HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Challenges of Advancing Coarse-Grained Reconfigurable Arrays from Embedded to High-Performance Computing

Markus Weinhardt

CGRA4HPC'23 - St. Petersburg/FL, USA - May 15, 2023

Outline

- CGRAs for Embedded Systems
- CGRAs for High-Performance Computing: Challenges
- HiPReP:
  <u>Hi</u>gh-<u>P</u>erformance <u>Re</u>configurable <u>P</u>rocessor
- Related Work, Future Work and Conclusion

CGRAs for Embedded Systems

CGRAs for Embedded Systems

Basic Principles of CGRAs

- ► Alternative to multicore processors for exploiting instruction-level and loop-level parallelism
- ► 2D array of *Processing Elements* containing ALUs and interconnect
- ► Similar to FPGAs, but with ALUs instead of look-up tables, word-wide instead of bit-wide connections
- ► In most cases, dataflow graphs (DFGs) of inner loop kernels are mapped to the Processing Elements (PEs)
  - Parallelization/Vectorization required to enable loop pipelining!

Implementation

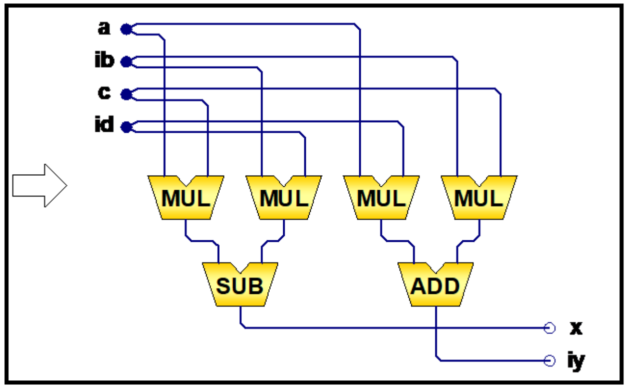
- ► As *ASIC* or part of a *SoC*
- ► As *FPGA overlay* („Virtual CGRA“)
  ➔ flexible, but less efficient (introduces another configuration layer)

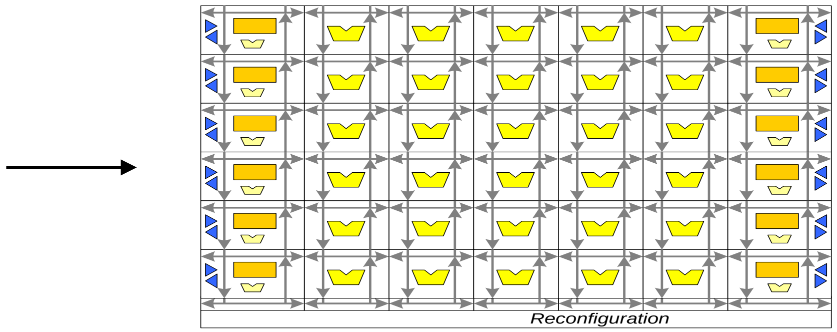
CGRAs for Embedded Systems: Single-Context

- Early CGRAs directly mapped DFGs to ALUs (1:1), aiming at a balanced pipeline → high parallelism, high throughput
  - Placement and Routing similar to FPGA tools
- Examples
  - **Xputer** (rDPU/KressArray) – Prof. Hartenstein, Univ. Kaiserslautern/Germany
  - **PACT XPP** – PACT XPP Technologies AG, Munich/Germany

PACT XPP Dataflow Array

- Optimized for signal/image processing codes
- Data stream synchronization in hardware (handshake protocol)



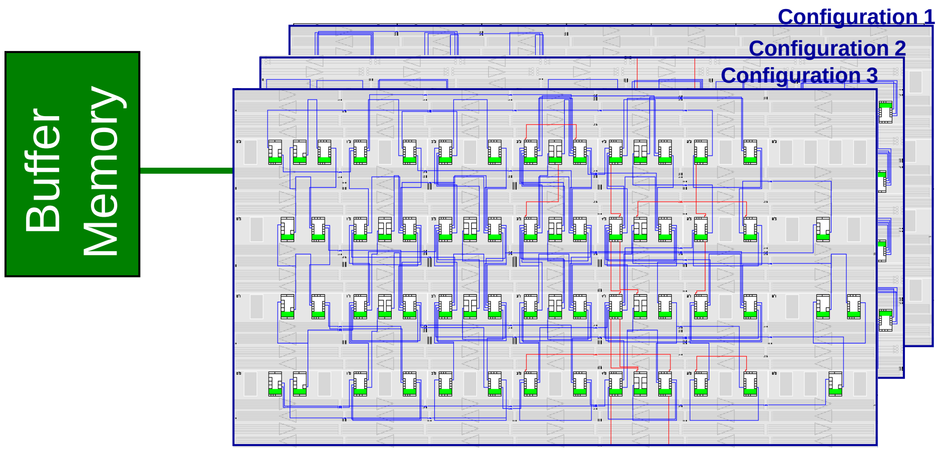


RAM-PAEs and ALU-PAEs (Processing Array Elements)

