Parallel H.264/AVC Motion Compensation for GPUs using OpenCL

Biao Wang, Mauricio Alvarez-Mesa, Chi Ching Chi, Ben Juurlink

Embedded Systems Architecture
Technische Universität Berlin
Berlin, Germany

January 20, 2014

Outline

- Introduction
- H.264 Motion Compensation Kernel Design
- Experimental Results
- Conclusions

Motivation

Computational demand of video codec:

- increased compression rate: H.263, **H.264**, HEVC, ...
- higher resolution pursuit: 1080p (FHD), 2160p (QHD), ...
- single thread CPU can't achieve real time decoding

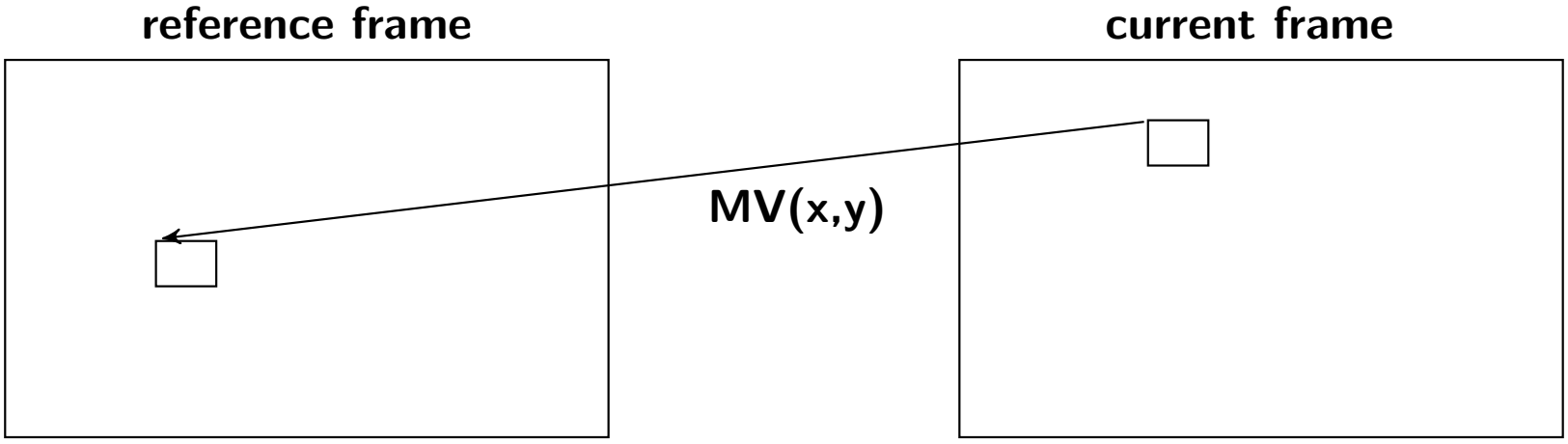
Solution: **GPU** acceleration for decoding

- higher computational power
- massive parallelism and low divergence

Targeting **motion compensation** in decoding

- one of the most time-consuming part
- exposes massive parallelism
- divergence can be high

H.264 Motion Compensation



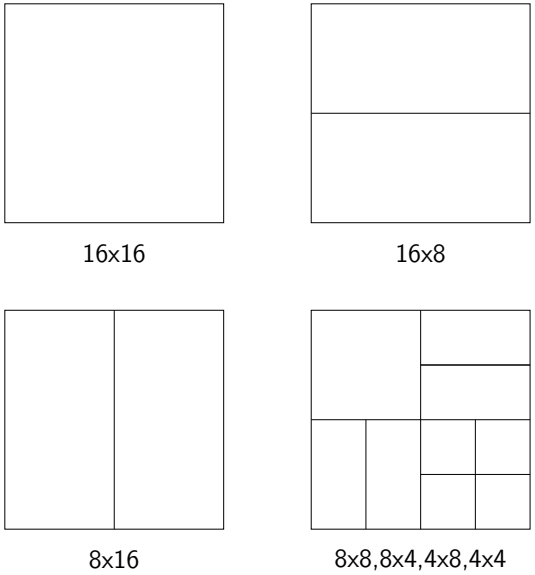
using pixels from reference frames to "predict" pixels of current frame

- motion vector
  - block size: 16×16 pixels down to 4×4.
  - resolution: 1/4 pixel , sub-pel is interpolated.
- reference index.
  - indicates reference frame.

