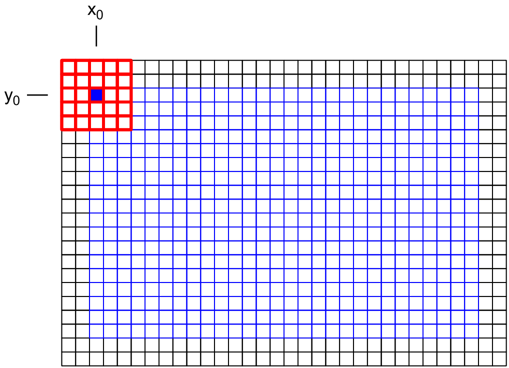
Border Handling



Problem: Divergence because of border handling

Convolution

Border Handling (cont.)



(One possible) solution: Reduction of the size ot the output image so that the mask can always be fully applied to the input image (output image = blue pixels)

Convolution

Misc. Convolution Kernels

$$M_{\text{Identity}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$M_{\text{Laplace}} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

$$M_{\text{Blurr}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1.5 & 2 & 1.5 & 1 \\ 1 & 2 & 10 & 2 & 1 \\ 1 & 1.5 & 2 & 1.5 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

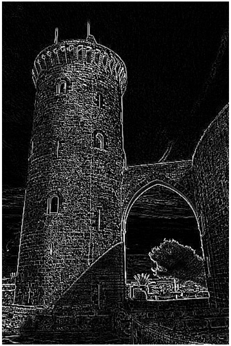
Remark: Renormalization of image data not considered here!

Convolution

Result of Simple Convolution Operations

Original

Laplace



Blurr

Convolution

Sobel Edge Filter (Scharr Version)

Convolution Kernels

$$M_{\text{Sobel,x}} = \begin{pmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{pmatrix} \qquad M_{\text{Sobel,y}} = \begin{pmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{pmatrix}$$

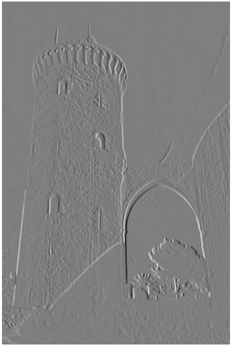
Convolution and Computation of the Output Image

$$\begin{aligned} I_x &= M_{\text{Sobel,x}} * I \\ I_y &= M_{\text{Sobel,y}} * I \\ I_{\text{out}} &= \sqrt{I_x^2 + I_y^2} \quad \text{(pixel-wise)} \end{aligned}$$

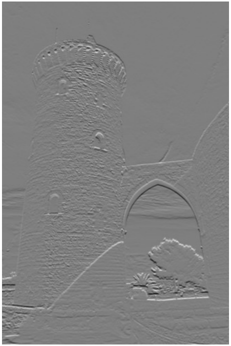
Remark: Renormalization of image data not considered here!

Convolution
Result of the Sobel Edge Filter

I_x



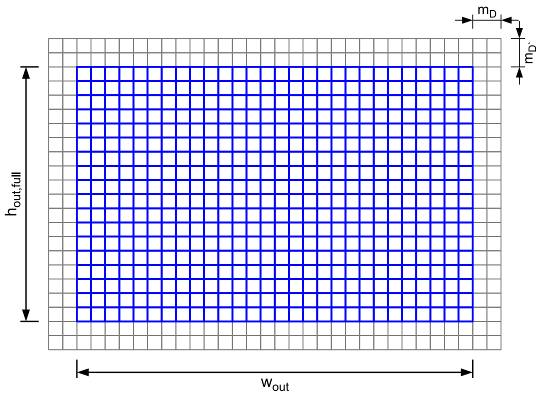
I_y



I_{out}

Data Partitioning

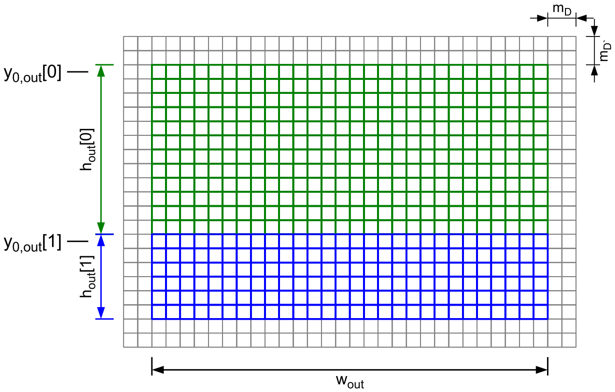
Output Image



$h_{out,full}$: Full height of the output image
w_{out} : Width of the output image