CGRAs for Embedded Systems: Single-Context (cont‘d)

PACT XPP Dataflow Array (cont‘d)

- Single context (i. e. configuration of PAEs/bus connects) running at a time
- Fast, partial reconfiguration (in µ-secs), overlapping with execution (configuration stored outside CGRA and loaded sequentially)
- Local buffer memories / FIFOs typically used for intermediate results → uses fast SRAM (on-chip or off-chip)
- Reconfiguration after larger phases (e. g. in MPEG decoder)



Reference:

V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt: *PACT XPP - A Self-Reconfigurable Data Processing Architecture*, Journal of Supercomputing, Vol. 26, No. 2, Sept. 2003, Kluwer Academic Publishers

CGRAs for Embedded Systems: Multi-Context

- Problem with single-context CGRA
  - Large DFGs must be split into several configurations
    → reconfiguration overhead, buffer memory required
  - Cyclic DFGs (accumulators etc.) reduce throughput/effective PE utilization
- Solution: Multi-context CGRA
  - Every PE locally stores several instructions (contexts) which can be changed every cycle (*single-cycle configuration*); local register file for data reuse/ feedback
    → large DFGs fit on smaller array (with reduced throughput)

Seminal work: **ADRES Template** (IMEC, Belgium)

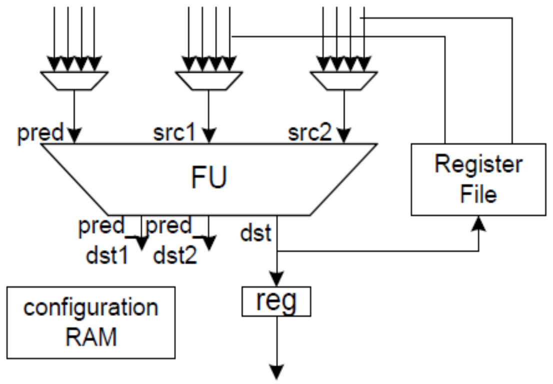
- bought and used by Samsung!
- new compilation methods (e. g. DRESC based on modulo-scheduling) were developed for multi-context CGRAs

Reference:

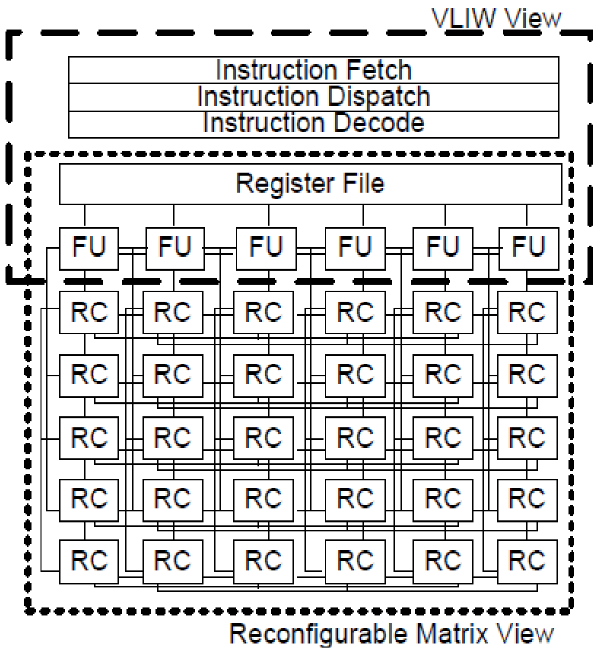
B. Mei, S. Vernalde, D. Verkest, H. D. Man and R. Lauwereins, *ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix*, in Proc. 13th Intern. Conference on Field-Programmable Logic and Applications, FPL 2003

CGRAs for Embedded Systems: Multi-Context (cont‘d)

ADRES Template (cont’d)



Reconfigurable Cell (RC)
(a)



(b)

Source:
FPL 2003

- ALU (FU) performs integer operations in one cycle
- DRESC compiler maps (several) operations to one PE (RC) at different time steps (contexts). Scheduler cycles through contexts (in lock-step).
- Results are stored in local register file (for reuse/feedback cycles) and/or forwarded to neighboring PEs.
- Uses modulo routing resource graph (MRRG) for scheduling and simulated annealing for placement.

