CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

SoC programming in the era of the Internet of Things, machine learning and emerging technologies

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

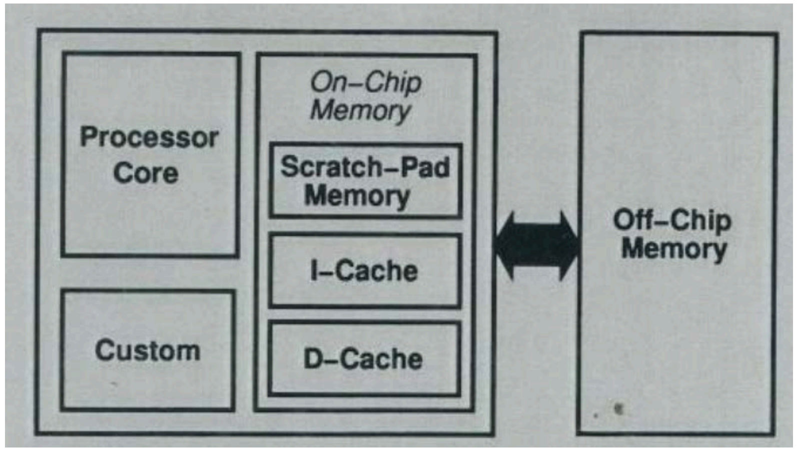
Groupement De Recherche SOC2: System On Chip, Systémes embarqués et Objets Connecté

Montpellier, France. June 20 2019

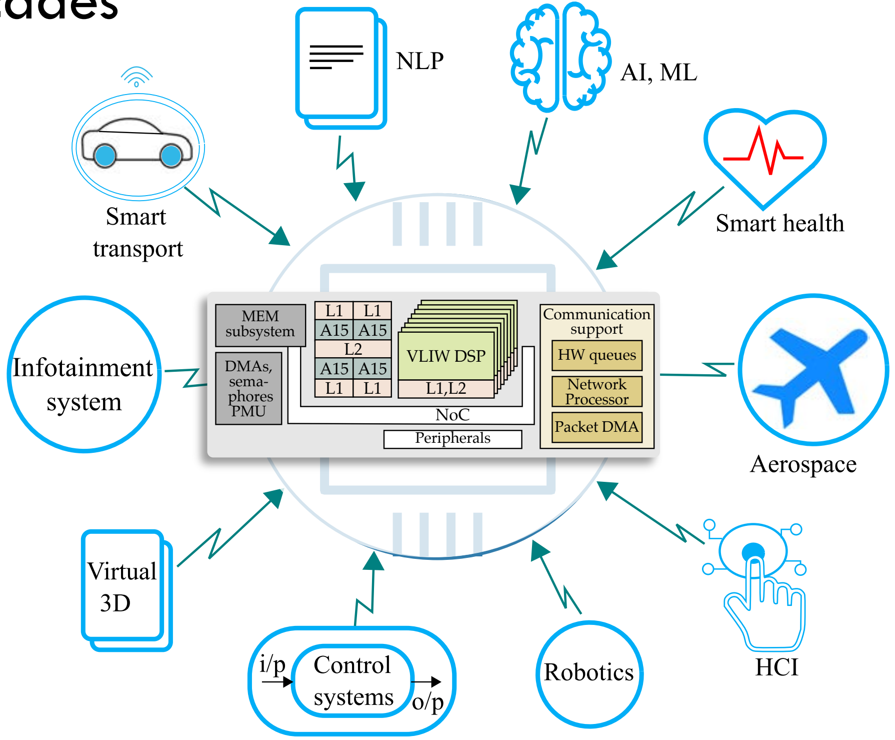
TECHNISCHE UNIVERSITÄT DRESDEN   DRESDEN concept   DFG   WR WISSENSCHAFTSRAT

Systems on Chip (SoC): Evolution

- SoCs: Long history of specialization and interaction with environment
- Incredible evolution over the last decades

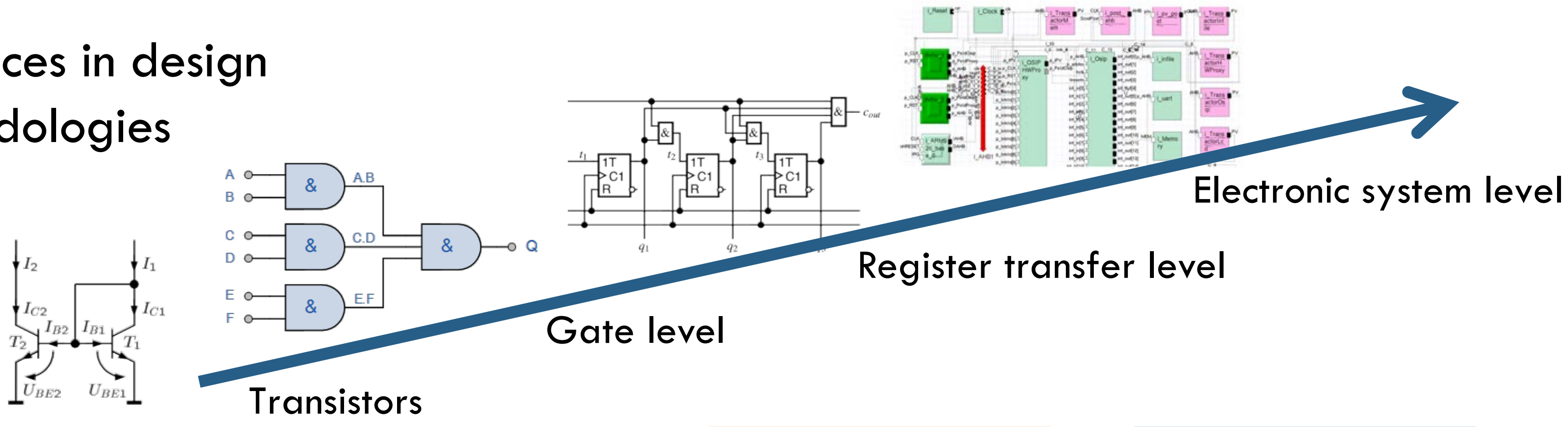


Panda, P. R., Dutt, N. D., & Nicolau, A. Memory issues in embedded systems-on-chip: optimizations and exploration. Springer Science & Business Media. **1999**

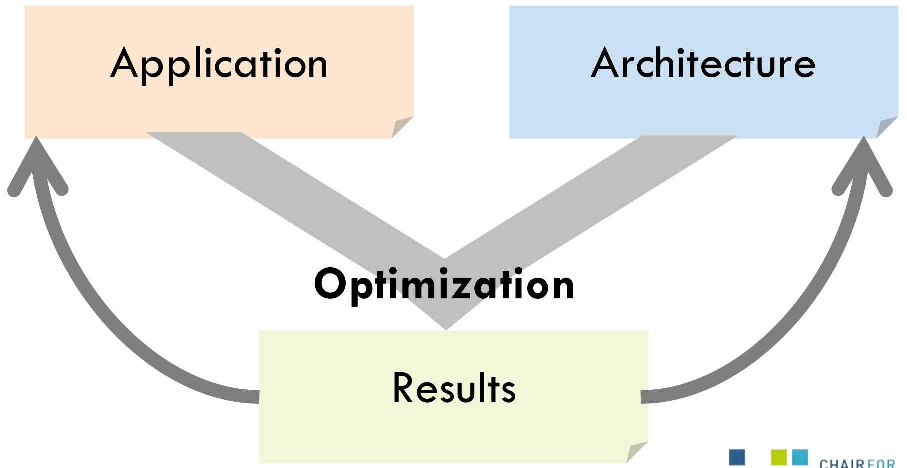


SoC design and programming: Handling complexity

- Advances in design methodologies



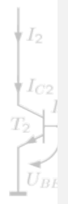
- Model-based programming



SoC design and programming: Handling complexity

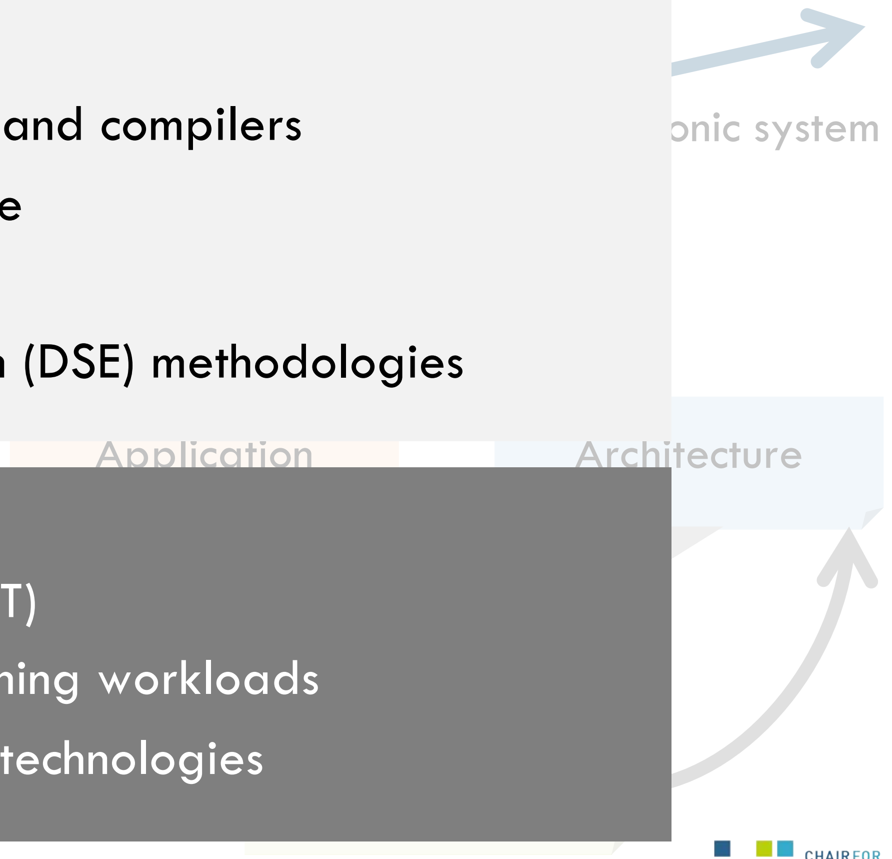
- Advance
  methodo

onic system level



Exciting innovations in

- Modeling languages
- Programming languages and compilers
- Costs models of hardware
- System simulators
- Design space exploration (DSE) methodologies



