Hochschule Osnabrück
University of Applied Sciences

An Industrial Case Study on Data Visualization combining CPU, FPGA and GPU with the SAccO Interface

Markus Weinhardt

Small-Scale Heterogeneous Multiprocessing

Thematic Session, Oct. 10, 2014

Outline

- **Background: HPVis Project**
- **SAccO Accelerator Framework**
- **GPU Extension**
- **Performance Analysis and Results**
- **Conclusions and Future Work**

Publication:
M. Weinhardt, A. Krieger, Th. Kinder: *A Framework for PC Applications with Portable and Scalable FPGA Accelerators*
Proc. Int. Conf. on ReConFigurable Computing and FPGAs, Cancun, Mexiko, Dec. 2013

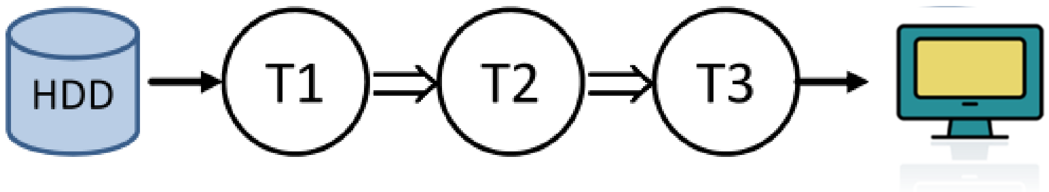
Background: HPVis Project

HPVis: High-Performance Processing and Visualization of High-Volume Data

- ▸ Application recast as task-parallel software on standard PCs where some tasks *can* be mapped to FPGA; the other tasks remain unchanged.
- ▸ Tasks communicate over streaming interface.

Example visualization task graph:

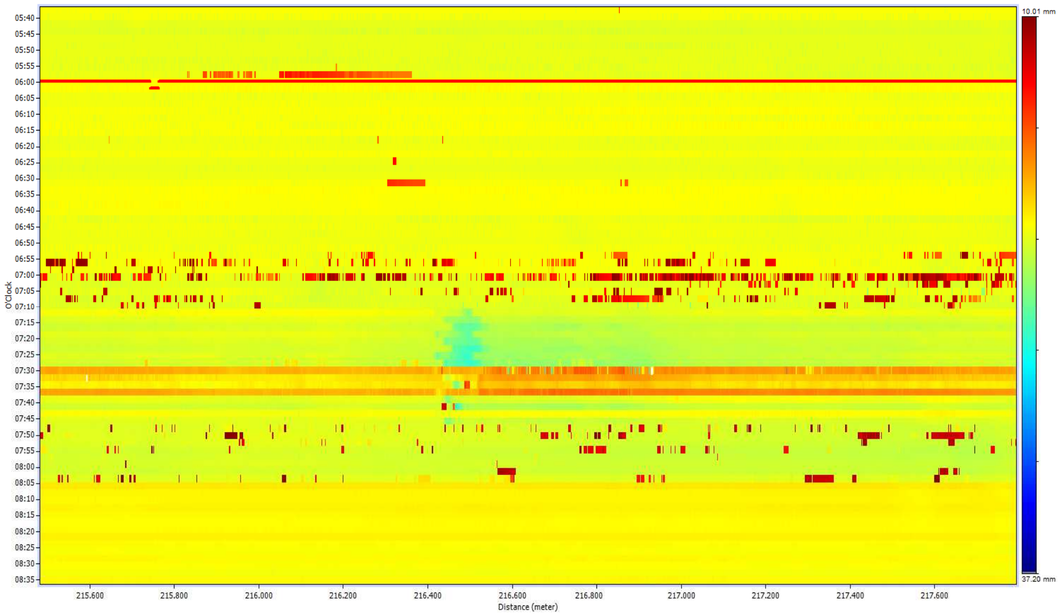
(restricted to linear task graphs/pipelines, but multiple channels possible)



- ▸ **Project requirements:**
  - For hardware efficiency, application kernels implemented in optimized VHDL (RTL)
  - CPU and FPGA communicate over PCI-Express (PCIe)
  - Applications (SW and HW) should be portable and scalable
  - Later extended by processing on a GPU (using CUDA)

