CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Dataflow and higher level abstractions for parallel programming

Jeronimo Castrillon

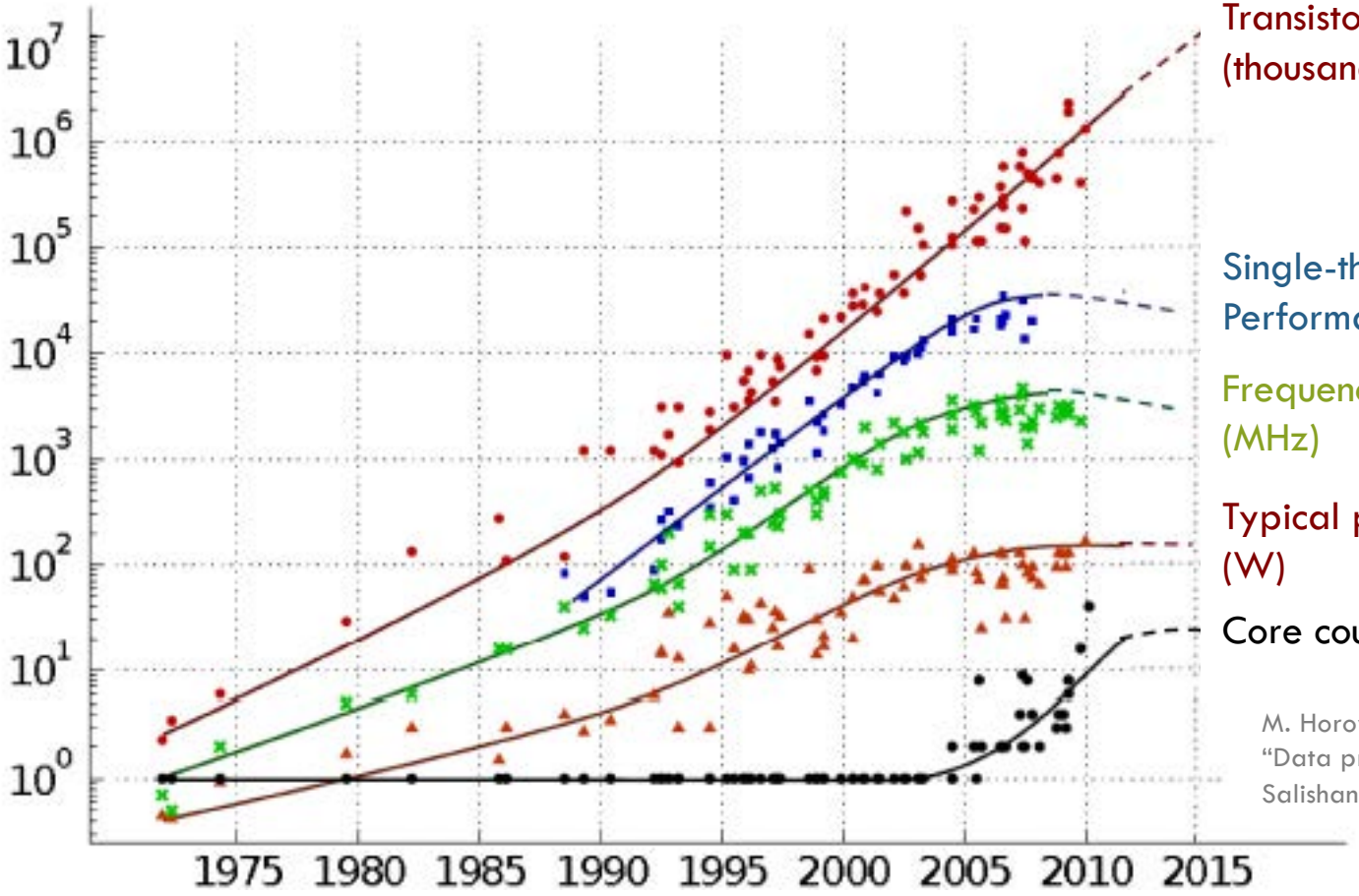
Chair for Compiler Construction (CCC)