- Model-b

New challenges

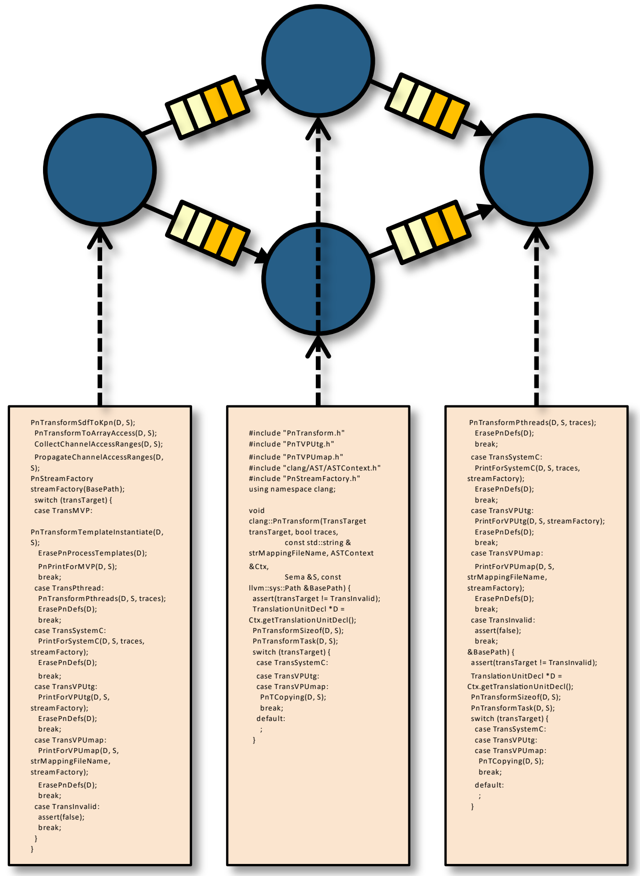
- System dynamics (e.g., IoT)
- Ubiquity of machine learning workloads
- Complexity of emerging technologies

CHAIR FOR
COMPILER
CONSTRUCTION

Dataflow and Hybrid DSE

Dataflow programming

- Graph representation of applications
  - Implicit repetitive execution of tasks
  - Good model for streaming applications
  - Good match for signal processing & multi-media
- The why
  - Explicit parallelism
  - Often: Determinism
  - Better analyzability (scheduling, mapping, optimization)



```
PnTransformSdfToKpn(D, S);
  PnTransformToArrayAccess(D, S);
  CollectChannelAccessRanges(D, S);
  PropagateChannelAccessRanges(D,
S);
PnStreamFactory
streamFactory(BasePath);
  switch (transTarget) {
  case TransMVP:

PnTransformTemplateInstantiate(D,
S);
    ErasePnProcessTemplates(D);
    PnPrintForMVP(D, S);
    break;
  case TransPthread:
    PnTransformPthreads(D, S, traces);
    ErasePnDefs(D);
    break;
  case TransSystemC:
    PrintForSystemC(D, S, traces,
streamFactory);
    ErasePnDefs(D);
    break;
  case TransVPUtg:
    PrintForVPUtg(D, S,
streamFactory);
    ErasePnDefs(D);
    break;
  case TransVPUmap:
    PrintForVPUmap(D, S,
strMappingFileName,
streamFactory);
    ErasePnDefs(D);
    break;
  case TransInvalid:
    assert(false);
    break;
  }
}
```

```
#include "PnTransform.h"
#include "PnTVPUtg.h"
#include "PnTVPUmap.h"
#include "clang/AST/ASTContext.h"
#include "PnStreamFactory.h"
using namespace clang;

void
clang::PnTransform(TransTarget
transTarget, bool traces,
          const std::string &
strMappingFileName, ASTContext
&Ctx,
          Sema &S, const
llvm::sys::Path &BasePath) {
  assert(transTarget != TransInvalid);
  TranslationUnitDecl *D =
Ctx.getTranslationUnitDecl();
  PnTransformSizeof(D, S);
  PnTransformTask(D, S);
  switch (transTarget) {
    case TransSystemC:
    case TransVPUtg:
    case TransVPUmap:
      PnTCopying(D, S);
      break;
    default:
      ;
  }
```

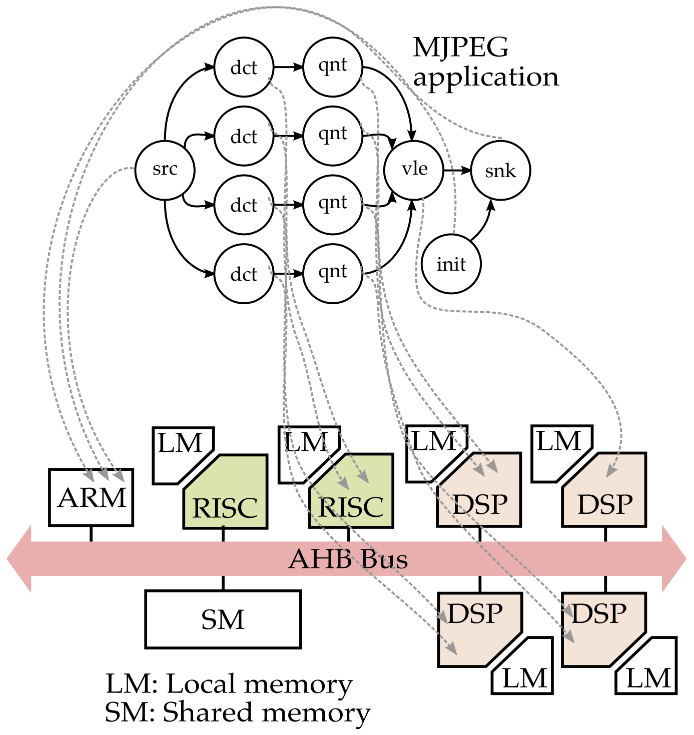
```
 PnTransformPthreads(D, S, traces);
    ErasePnDefs(D);
    break;
  case TransSystemC:
    PrintForSystemC(D, S, traces,
streamFactory);
    ErasePnDefs(D);
    break;
  case TransVPUtg:
    PrintForVPUtg(D, S, streamFactory);
    ErasePnDefs(D);
    break;
  case TransVPUmap:
    PrintForVPUmap(D, S,
strMappingFileName,
streamFactory);
    ErasePnDefs(D);
    break;
  case TransInvalid:
    assert(false);
    break;
&BasePath) {
  assert(transTarget != TransInvalid);
  TranslationUnitDecl *D =
Ctx.getTranslationUnitDecl();
  PnTransformSizeof(D, S);
  PnTransformTask(D, S);
  switch (transTarget) {
    case TransSystemC:
    case TransVPUtg:
    case TransVPUmap:
      PnTCopying(D, S);
      break;
    default:
      ;
  }
```

Dataflow compilation

- Plenty of research on
  - Language, compiler and mapping algorithms
  - Hardware modeling, performance estimation
  - Runtime systems
  - Code generation for heterogeneous multicores

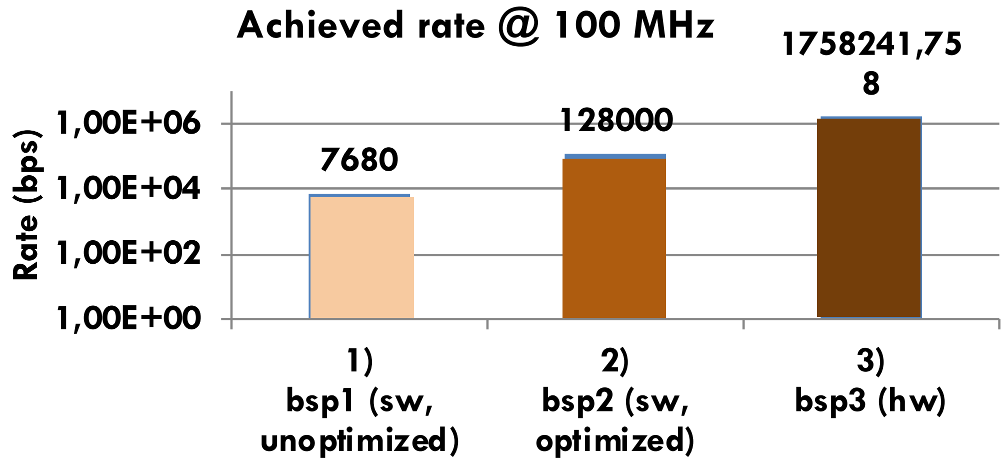
[Springer14]

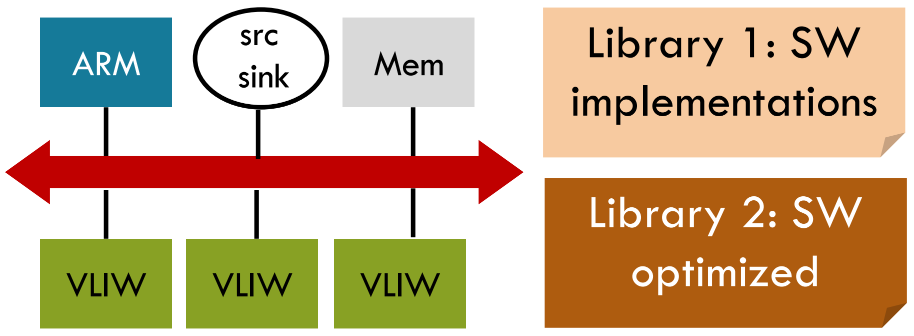
SILEXICA

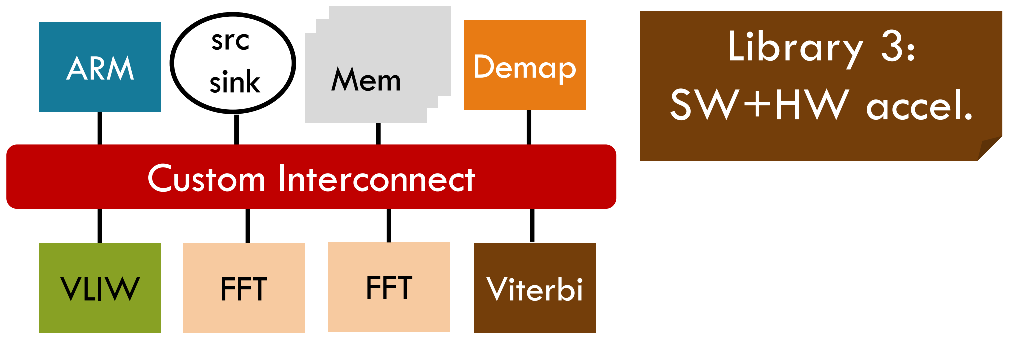


Example: HW acceleration and SW-defined radio

- Application: MIMO OFDM receiver
- Hardware
  - Platform 1: Baseline software
  - Platform 2: Optimized software
  - Platform 3: Optimized SW + HW



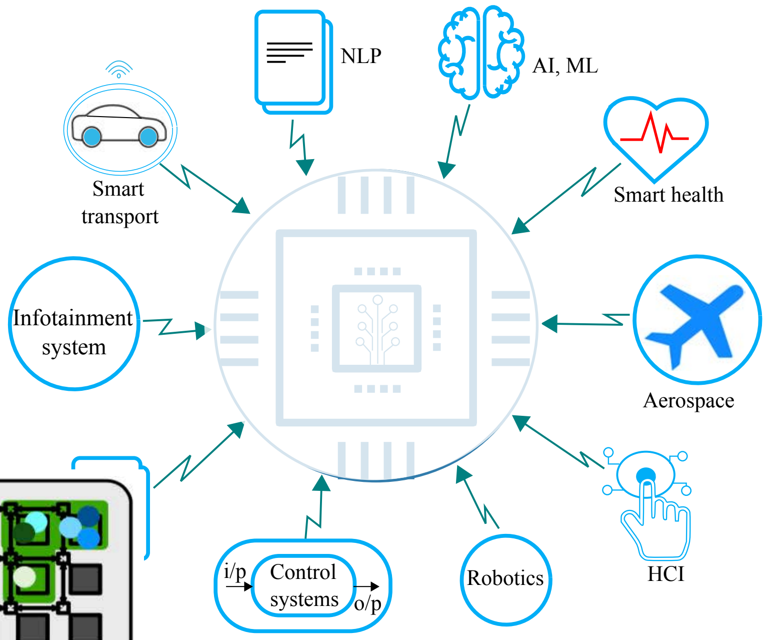


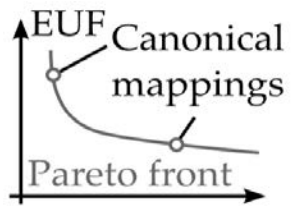


[SDR'10, ALOG'11]