Data Partitioning

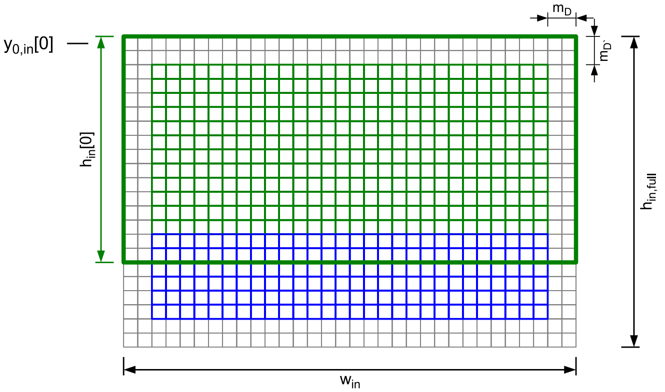
Output Image → Device 0 and 1



$h_{out}[0]$: Height of part of the output image for device 0
$h_{out}[1]$: Height of part of the output image for device 1

Data Partitioning

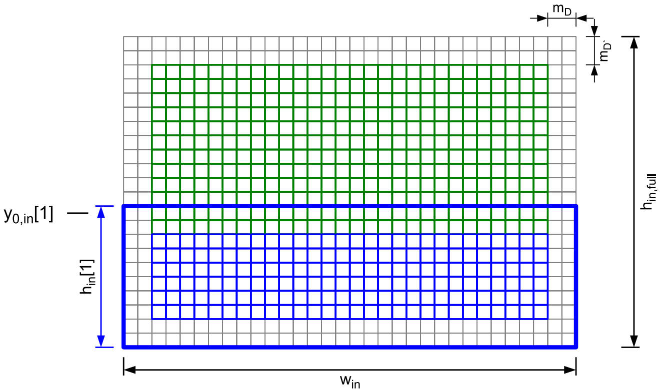
Input Image → Device 0 and 1



$h_{in,full}$: Full height of the input image
w_{in} : Width of the input image
$h_{in}[0]$: Height of part of the input image for device 0

Data Partitioning

Input Image → Device 0 and 1



$h_{in,full}$: Full height of the input image
w_{in} : Width of the input image
$h_{in}[1]$: Height of part of the input image for device 1

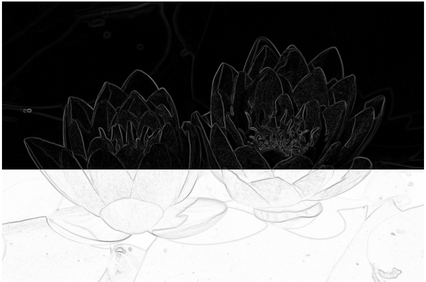
→ Data in border region has to be available on both devices!

Data Partitioning

Result for an Example Image

Original Image

After Filtering



- Partitioning between device 0 and device 1: 60 % zu 40 %
  - Black background: Device 0
  - Inverse colors: Device 1

Multi–Device: Load–Balancing

Load Balancing

Load Balancing

Distribution of computational load on several devices with the goal to use all devices evenly and to minimize the overall computation time

Example

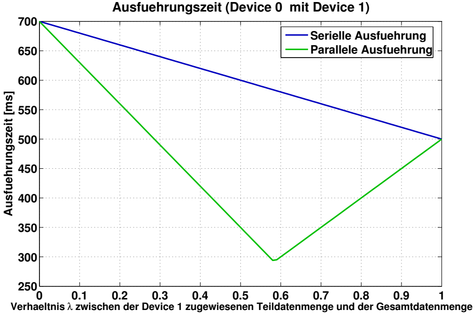
Image filtering executed by two devices in parallel (see preceding slides)

Study on Load Balancing

Theoretical Analysis

- **Overall problem size:** N
- **Distribution of the problem on two devices:** Device 0 (N_0) und Device 1 (N_1)
  - $N = N_0 + N_1$
  - $\lambda = \frac{N_1}{N}$
- **Computation times:**
  - Time required for device 0 to solve overall problem N: T_0
  - Time required for device 1 to solve overall problem N: T_1

