Institute of Microelectronic Systems

Leibniz Universität Hannover

Implementation and Analysis of the Histograms of Oriented Gradients Algorithm on a Heterogeneous Multicore CPU/GPU Architecture

Oliver Jakob Arndt, Tobias Linde, and Holger Blume

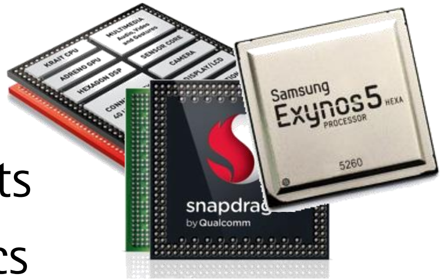
GlobalSIP'15, 16.12.2015

Outline

- Motivation
- Algorithm
- Parallelization
- GPU implementation
- Offload strategies
- Conclusion

Multicore applications

- Multicore processor SoCs
  - Heterogeneous, application-specific processing units
  - High performance at improved power characteristics

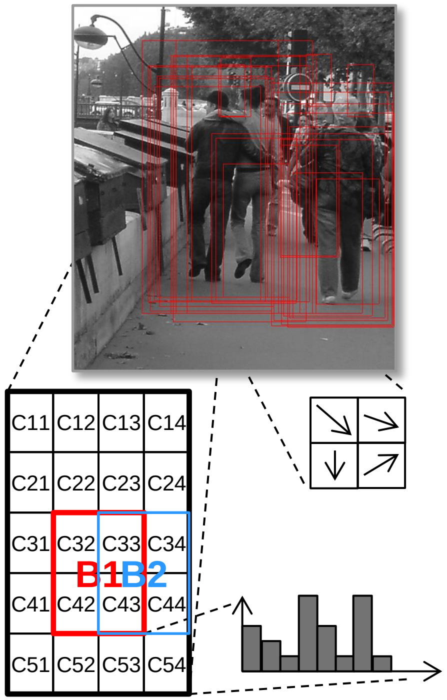


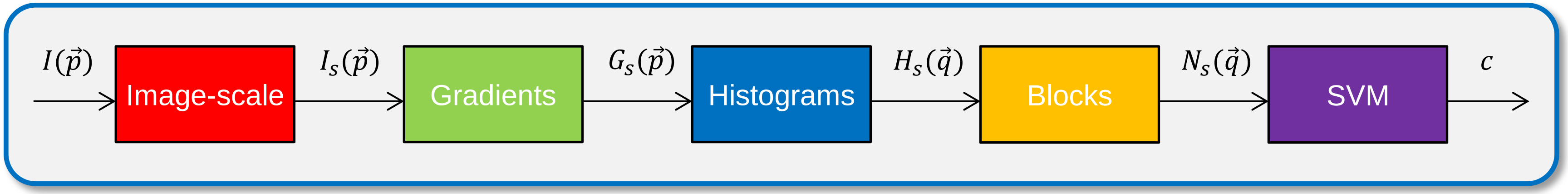
- Driver-assistance systems
  - High computational complexity: algorithmic, memory
  - Strict real-time constraints: throughput, latency
- Portable and scalable implementations
  - Numerous partitioning and mapping strategies
  - Experienced developers required

Case study: (1) how much effort to gain benefit and (2) evaluation of implementation and offloading strategies.

Algorithm: pedestrian detection

- **Traversing:** detection windows (DW)
- **Feature extraction:** Histograms of Oriented Gradients [1] (HOG)
    - Oriented gradients
    - Histogram generation in cells
    - Block-normalization
- **Classification of each DW:** Support Vector Machine (SVM) trained with pedestrians