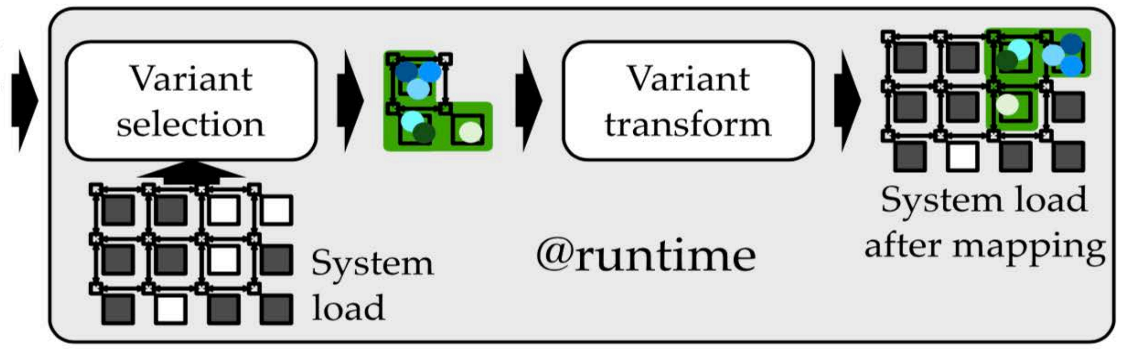
System dynamics

- Applications not so static anymore!
- Hybrid DSE: a compile and run-time approach
  - Enable adaptivity: malleable, multi-variant
  - Run-time predictability, robustness & isolation



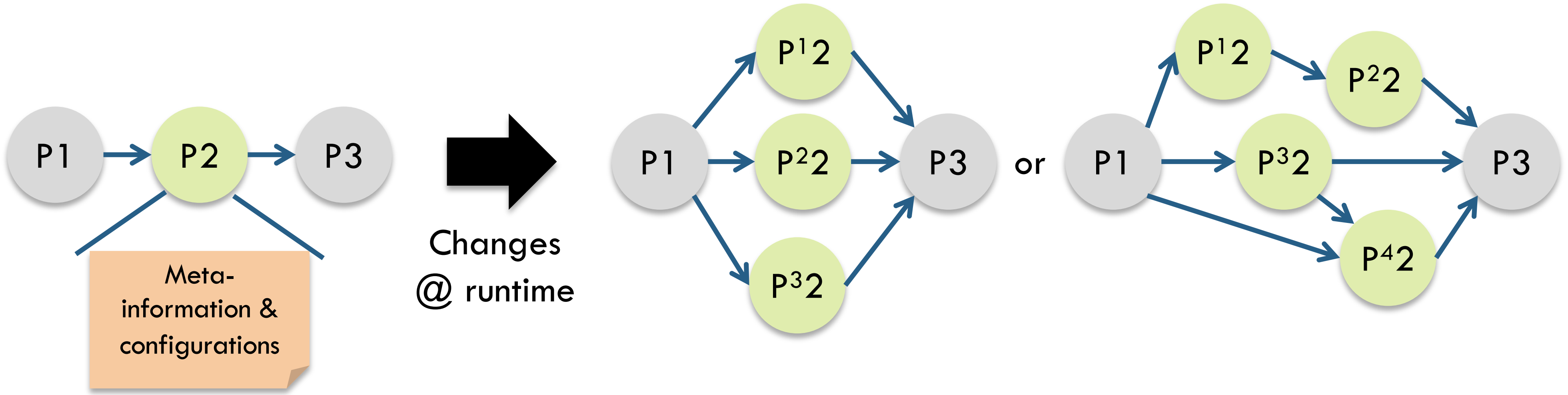


EUF Pareto front @ compile-time



Data-level parallelism: Scalable and adaptive

- Change parallelism from the application specification
- Static code analysis to identify possible transformations (or via annotations)
- Implementation in FIFO library (semantics preserving)



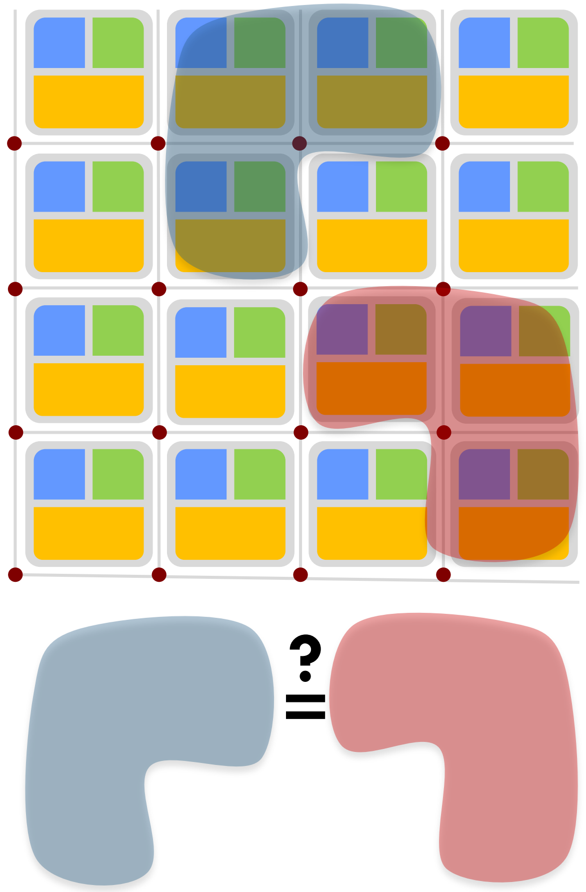
[PARMA-DITAM'18]

Exploiting symmetries

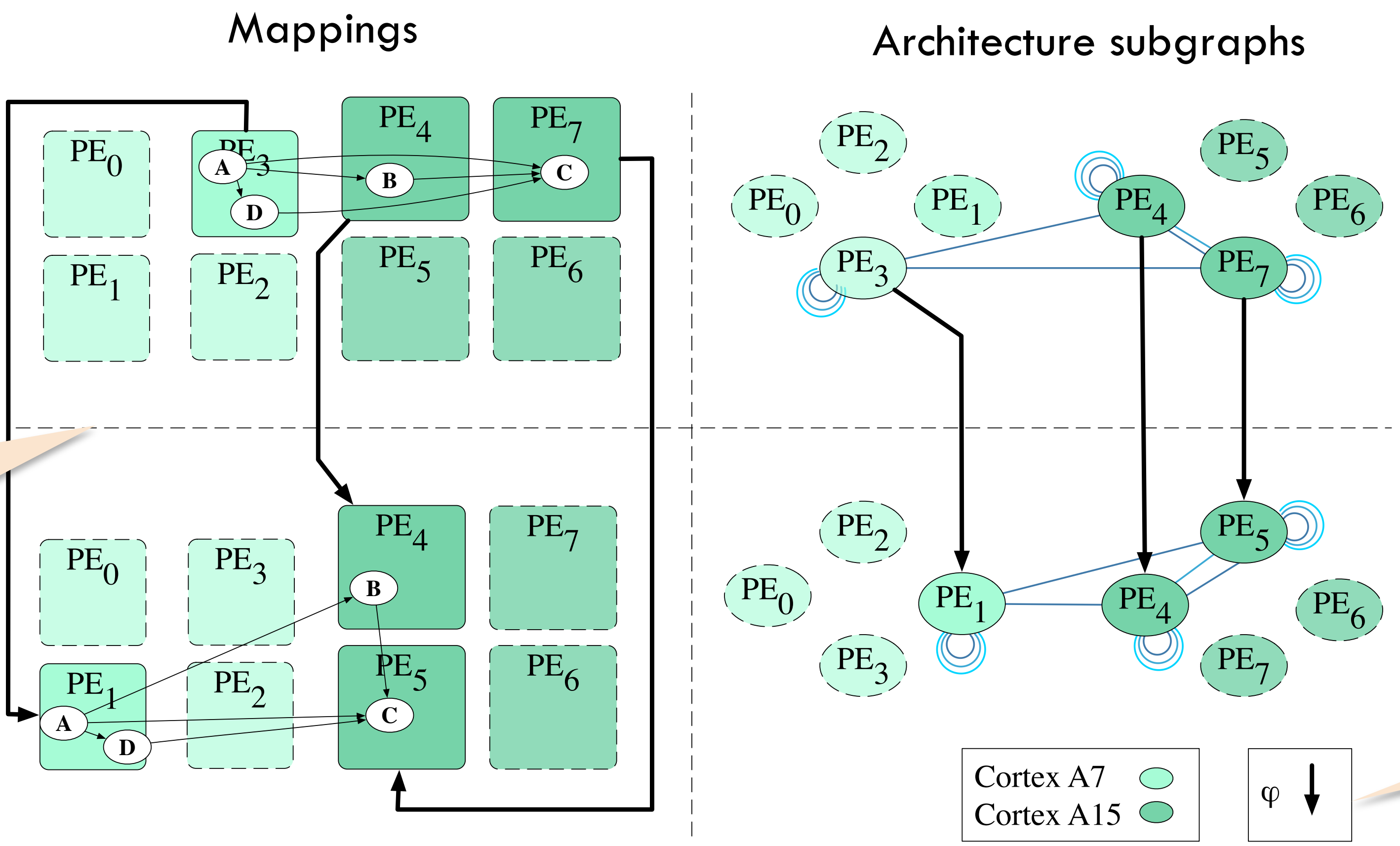
- Intuition
  - SW: Some tasks/processes/actors may do the same
  - HW: Symmetric latencies (CoreX ←→ CoreY)
  - Symmetry: Allows **transformations** w/o changing the **outcome**

→ No need to analyze all possible mappings (prune search space)