TU Dresden, Germany

Cyber-Physical Systems Summer School 2019

Alghero, Sardinia, Italy. September 25, 2019

Evolution of Hardware



Transistors (thousands)

Single-thread Performance

Frequency (MHz)

Typical power (W)

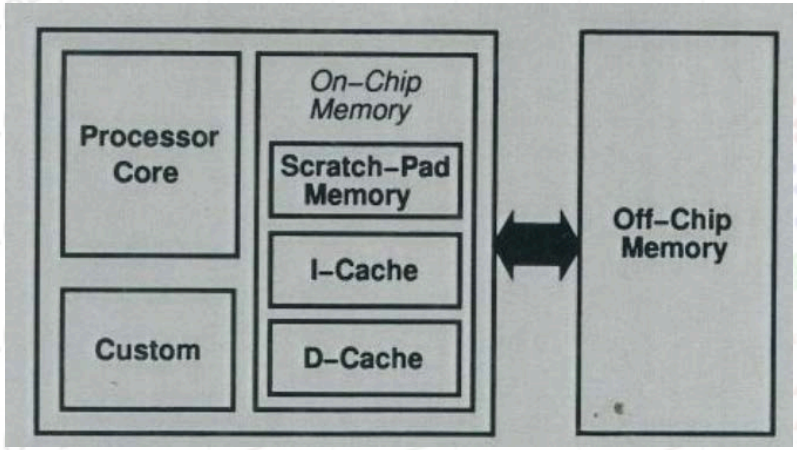
Core count

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

Evolution of Hardware: Great complexity

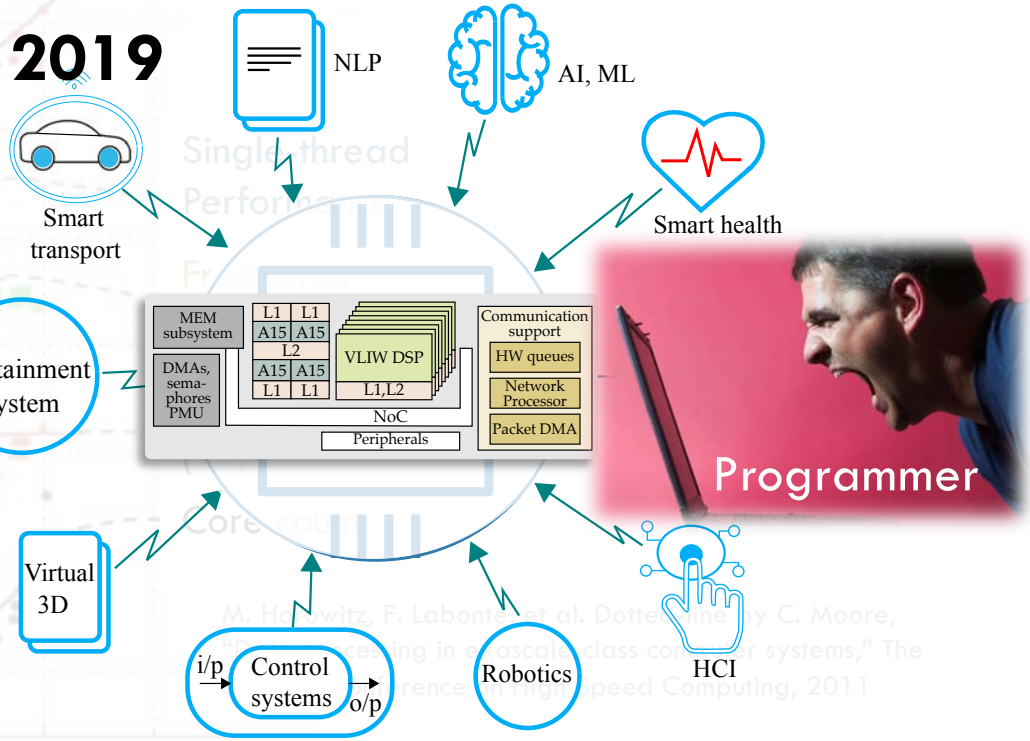
- Heterogeneous many-cores, scalable platforms, complex memory hierarchies, domain-specific accelerators, …