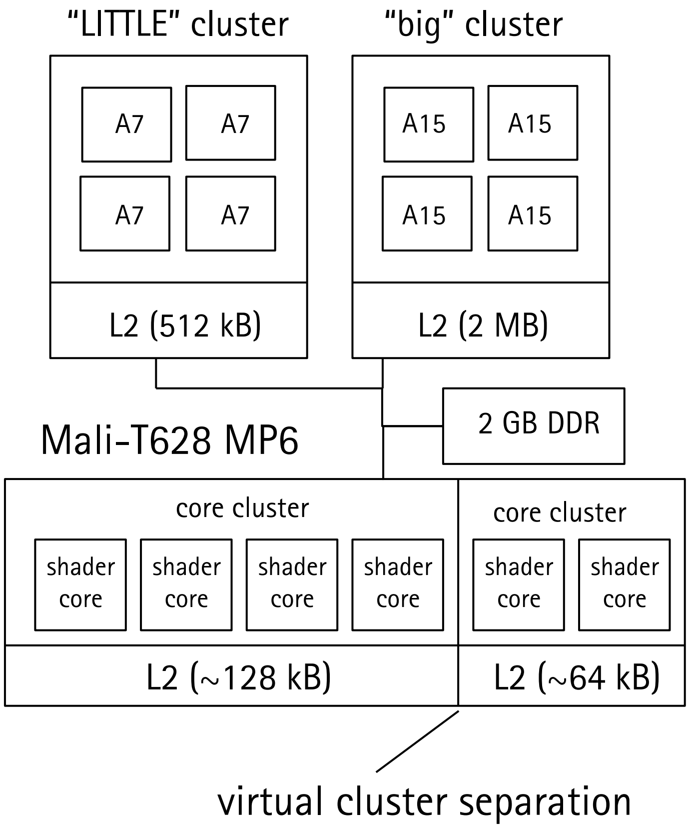




[1] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In Intl. Conf. Computer Vision and Pattern Recognition. IEEE, 2005

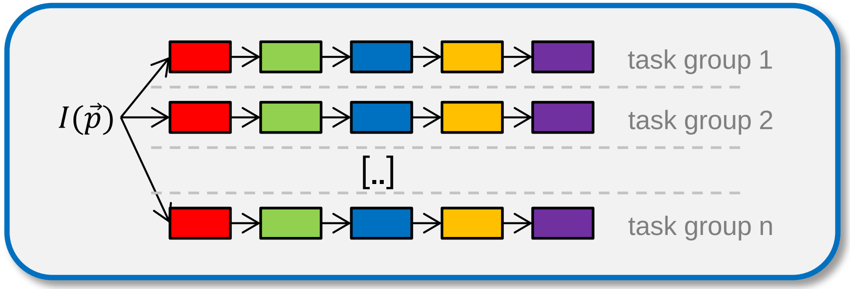
Platform

- **Samsung Exynos 5 Octa 5422**
- **CPU: ARM big.LITTLE**
  - 4 x Cortex A7 @ 1,4 GHz
    (1 core reserved for GPU management)
  - 4 x Cortex A15 @ 2,0 GHz
- **GPU: ARM Mali**
  - 6 shader cores @ 600MHz
  - 128-bit vector registers
  - OpenCL 1.1

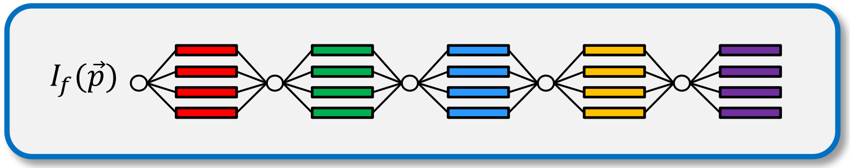


CPU parallelization scheme

- **Splitting by image-scale factor**
  - Insufficient concurrencies
  - Uneven work distribution

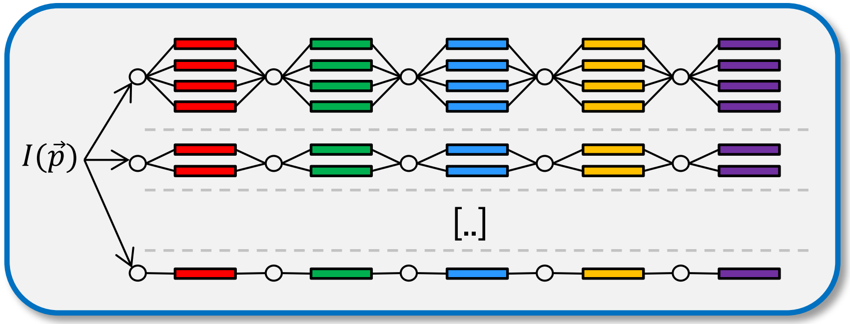


- **Splitting by domain decomposition**
  - Insufficient concurrencies



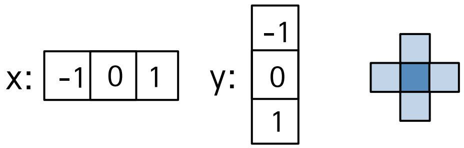
- **Combination: nested parallelization**
  - Work distribution model:

$$T_{total} = \sum_{s=s_{min}}^{s_{max}} T(s)\,, \quad T(s) = \frac{1}{s^c}\,, \quad c \approx 2$$

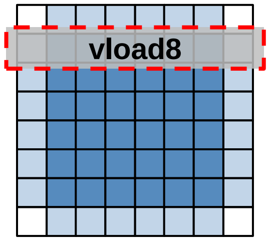


GPU kernel: oriented gradients

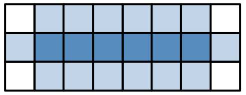
- Pixel-wise gradient calculation: loading one by one
  - ✓ OpenCL buffer uses HW interpolation
  - ✘ 4 load OPs per gradient



- Vectorized load: large work items (e.g., 8x8)
  - ✓ 8 vload OPs per 36 gradients
  - ✘ High memory consumption: performance drop



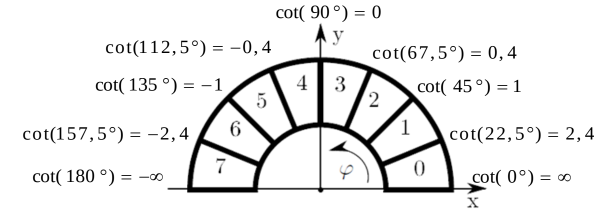
- Optimized load: small work items (e.g., 3x8)
  - ✓ 3 vload OPs per 6 gradients
  - ✓ Data fit register size: best performance



*Optimization parameter: **task granularity***

GPU kernel: spherical coordinates

- **CPU:** transforming Cartesian to polar coordinates: $\cot(\varphi) = \frac{x}{y}$
  - Quantification boundaries $\frac{x}{y}$ stored in LUT[2]
  - Find binning by linear search



- **GPU:** low performance of branches
  - GPU vector operations count positive signed elements
  - Two vector operations identify quantified angle (binning)

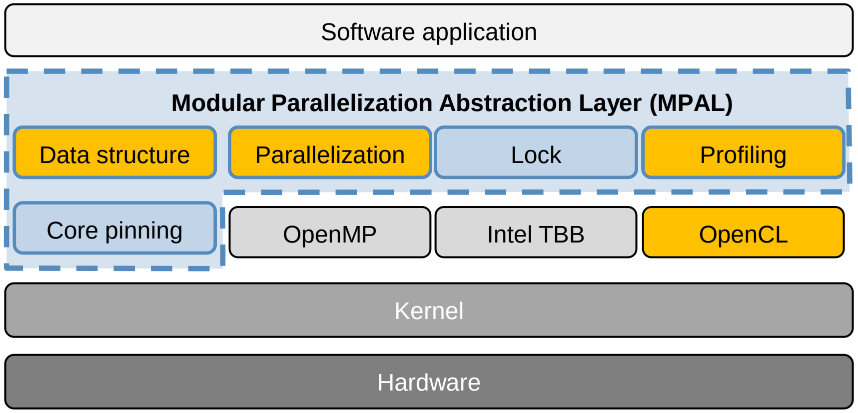
			e.g., ⇒					
Cot(22,5°)	−	gradX/gradY		2,414	−	0,82	=	+ 1,594
Cot(45°)		gradX/gradY		1		0,82		+ 0,18
Cot(67,5°)		gradX/gradY		0,414		0,82		− 0,406
...		...		...		...		...

Optimization restriction: ***instruction set***

[2] O. J. Arndt, D. Becker, F. Giesemann, G. Paya-Vaya, C. Bartels, and H. Blume. Performance evaluation of the Intel Xeon Phi manycore architecture using parallel video-based driver assistance algorithms. In International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IEEE, 2014.