→ Work on formalization via inverse semi-groups and efficient algorithms



Symmetries in Odroid: Example

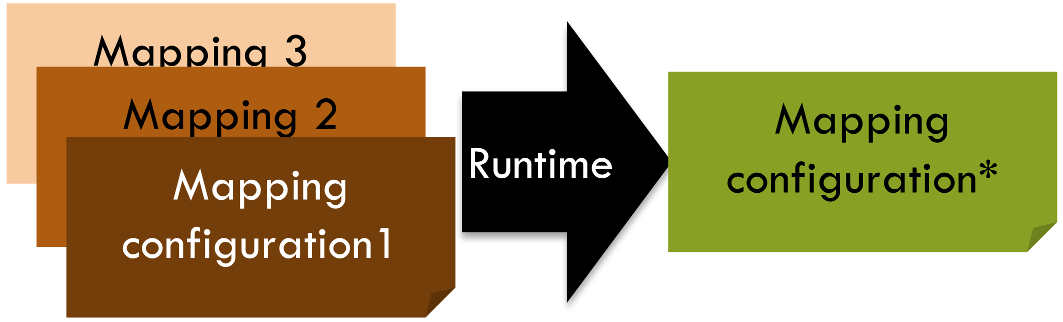


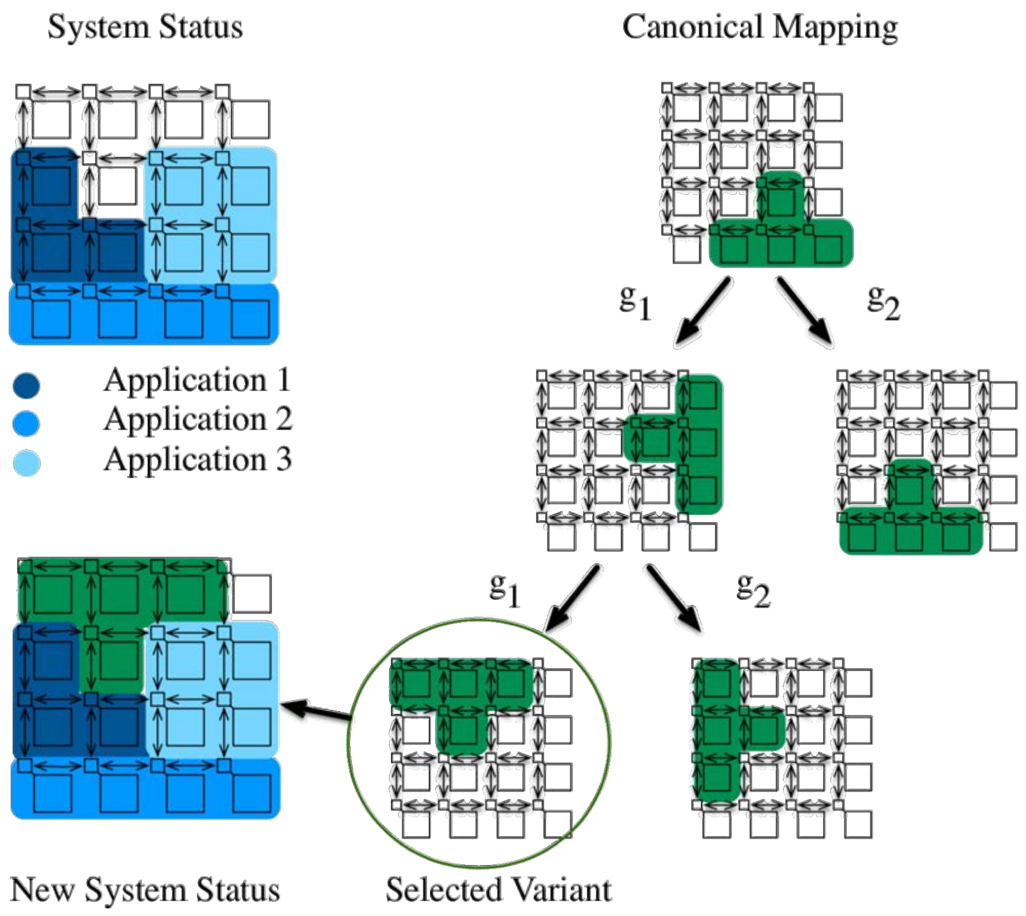
Equivalent mappings

Graph isomorphism

[IESS'15, ACM TACO'17]

Flexible mappings: Generalized Tetris



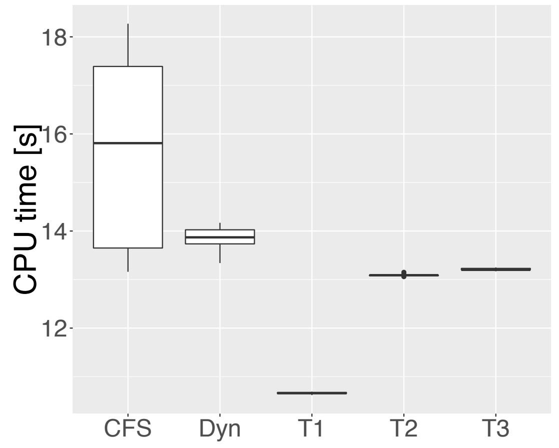


- Given multiple **canonical** configs by compiler, select one at run-time
- Exploit mapping **equivalences** and **similarities**

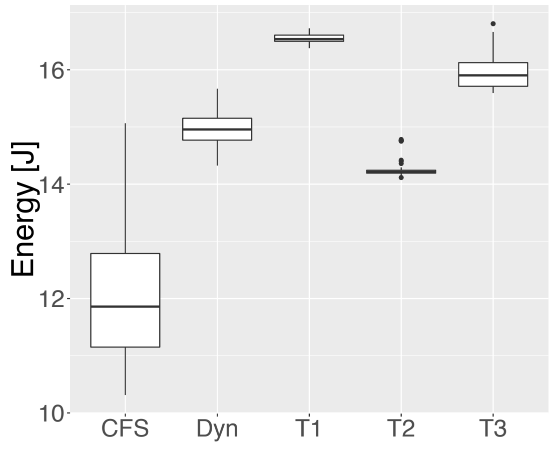
[SCOPES'17b]

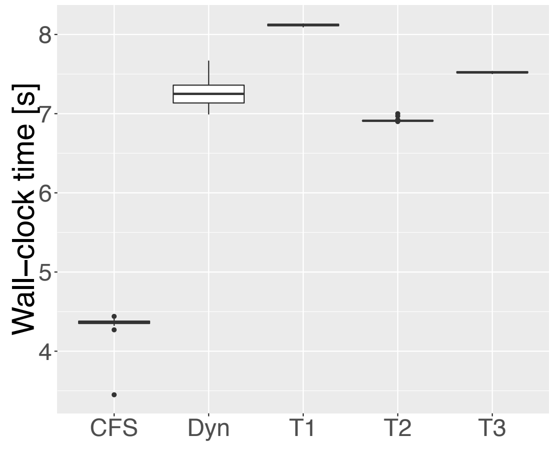
Flexible mappings: Run-time analysis

- Linux kernel: symmetry-aware
- Target: Odroid XU4 (big.LITTLE)
- Multi-application scenarios: audio filter (AF) and MIMO
  - 1x AF
  - 4 x AF
  - 2 x AF + 2 x MIMO
- 3 mappings to two processors
  - T1: Best CPU time
  - T2: Best wall-clock time
  - T3: GBM heuristic [Castrill12]



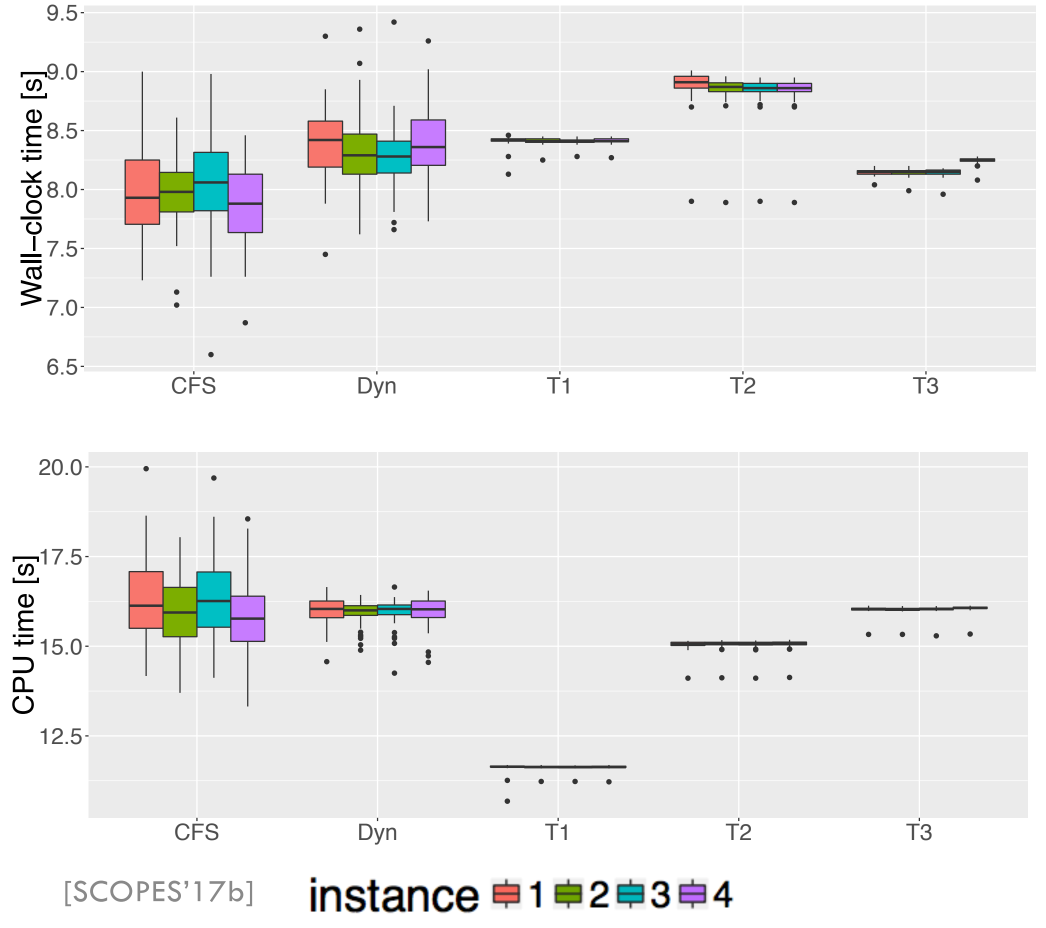
Single AF

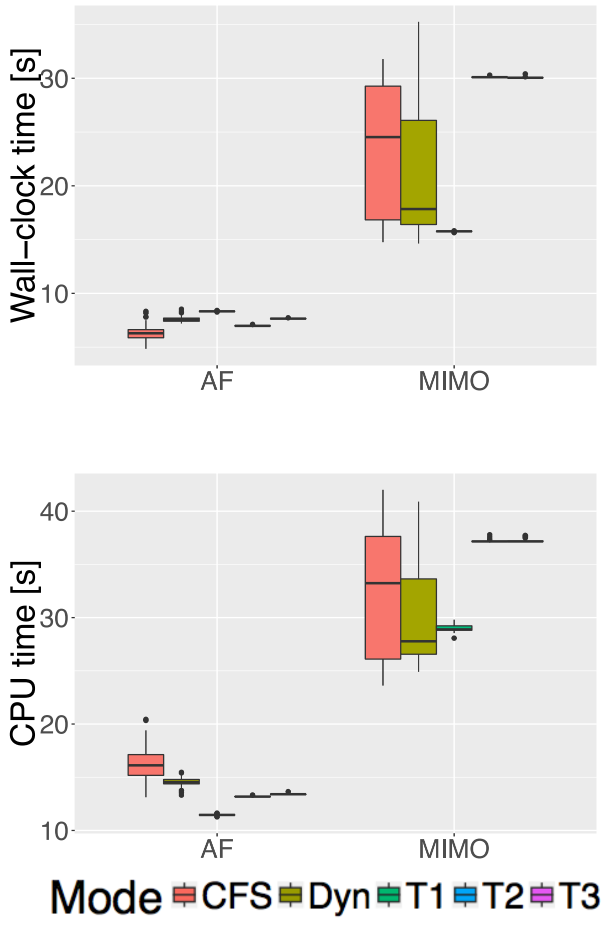




[SCOPES'17b]