H.264 Motion Compensation

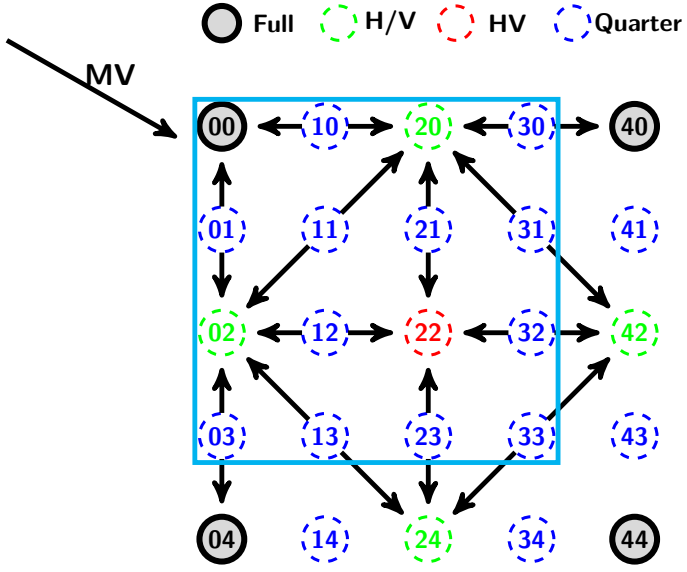
Partition scheme

- multiple partition in marcoblock (16×16 pixels)
- each partition has its own motion vector
- extreme: 16 partitions of 4×4



Interpolation modes

- full pel: copy
- half pel: 6-tap FIR filter
- quarter pel: bi-linear filter



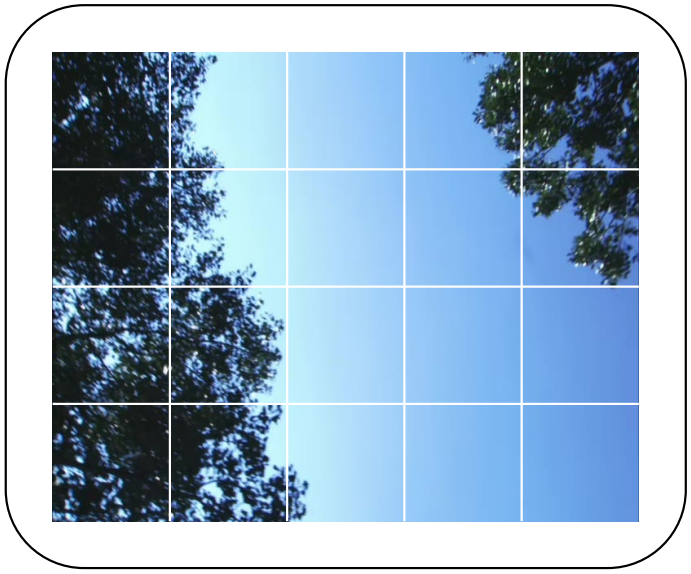
Outline

- Introduction
- H.264 Motion Compensation Kernel Design
- Experimental Results
- Conclusions