Abstraction layer

- **MPAL: Modular Parallelization Abstraction Layer[3]**
  - Abstracting CPU parallelization (shared memory)
  - No framework specific syntax (e.g., OpenMP, TBB)
  - Config flag specifies applied framework
- **Integration of GPU support (OpenCL)**
  - GPU compatible memory (self-mapping memory)
  - Measuring GPU events for automated profiling

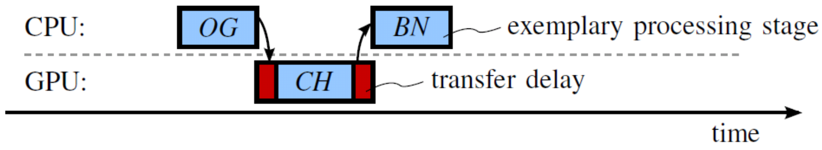


Generic and flexible offload strategies through abstraction.

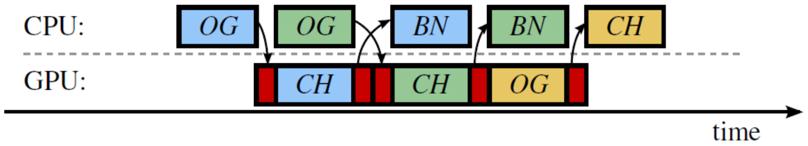
[3] O. J. Arndt, T. Lefherz, and H. Blume. Abstracting Parallel Programming and its Analysis Towards Framework Independent Development. Intl. Symp. Embedded Multicore/Many-Core System-on-Chip (MCSoC-15), IEEE, 2015

Partitioning and mapping strategies

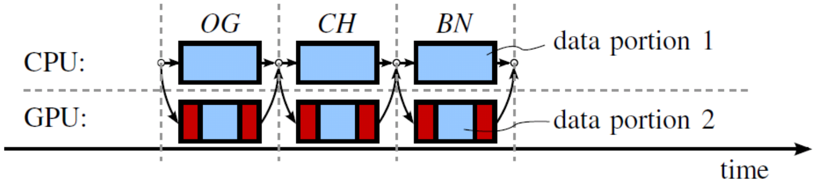
- **Single stage offload:**
  - Low implementation effort
  - Low resource exploitation



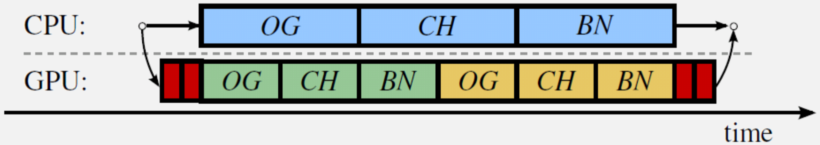
- **Workload adaptivity**
  - High processing stage fluctuation
  - High data transfer rates



- **Data decomposition:**
  - High management overhead
  - Missing memory coherency

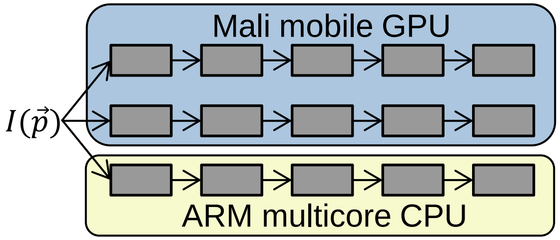


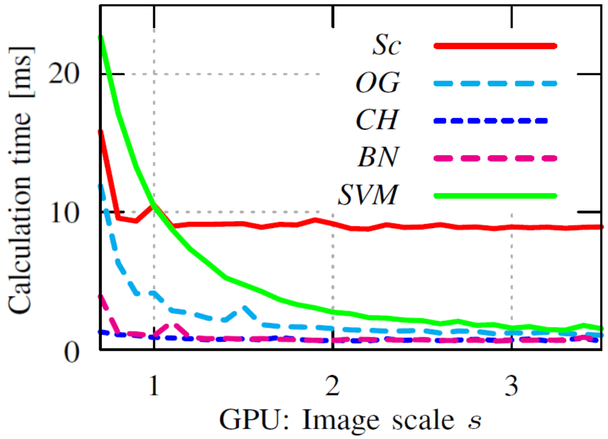
- **Static work split:**
  - Offload scheme well evaluated
  - Low management overhead



