CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Embedded manycore programming: From auto-parallelization to domain specific languages

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

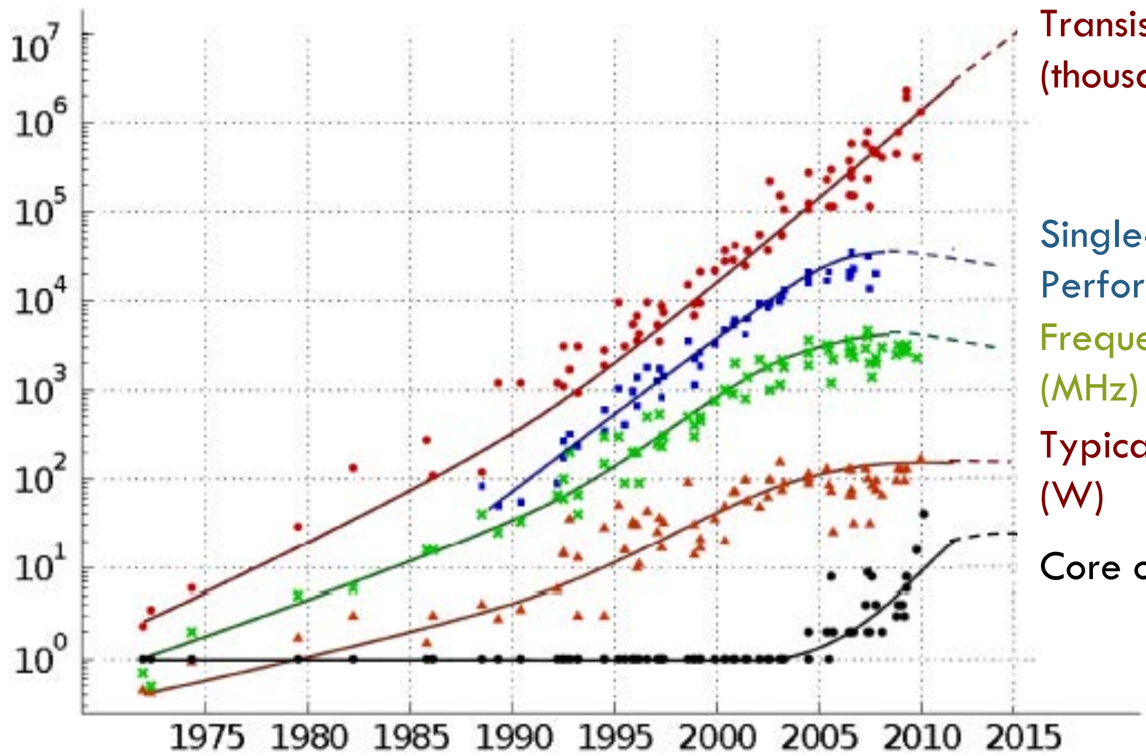
Keynote: International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC'19)

Nanyang Technical University, Singapore. October 2 2019

TECHNISCHE UNIVERSITÄT DRESDEN · DRESDEN concept · DFG · WR WISSENSCHAFTSRAT

Systems on Chip (SoC): Evolution (1)

- Naturally powered by: Moore's law, power density wall, …

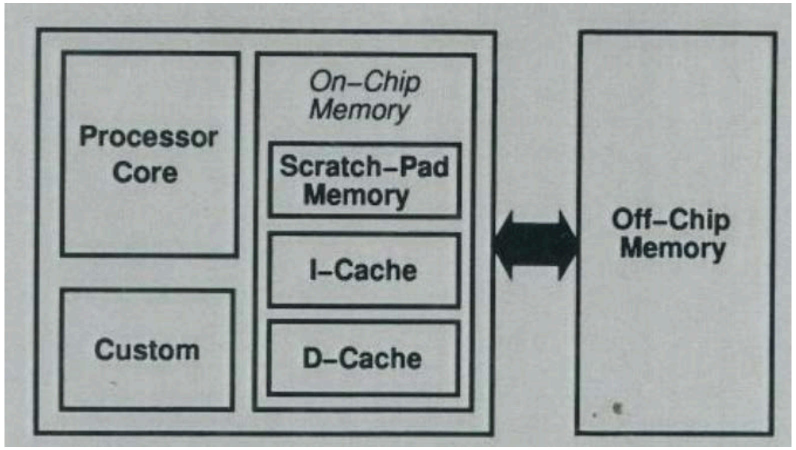


M. Horowitz, F. Labonte, et al. Dotted-line by C. Moore, "Data processing in exascale-class computer systems," The Salishan Conference on High Speed Computing, 2011

Systems on Chip (SoC): Evolution (2)

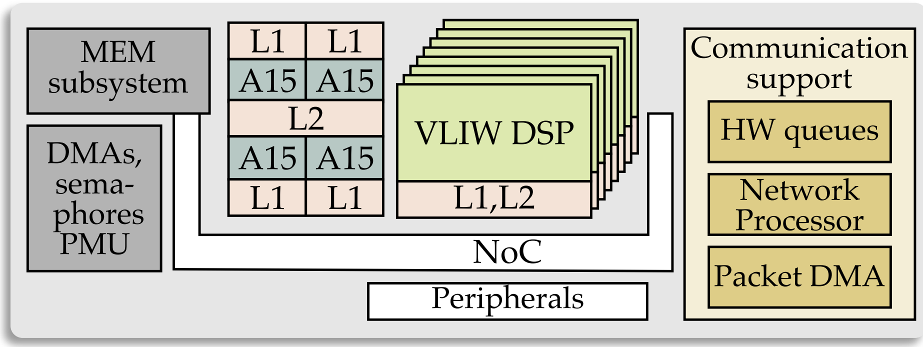
- Incredible evolution over the last decades
- SoCs: Long history of specialization and interaction with environment

1999



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

2019

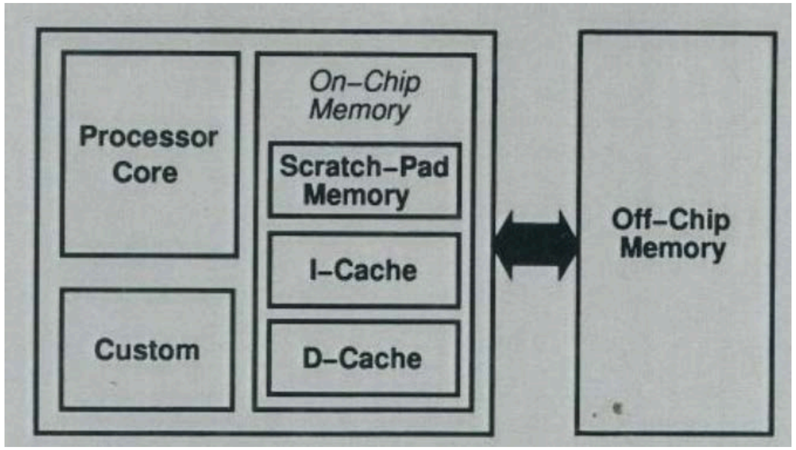


Sketch of TI Keystone II

Systems on Chip (SoC): Evolution (3)

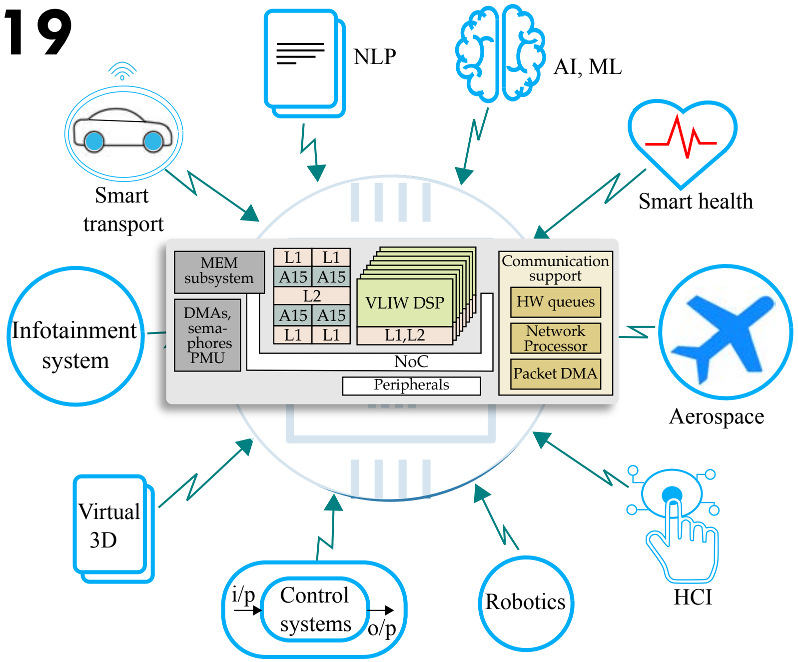
- Incredible evolution over the last decades
- SoCs: Long history of specialization and interaction with environment

1999



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

2019

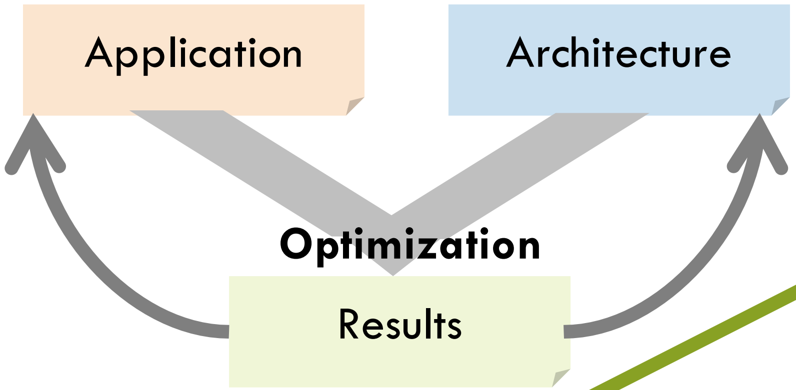


SoC programming: Evolution

- Auto-parallelization
- Formal model-based code/HW generation
- Higher-level programming abstractions

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

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




```
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;
 }
}
```

Keep it sequential

```
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;
 }
}
```

Sounds easy

Sequential code → Compiler → Parallel code (OpenMP, Pthreads,…)

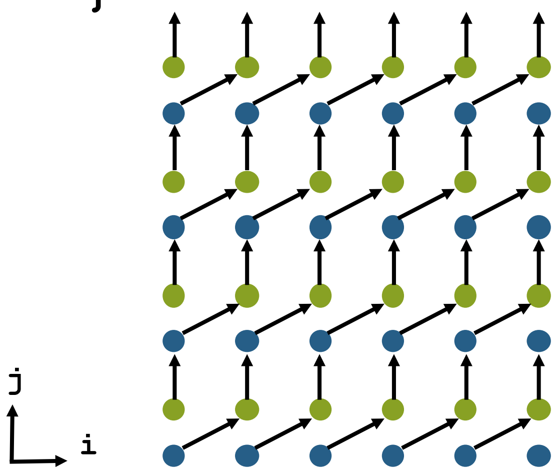
Theorem (Allen/Kennedy): Any reordering transformation that preserves every dependence in a program preserves the meaning of that program

Problems for auto-parallelizing compilers

1) Find all dependencies?

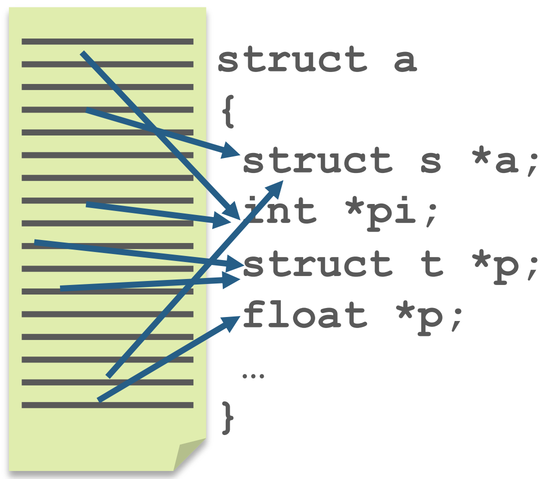
More often than not, impossible!

```
    for (i = 1; i <= 100; i++)
        for (j = 1; j <= 100; j++) {
S1:        X[i][j] = X[i][j] + Y[i-1][j];
S2:        Y[i][j] = Y[i][j] + X[i][j-1];
        }
```



Example: **Polyhedral compilation**

2) Coding style and the illusion of infinite shared memory



Problems for auto-parallelizing compilers (2)

1) Find all dependencies?