CGRAs for High-Performance Computing: Challenges

CGRAs for HPC: Challenges

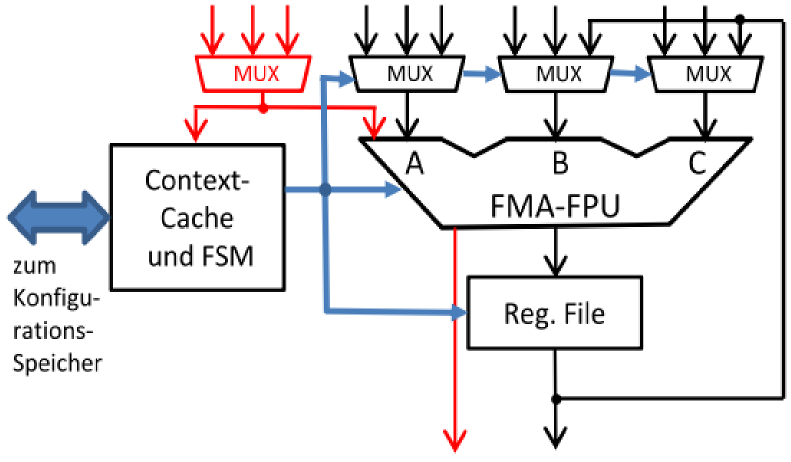
- **Support for Floating-Point Operations**
  - Older CGRAs provide no or limited support for FP operations (e. g. by combination of two integer ALUs)
    → Extend CGRAs with Floating-Point Units (FPUs)
  - Hardware and compiler must handle multi-cycle operations (with and without pipelining) or operations with varying latency (e. g. FP division)
    → no simple, predictable schedule
- **High Memory-Bandwidth Requirements**
  - CGRA must be integrated in memory hierarchy of host system and/or access (large) DRAM blocks
    → varying memory access time must be handled
    → no static, fixed schedules, or complete CGRA halt required when memory stalls
  - Complex address generation for multi-dimensional array accesses required
    → Dedicated Address-Generator Units (AGUs) save precious PEs with FPUs for FP computations
- **Irregular Codes (e.g. graph algorithms)**

HiPReP:

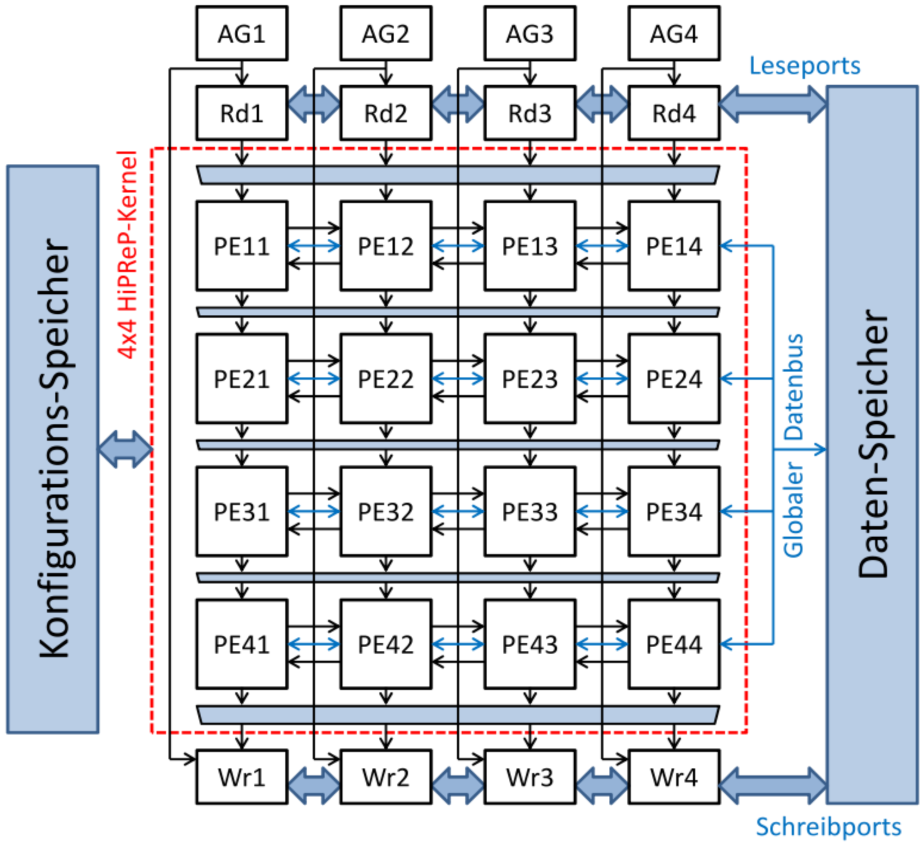
<u>Hi</u>gh-<u>P</u>erformance <u>Re</u>configurable <u>P</u>rocessor