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



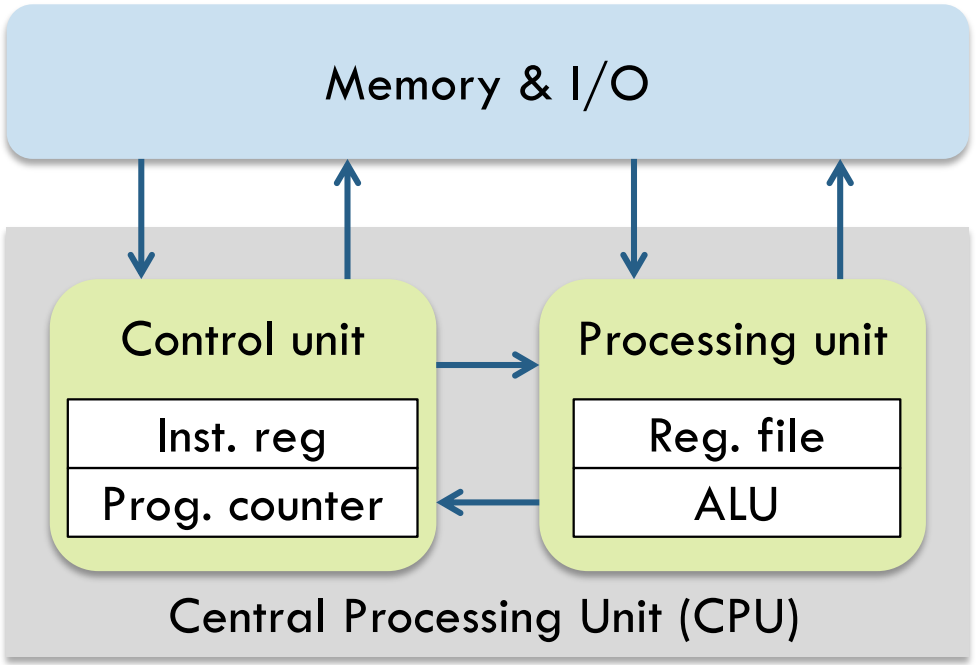
Models: Von Neumann

- Program: sequence of instructions in memory

```
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 (unsig  i9 = 0; i9 < 2; i9++) {
      t9[0] = 0.
      for (unsigne                0; i12_contr < 3; i12_contr++
        t9
      }
      t8[
      dou
      t10
      v[(
    }
  }
}
```

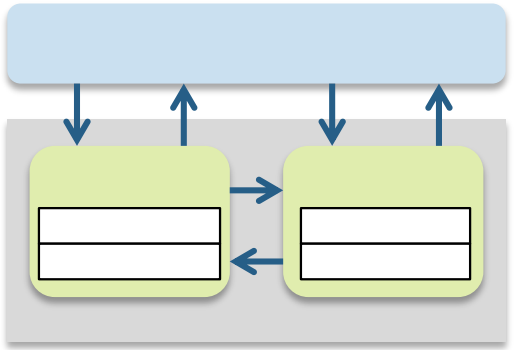
Good abstractions:

- "High-level" languages (C, …)
- Efficient HW implementation



Models: Von Neumann and Modern HW

- Different sources of parallelism
  - SIMD, threads, GPUs, …
- Fundamentally different resources
  - Dataflow accelerators, Tensor processing units, …