2) Coding style and the illusion of infinite shared memory

3) Dependencies can sometimes be violated!
(they are artifacts of style)

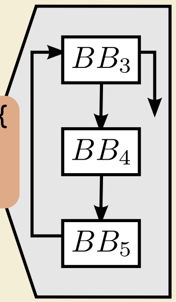
```
while(!queue.empty())
{
  // Dequeue a vertex from queue
  s = queue.front();
  queue.pop_front();

  // Apply function f to s, accumulate values
  result += f(s);

  // Get all adjacent vertices of s.
  // If an adjacent node hasn't been visited,
  // then mark it as visited and enqueue it
  for(i=adj[s].begin(); i!=adj[s].end(); ++i)
  {
    if(!visited[*i])
    {
      visited[*i] = true;
      queue.push_back(*i);
    }
  }
}

return result;
}
```

[Edler18]

Dynamic information for auto-parallelization

- Static analysis: Limited applicability (cf. Polyhedral compilation)
- Dynamic analysis: Extract dependencies based on application tracing
  - Summarize information in form weighted control-data flow graphs

```
int A[10];
int foo() { int A[10]; return A[5];}
int main() {
    int s = 0, i;
    A[8] = foo();
    for (i = 0; i < 2; i++) {
        s += A[i * 4];
    }
    A[2] = foo();
    return 0;
}
```




```
1:s:0:enter:main:ex.bc
2:2
3:s:16:enter:foo:ex.bc
4:1
5:m:5|r||l|foo|1|A|8|20
6:exit:foo:ex.bc
7:m:17|w||g|A|8|32
8:3, 4
9:m:27|r||g|A|8|0
10:5, 3, 4
11:m:27|r||g|A|8|16
12:5, 3, 6
13:s:36:enter:foo:ex.bc
14:1
15:m:5|r||l|foo|2|A|8|20
16:exit:foo:ex.bc
17:m:37|w||g|A|8|8
18:exit:main:ex.bc
```

[DAC08, Springer14]

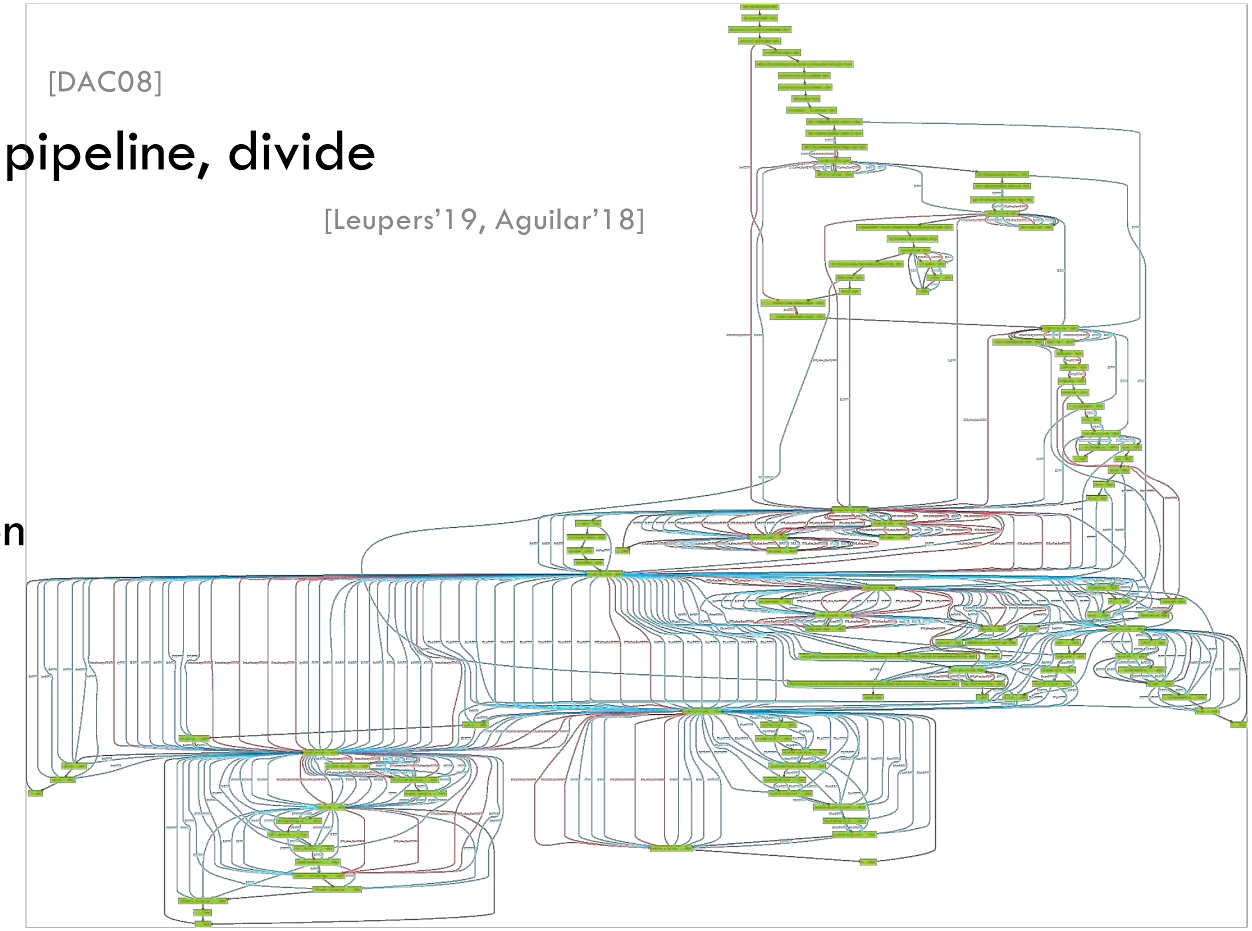
- Related approaches: Profile-driven and ML-based mapping, [Tournavitis09] hierarchical task graphs, and Sambamba [Streit12]

[Cordes10]

Parallelism extraction

- Clustering: Right granularity? [DAC08]
- Identify patterns: data, task, pipeline, divide and conquer, … [Leupers'19, Aguilar'18]

- Requires
  - Cost model of computation
  - Cost model of communication
  - Abstract notion of time for dependencies



Results from a decade of work

Step	Speedup	No. of PEs	Parallel Efficiency
1	3.61x	16	22.58%
2	5.48x	17	32.3%
3	5.48x	16	34.3%
manual	9.43x	19	49.6%

Table 1: Summary of JPEG Encoder Parallelization by MAPS

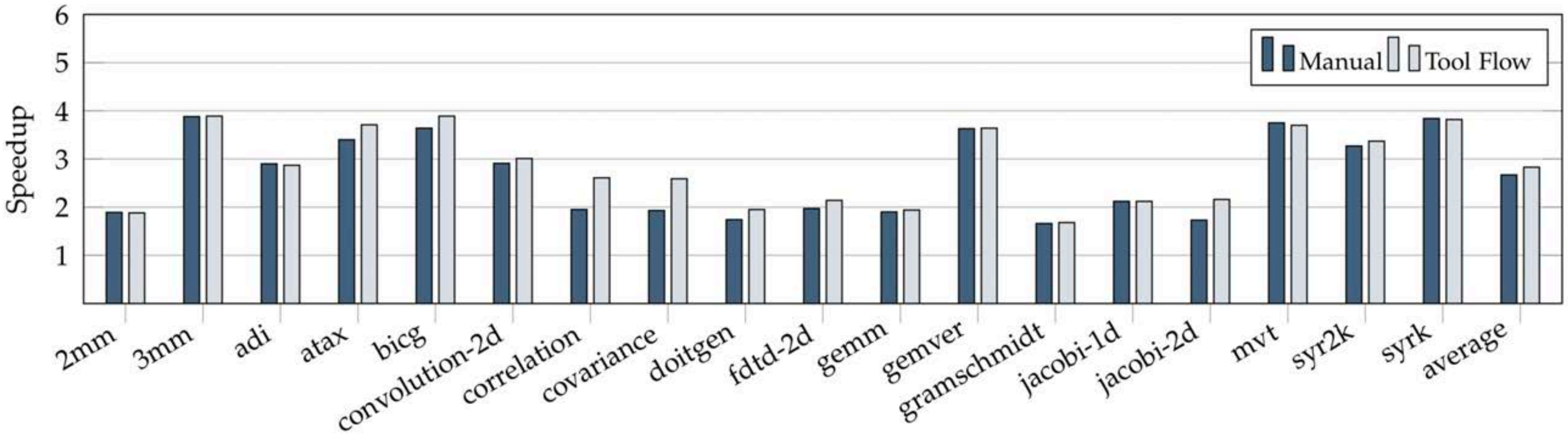
2008

Experimental multi-core from Tokyo Institute of Technology (Prof. Isshiki)

[DAC08]

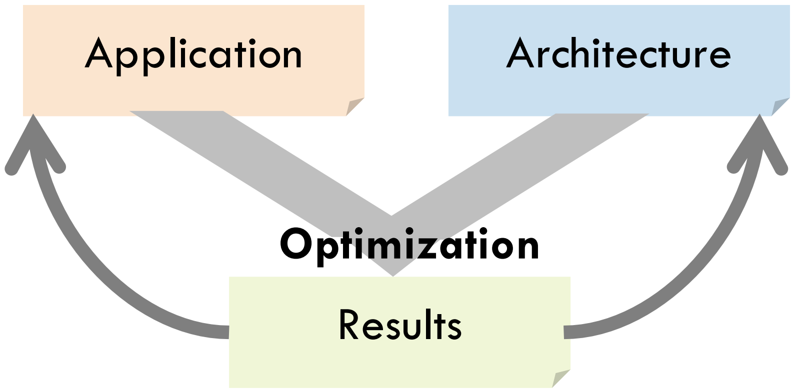
2018

Polybench on real Jetson TX1



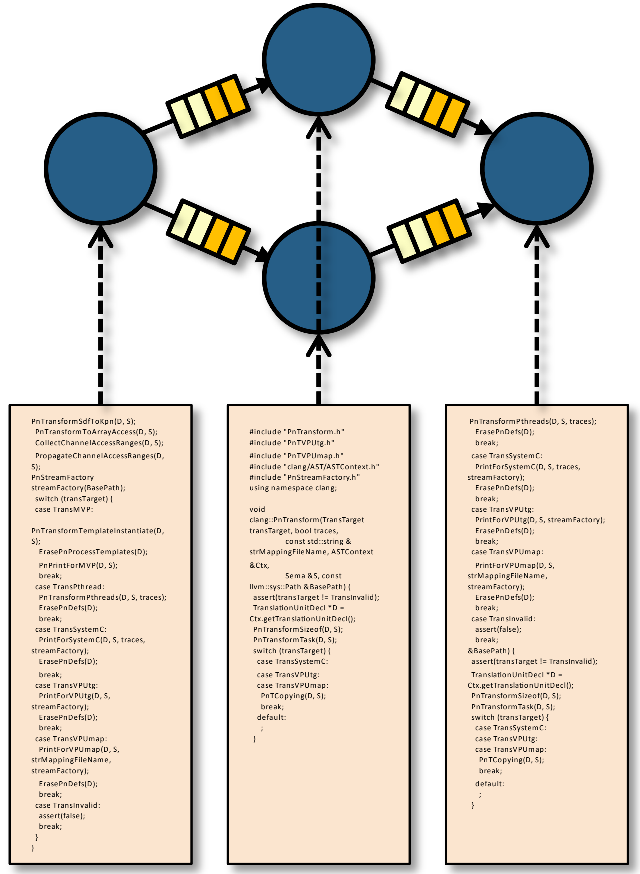
[Aguilar'18]