HiPReP: High-Performance Reconfigurable Processor

- Project at *Osnabrück UAS* (funded by German Research Found. - DFG)
- Goals:
  - Hardware design and C compiler for CGRA with FPUs
  - Combining ideas from ADRES and XPP
- First idea:



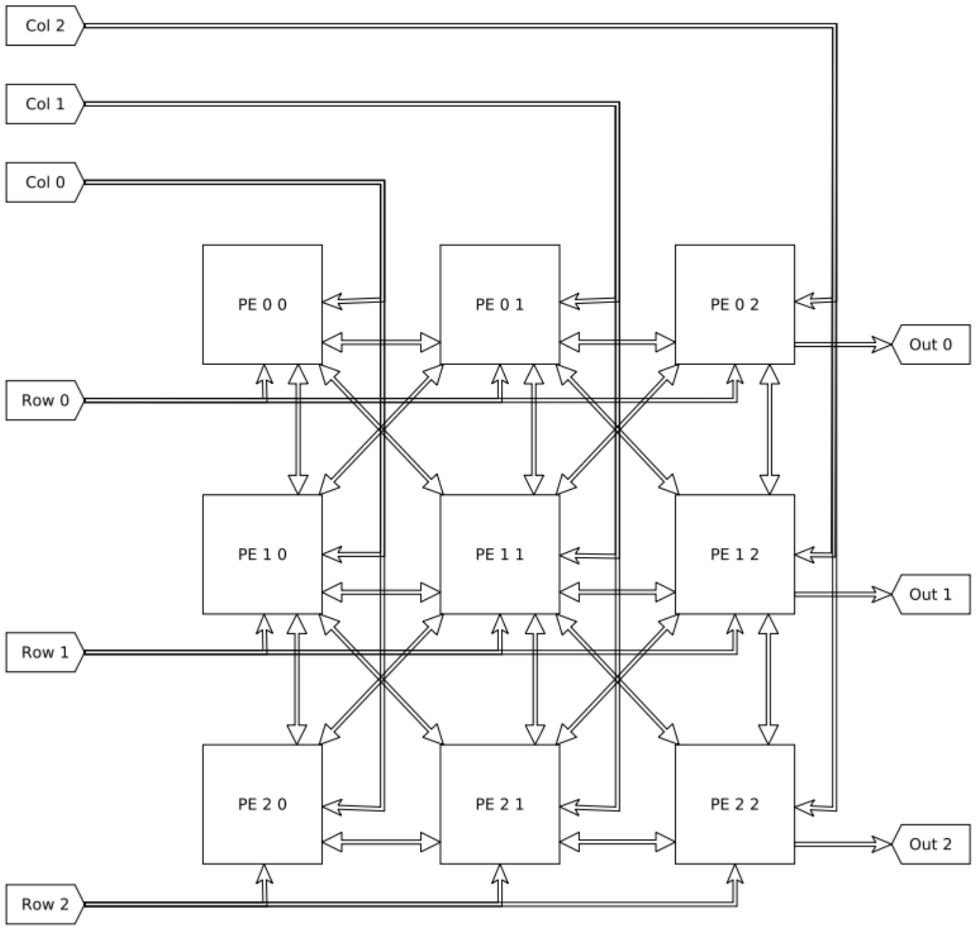
Processing Element



HiPReP Architecture Template – Array Architecture

CGRA template:

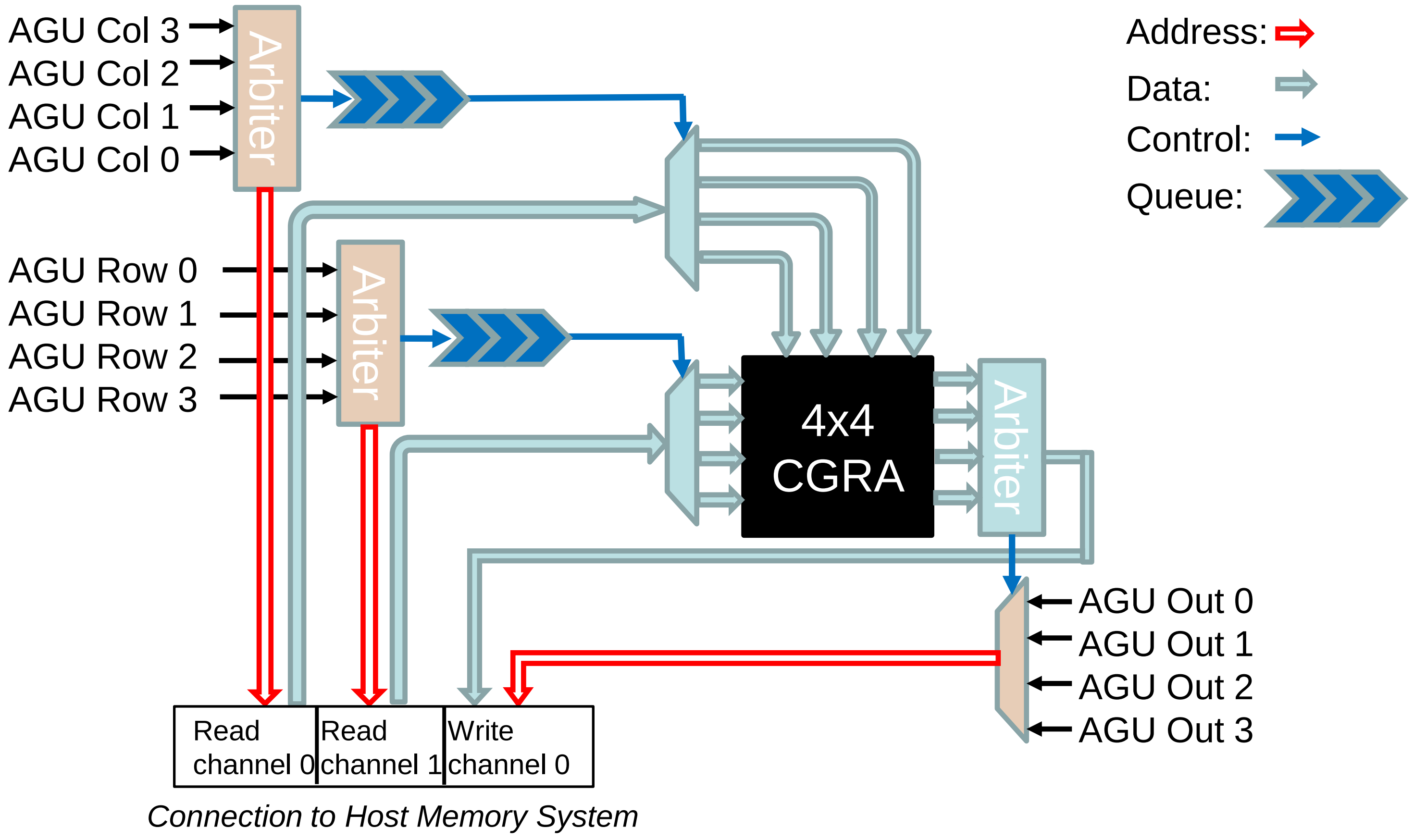
- *Direct communication with 8 nearest neighbors* (bidirectional *32-bit channels*)
- All connections auto-synchronize via *handshake signals*
- **Streamed load/store:**
  Memory accessed by AGUs,
  → array access for two nested loops
  Read-AGUs: Can broadcast to entire row (Row 0-2) or column (Col 0-2), respectively
  Write-AGUs: Connected to rightmost PEs only (Out 0 -2)