Background: HPVis Project

- **Main Application:**
  - Visualization of sensor data gathered by pipeline inspection "pigs" (mainly for oil and gas pipelines)
  - Performance of zooming and scrolling is not sufficient for human inspectors on PCs, should be accelerated



Color Scan View

(Source: Rosen Technology and Research Center)

SAccO Accelerator Framework: API

SAccO: Scalable Accelerator platform Osnabrück

- Starting Point: Task-parallel implementation of streaming application, completely in software
- Uses high-level API for both socket-based SW-SW and PCIe-based SW-HW communication

Data Flow (DF) Transfers

```
uint8_t WriteDF(uint8_t num, void *pval)
uint8_t ReadDF(uint8_t num, void *pval)
uint8_t StreamWriteDF(uint8_t num, void* pdata, uint16_t size)
uint8_t StreamReadDF(uint8_t num, void* pdata, uint16_t size)
uint8_t StreamReadWriteDF(uint8_t rnum, void* prdata, uint16_t rsize,
                          uint8_t wnum, void* pwdata, uint16_t wsize)
```

- Automatically detects FPGA board and redirects communication accordingly (also for cheap PCs w/o FPGA card)

SAccO Accelerator Framework: HW Implementation Rules

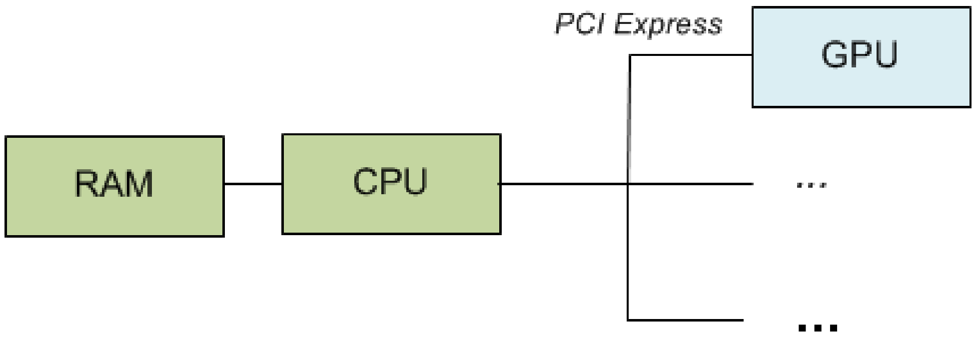
To ensure portability and scalability, some rules are imposed for the VHDL implementation of the kernels in hardware:

- **Portability**
  - communication: FPGA and board specific features are encapsulated in PCIe wrapper; handshake protocol for synchronization with PCIe data streams
  - no direct instantiation of FPGA specific components (Block RAM, DSP etc.) ➔ infer by synthesis!
  - ensure minimal user clock frequency (currently f_{user} = 125 MHz) for all designs

- **Scalability**
  - if possible, designs consist of *Processing Elements (PEs)* which can be replicated in parallel to adapt to FPGA size and PCIe bandwidth ➔ implement as many parallel PEs as possible (semi-automatic, for details see ReConFig'13 paper)

SAccO Accelerator Framework: Hardware Architecture

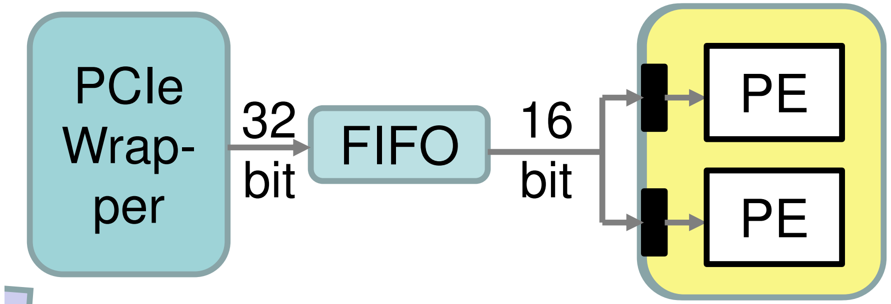
System Architecture:



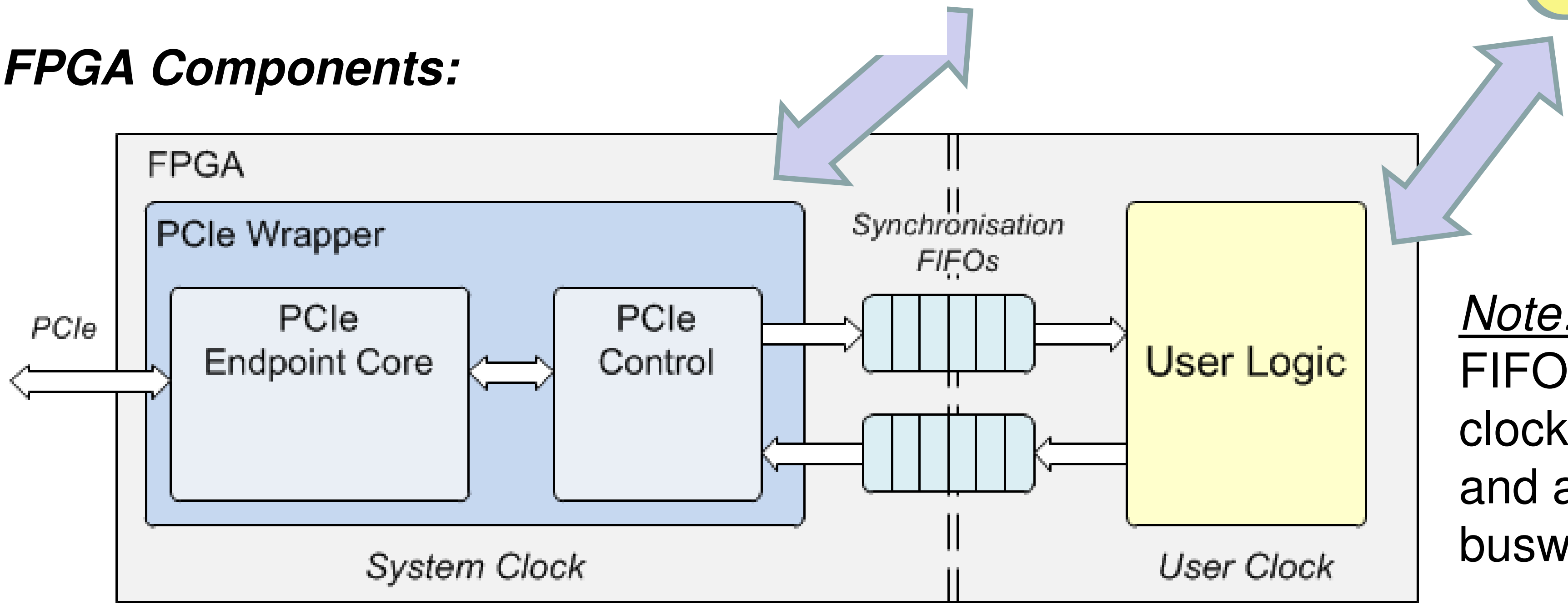
User Logic with PE Scaling:

User Logic (R=2)