Flexible mappings: Multi-application results (1)





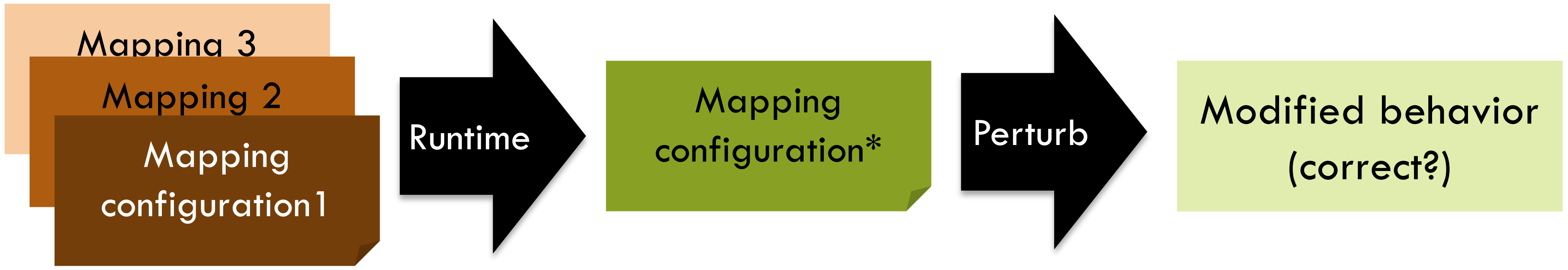
More predictable performance

Comparable performance to dynamic mapping

[SCOPES'17b]

Robustness

- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions

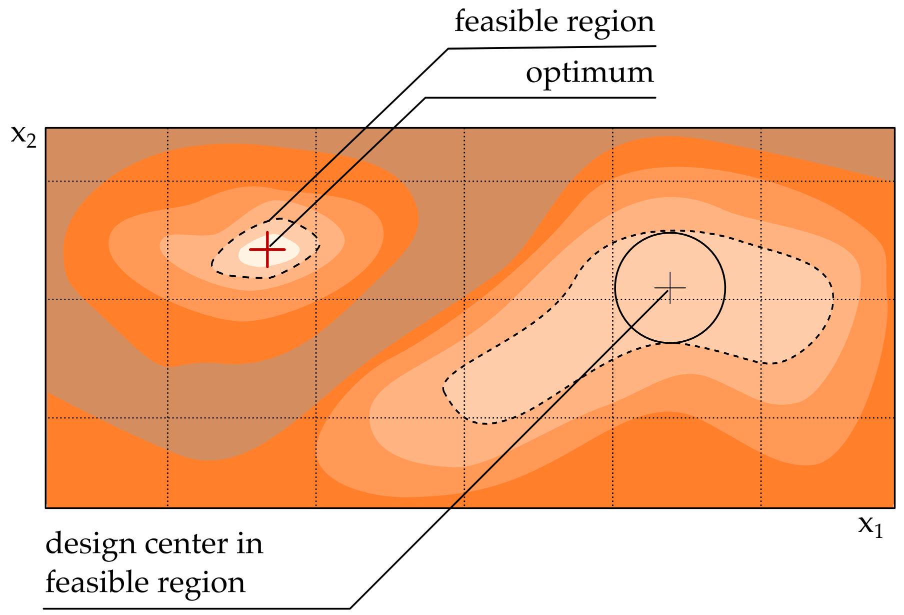


→ Can we reason about robustness of mapping to external factors?

Design centering

- Design centering: Find a mapping that can better tolerate **variations** while staying feasible
- Studied field, in e.g., biology, circuit design or manufacturing systems.

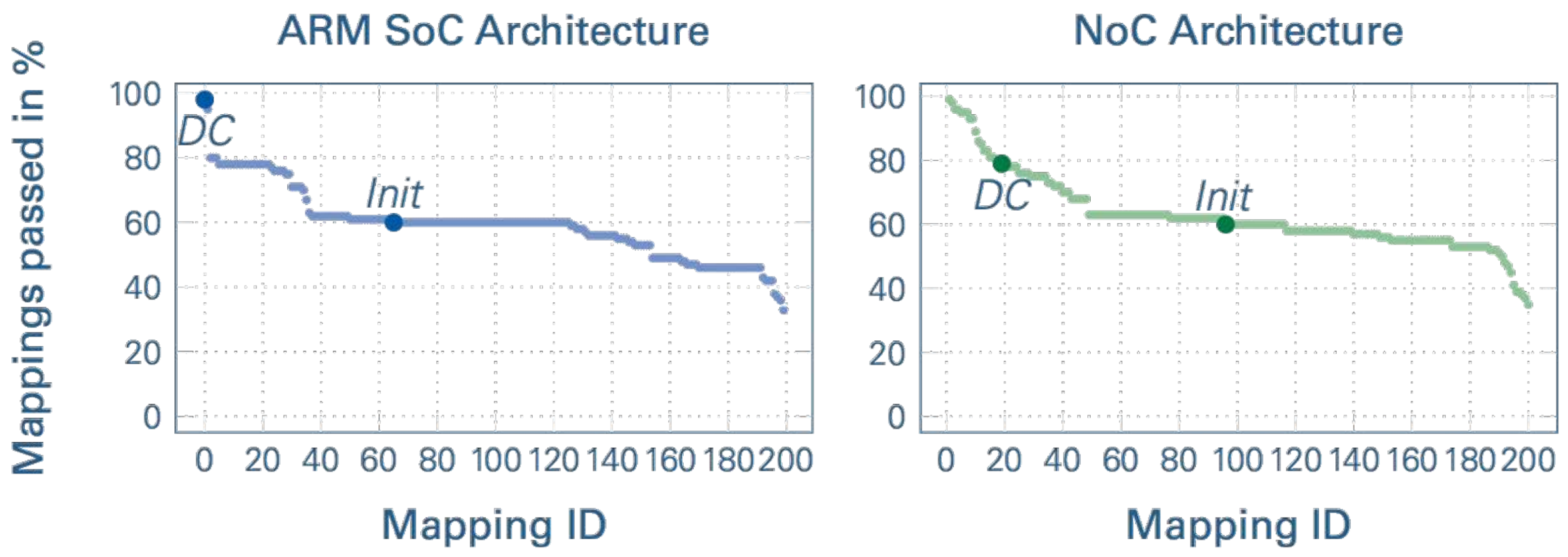
- Currently
  - Using a bio-inspired algorithm
  - Robust against OS changes to the mapping



[SCOPES'17a]

Evaluation

- Analyze how robust the center really is
  - Perturbate mappings and check how often the constraints are missed
  - Signal processing applications on clustered ARM manycore and NoC manycore (16)



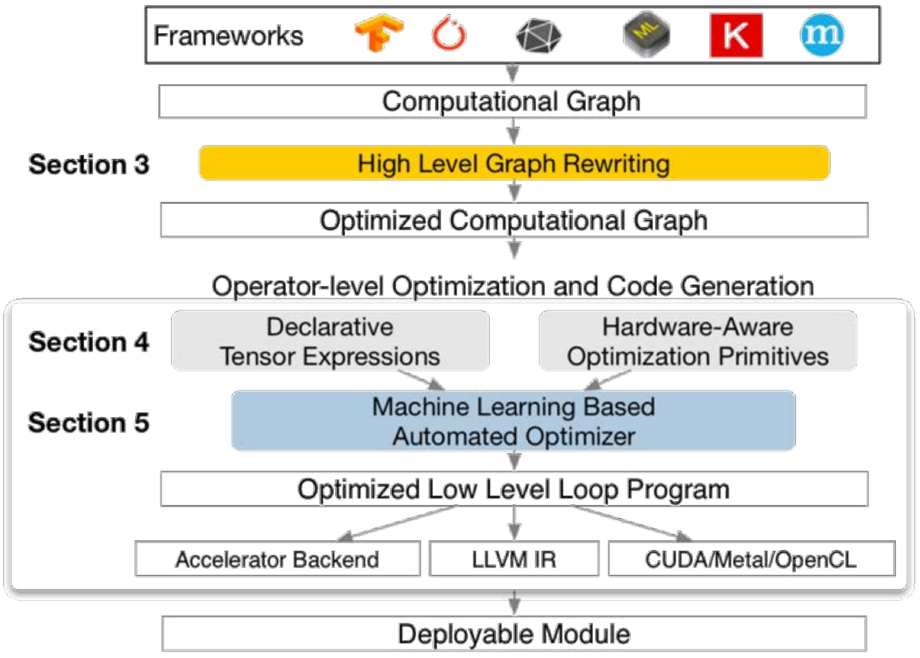
MIMO-OFDM

[SCOPES'17a]

Machine Learning & emerging technologies

ML revolution: Frameworks and architectures

- Many existing frameworks, e.g., TVM, Tensor Comprehensions, TensorFlow, …
- Lots of traction in hardware architectures: TPU, V100, …
- **Lot's of resources for training, less on inference on edge-devices!**


Frameworks
Computational Graph
Section 3
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Section 4
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Section 5
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Example flow: TVM [Chen, OSDI'18]

Domain-specific abstractions

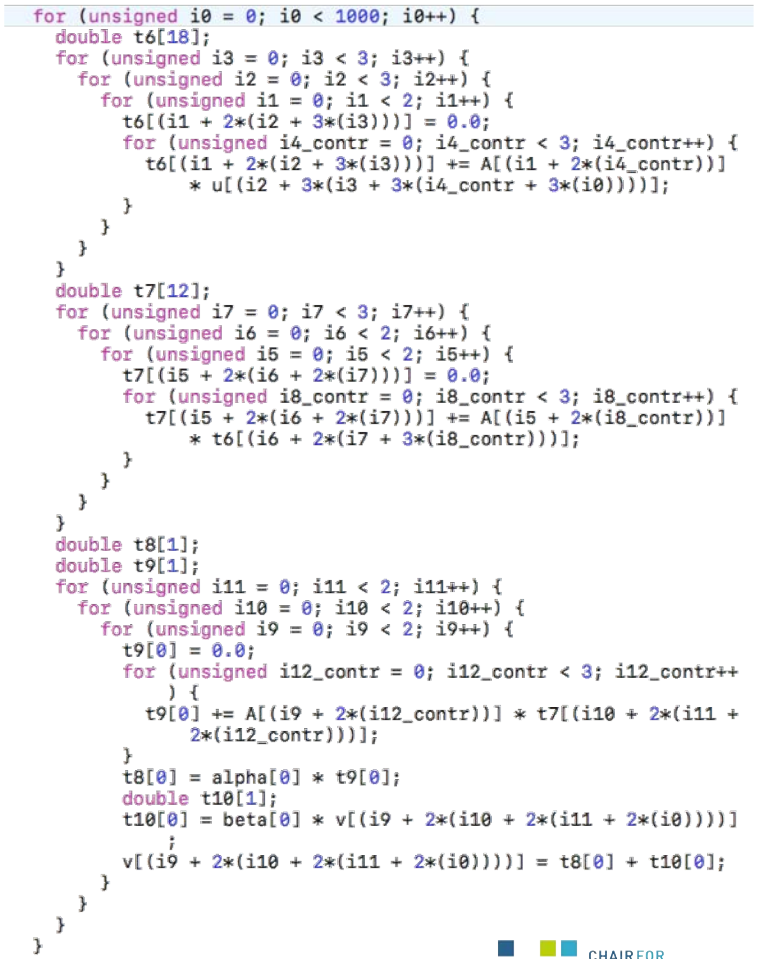
- ❑ Commonality: Tensor expression languages
- ❑ Increase programmer's productivity
- ❑ From compiler perspective: No abstraction toll
  - ❑ Easier access to information
  - ❑ Larger score for optimization

```
var input A    : matrix          &
var input u    : tensorIN        &

v = (A # A # A # u .
    [[5 8] [3 7] [1 6]])
```