3x3 CGRA

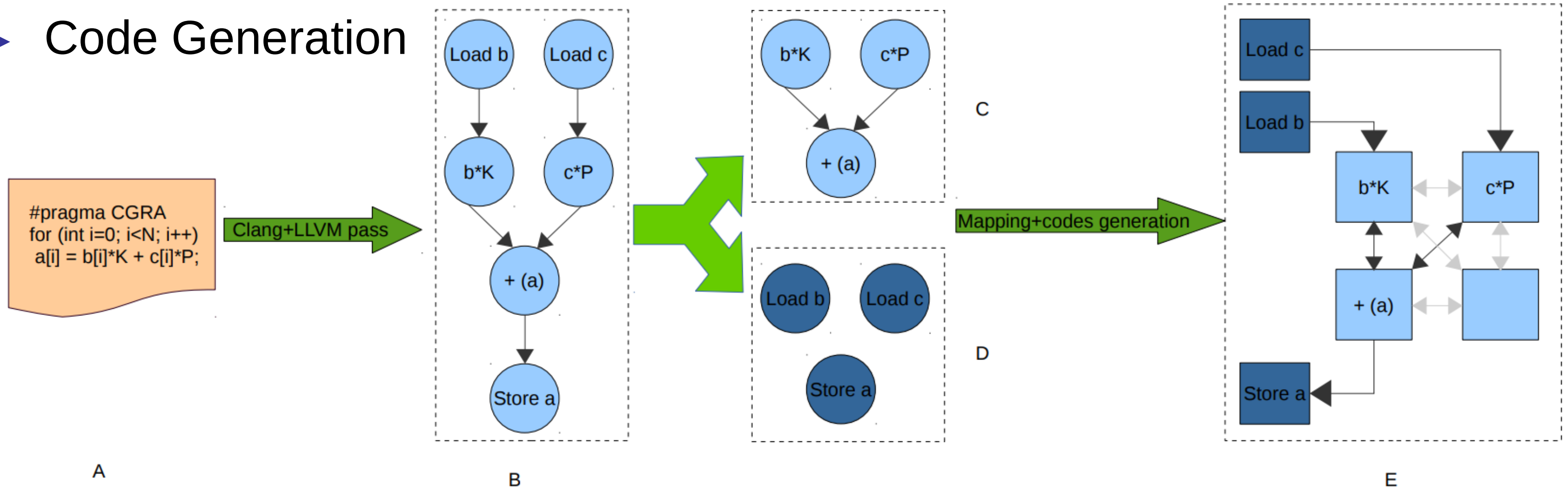
HiPReP Architecture Template – Memory Access

- **Arbiters** combine AGUs to channels, connect to host/memory system
- Same buses also used to configure context memories
- Scalable # of channels, e.g. 4x4 CGRA with 2 read and 1 write channel:



CHiPReP C Compiler

- Based on *LLVM* and *CCF* compilation frameworks
- Annotated inner loops (**A**) mapped to Data Dependence Graph (**B**) and *split into execution part* (**C**), mapped to PEs, *and memory-movement part* (**D**), mapped to AGUs
- *Clustering heuristic* combines DDG nodes in one PE (increases PE utilization, but decreases throughput!)
- Combined Placement, Routing and Pipeline Balancing maps clusters on PEs and memory accesses on AGUs (**E**), optimized by Simulated Annealing
- Code Generation



```
#pragma CGRA
for (int i=0; i<N; i++)
 a[i] = b[i]*K + c[i]*P;
```

HiPReP Architecture Template – Processing Element (PE)

- Differs from usual CGRA-PEs
  → 32-bit integer ALU *and* single-precision, pipelined Floating-Point Unit
  → Fused-Multiply Add (FMA) operator (for mul-add/mul-acc operations), 3rd operand from dedicated register, not as general as opds. 1 and 2!
- Homogenous and heterogeneous array can be generated/synthesized
- Each PE has private context memory, executes 32-bit RISC-like instructions → independent control flow (not in lock-step)
- Register File (32 32-bit registers):
  - Input operands are read from internal register file or output registers of neighboring PEs
  - PE's output written to internal register file or output registers
- *Hazard detector* (comparable to scoreboard) used for synchronizing operations with varying latencies (in-order issue/out-of-order completion)
- Instruction Set supports comparisons, conditional/unconditional jumps (no predicated instructions!), zero-delay jumps for infinite loops

HiPReP Architecture Template – Processing Element (PE)