➔ Lots of effort in auto-parallelization and vectorization

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

Static/dynamic control/data analysis: Sample results

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

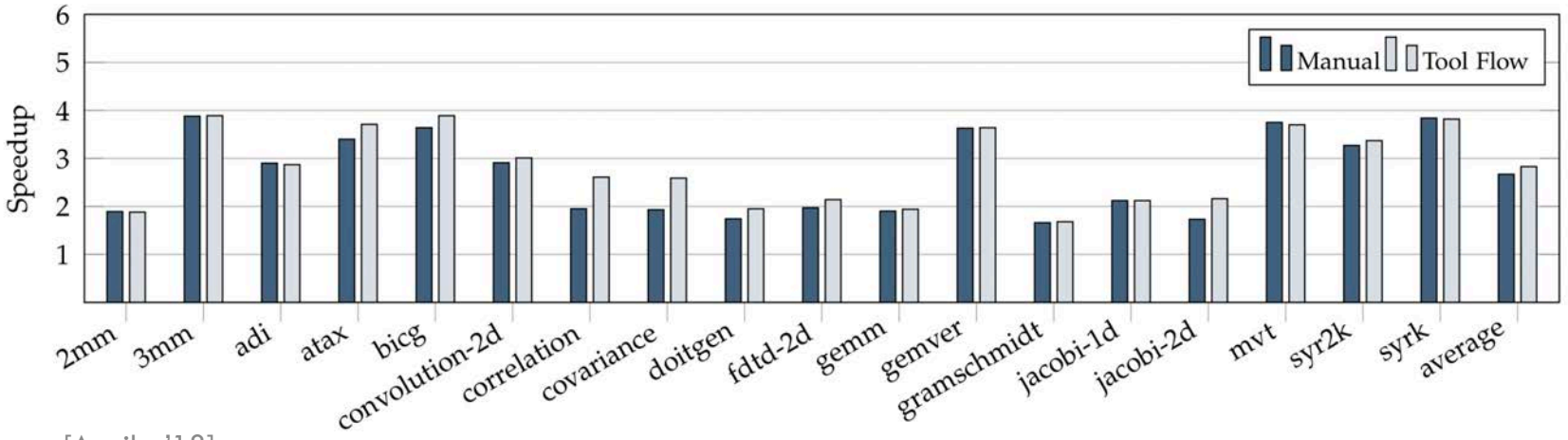
2008

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

[DAC'08]

2018

Polybench on real Jetson TX1



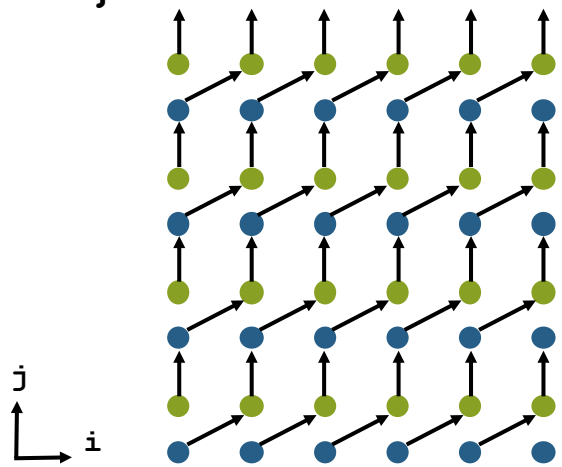
[Aguilar'18]

Problems for auto-parallelizing compilers

1) Find all dependencies?

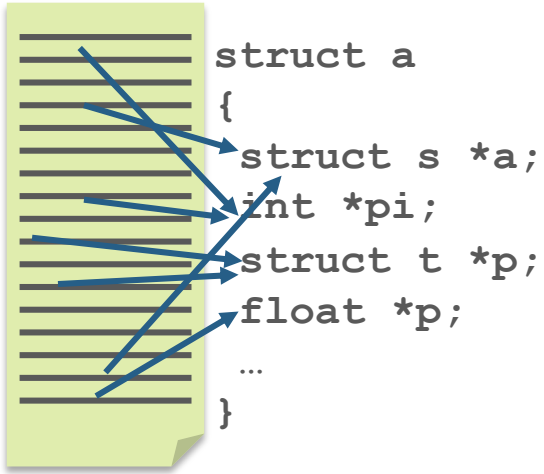
More often than not, impossible!

```
    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];
        }
```