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



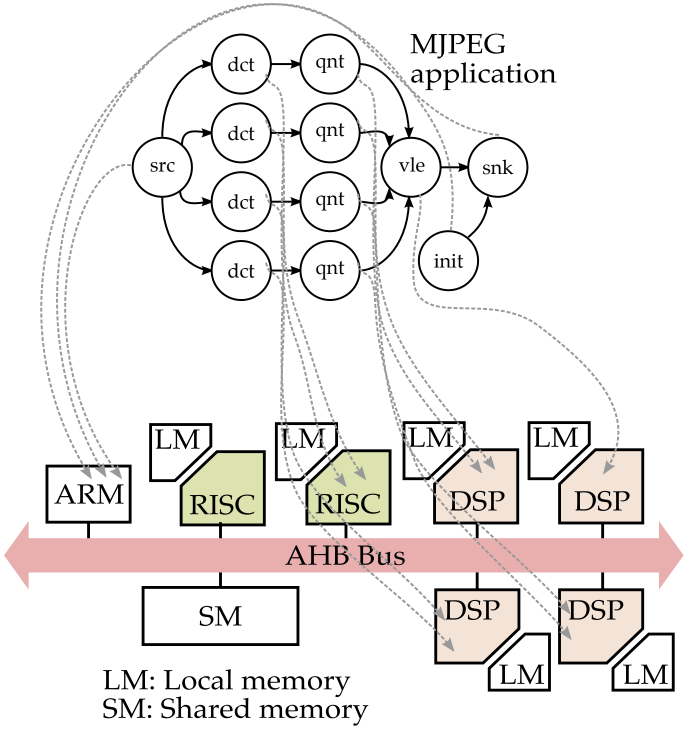
- The why
  - Explicit parallelism
  - Often: Determinism
  - Better analyzability (scheduling, mapping, optimization)

Dataflow compilation

- Plenty of research on
  - Formal Models of Computation (MoCs)
  - Language, compiler and mapping algorithms
  - Hardware modeling, performance estimation
  - Runtime systems
  - Code generation for heterogeneous multicores

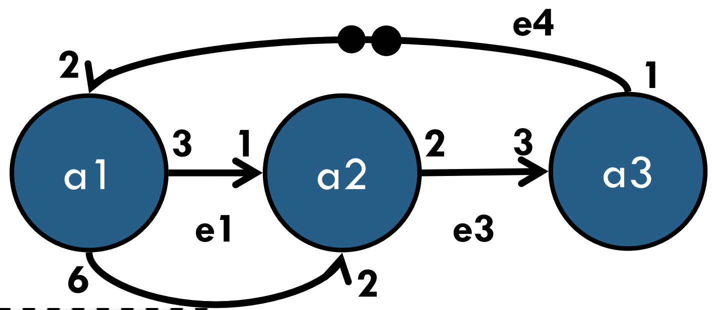
[IEEE TII'13, Springer14]

SILEXICA



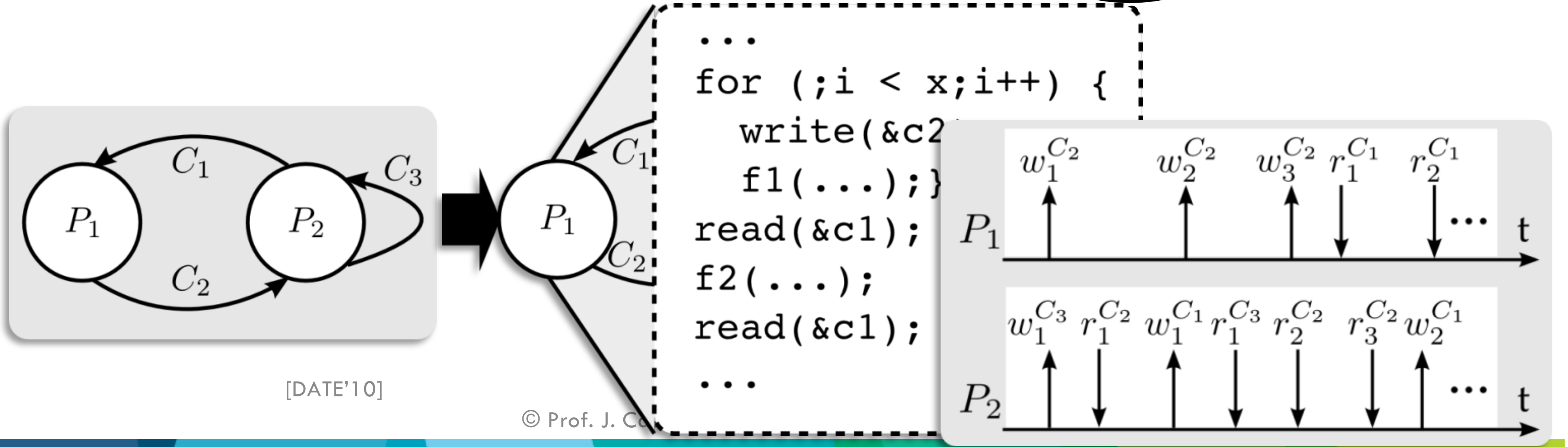
Static vs dynamic models

- Large body of successful work on static models like synchronous dataflow
  - Schedule and bounds computable at compile time!



[Lee87]

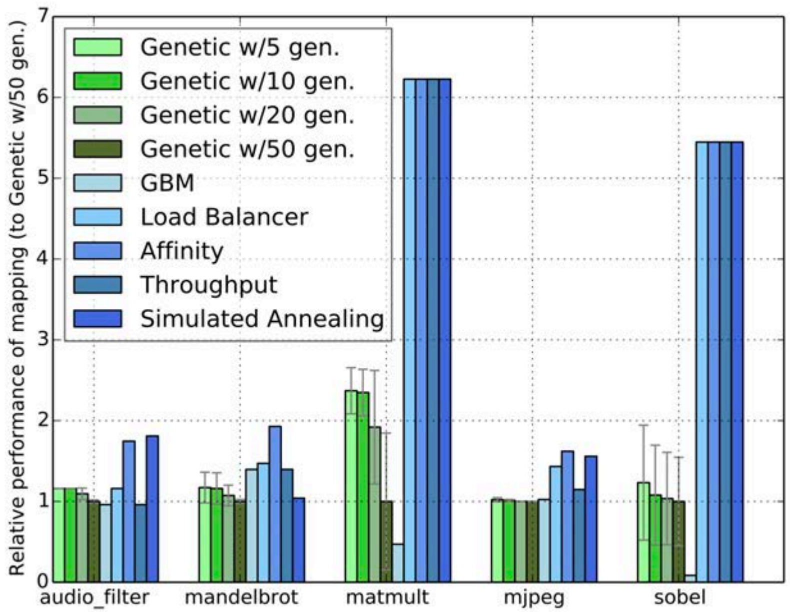
- Dynamic models: Data-dependent behavior

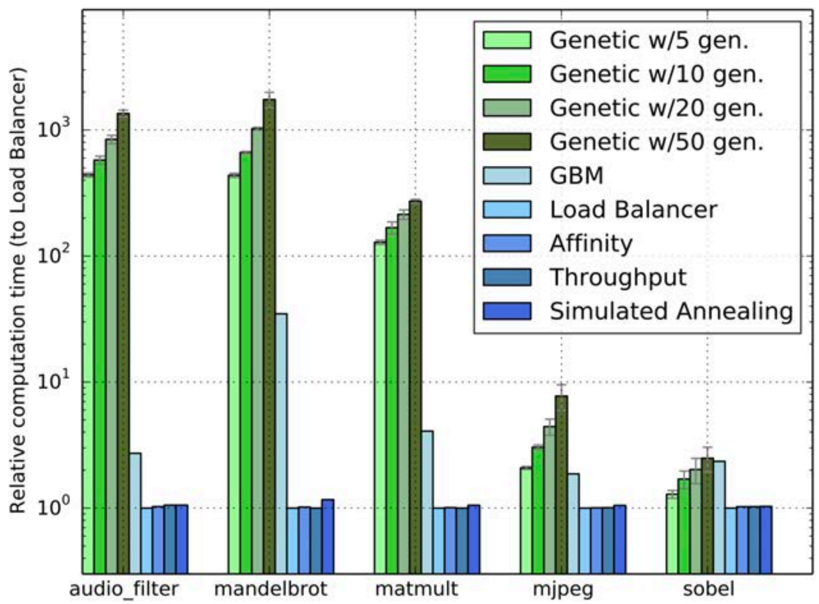


[DATE'10]

Automatic mapping to heterogeneous many-cores

- Lots of research on fixed design-time mapping algorithms (e.g., using genetic algorithms), e.g, Sesame, DOL [Erbas06, Thiele07]
- We worked on trace-based heuristics [DAC'12]



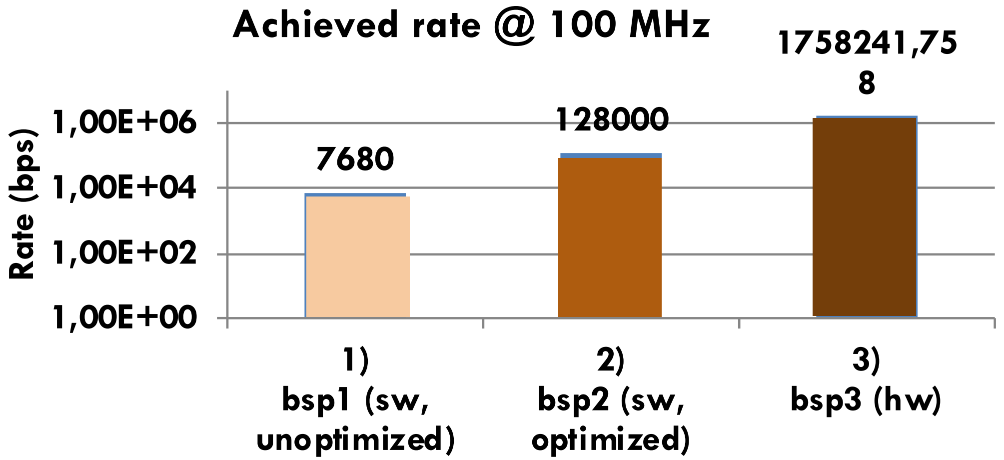


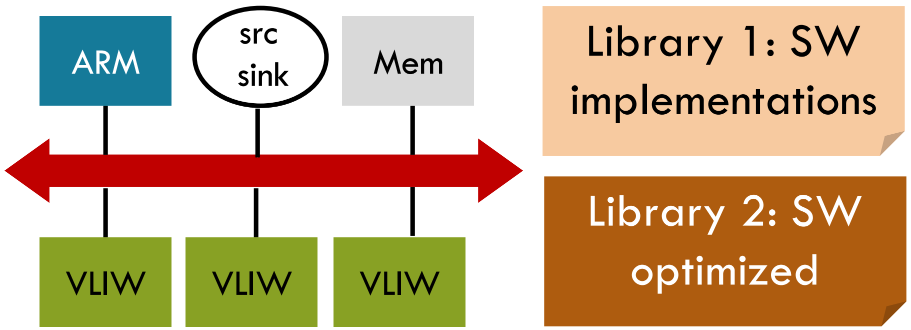
Multimedia apps on TI Keystone II

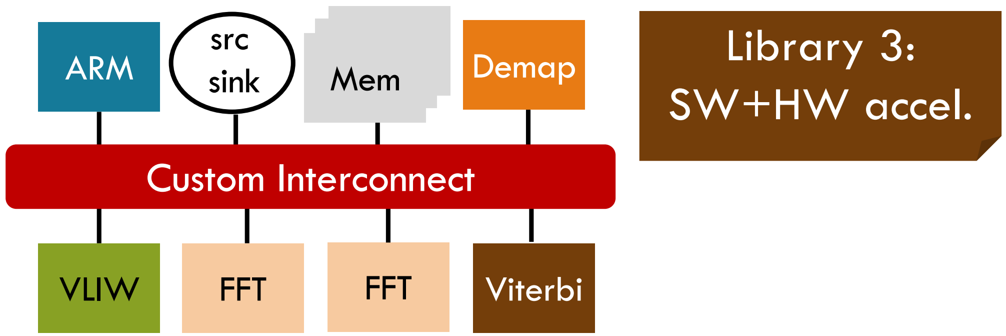
[MCSOC'16]