Device 0

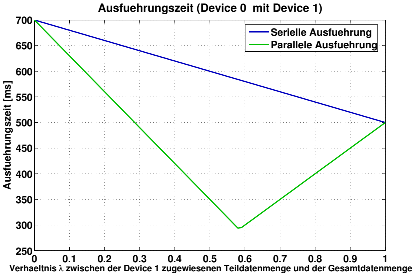


Device 1

Computation time when distributing load on both devices

- **Assumption:** Linear relationship between problem size and computation time ($T \in \mathcal{O}(N)$)

  $\Rightarrow$ Computation time on device 0: $t_0 = (1 - \lambda) T_0$
  $\Rightarrow$ Computation time on device 1: $t_1 = \lambda T_1$
- **Serial execution:** $T_{0+1}^{SER} = t_0 + t_1 = (1 - \lambda) T_0 + \lambda T_1$
- **Parallel execution:** $T_{0+1}^{PAR} = \max(t_0, t_1) = \max((1 - \lambda) T_0, \lambda T_1)$



Derivation of the optimal value for λ $(= \lambda^*)$

- **Assumption:** Perfectly parallel execution with

$$T_{0+1}^{PAR} = \max\left((1-\lambda)T_0, \lambda T_1\right) \ .$$

- **Approach:** Minimum computation time with

$$\begin{aligned} (1-\lambda^*)T_0 &= \lambda^* T_1 \ . \\ \Leftrightarrow (1-\lambda^*)T_0 - \lambda^* T_1 &= 0 \\ \Leftrightarrow \lambda^* T_0 + \lambda^* T_1 &= T_0 \\ \Leftrightarrow \lambda^* &= \frac{T_0}{T_0 + T_1} \end{aligned}$$

Study on Load Balancing

Design and Realization

OpenCL Configuration

- Two devices (when indicated from different platforms and different type)
- Separate contexts and queues (in–order, synchronization after last memory transfer)

Realization

- Sobel filtering on an RGB image with a size of 10 megapixels (thus: problem size equiv. to amount of data in output image)
- Measurement of the computation (wall)time (data transfer and kernel execution) as mean value over 20 trials
- Systematic variation of λ

Study on Load Balancing: Hints!

Correct Handling of Queues for two Devices

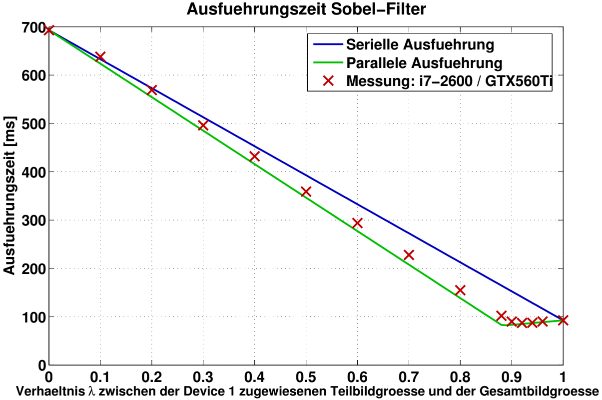
- **Queue 0:** Asynchronous call of memory transfer (H to D), kernel invocation, memory transfer (D to H)
- **Queue 1:** Asynchronous call of memory transfer (H to D), kernel invocation, memory transfer (D to H)
- **Queue 0:** `clFlush(..)`
- **Queue 1:** `clFlush(..)`
- **Queue 0:** Synchronization (`clFinish(..)` or `clWaitForEvents(..)`)
- **Queue 1:** Synchronization (`clFinish(..)` or `clWaitForEvents(..)`)

Attention!

- `clFlush(..)` enforces that all commands in the queue are sent to the device for immediate execution. Important to use!