[RWDSL'18] $\mathbf{v}_e = (\mathbf{A} \otimes \mathbf{A} \otimes \mathbf{A})\, \mathbf{u}_e$

VS



```
for (unsigned i0 = 0; i0 < 1000; i0++) {
  double t6[18];
  for (unsigned i3 = 0; i3 < 3; i3++) {
    for (unsigned i2 = 0; i2 < 3; i2++) {
      for (unsigned i1 = 0; i1 < 2; i1++) {
        t6[(i1 + 2*(i2 + 3*(i3)))] = 0.0;
        for (unsigned i4_contr = 0; i4_contr < 3; i4_contr++) {
          t6[(i1 + 2*(i2 + 3*(i3)))] += A[(i1 + 2*(i4_contr))]
              * u[(i2 + 3*(i3 + 3*(i4_contr + 3*(i0))))];
        }
      }
    }
  }
  double t7[12];
  for (unsigned i7 = 0; i7 < 3; i7++) {
    for (unsigned i6 = 0; i6 < 2; i6++) {
      for (unsigned i5 = 0; i5 < 2; i5++) {
        t7[(i5 + 2*(i6 + 2*(i7)))] = 0.0;
        for (unsigned i8_contr = 0; i8_contr < 3; i8_contr++) {
          t7[(i5 + 2*(i6 + 2*(i7)))] += A[(i5 + 2*(i8_contr))]
              * t6[(i6 + 2*(i7 + 3*(i8_contr)))];
        }
      }
    }
  }
  double t8[1];
  double t9[1];
  for (unsigned i11 = 0; i11 < 2; i11++) {
    for (unsigned i10 = 0; i10 < 2; i10++) {
      for (unsigned i9 = 0; i9 < 2; i9++) {
        t9[0] = 0.0;
        for (unsigned i12_contr = 0; i12_contr < 3; i12_contr++
            ) {
          t9[0] += A[(i9 + 2*(i12_contr))] * t7[(i10 + 2*(i11 +
              2*(i12_contr)))];
        }
        t8[0] = alpha[0] * t9[0];
        double t10[1];
        t10[0] = beta[0] * v[(i9 + 2*(i10 + 2*(i11 + 2*(i0))))]
            ;
        v[(i9 + 2*(i10 + 2*(i11 + 2*(i0))))] = t8[0] + t10[0];
      }
    }
  }
}
```

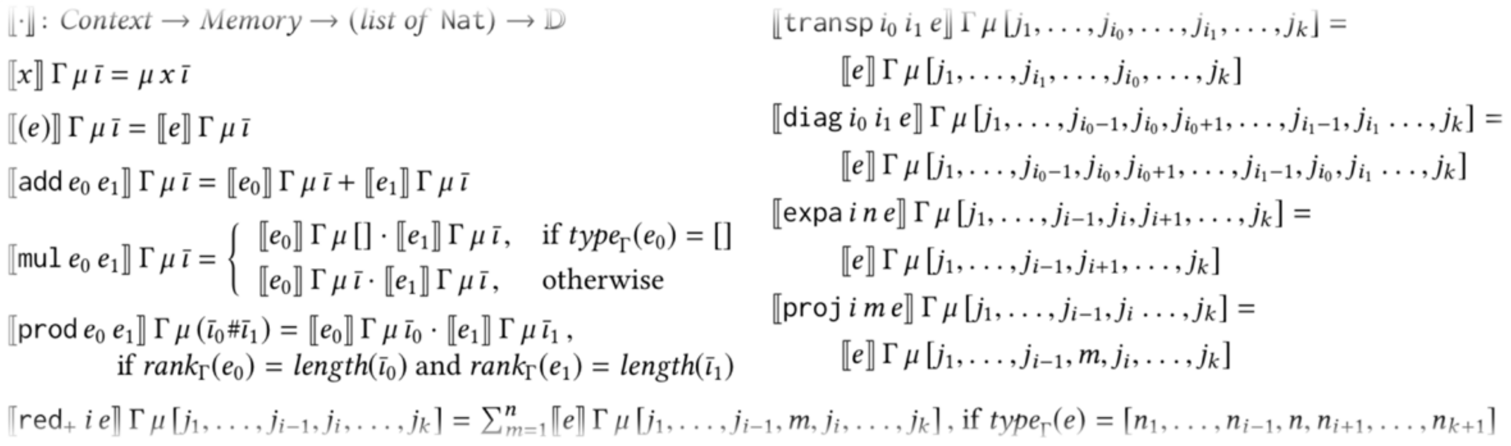
Correct by construction

- Especially important in embedded systems
  - Correct by design
  - No abstraction leaks

```
A = placeholder((m,h), name='A')
B = placeholder((n,h), name='B')
k = reduce_axis((0,h), name='k')
C = compute((m,n), lambda i, j:
      sum(A[k, i] * B[k, j], axis=k))
```

$$C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj}$$

- Current efforts in formal semantics for safe code generation



$$\llbracket \cdot \rrbracket : \mathit{Context} \to \mathit{Memory} \to (\mathit{list\ of}\ \mathsf{Nat}) \to \mathbb{D}$$

$$\llbracket x \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \mu\, x\, \bar{\imath}$$

$$\llbracket (e) \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \llbracket e \rrbracket\, \Gamma\, \mu\, \bar{\imath}$$

$$\llbracket \mathsf{add}\ e_0\ e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath} + \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}$$

$$\llbracket \mathsf{mul}\ e_0\ e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \begin{cases} \llbracket e_0 \rrbracket\, \Gamma\, \mu\, [\,] \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{if } \mathit{type}_\Gamma(e_0) = [\,] \\ \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath} \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{prod}\ e_0\ e_1 \rrbracket\, \Gamma\, \mu\, (\bar{\imath}_0 \# \bar{\imath}_1) = \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_0 \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_1\,, \quad \text{if } \mathit{rank}_\Gamma(e_0) = \mathit{length}(\bar{\imath}_0) \text{ and } \mathit{rank}_\Gamma(e_1) = \mathit{length}(\bar{\imath}_1)$$

$$\llbracket \mathsf{transp}\ i_0\ i_1\ e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0}, \ldots, j_{i_1}, \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_1}, \ldots, j_{i_0}, \ldots, j_k]$$

$$\llbracket \mathsf{diag}\ i_0\ i_1\ e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0-1}, j_{i_0}, j_{i_0+1}, \ldots, j_{i_1-1}, j_{i_1} \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i_0-1}, j_{i_0}, j_{i_0+1}, \ldots, j_{i_1-1}, j_{i_0}, j_{i_1} \ldots, j_k]$$

$$\llbracket \mathsf{expa}\ i\ n\ e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, j_{i+1}, \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_{i+1}, \ldots, j_k]$$

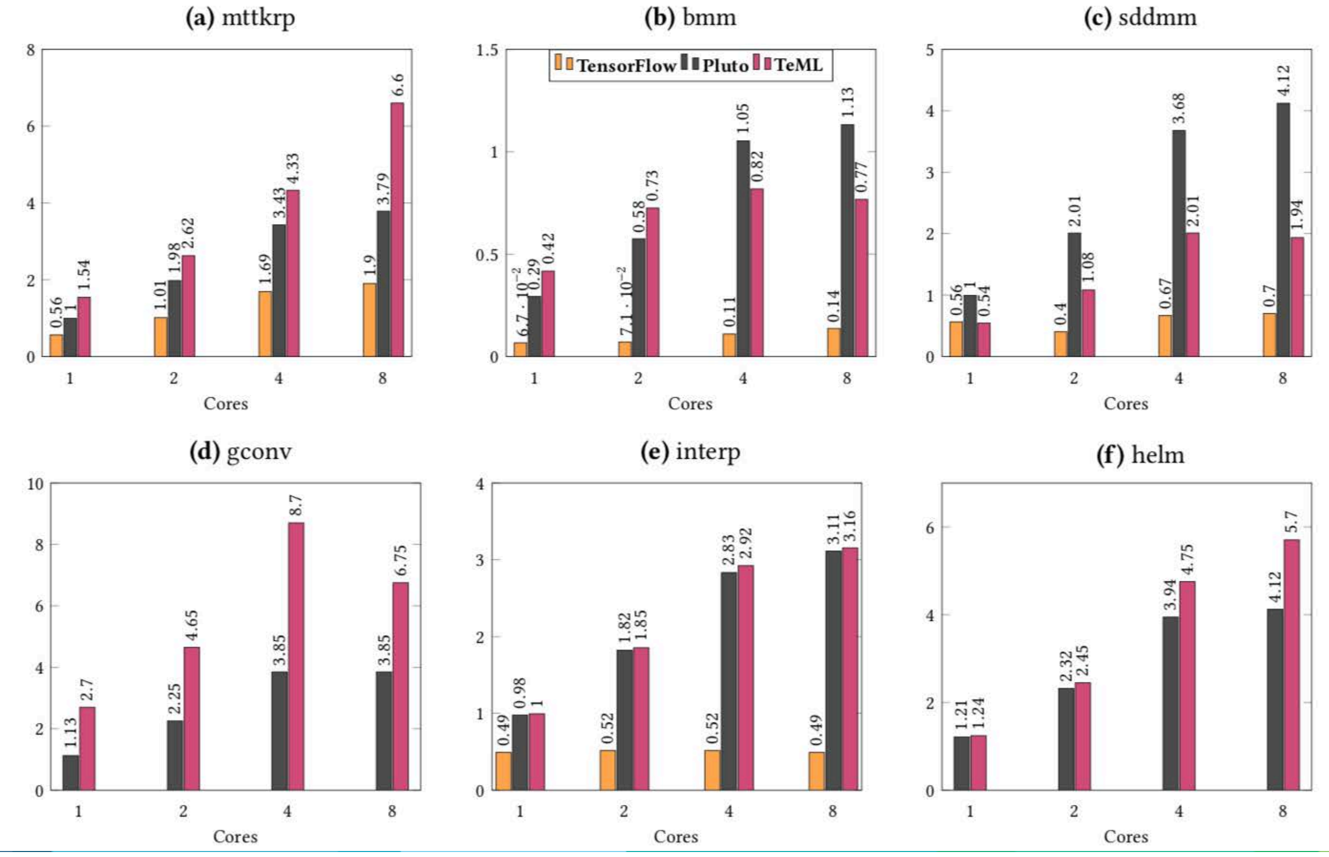
$$\llbracket \mathsf{proj}\ i\ m\ e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k]$$