Support for HW acceleration in SW-defined radio

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



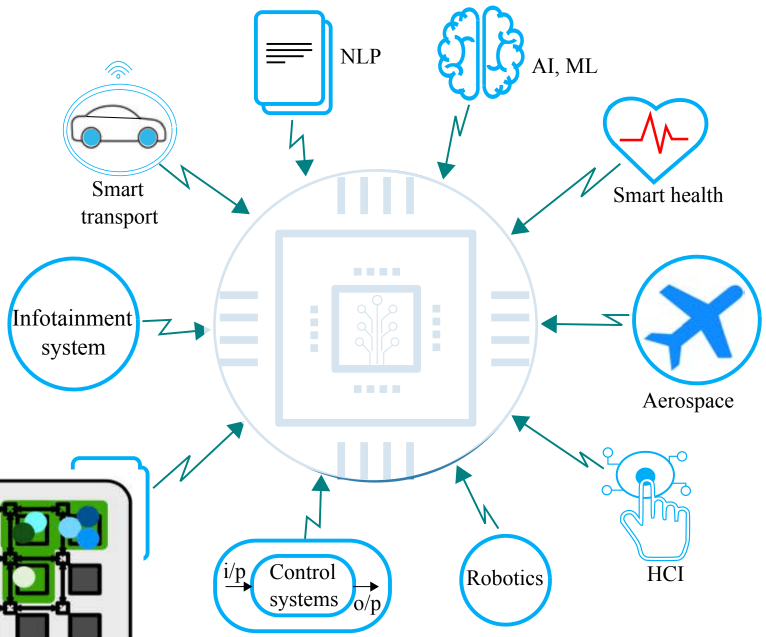


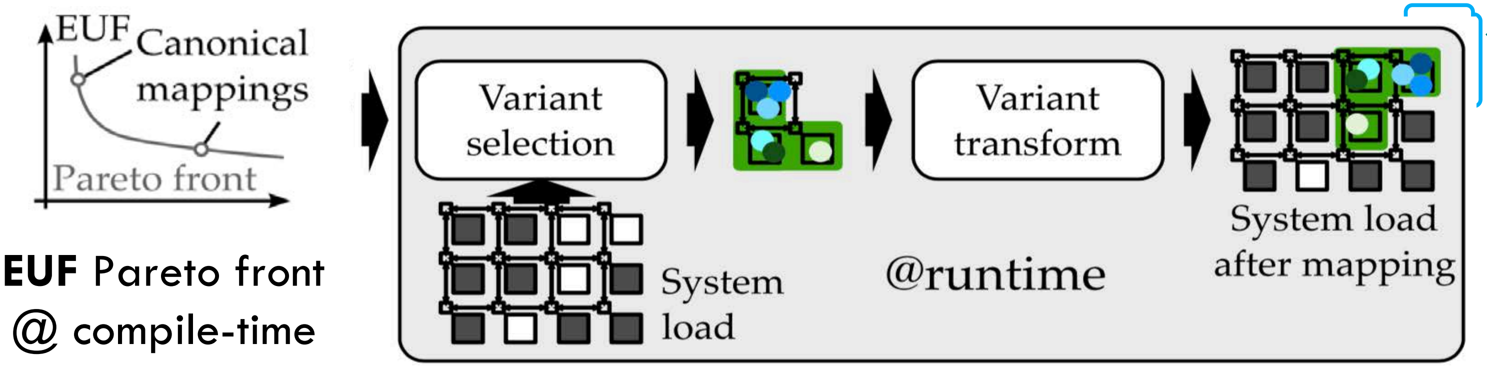


[SDR'10, ALOG'11]

System dynamics: Hybrid mapping

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



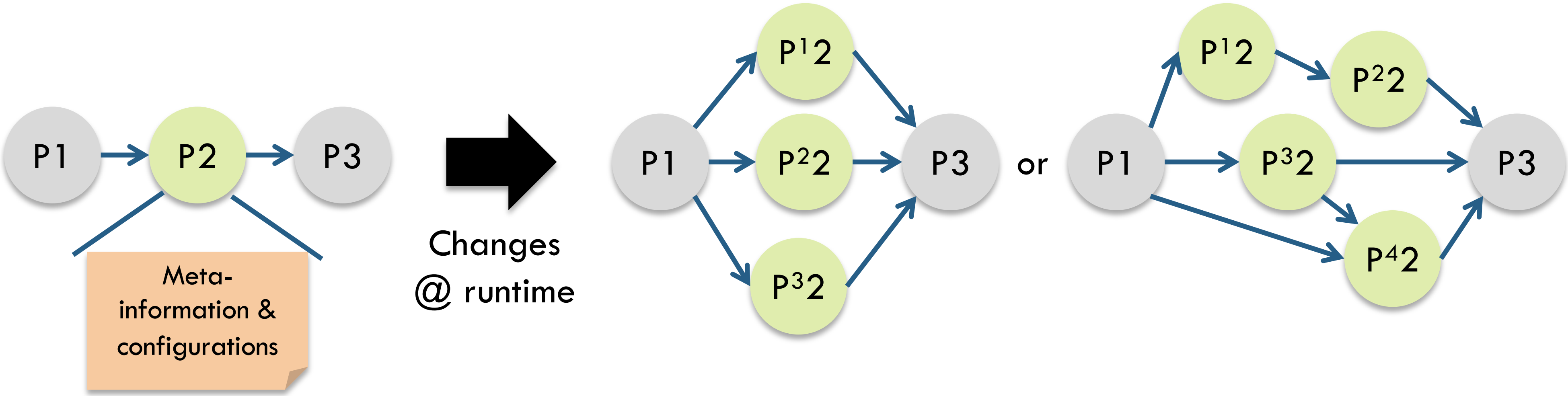


EUF Pareto front
@ compile-time

CHAIR FOR COMPILER CONSTRUCTION

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)
- Similar to AdaPNet or parameterized SDFs [Schor'14, Desnos'13]



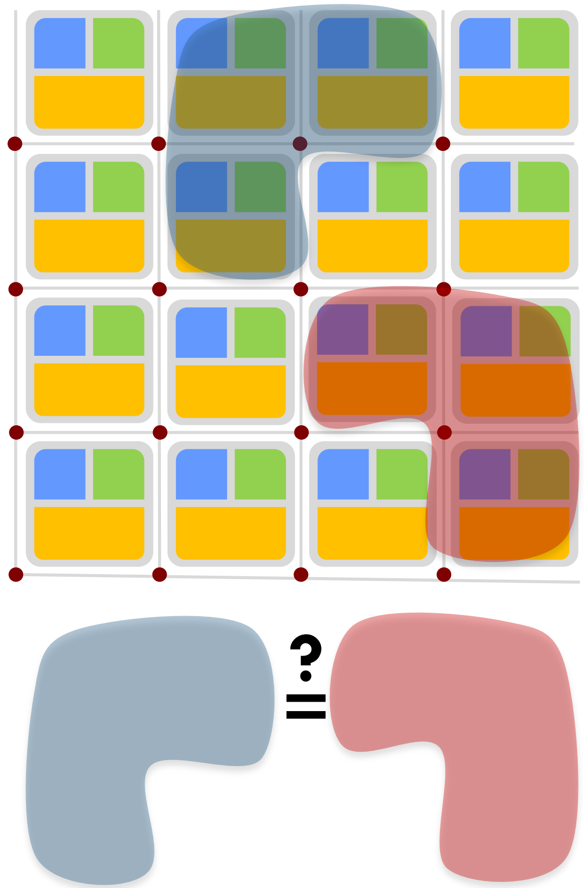
[PARMA-DITAM'18]

Exploiting symmetries

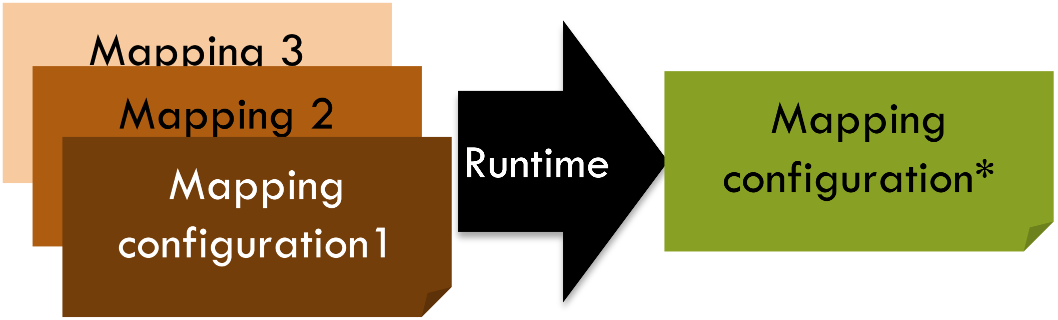
- Intuition
  - SW: Some tasks/processes/actors may do the same
  - HW: Symmetric latencies (CoreX ←→ CoreY)
  - Symmetry: Allows **transformations** w/o changing the **outcome**

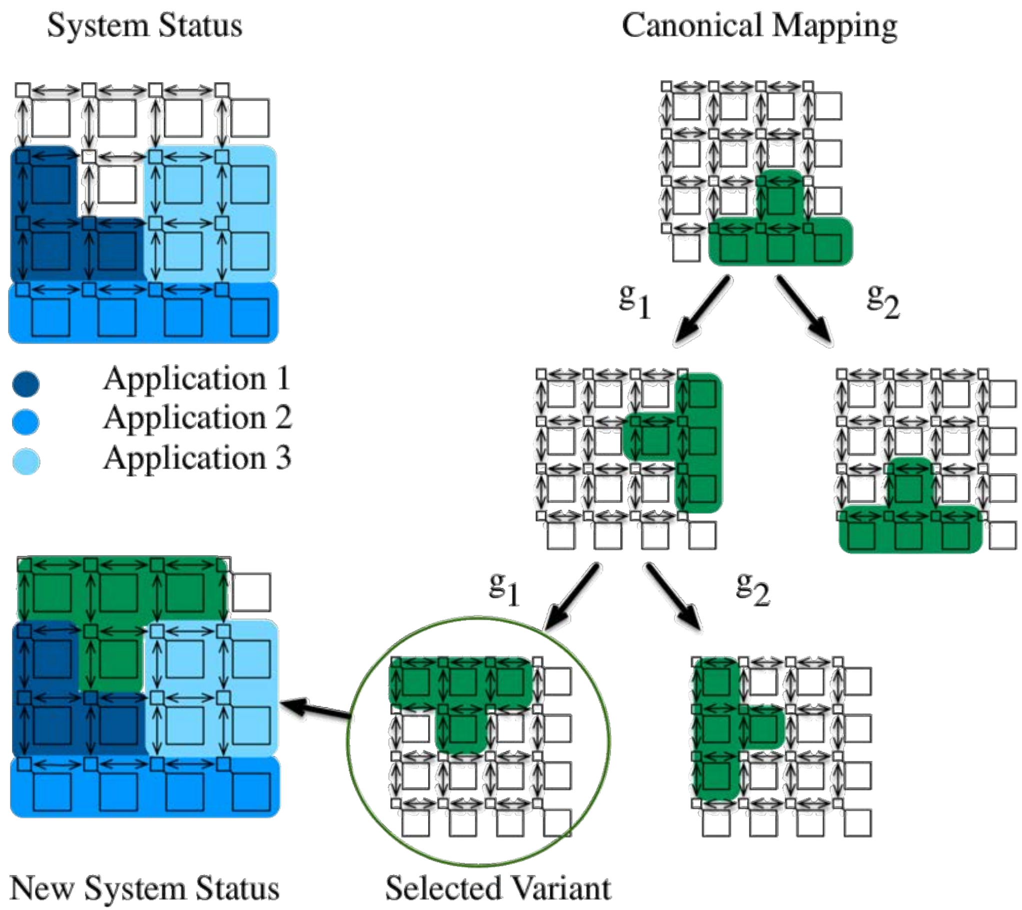
→ No need to analyze all possible mappings (prune search space)