Study on Load Balancing

Core i7–2600 (CPU) vs. NVIDIA GTX 560Ti (GPU)

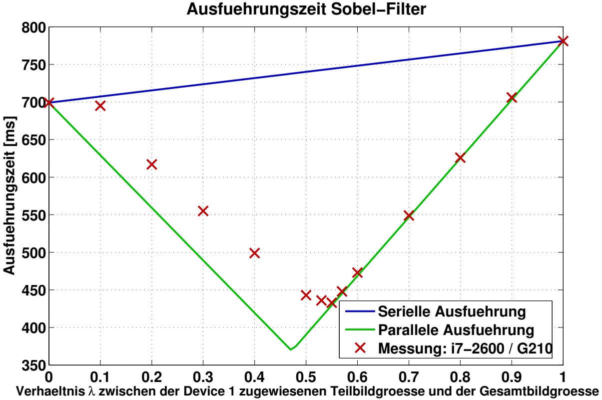


- **Platforms:** Intel, NVIDIA

➔ Only minimal gain if devices are from very different performance classes

Study on Load Balancing

Core i7–2600 (CPU) vs. NVIDIA G210 (GPU)



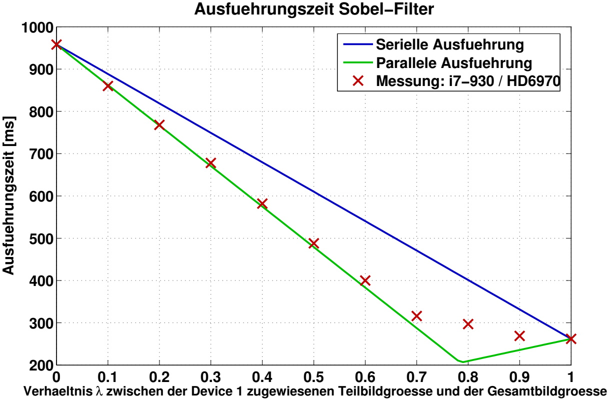
- **Platforms:** Intel, NVIDIA

➔ Nice gain if CPU and GPU operate at same performance level

➔ Worse than ideal parallel execution because CPU is also required as host for GPU processing

Study on Load Balancing

Core i7–930 (CPU) vs. AMD HD 6970 (GPU)

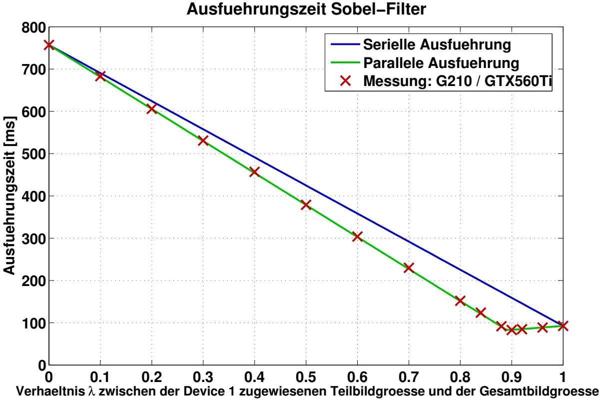


- **Platforms:** AMD

➔ Parallel execution, but worse than ideal model because CPU is also required as host for GPU processing

Study on Load Balancing

NVIDIA G210 (GPU) vs. NVIDIA GTX 560Ti (GPU)

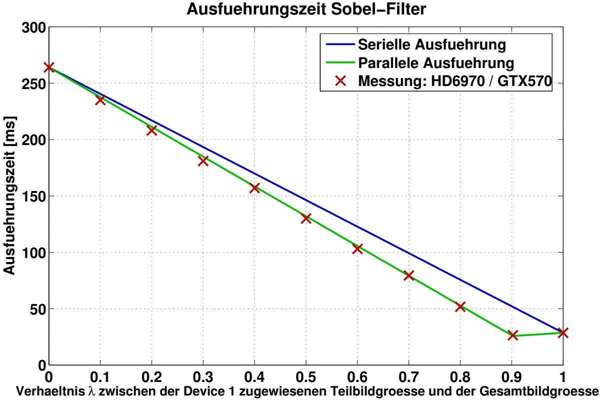


- **Platforms:** NVIDIA

➔ Ideal parallel execution with two GPUs from the same manufacturer, but only minimal gain because of very different base performance

Study on Load Balancing

AMD HD 6970 (GPU) vs. NVIDIA GTX 570 (GPU)