PE: 8-bit input port

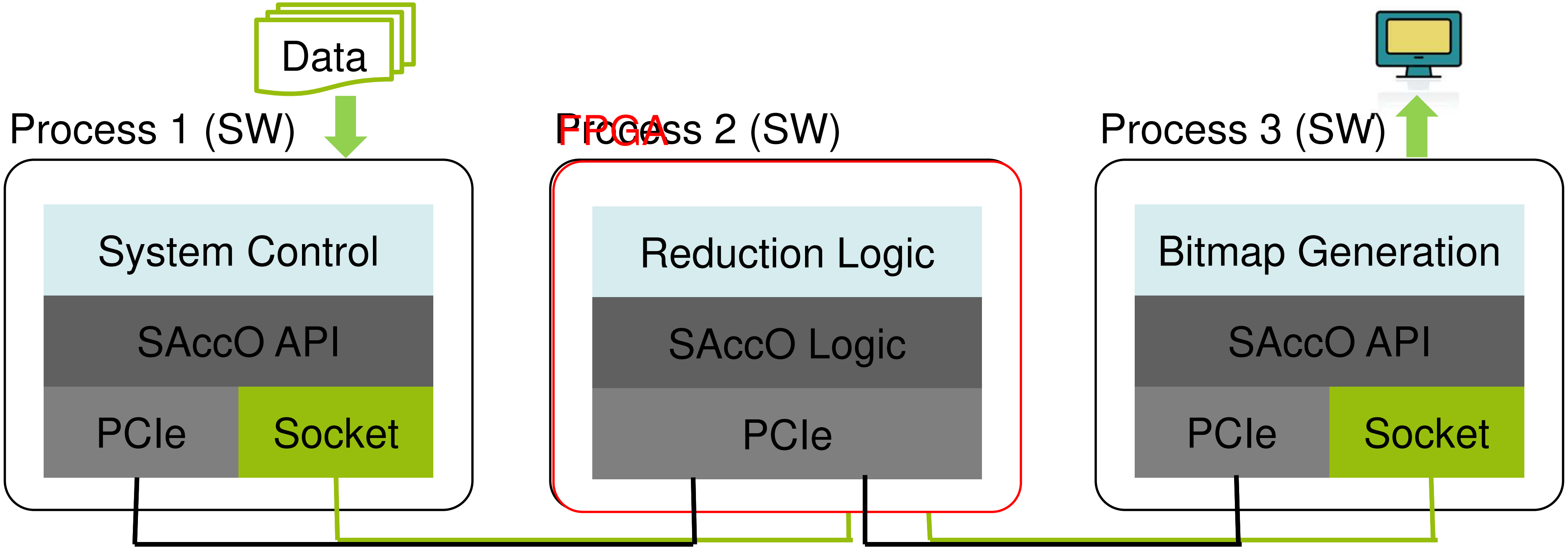


FPGA Components:

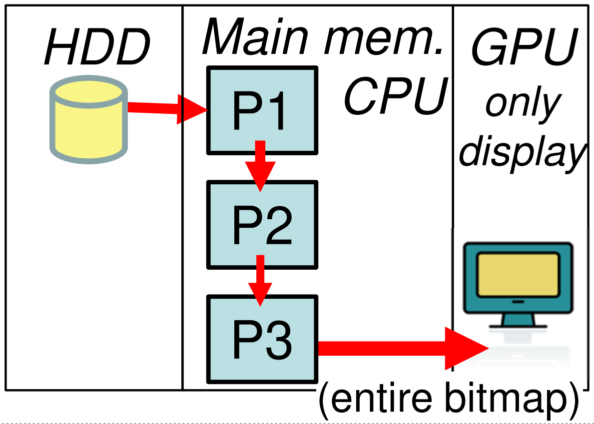


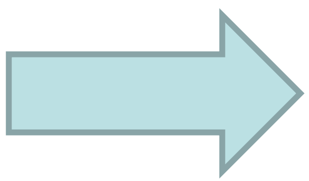
<u>*Note:*</u>
FIFOs traverse clock domains and adjust buswidth!

SAccO Accelerator Framework: Visualization Application

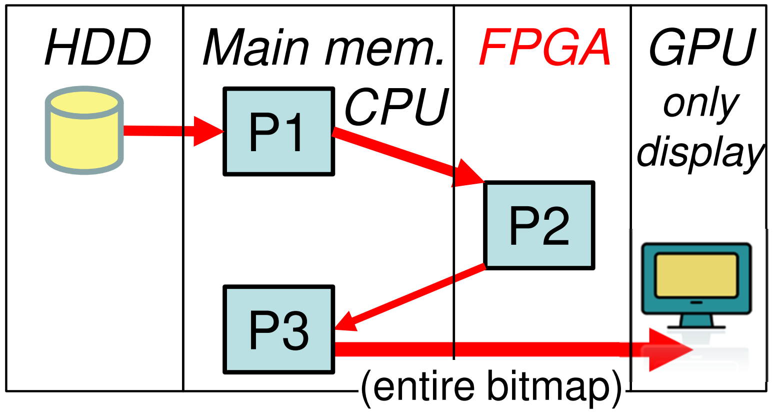


If API detects FPGA board, socket communication is replaced by PCIe communication. (HW processes are marked in application setup file.)