Successful 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

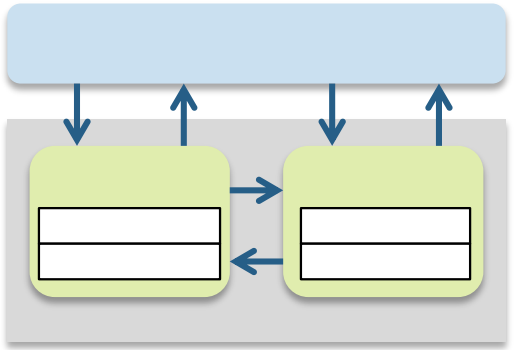
3) Dependencies can sometimes be violated! (they are artifacts of style)

```
19  while(!queue.empty())
20  {
21    // Dequeue a vertex from queue
22    s = queue.front();
23    queue.pop_front();
24
25    // Apply function f to s, accumulate values
26    result += f(s);
27
28    // Get all adjacent vertices of s.
29    // If an adjacent node hasn't been visited,
30    // then mark it as visited and enqueue it
31    for(i=adj[s].begin(); i!=adj[s].end(); ++i)
32    {
33      if(!visited[*i])
34      {
35        visited[*i] = true;
36        queue.push_back(*i);
37      }
38    }
39  }
40
41  return result;
42 }
```

[Edler'18]

Models: Von Neumann and Modern HW

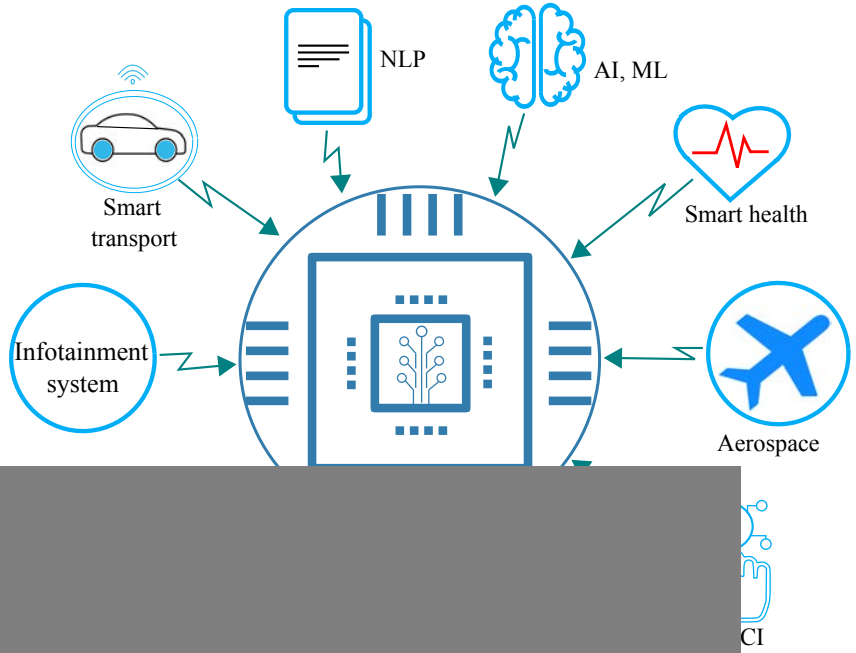
- Different sources of parallelism
  - SIMD, threads, GPUs, …
- Fundamentally different resources
  - Dataflow accelerators, Tensor processing unit, …



- **Von Neumann and sequential programming**
  - **Difficult to automatically extract parallelism**
  - **Difficult to recognize high-level operations**
  - **Difficult to ensure correctness (functional, timing, …)**

Models and Programming

- Specially important for CPS!
  - Domain specific
  - Interconnected systems
  - Interaction with physical world (Reactive)
  - Real-time (WCET, predictable HW)
  - Dynamic changing conditions
  - …



In this lecture

- Classic dataflow modeling
- Current work on extensions: dynamic, adaptable… (CPS)
- Raise-level of abstraction: Domain-specific languages