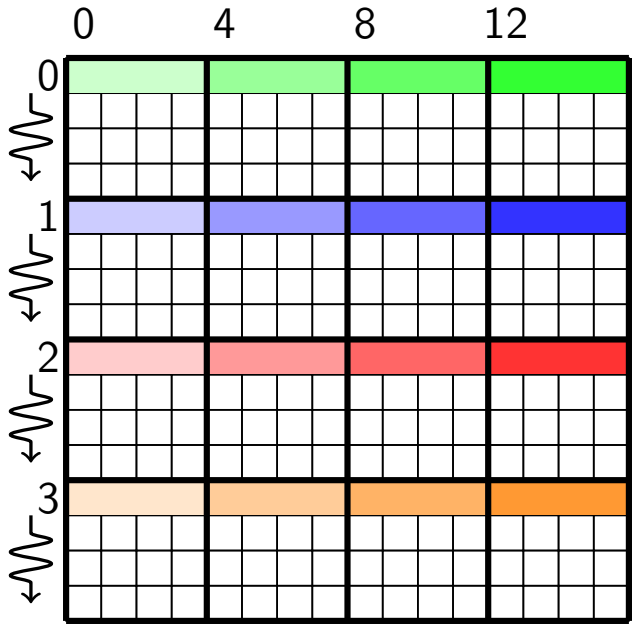
Thread Mapping

Hierarchical thread mapping in OpenCL

NDRange→Frame



Workgroup→Macroblock



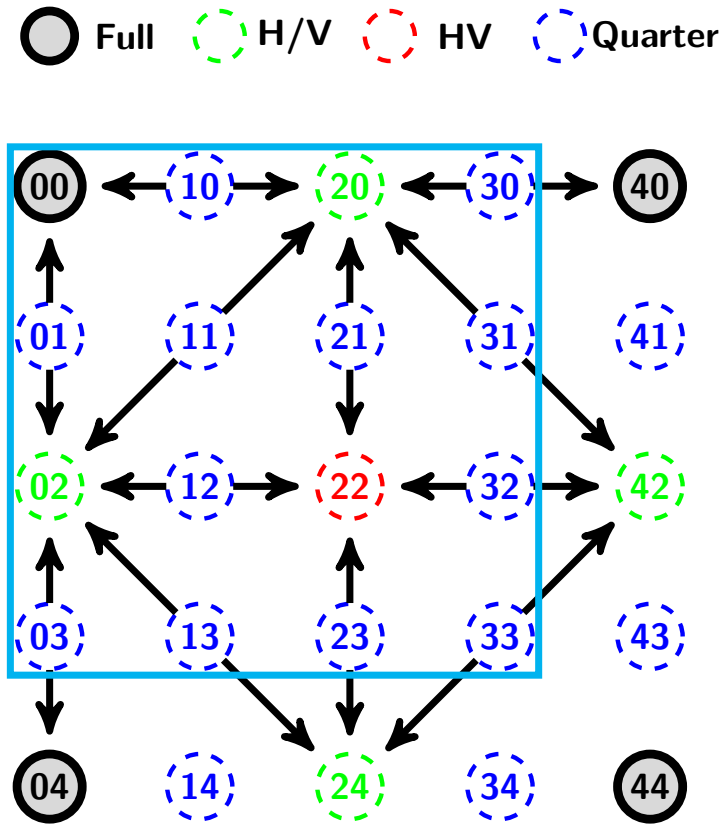
- Every 4 threads → 4×4 block
- Every thread →Each column of 4 pixels

Multiple Stages Design

Straight forward implementation

```
switch(mode){
        case mode00:
                do_MC_mode00;
                break;
        case mode01:
                do_MC_mode01;
                break;
        .
        .
        .
        case mode33:
                do_MC_mode33;
                break;
}
```

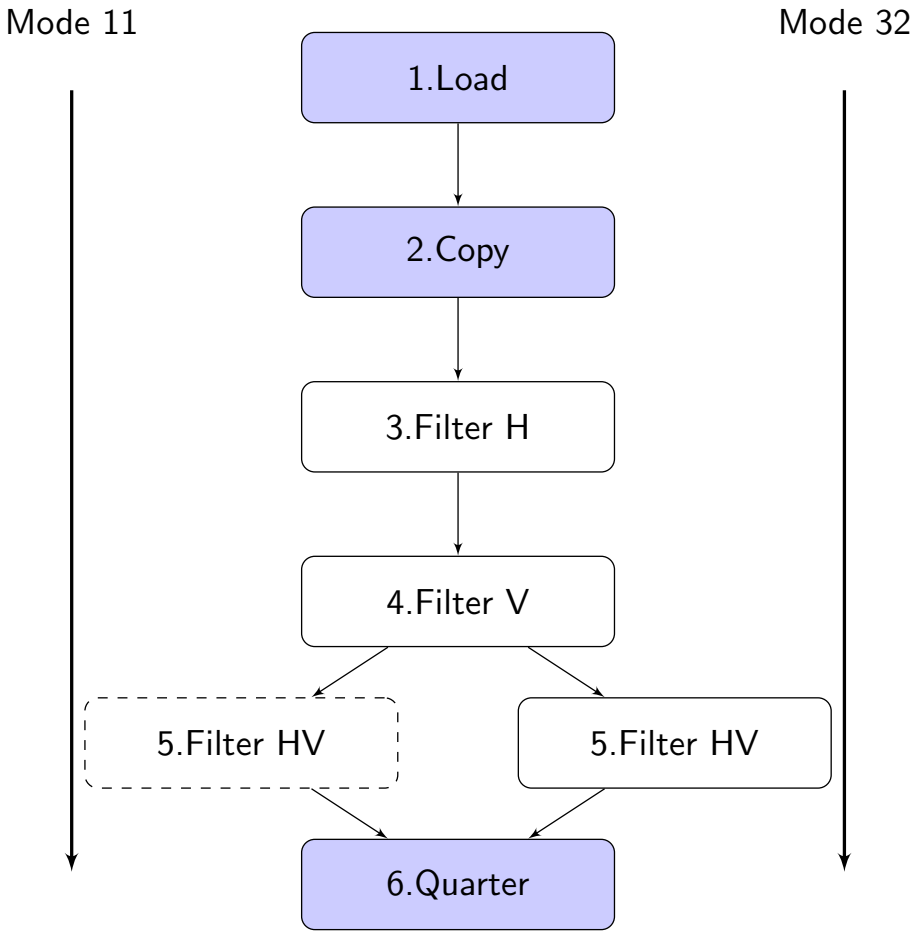
Divergence!