[Array'19]

$$\llbracket \mathsf{red}_+\ i\ e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, \ldots, j_k] = \textstyle\sum_{m=1}^{n} \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k]\,, \text{ if } \mathit{type}_\Gamma(e) = [n_1, \ldots, n_{i-1}, n, n_{i+1}, \ldots, n_{k+1}]$$

TeML: Results

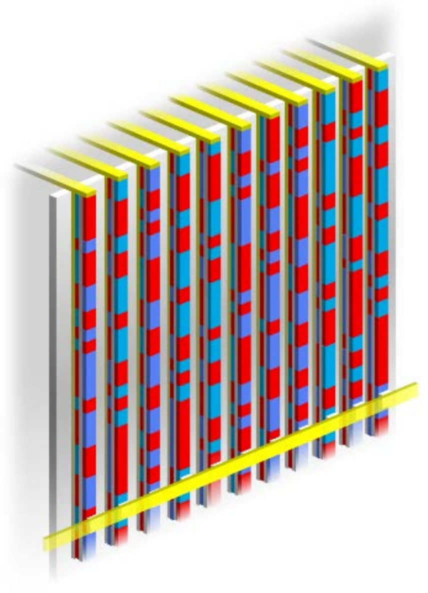
- Extra control allows for new optimization (vs pluto): changing shapes
- General tensor semantics allows covering more benchmarks than TensorFlow



[GPCE'18]

Emerging technologies: Racetrack memories

- Racetrack memories: one of many future alternatives
  - Cf. STT/Re-RAM, hybrid architectures [TVLSI'18, TVLSI'19]
- Predicted extreme density at low latency
  - 3D nano-wires with magnetic domains
  - One port shared for many bits
  - Domains move at high speeds (1000 ms^{-1})
- Sequential: Game changer for current HW/SW stack
  - Memory management
  - Integration with other memory architectures
  - Data layout and allocation


[Parkin-Nature'15]

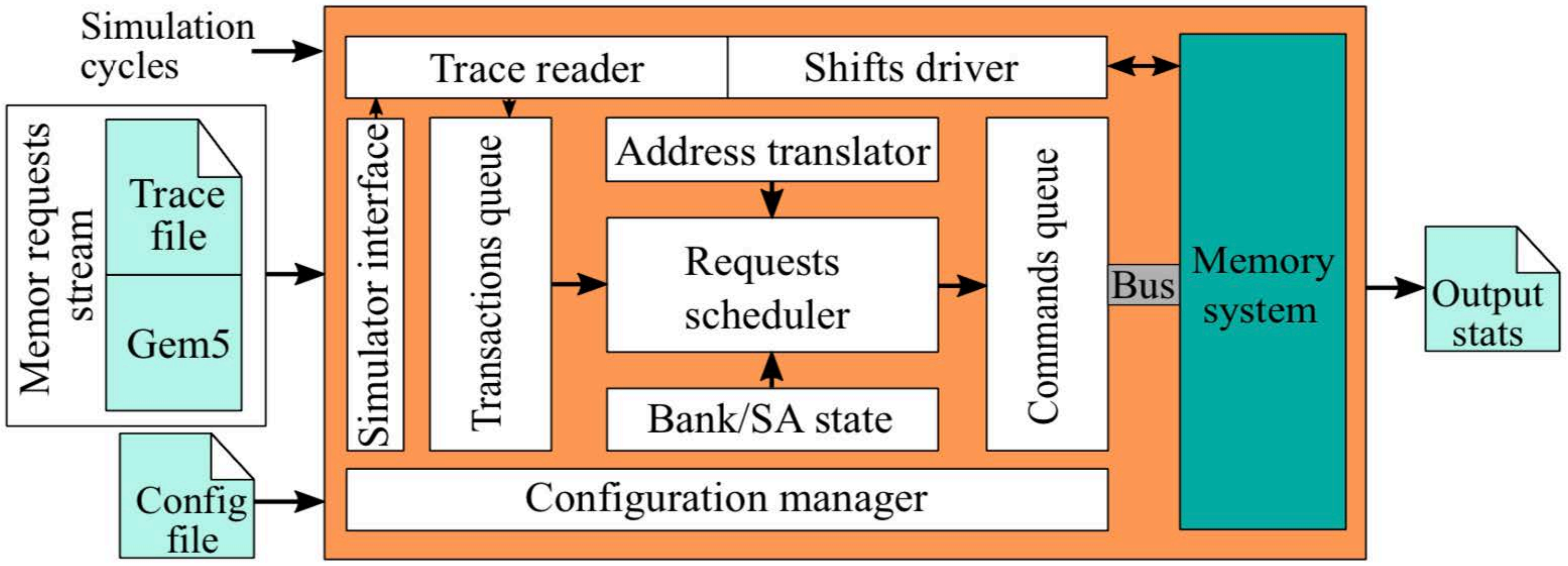
Compiler research on placement

- Compiler pass to reorganize placement of instructions
  - Instruction fetch is naturally sequential!
  - Layout instructions to reduce shift operations [ISLPED'19]

- Compiler pass to reorganize data for higher-level objects
  - Variable allocation [ArXiv'19]
  - Tensor allocation and program transformation [LCTES'19]

Simulating RTMs

- RTSim: Configurable racetrack simulator
  - Allows running software benchmarks
  - Built on stop of other simulator technology: NVMAIN 2, Gem5, SystemC, …

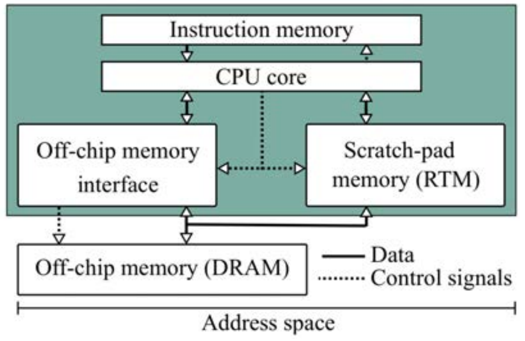


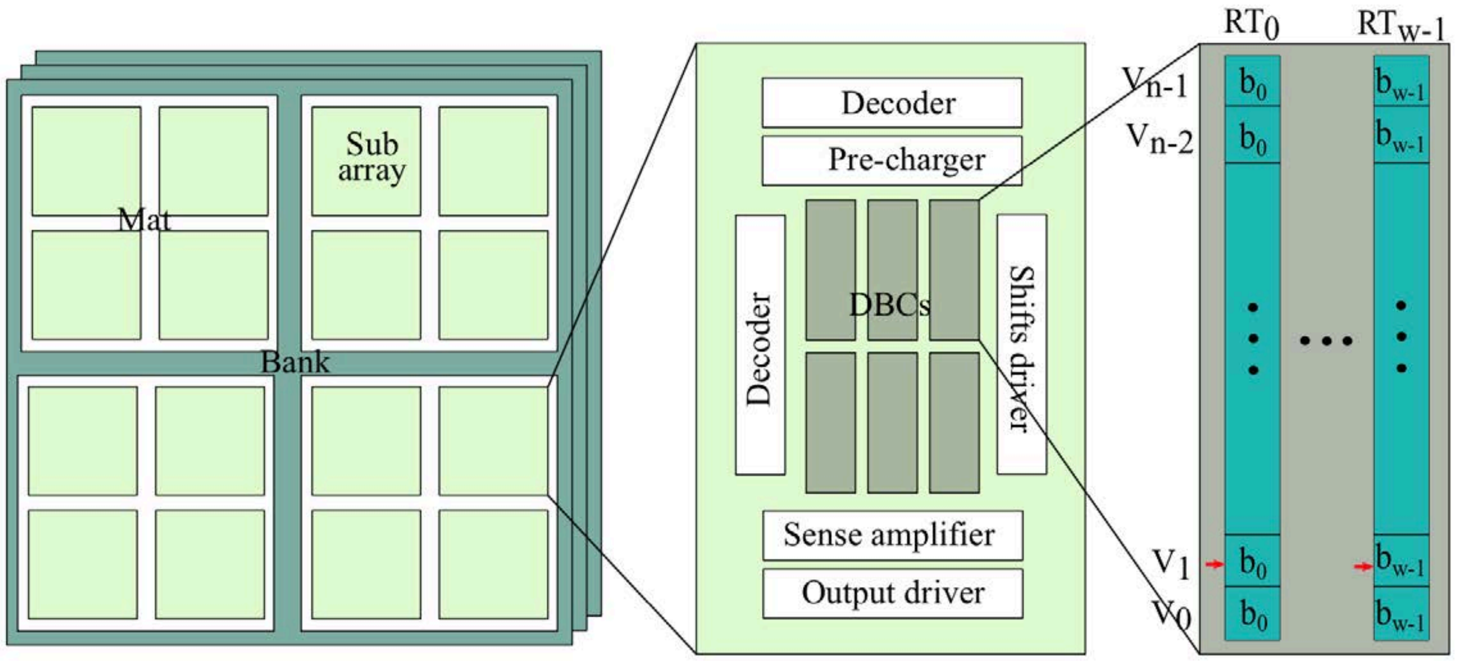
[IEEE CAL'19]

https://github.com/tud-ccc/RTSim

Architecture and data layout optimization

- Architecture – software co-optimization
  - Embedded system for inference: RTM as scratchpad with pre-shifting and other optimizations