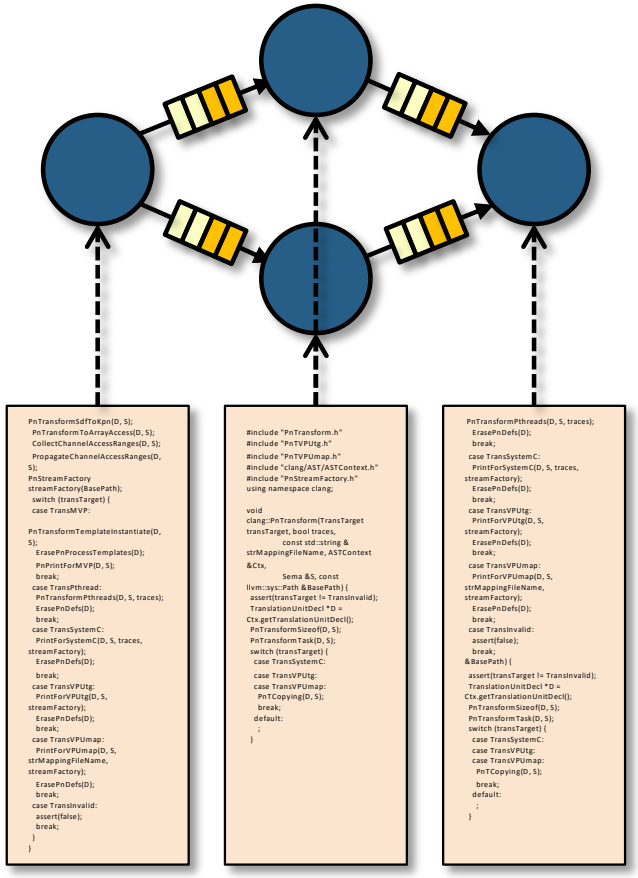
CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Dataflow programming

Dataflow programming

- Graph representation of applications: Long history
  - 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)

Example: Synchronous dataflow graph (SDF)

Def.: An **SDF** is an annotated multi-graph $G=(V,E,W)$. V is the set of actors and $E \subseteq V \times V$ the set of channels. $W = \{w_1,...,w_{|E|}\} \subset \mathbb{N}^3$ is a set of annotations for every channel $e = (a_1,a_2)$, $w_e=(p_e,c_e,d_e)$: p_e is the number of tokens produced by a firing of a_1, c_e the tokens consumed by a_2 and d_e the amount of delay tokens

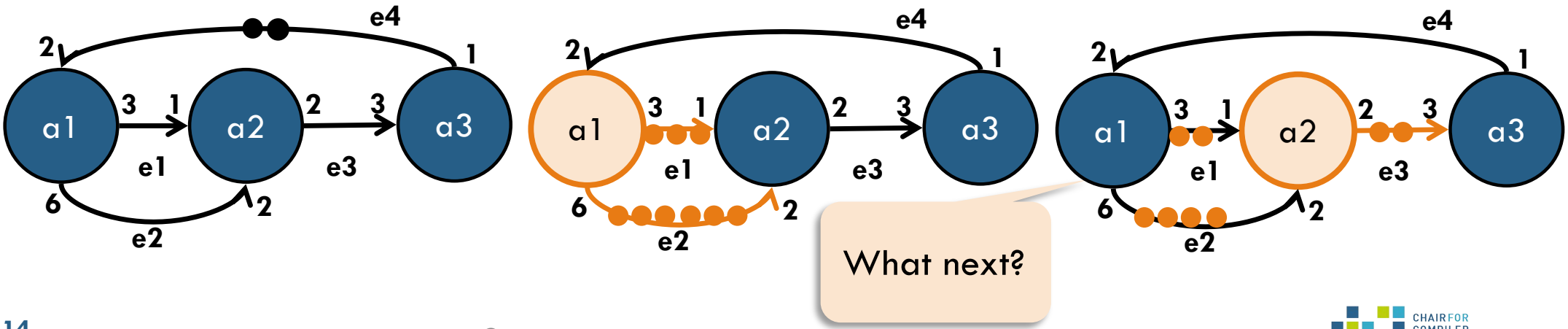
Originally in [Lee'87]

Example: Synchronous dataflow graph (SDF)

Def.: An **SDF** is an annotated multi-graph G=(V,E,W). V is the set of actors and E $\subseteq$ VxV the set of channels. W = $\{w_1,...,w_{|E|}\} \subset N^3$ is a set of annotations for every channel e = (a_1,a_2), $w_e=(p_e,c_e,d_e)$: p_e is the number of tokens produced by a firing of a_1, c_e the tokens consumed by a_2 and d_e the amount of delay tokens