Evaluation of the processing units

- Execution time exclusively on A15: **270 ms** (reference time per frame)
  - A7: **715 ms** (speedup: 0.38)
  - GPU: **700 ms** (speedup: 0.39)
- Offload entire processing chains (image scales)
  - Best work distribution: factors 0.7 – 1.1
  - Execution time: **200 ms** (speedup: 1.35)
- Particular processing stages on GPU:
  - Scaling and SVM not efficient (403 ms)
  - Extreme variations in speedups



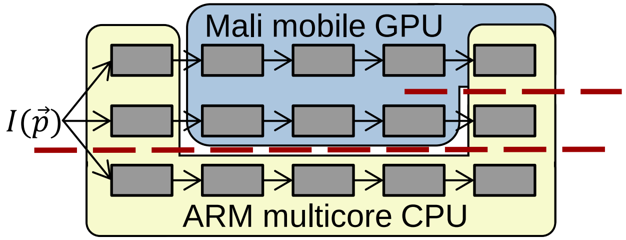


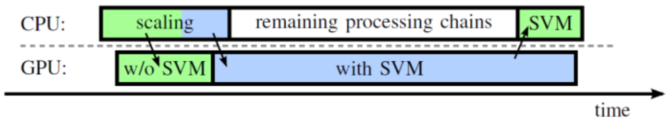
Offload only selected stages

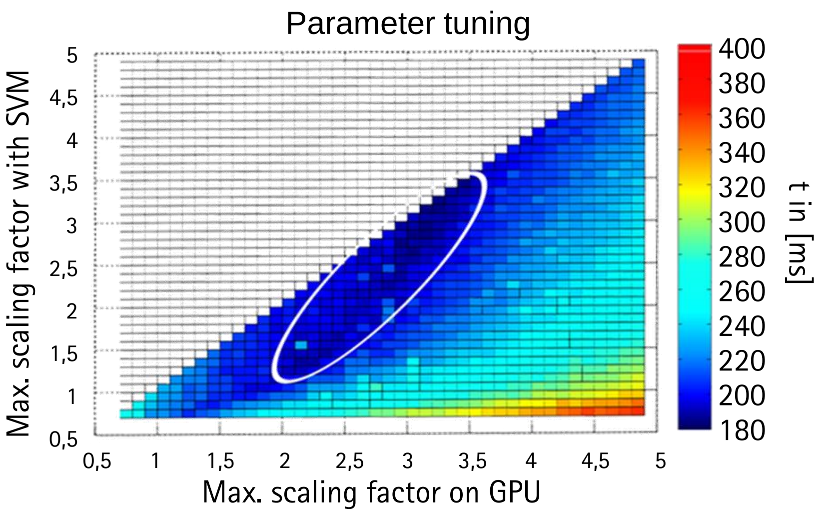
Tuning the offloading

- **Only partially offloading processing stages**
  - **CPU:** Sc, SVM; **GPU:** OG, CH, BN: **200 ms**
  - Work distribution: GPU idling for 153 ms
- **Splitting processing chains**
  - Multiple degrees of freedom
  - Full search DSE: **181 ms**
- **Manual memory optimization**
  - Reduced code flexibility
  - Execution time: **161 ms**

Individual speedups necessitate costly DSE of mapping parameters.







Conclusion

- Heterogeneous implementation:
  - GPU kernel implementation of the HOG algorithm
  - Abstraction layer extension for heterogeneity
- Shared memory (zero-copy) platform:
  - Missing coherency and transfer (mapping) complicates data sharing
  - Application specific units require costly evaluation of mapping parameters
- Evaluation results (557x631 px):
  - Statically splitting processing chains: **161 ms** (speedup **1.68**) at 7.15 W
  - Transfer and management overhead: **30 %**
  - Gain in performance per watt ratio of **53 %**