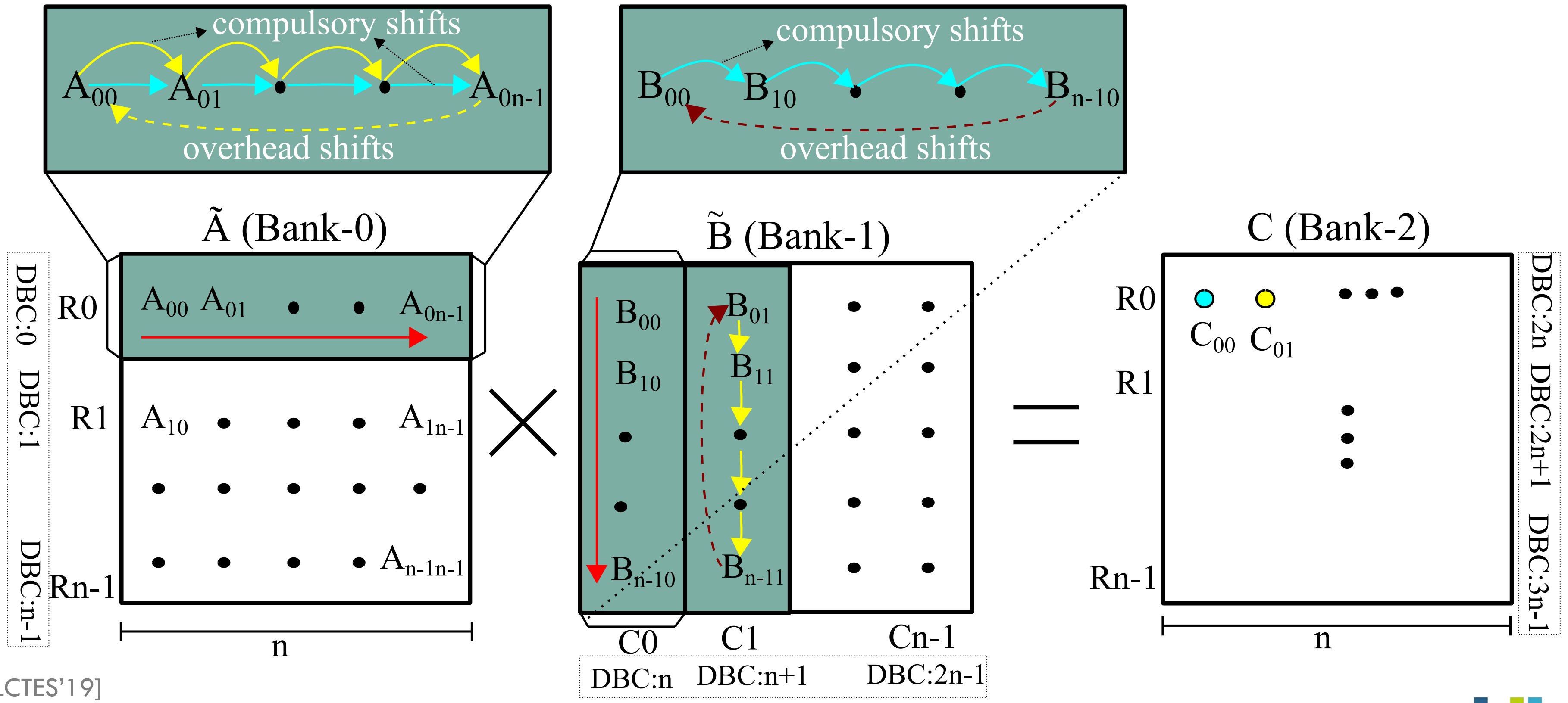




[LCTES'19]

Architecture and data layout optimization

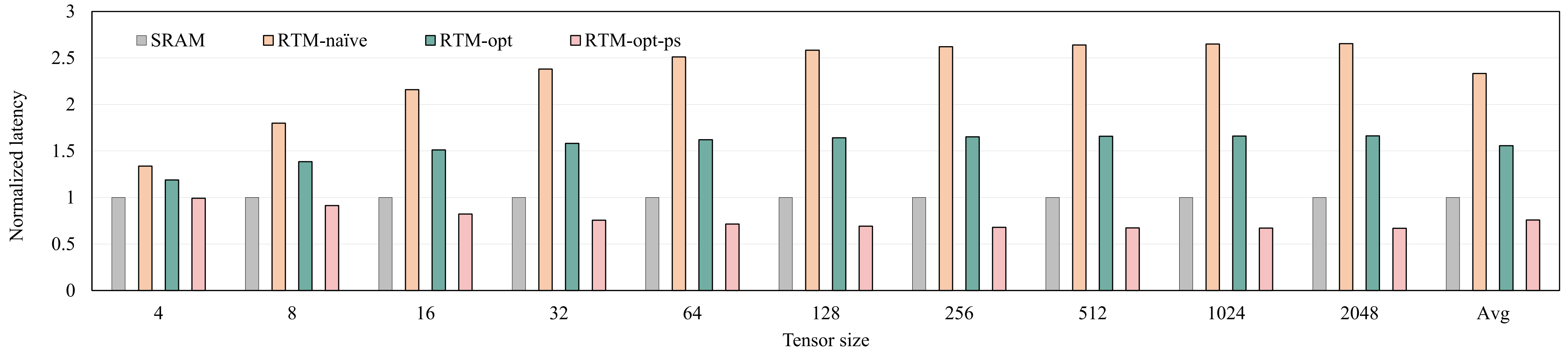
- Data-layout: Reduce the number of shifts



[LCTES'19]

Latency comparison vs SRAM

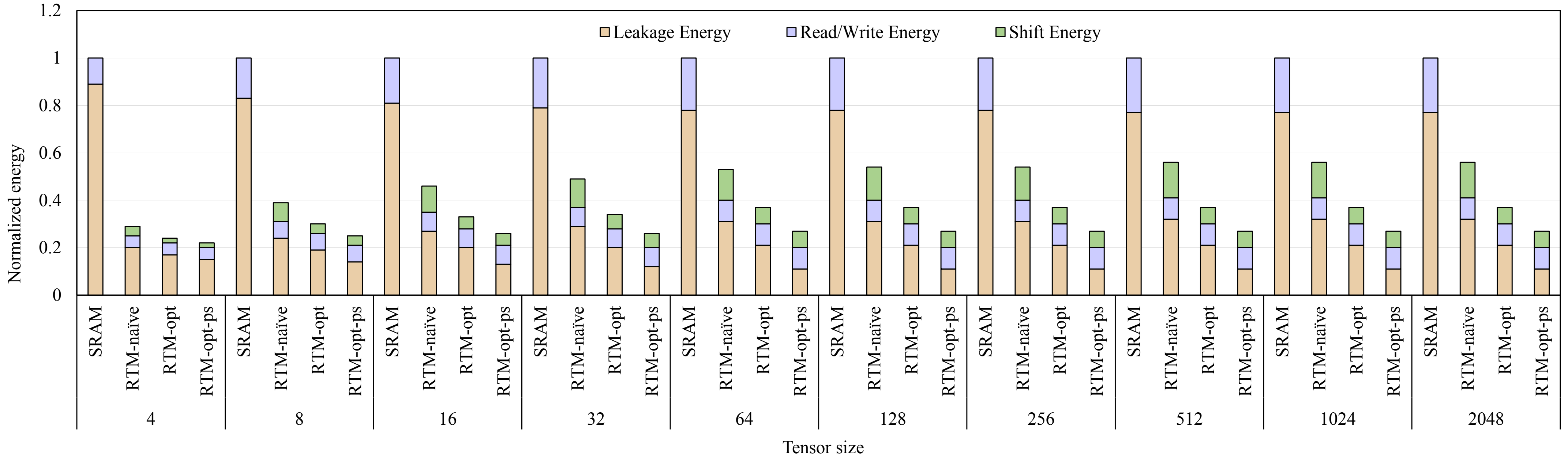
- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)



[LCTES'19]

Energy comparison vs SRAM

- Higher savings due to less leakage power
- 74% average improvement



[LCTES'19]

CHAIR FOR COMPILER CONSTRUCTION

CENTER FOR ADVANCING ELECTRONICS DRESDEN

Discussion

Summary

- System dynamics: More complex in distributed IoT scenarios
  - Hybrid compile-runtime methodologies: Difficult balance, new interfaces
  - Strive at retaining time predictability
- New workloads (e.g., ML) + new techs: Harness domain-specific abstractions
  - More complex decision making (e.g., het. Memory systems)
  - Expression DSLs to ease high-level manipulation and transformation
- **Exciting times for DSE, SW/HW co-design formal languages for emerging platforms (example: RTM-scratchpads)**

References

[Springer'14] J. Castrillon, R. Leupers, "Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap" , Springer, pp. 258, 2014.

[SDR'10] J. Castrillon, S. Schürmans, A. Stulova, W. Sheng, T. Kempf, R. Leupers, G. Ascheid, and H. Meyr, "Component-based waveform development: The nucleus tool flow for efficient and portable SDR," Wireless Innovation Conference and Product Exposition (SDR), 2010

[ALOG'11] J. Castrillon, S. Schürmans, A. Stulova, W. Sheng, T. Kempf, R. Leupers, G. Ascheid, and H. Meyr, "Component-based waveform development: The nucleus tool flow for efficient and portable software defined radio", Analog Integrated Circuits and Signal Processing, vol. 69, no. 2–3, pp. 173–190, 2011

[ACM TACO'17] Goens, A. et al. "Symmetry in Software Synthesis". In: ACM Transactions on Architecture and Code Optimization (TACO) (2017).

[IESS'15] A. Goens and J. Castrillon, "Analysis of Process Traces for Mapping Dynamic KPN Applications to MPSoCs", In IFIP International Embedded Systems Symposium (IESS), 2015, Foz do Iguaçu, Brazil, 2015.

[PARMA-DITAM'18] R. Khasanov, et al, "Implicit Data-Parallelism in Kahn Process Networks: Bridging the MacQueen Gap" , PARMA-DITAM 18, ACM, pp. 20–25, Jan 2018.

[SCOPES'17a] G. Hempel, et al, "Robust Mapping of Process Networks to Many-Core Systems Using Bio-Inspired Design Centering" SCOPES'17.

[SCOPES'17b] Goens, A. et al. "TETRiS: a Multi-Application Run-Time System for Predictable Execution of Static Mappings", SCOPES'17.

[Chen, OSDI'18] Chen, Tianqi, et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning." 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). 2018.

[RWDSL'18] N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL'18.

[Array'19] N.A. Rink, N. A. and Jeronimo Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY), ACM, 2019, 57-68

[GPCE'17] A. Susungi, et al. "Towards Compositional and Generative Tensor Optimizations" GPCE'17, 169-175.

[GPCE'18] A. Susungi, et al. " "Meta-programming for cross-domain tensor optimizations" GPCE'18, 79-92.

[TVLSI'18] F. Hameed, A. A. Khan, J. Castrillon, "Performance and Energy Efficient Design of STT-RAM Last-Level-Cache" , In IEEE Transactions on Very Large Scale Integration Systems (TVLSI), vol. 26, no. 6, pp. 1059–1072, Jun 2018.

[TVLSI'19] F. Hameed, J. Castrillon, "A Novel Hybrid DRAM/STT-RAM Last-Level-Cache Architecture for Performance, Energy and Endurance Enhancement" , In IEEE Transactions on Very Large Scale Integration Systems (TVLSI), pp. 13pp, Jun 2019.

[Parkin-Nature'15] S. Parkin, and See-Hun Yang. "Memory on the racetrack." Nature nanotechnology 10.3 (2015): 195.

[IEEE CAL'19] A. A. Khan, F. Hameed, R. Bläsing, S. Parkin, J.Castrillon, "RTSim: A Cycle-accurate Simulator for Racetrack Memories" In IEEE Computer Architecture Letters, Feb 2019.

[ISPLED'19] J. Multanen, A. A. Khan, P. Jääskeläinen, F. Hameed, J. Castrillon, "SHRIMP: Efficient Instruction Delivery with Domain Wall Memory", International Symposium on Low Power Electronics and Design (ISLPED), ACM, 2019, 10pp

[LCTES'19] Khan, A. A., Rink, N. A., Hameed, F., Castrillon, J. "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", Proceedings of LCTES'19, ACM, pp. 12pp, New York, NY, USA, Jun 2019

[ArXiv'19] Khan, A. A., Hameed, F., Bläsing, R., Parkin, S., Castrillon, J. "ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0", ArXiv arXiv:1903.03597, Mar 2019.