Multiple Stages Design

Table : complete stages

Mode	Stage					
xy	Load	Copy	H	V	HV	Quarter
00	X	X				X
10	X	X	X			X
20	X	X	X			X
30	X	X	X			X
01	X	X		X		X
11	X	X	X	X		X
21	X	X	X		X	X
31	X	X	X	X		X
02	X	X		X		X
12	X	X	X	X	X	X
22	X	X	X		X	X
32	X	X	X	X	X	X
03	X	X		X		X
13	X	X	X	X		X
23	X	X	X		X	X
33	X	X	X	X		X



- divergence is mitigated by stage sharing
- stages of filter H, V, and HV can be skipped

Optimizations

Zero copy for integrated GPUs

- discrete GPUs: separate memories of CPUs and GPUs
- integrated GPUs: shared physical memory of CPUs and GPUs
  - AMD APU
  - Intel Ivybridge, Haswell
  - memory copy can be avoided

Overlapped Execution

- implemented both luma and chroma kernels
- CPU2GPU $\rightarrow$ kernel execution $\rightarrow$ GPU2CPU
- overlapped kernel execution and memory copy

Outline

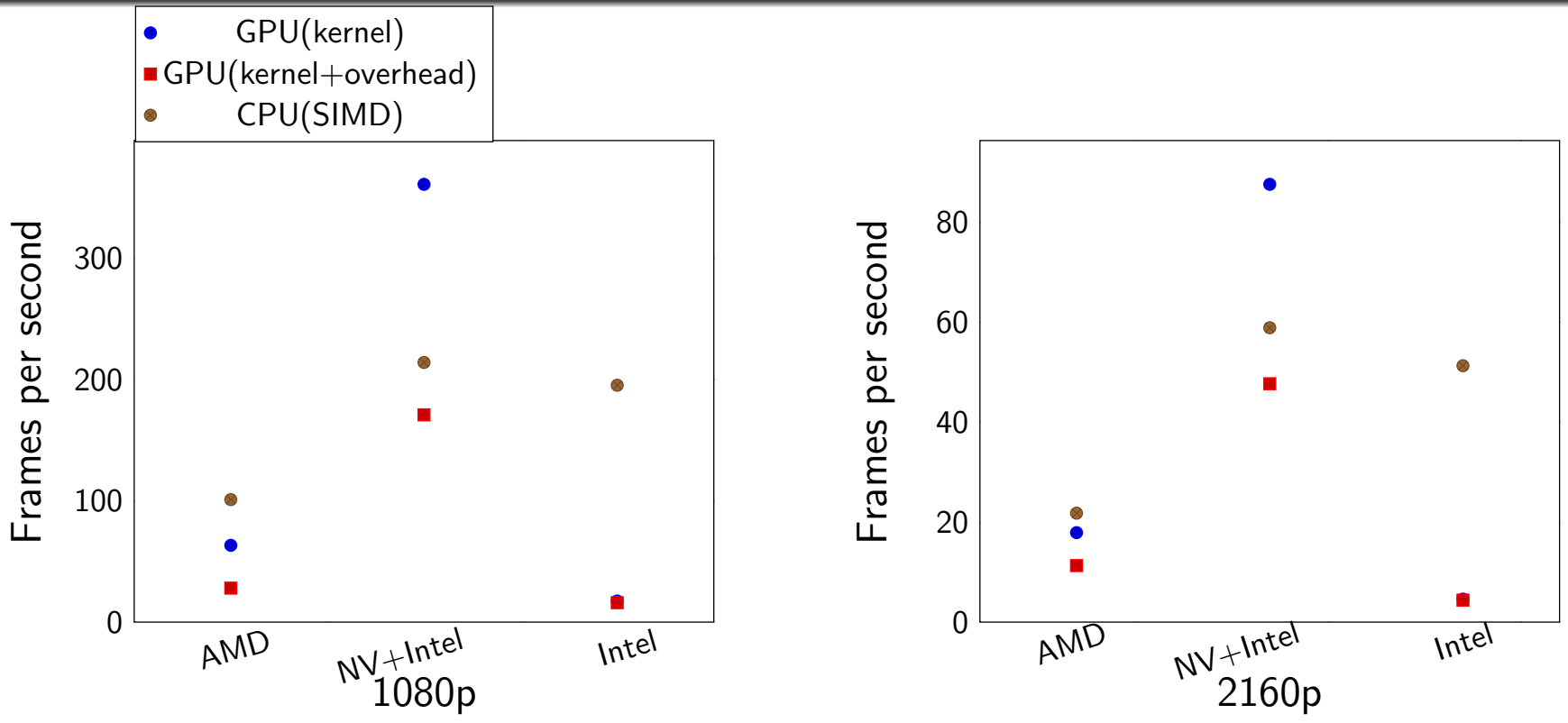
- Introduction
- H.264 Motion Compensation Kernel Design
- **Experimental Results**
- Conclusions

Experimental Setup

GPU	Quadro 3000M	i5-3317U (HD4000)	A6-4455M (HD7500G)
Vendor	NVIDIA	Intel	AMD
Process Elements	**240**	128	256
Freq(MHz)	**900**	350	327
Mem BW(GB/s)	**80**	25.6	21.3
Zero copy support	**No**	Yes	Yes
CPU	i7-2760QM	i5-3317U	A6-4455M
Vendor	Intel	Intel	AMD
Freq(GHz)	2.4	1.7	2.1
OS	ubuntu 12.10	windows 7	windows 7

- videos : 4 (1920×)1080p, 2 (3840×)2160p
- encoder: x264, 4 QPs (22, 27, 32, 37)
- decoder: ffmpeg with MC offloaded onto GPUs

Multiple Platforms Testing

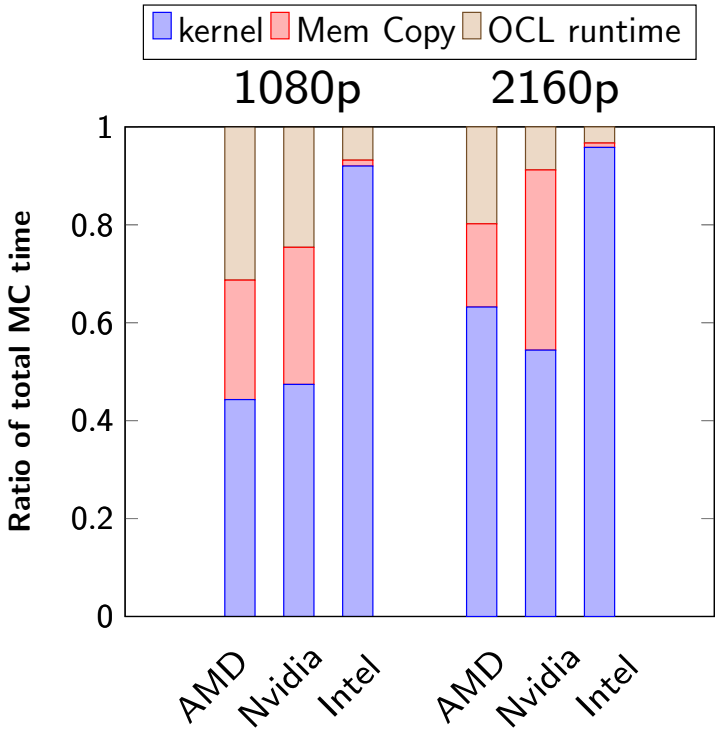


- discrete GPU attains the highest performance, 1.7 speedup
- integrated GPUs are slower than their CPU parts.
- significant performance penalty when including the overhead