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



Flexible mappings: Generalized Tetris



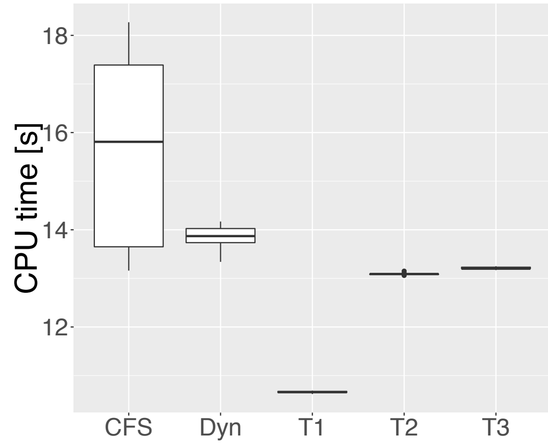


- Given multiple **canonical** configs by compiler, select one at run-time
- Exploit mapping **equivalences** and **similarities**

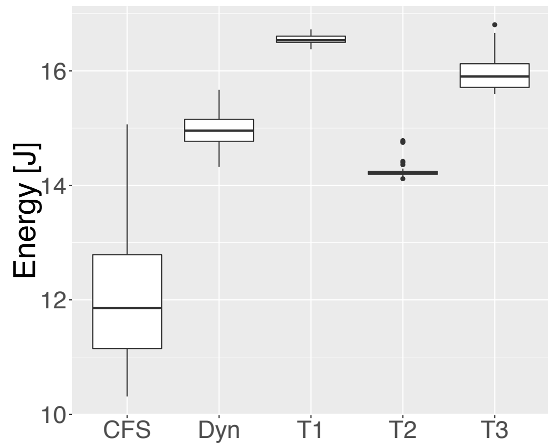
[SCOPES'17b]

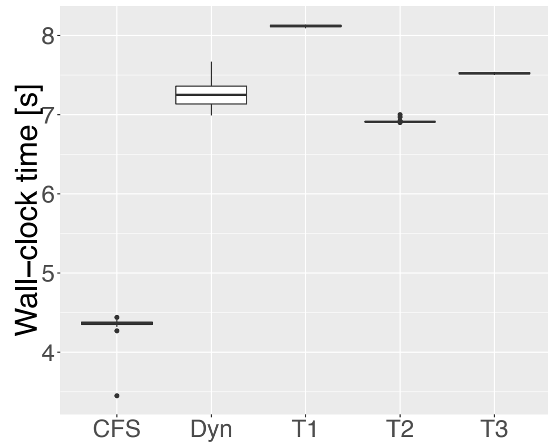
Flexible mappings: Run-time analysis

- 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 [DAC'12]



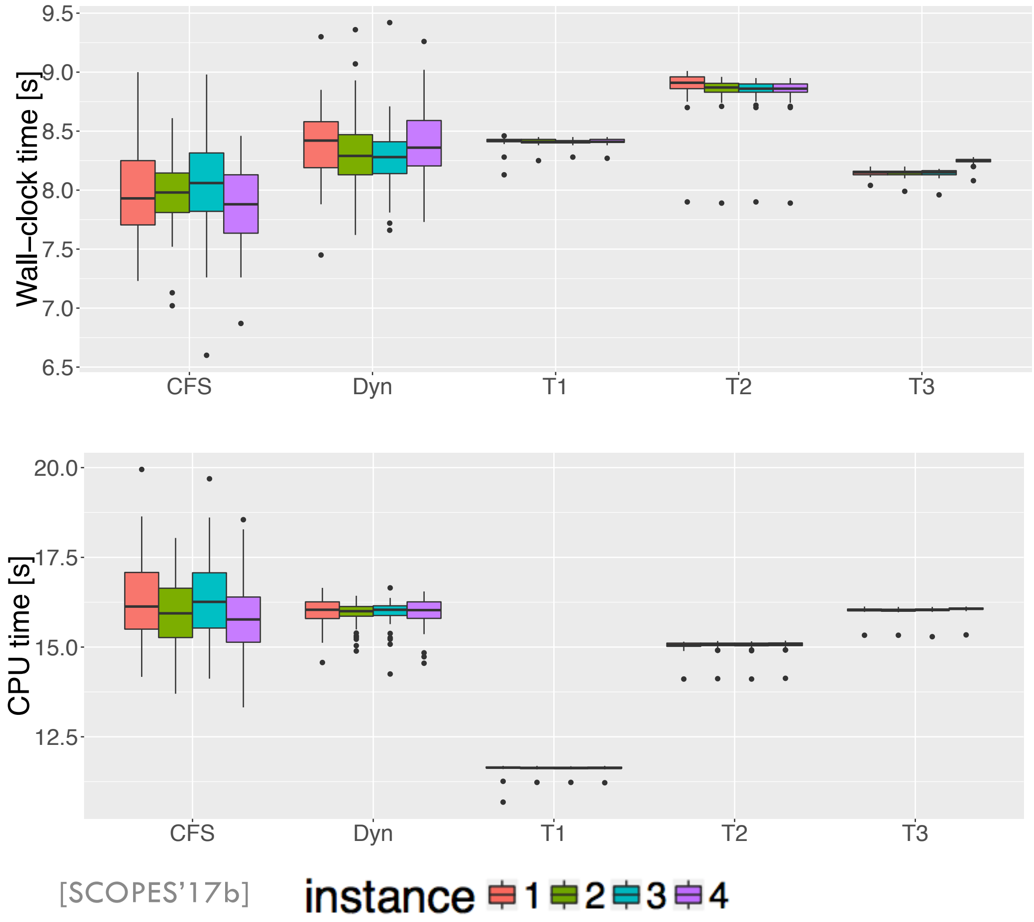
Single AF

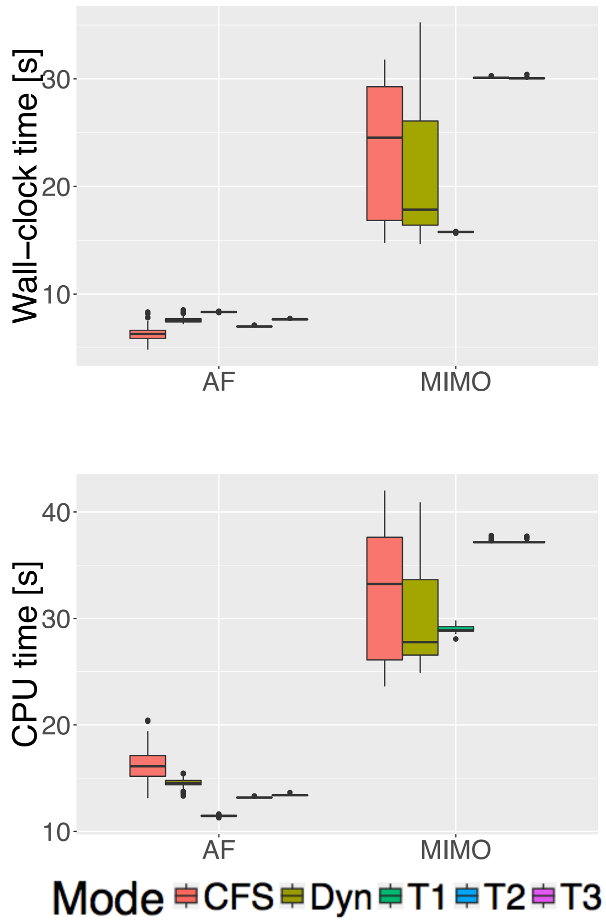




[SCOPES'17b]

Flexible mappings: Multi-application results (1)





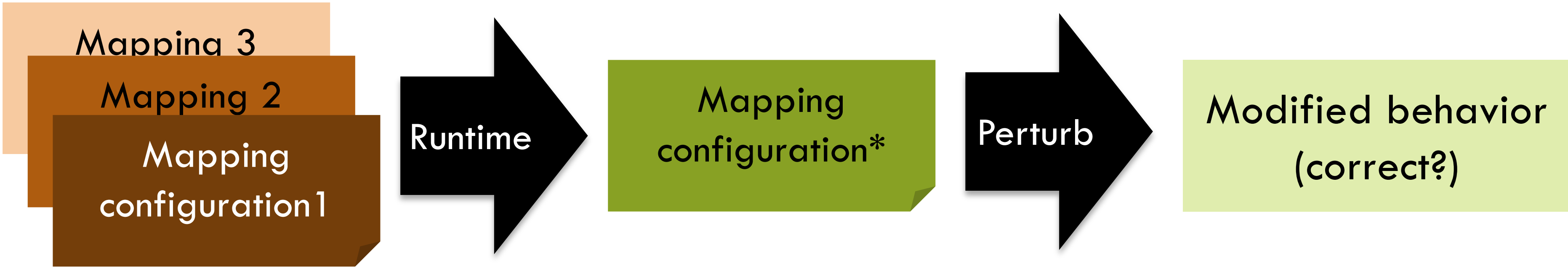
More predictable performance

Comparable performance to dynamic mapping

[SCOPES'17b]

Robustness

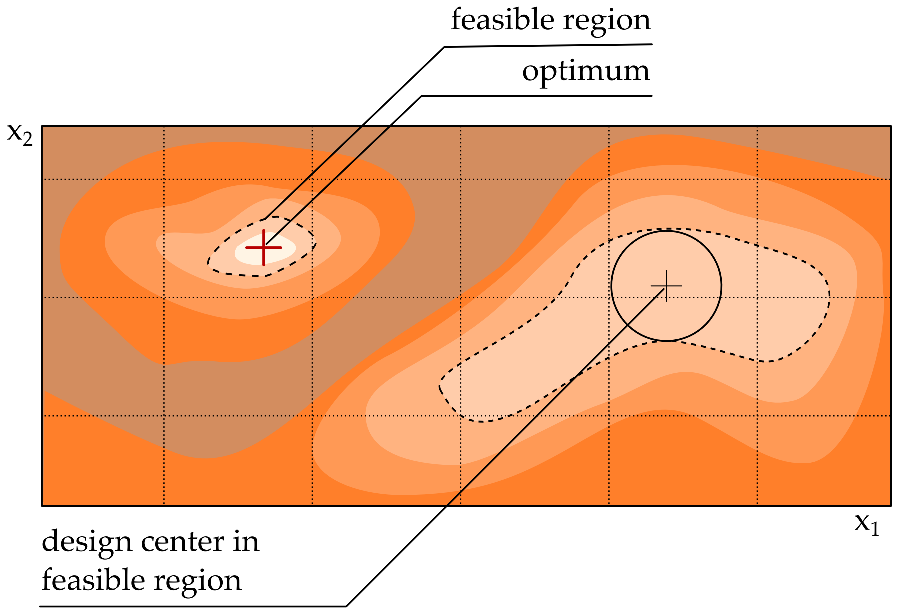
- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions



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

Design centering

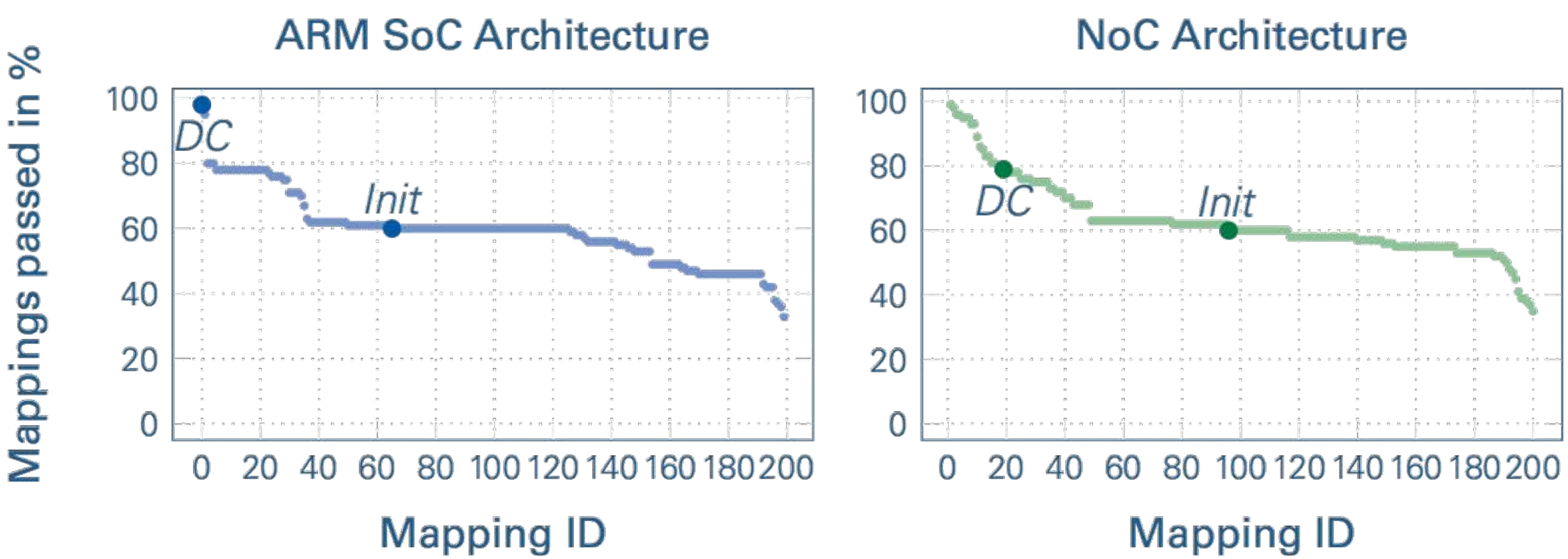
- Design centering: Find a mapping that can better tolerate **variations** while staying feasible
- Studied field, in e.g., biology, circuit design or manufacturing systems.
- Currently
  - Using a bio-inspired algorithm
  - Robust against OS changes to the mapping



[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)

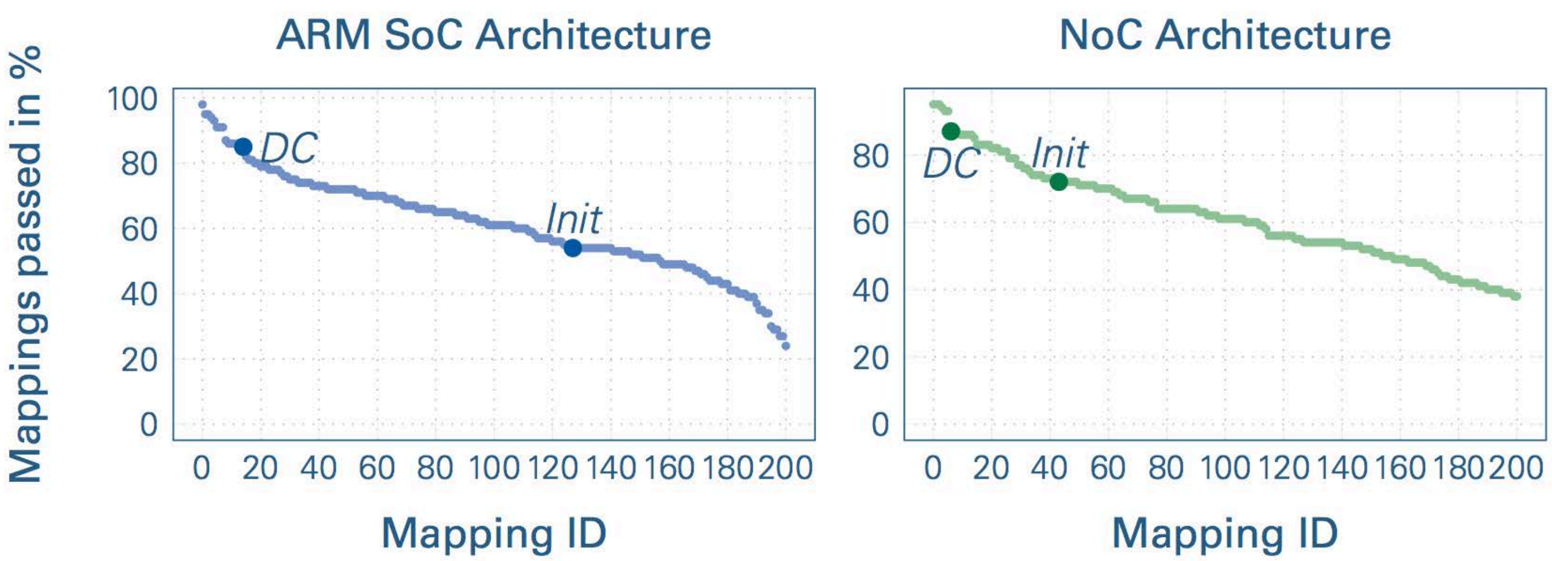


MIMO-OFDM

[SCOPES'17a]

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)

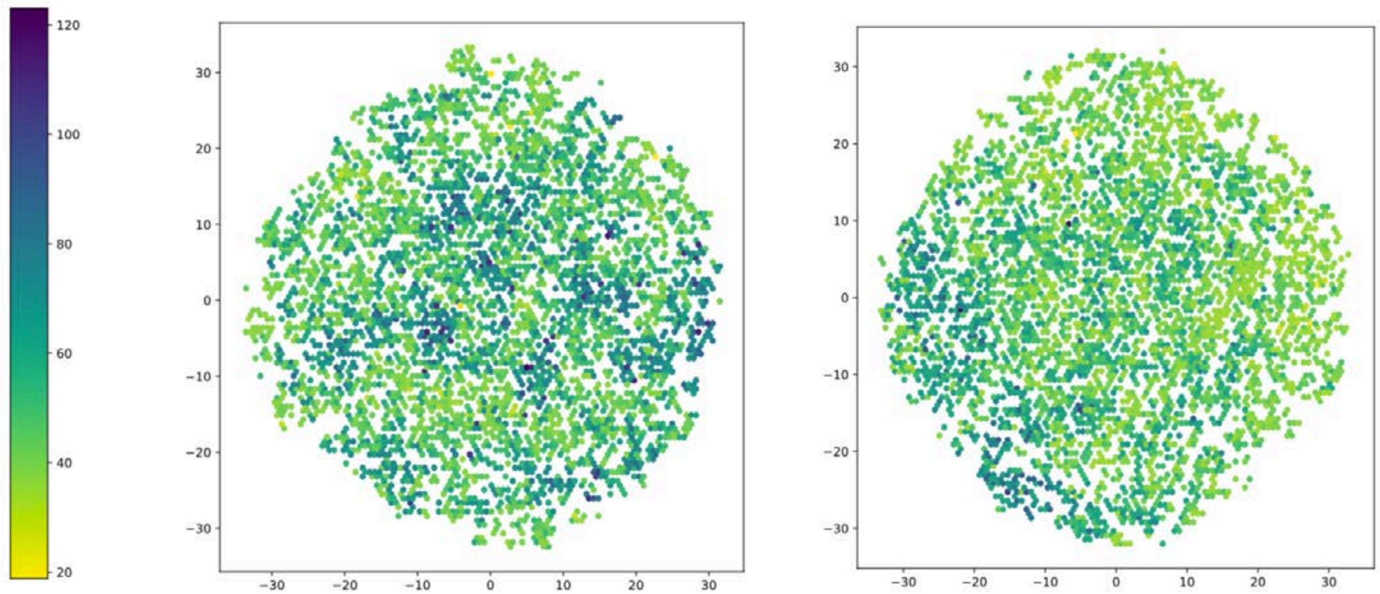


Audio-filter

[SCOPES'17a]

Ongoing work: Improve representations

- Work on embeddings: Architectures → Real numbers
- Novel mapping representations: faster heuristics & more efficient heuristics?
- Example: T-SNE Visualization for mappings space (8 tasks on Odroid XU4)



[MCSoC'18]

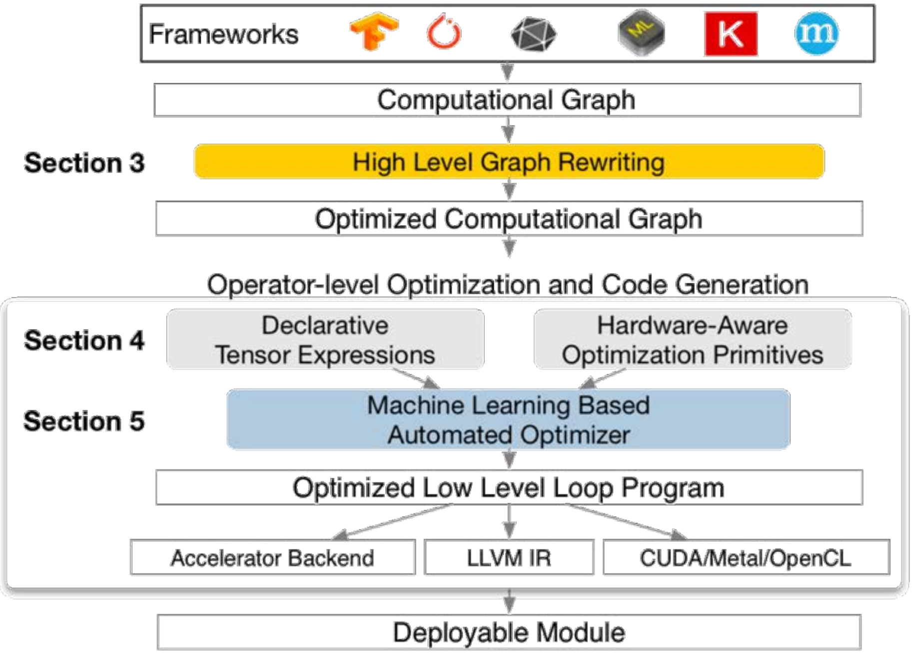
Higher-level programming abstractions

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

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

ML revolution: Frameworks and architectures

- Many existing frameworks, e.g., TVM, Tensor Comprehensions, TensorFlow, …
- Lots of traction in hardware architectures: TPU, V100, …