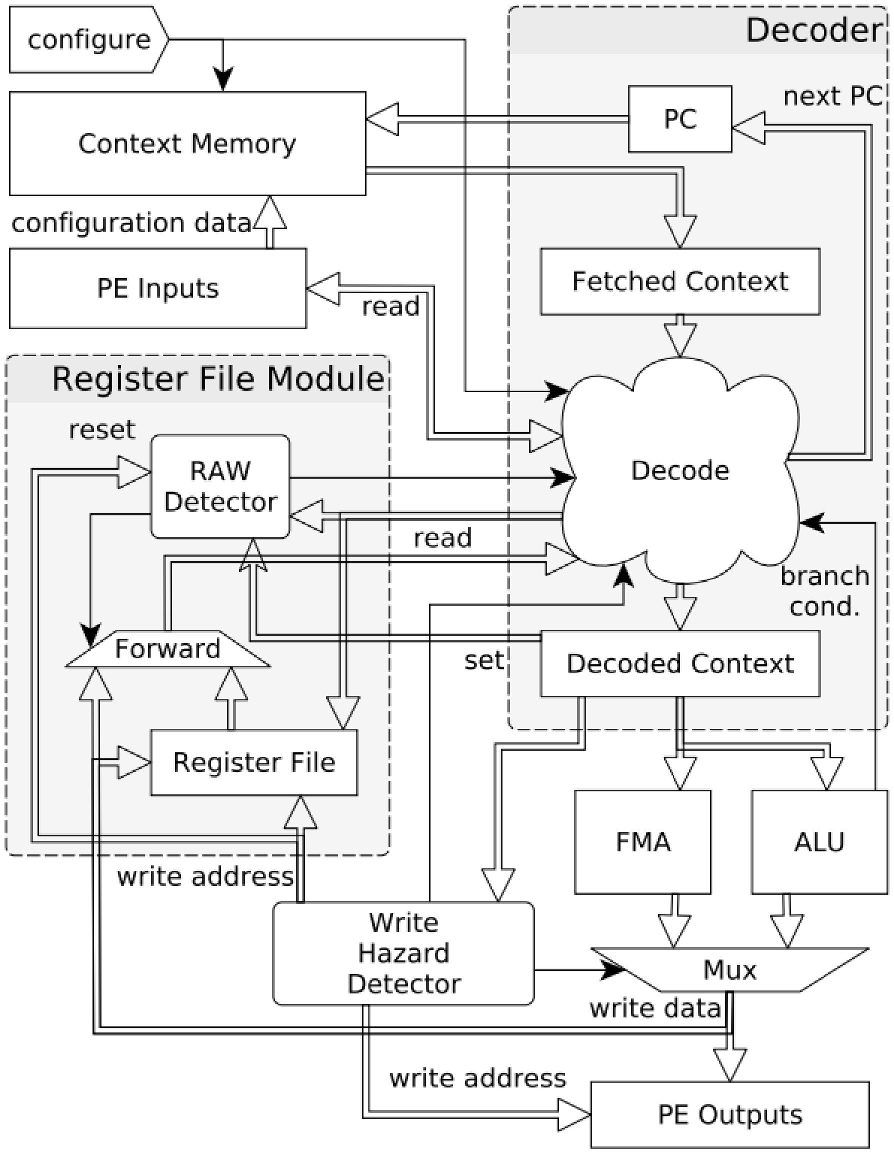
PE Components

- ▶ FMA Unit performs FP instructions (add, sub, mul, cmp, macc, fma, fms, int2fp, fp2int)
- ▶ RAW Detector enforces RAW dependences
- ▶ Write Hazard Detector enforces correct out-of-order completion