Performance Analysis

Total (kernel+overhead)

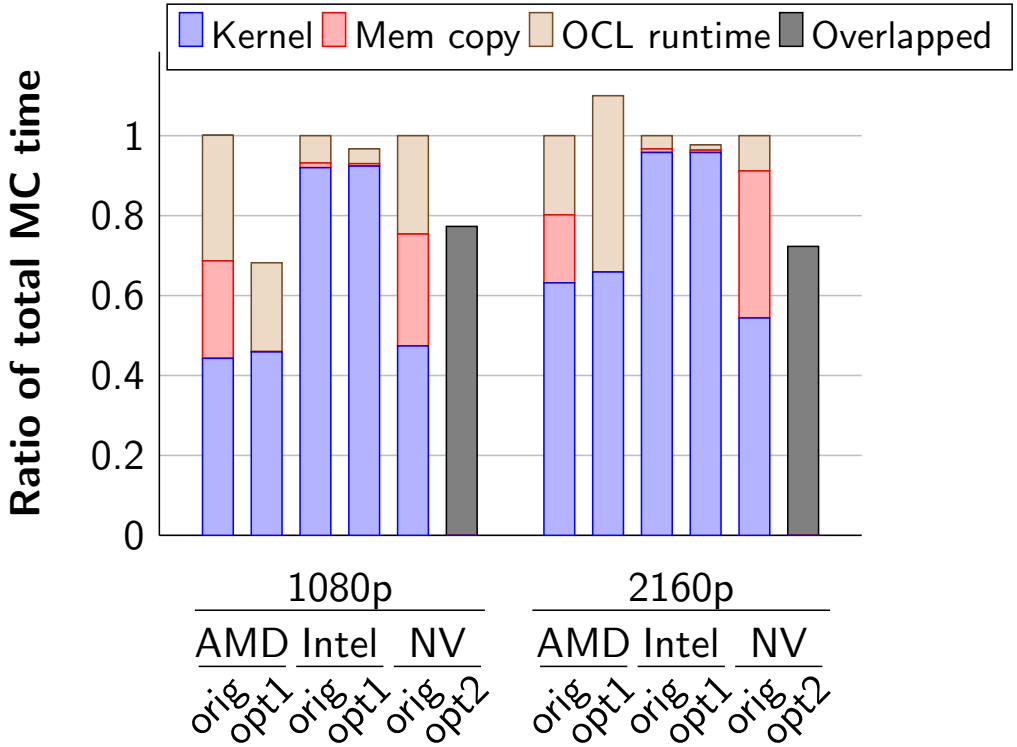
- Kernel execution
- Memory copy
- OpenCL runtime



Kernel level bottlenecks

- Intel: memory bound
  - low memory bandwidth of accessing 8-bit and 16-bit data types
- AMD : instruction level parallelism
  - VLIW4 packing efficiency 50%
- Nvidia: occupancy
  - bound by shared memory size

Optimizations



- Opt1: zero copy, up to 32% ↓
- Opt2: overlapped execution, up to 28% ↓

Outline

- Introduction
- H.264 Motion Compensation Kernel Design
- Experimental Results
- Conclusions

Conclusions and Future Work

Parallelize motion compensation on GPU architecture

- MC for GPU architecture:
  - mitigating divergence by multi-stage design
  - GPU is not appropriate for MC
- GPU computation limitation:
  - Intel: improve memory access for smaller data types
  - Nvidia: eliminate memory copy, improve OpenCL runtime
  - AMD: remove limitation of ILP for VLIW4
- solutions for memory copy:
  - integrated GPUs: zero copy, 32%
  - discrete GPUs: overlapped, 28%

Future work: HEVC (h.265)

Questions?

- `http://www.aes.tu-berlin.de/`