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 scope 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];
      }
    }
  }
}
```

Example: Interpolation operator

- Interpolation: $\mathbf{v}_e = (\mathbf{A} \otimes \mathbf{A} \otimes \mathbf{A})\, \mathbf{u}_e$

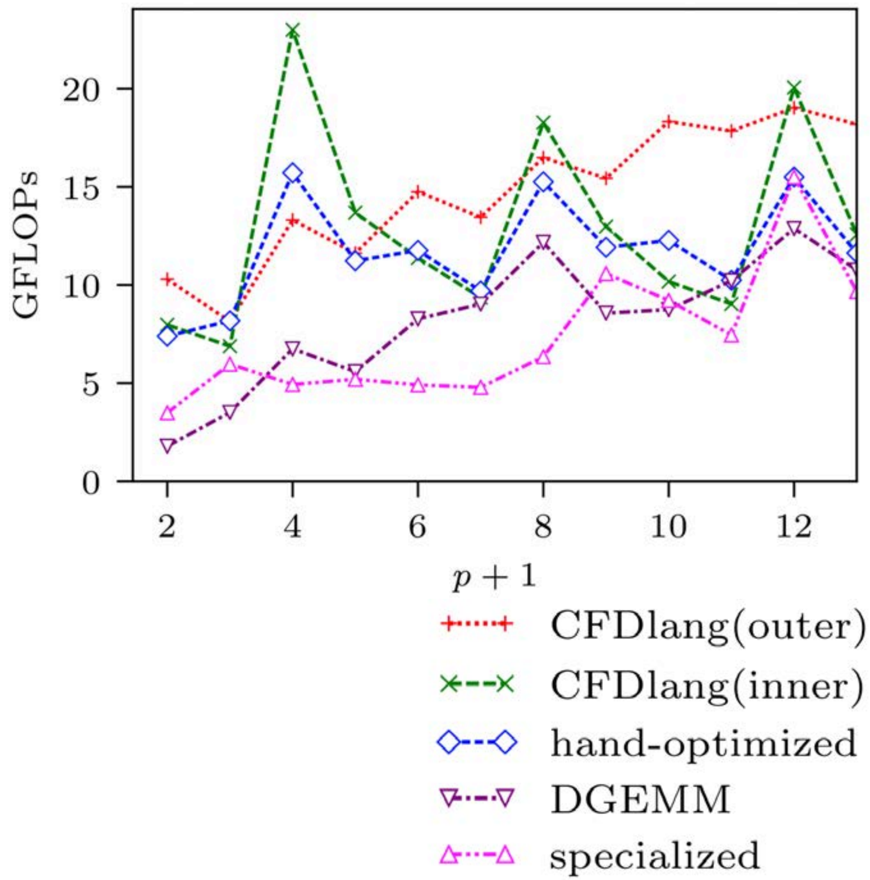
$$v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}$$

- Three alternative orders (besides naïve)

E1: $v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot (A_{jm} \cdot (A_{il} \cdot u_{lmn})))$

E2: $v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn})$

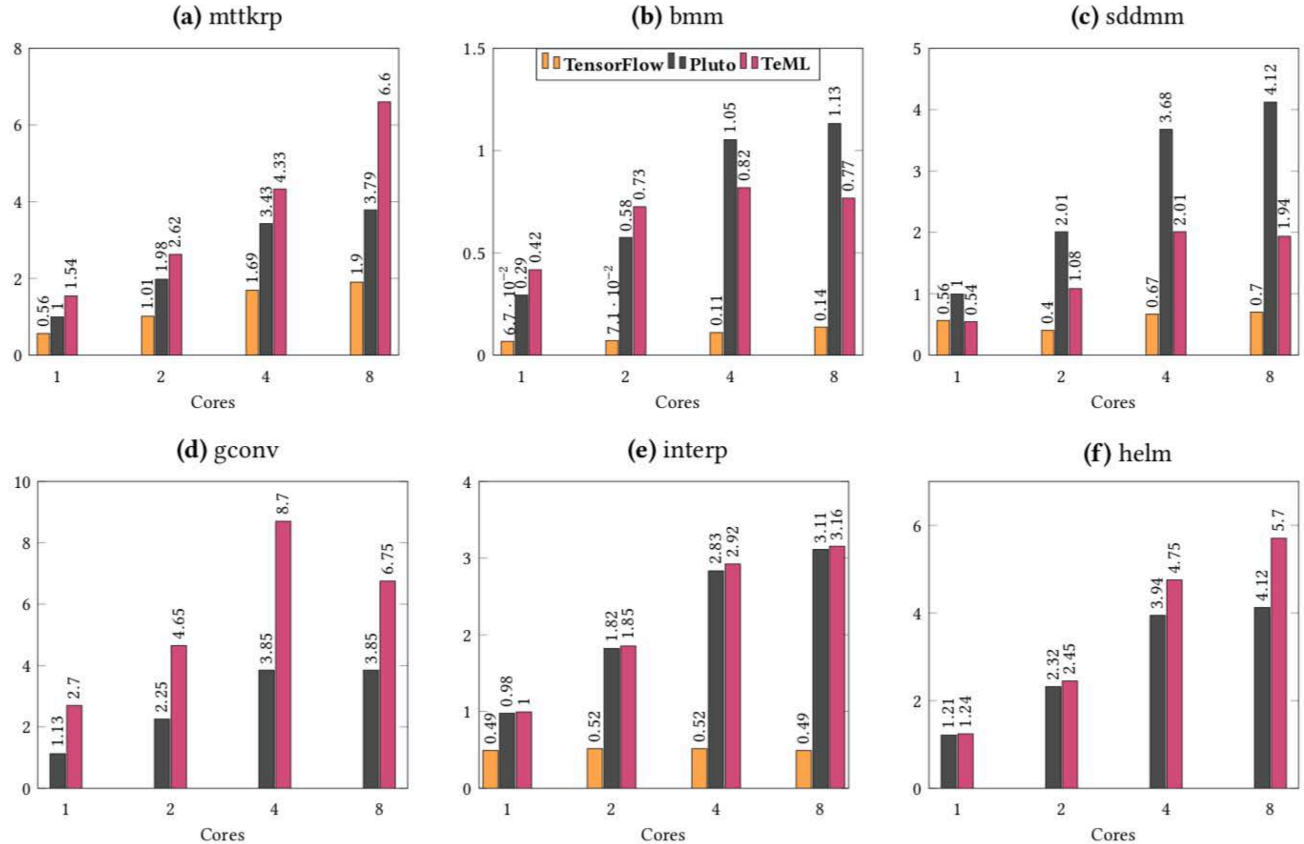
E3: $v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))$



[RWDSL'18, GPCE'17]

TeML: Results

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

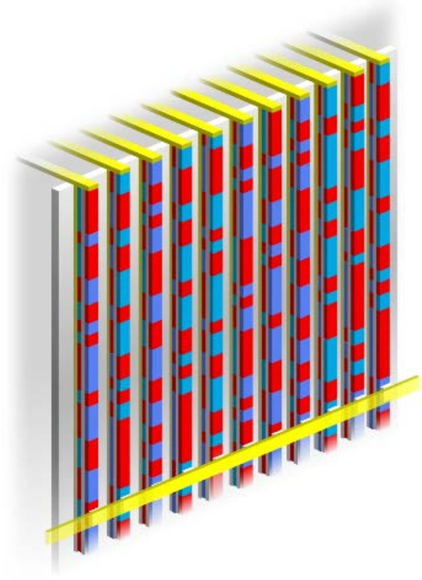


[GPCE'18]

CHAIR FOR COMPILER CONSTRUCTION

Emerging technologies: Racetrack memories

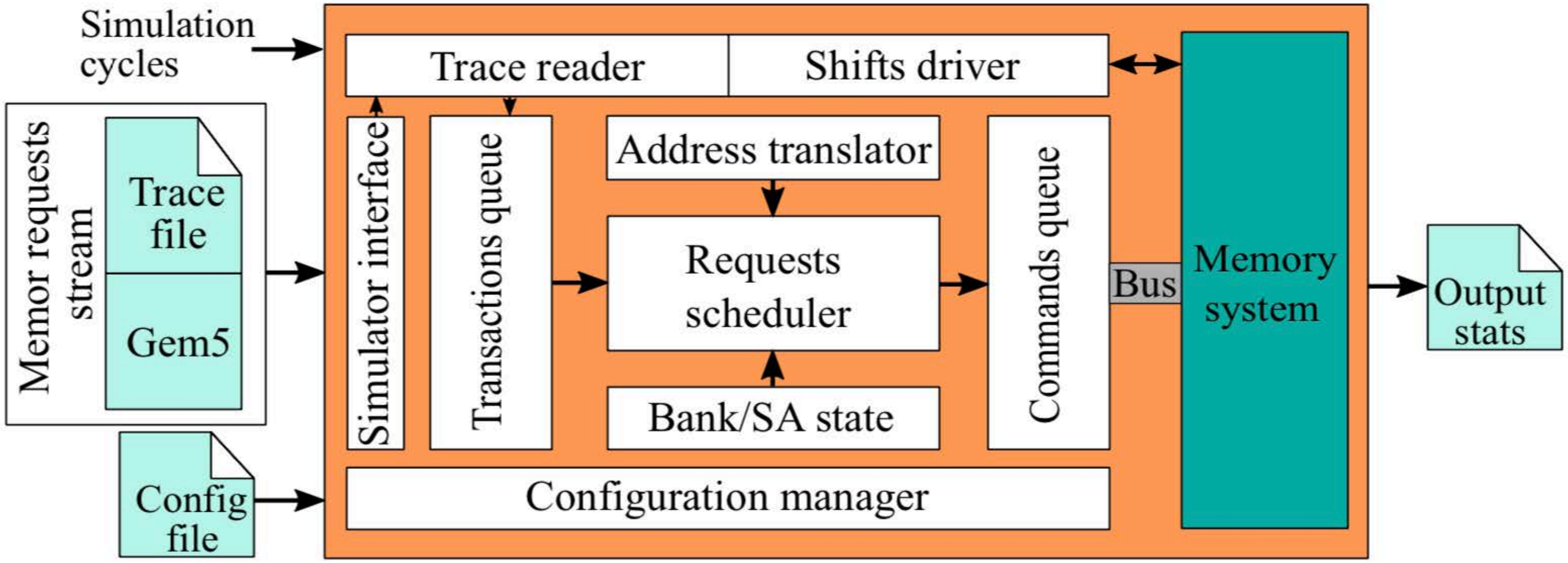
- Predicted extreme density at low latency
  - 3D nano-wires with magnetic domains
  - One port shared for many bits
  - Domains move at high speeds (1000 ms^{-1})
- Sequential: Game changer for current HW/SW stack
  - Memory management
  - Integration with other memory architectures
  - Data layout and allocation



[Parkin-Nature'15]

Simulating RTMs

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

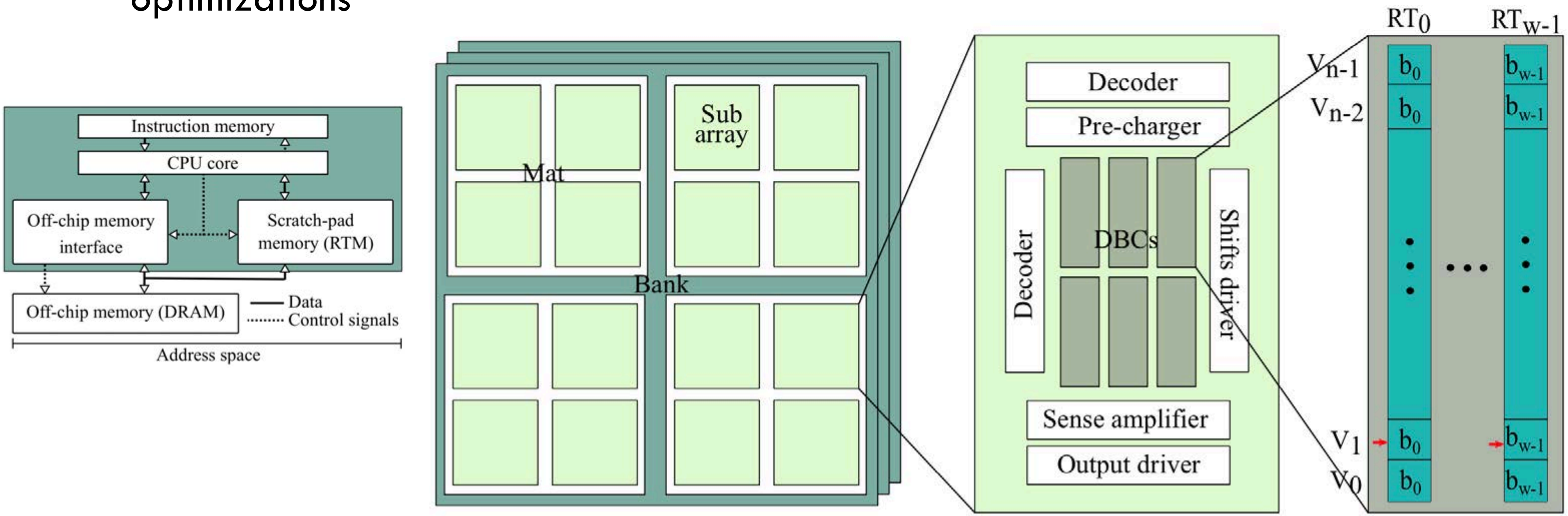


https://github.com/tud-ccc/RTSim

[IEEE CAL'19]

Architecture and data layout optimization

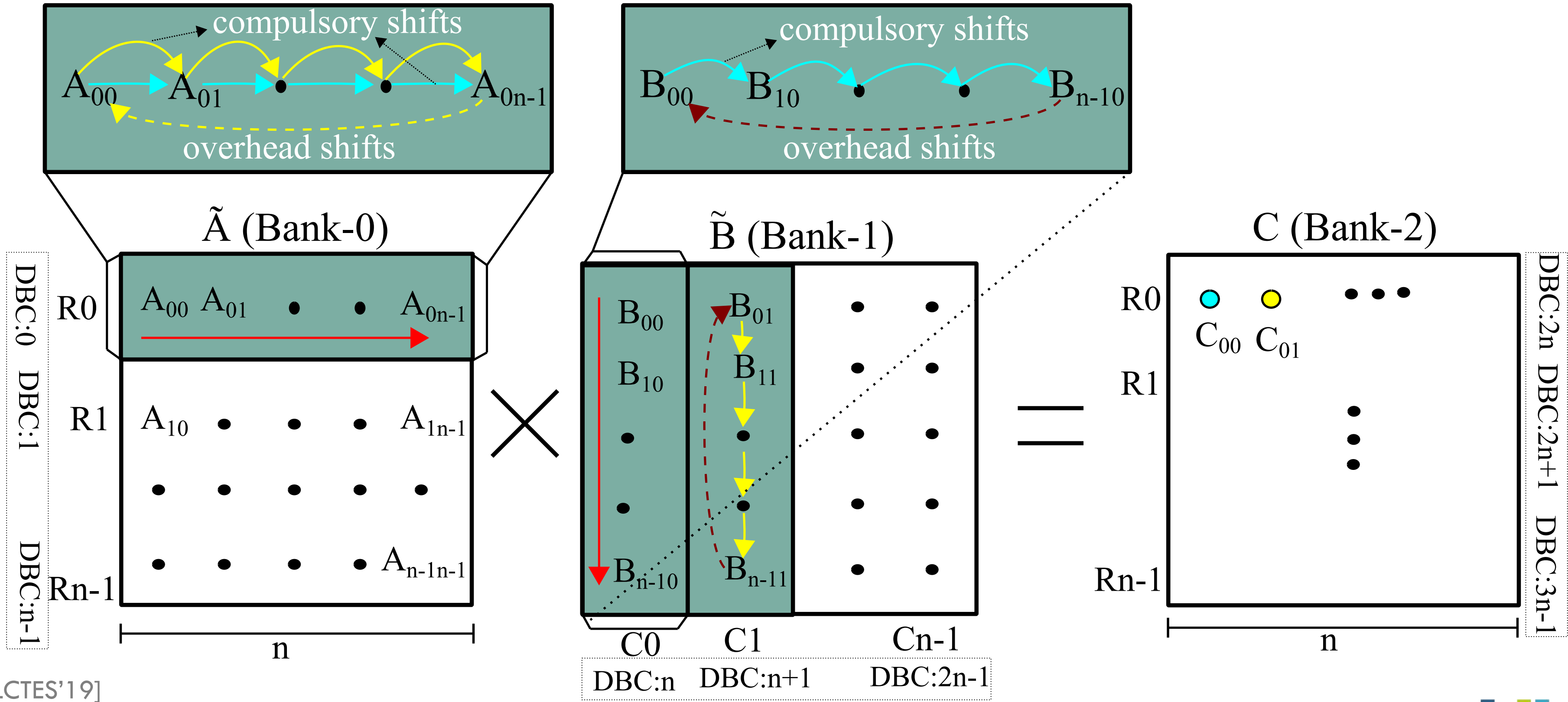
- Architecture – software co-optimization
  - Embedded system for inference: RTM as scratchpad with pre-shifting and other optimizations



[LCTES'19]

Architecture and data layout optimization

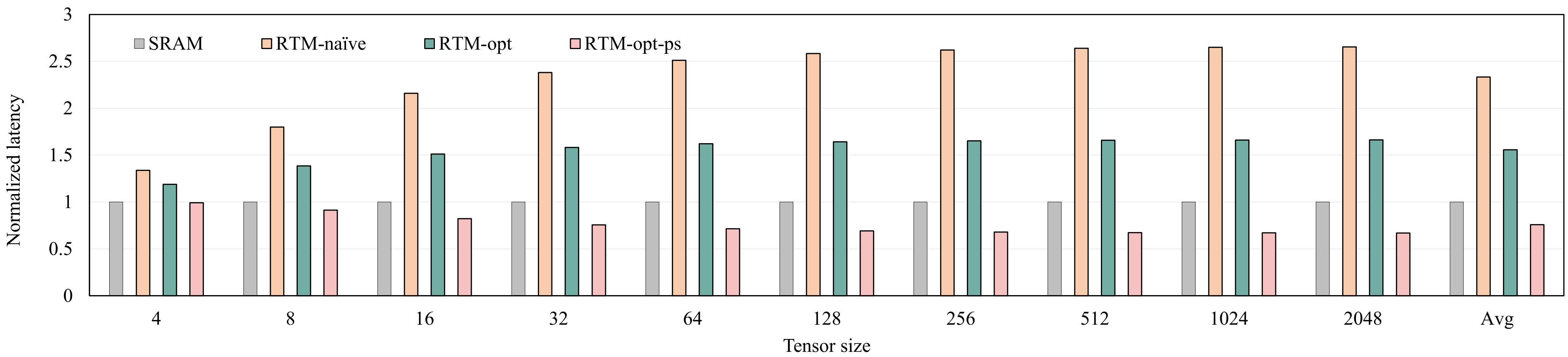
- Data-layout: Reduce the number of shifts



[LCTES'19]

Latency comparison vs SRAM

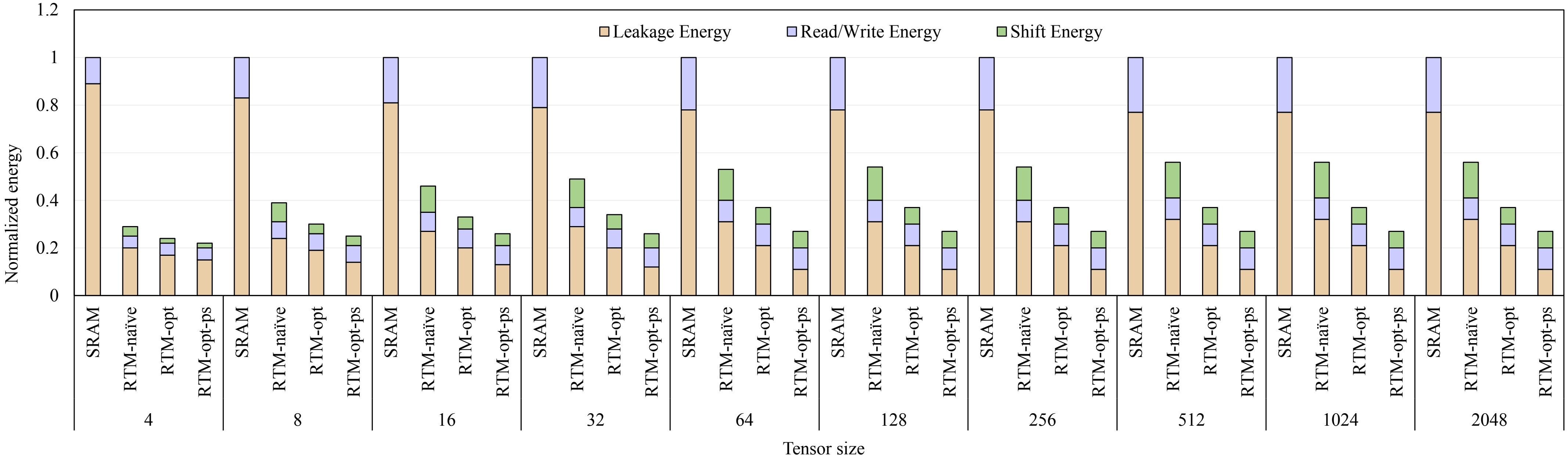
- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)



[LCTES'19]

Energy comparison vs SRAM

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



[LCTES'19]

Discussion

Summary

- Past ten years of efforts to handle the ever increasing complexity of SoCs
  - Advances in auto-parallelization (polyhedral, dynamic analysis)
  - Explicit parallel MoC-based programming models
  - Quest for more adaptivity but retaining time predictability
  - Higher-level abstractions: Example for tensor-based computations for accelerators

- Lots of challenges remain (thankfully)
  - Cost models and characterization of trade-offs (vs. blind searches)
  - Understand impact of emerging technologies
  - Syntax and semantics (for correctness): Lots of open questions
  - Time-semantics in programming and execution environments

References

[DAC'08] Ceng, J., et al. "MAPS: an integrated framework for MPSoC application parallelization." Proceedings of the 45th annual Design Automation Conference. DAC'08.

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

[Aguilar'18] Aguilar, M. A., Software Parallelization and Distribution for Heterogeneous Multi-Core Embedded Systems. RWTH Aachen University, RWTH Aachen University, 2018, 181pp