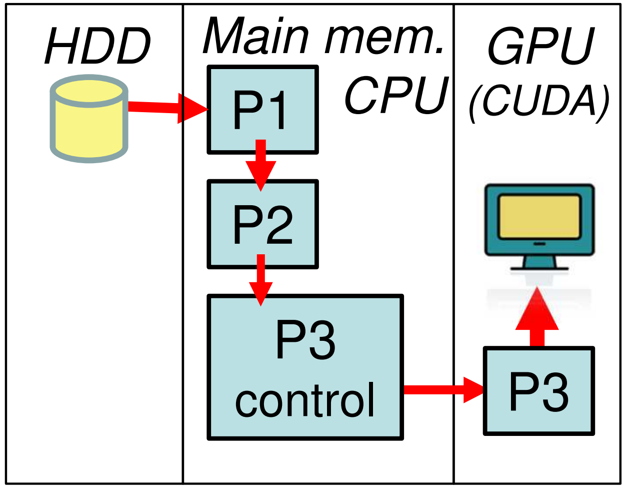
Dataflow extended by FPGA component



GPU Extension

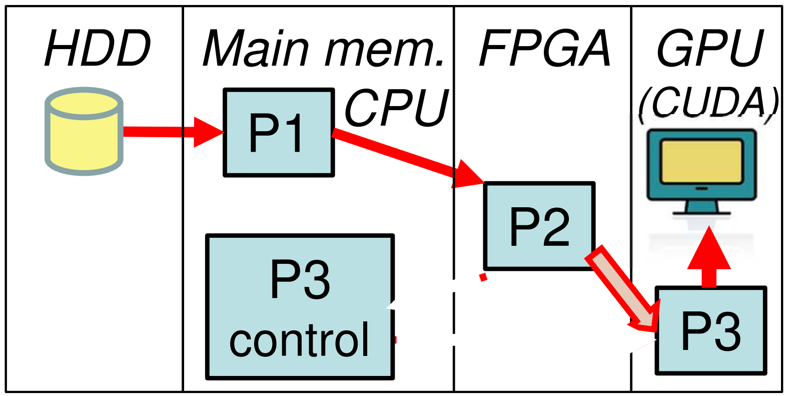
- **Project extension: Process 3 (Bitmap Generation) accelerated by CUDA kernels on a NVIDIA GPU**
  - Not entire task on GPU, just kernels
  - Remaining parts of Process 3 on CPU
  - Data copied in blocks by `cudaMemcpyAsync()`, not as streams with the SAccO API
  - Bitmap directly copied to OpenGL texture and visualized

Dataflow CPU+GPU:



- **Combination with FPGA and GPU**
  - No direct FPGA→GPU transfer implemented yet
  - Next step: use GPUdirect RDMA? (for NVIDIA Kepler/Maxwell GPUs)

Dataflow CPU+FPGA+GPU:



Performance Analysis and Results

Simplifying assumption: All processes and transfers overlap
➔ Overall performance limited by component with lowest throughput

- **HDD→main memory (host) transfer rate:** ≈ 100 MB/s
- **P1 (host):** No processing, limited by transfer rates
- **Host→FPGA:** ≈ 137 MB/s (SP605, PCIe x1 Gen1),
  ≈ 1099 MB/s (ML605, PCIe x8 Gen1)
- **P2 (FPGA)** - *not optimized*: Input values consumed at ≈ 32 MB/s
- **FPGA→host:** ≈ 172 MB/s (SP605), ≈ 1373 MB/s (ML605)
- **Host→GPU:** ≈ 12 GB/s (GTX750 Ti, PCIe x16 Gen3)
- **P3 (GPU):** Pixels produced in CUDA mem. at ≈ 30 GB/s (GTX750 Ti)
- **CUDA memory→OpenGL texture:** ≈ 80 GB/s
  Note: Not overlapped with CUDA bitmap generation!

Results:

- Speedup 6x - 9x achieved for P3 on GPU including Host→GPU transfer. Overall system speedup depends on data set, PCIe system.
- P2 on FPGA is a performance bottleneck, must be further optimized.

Conclusions and Future Work

Conclusions

- Our portable and scalable approach simplifies implementation of streaming applications on PCs without and with FPGA boards, can be extended with CUDA kernels on GPU.
- For such heterogeneous systems, many manual optimizations are required, e. g. further optimization of FPGA design (P2).
- In our application, GPU processing (P3) is bandwidth-limited.

Future Work

- Implement Reduction Process P2 on GPU as well
  ➔ only one PCIe Gen3 transfer (will probably be fastest solution)
- Direct PCIe communication FPGA→GPU (GPUdirect or other)
- Use High-Level Synthesis for FPGA
- Include memory access on FPGA boards

Thank you! Time for questions…