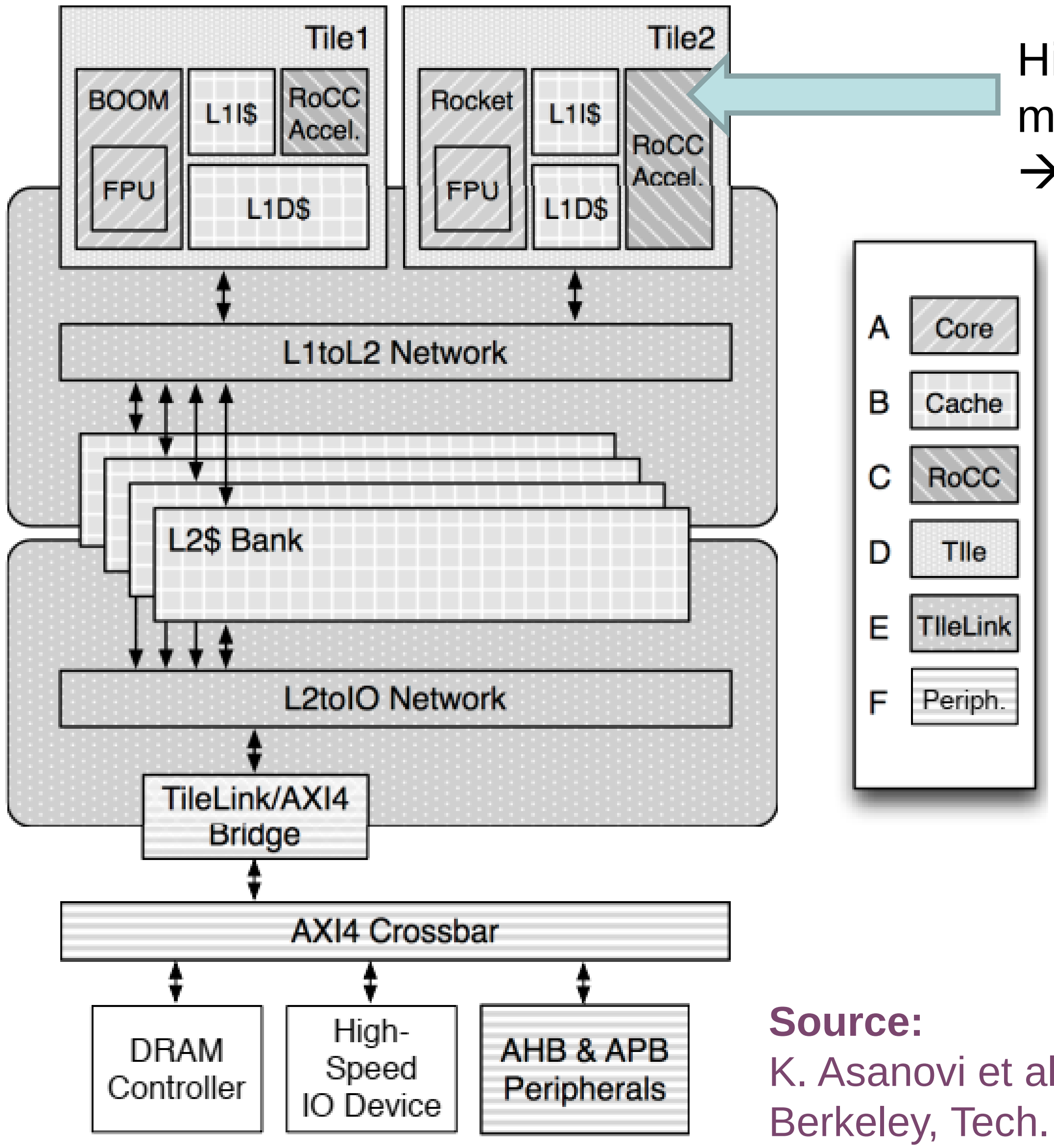
References:

P. Käsgen, M. Weinhardt, C. Hochberger:

- *A Coarse-Grained Reconfigurable Array for High-Performance Computing Applications*, Intern. Conf. on ReConFigurable Computing and FPGAs (ReConFig), 2018
- *Dynamic Scheduling of Pipelined Functional Units in Coarse-Grained Reconfigurable Array Elements*, Intern. Conf. on Architecture of Computing Systems (ARCS 2019), 2019



Integration in Rocket Chip / Synthesis Results



HiPReP Core (synthesizable Chisel model) used as RoCC Accelerator
→ direct access to L1 cache!
(i.e. 1 read and 1 write channel combined)

Synthesis Results

3x3 CGRA including AGUs, Synopsys Design Compiler on UMC 65 nm LL process:

- Area: ~ 1 mm^2
- Frequency: ~ 770 MHz (wireload model WL20)

Source:
K. Asanovi et al.: *The Rocket Chip Generator*, UC Berkeley, Tech. Report No. UCB/EECS-2017-17

Related Work, Future Work and Conclusion

Related Work, Future Work and Conclusion

- **Related Work: Riken High-Performance CGRA (RHP-CGRA)**
  - 1:1 mapping of DFG to PEs
  - High memory-bandwidth through Address Generators (AGs) connected to external memory via Memory Controllers and Interconnect Network
  - Tiles of CGRAs and On-Chip SRAM envisioned

  Reference:
  A. Podobas, K. Sano, S. Matsuoka*: A Template-based Framework for Exploring Coarse-Grained Reconfigurable Architectures*, Proc. Intern. Conf. on Application-Specific Systems, Architectures and Processors (ASAP) 2020

- **Future Work (HiPReP)**
  - Increase memory bandwidth with individual D-caches (or with scratchpad memory) for several memory channels
  - Benchmark Analysis and Design Space Exploration

- **Conclusion**
  - There are promising approaches to extend CGRAs to HPC, but no complete, ready-to-use system avalable yet.
    ➔ Especially solutions for irregular code are lacking!

THE END

Thank you for your attention!

Any questions?