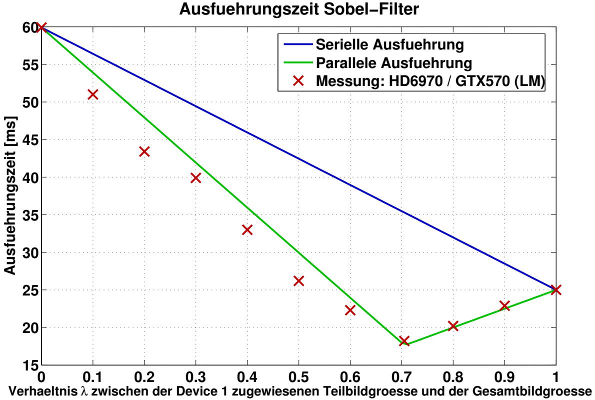


- **Platforms:** AMD und NVIDIA

➔ Ideal parallel execution with two GPUs from different manufacturers!

Exkursus: Usage of Local Memory

AMD HD 6970 (GPU) vs. NVIDIA GTX 570 (GPU)

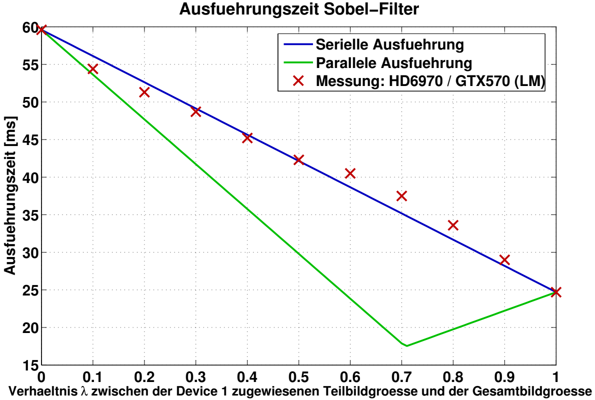


- Modified kernel using local memory as cache speeds up execution on the Cayman–XT GPU

➔ Parallel execution even faster than expected from theoretical model!

Exkursus: Omission of `clFlush(..)`

AMD HD 6970 (GPU) vs. NVIDIA GTX 570 (GPU)



- **Setting:** Different platforms and GPUs, kernel with local memory

→ Without `clFlush(..)` execution on GPUs is only serial!

Study on Load Balancing

Conclusions

- **CPU with GPU** („Heterogeneous Computing")
  - Parallel execution
  - Performance not as good as predicted by ideal parallel model because CPU is required as host for GPU processing
  - Especially useful if CPU and GPU have roughly the same base performance (see APU concept)
- **GPU with GPU**
  - Parallel execution as predicted by ideal parallel model (sometimes even better)
  - Can be used with GPUs from the same platform or from different platforms (manufacturers)
  - Even a weak GPU can support a strong GPU up to a measurable effect (as predicted by the ideal parallel model; however, not that many good use cases exist for this scenario)

Load Balancing: General Hints

Consideration of the Capacity/Latency/Speed of the Devices

- Excessive demand on a weak device may hinder overall execution
- Startup latency may become a limiting factor if data pieces are too small

Approach No 1

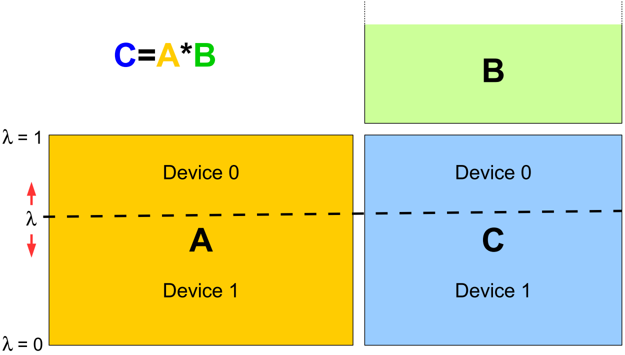
- Tests with small amounts of data/problem sizes
- Profiling on various devices
- Extrapolation for larger amounts of data/problem sizes (based on an analytical or empirical performance model)
- Production runs with enlarged data amounts/problem sizes to minimize overhead

Approach No 2

- If one device is clearly superior to all others: Just use this one. . .

Exercise

Exercise: Matrix Multiplication on two Devices



1. Modify example code so that both devices compute part of the target matrix **C**.
2. Determine λ value with minimum overall execution time according to theoretical model.
3. Check if this corresponds to the empirical minimum.