[Leupers'19] R. Leupers, M. A. Aguilar, J. Castrillon, W. Sheng, "Software Compilation Techniques for Heterogeneous Embedded Multi-Core Systems" , Chapter in Handbook of Signal Processing Systems (3rd Edition) (Bhattacharyya, S. S. and Deprettere, E. F. and Leupers, R. and Takala, J.) , Springer New York, pp. 1021–1062, Sep 2018.

[Tournavitis09] G. Tournavitis and et al."Towards a Holistic Approach to Auto-Parallelization -- Integrating Profile-driven Parallelism Detection and Machine-Learning Based Mapping" PLDI 09.

[Cordes10] D. Cordes, P. Marwedel, A. Mallik. "Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming." Conference on HW/SW codesign and system synthesis. ACM, 2010.

[Streit12] Streit, K.; Hammacher, C.; Zeller, A. & Hack, S. Sambamba: A runtime system for online adaptive parallelization. Compiler Construction, 2012, 240-243

[Edler18] T. J.K. Edler von Koch, S. Manilov, C. Vasiladiotis, M. Cole, and B.Franke. Towards a Compiler Analysis for Parallel Algorithmic Skeletons. Compiler Construction (CC'18), Vienna, 2018.

[IEEE TII'13] J. Castrillon, R. Leupers, and G. Ascheid, "MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs," IEEE Transactions on Industrial Informatics, vol. 9, no. 1, pp. 527–545, 2013.

[Lee87] E.A. Lee and D. G. Messerschmitt. "Synchronous data flow." Proceedings of the IEEE 75.9 (1987)

[DATE'10] J. Castrillon, R. Velásquez, et al. "Trace-based KPN Composability Analysis for Mapping Simultaneous Applications to MPSoC Platforms", Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2010, 753-758

[Thiele07] Thiele, L.; Bacivarov, I.; Haid, W. & Huang, K. "Mapping Applications to Tiled Multiprocessor Embedded Systems" ACSD '07: Proceedings of the Seventh International Conference on Application of Concurrency to System Design, IEEE Computer Society, 2007, 29-40

[Erbas06] Erbas, C.; Cerav-Erbas, S. & Pimentel, A. Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design IEEE Transactions on Evolutionary Computation, 2006, 10, 358 – 374

[DAC'12] J. Castrillon, A. Tretter, R. Leupers, G. Ascheid. "Communication-aware Mapping of KPN Applications Onto Heterogeneous MPSoCs" Design Automation Conference, ACM, 2012, 1266-1271

[MCSoC'16] A. Goens, R. Khasanov, J. Castrillon, S. Polstra, A. Pimentel, "Why Comparing System-level MPSoC Mapping Approaches is Difficult: a Case Study" , MCSoC-16.

[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.

[MCSoC'18] A. Goens, C. Menard, J. Castrillon, "On the Representation of Mappings to Multicores", MCSoC-18, pp. 184–191, Hanoi, Vietnam, Sep 2018.

[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.

[Schor'14] Schor, L.; Bacivarov, I.; Yang, H. & Thiele, L. "AdaPNet: Adapting Process Networks in Response to Resource Variations", ESWEEK'14, 2014

[Desnos'13] Desnos, K., et al. "Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration." SAMOS), 2013.

[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.

[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.

[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.

[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, Jun 2019