- Example

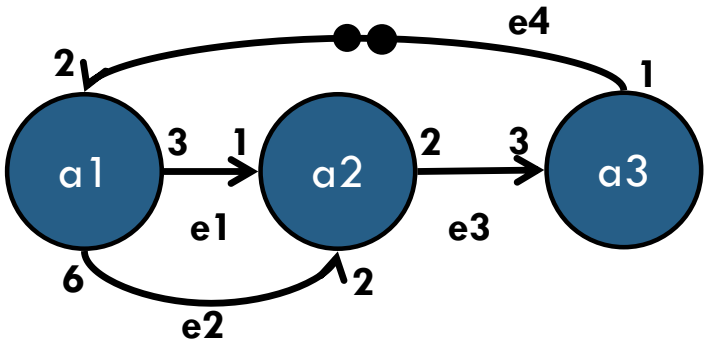
Originally in [Lee'87]



Topology matrix

Def.: Given an SDF G=(V,E,W), its **topology matrix $\mathbf{\Gamma}$** has a row for every channel and a column for every actor. $[\mathbf{\Gamma}_{ik}] = p_{ik} - c_{ik}$ (i.e. the amount of tokens produced minus those consumed by actor k on/from channel i)

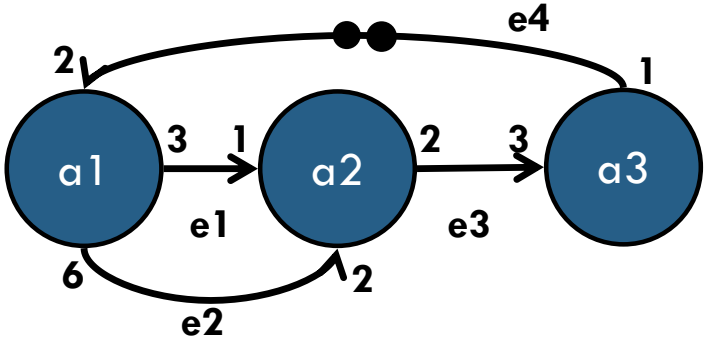
- Example

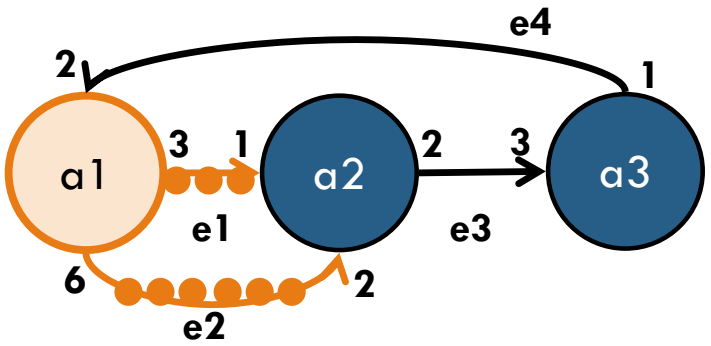


$$\Gamma = \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix}$$

Topology matrix (2)

- **State** of an SDF: tokens in the queues
- The topology matrix can be used to update the state after a *firing*



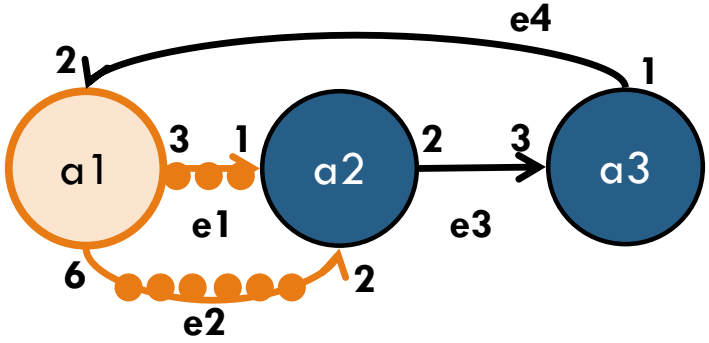


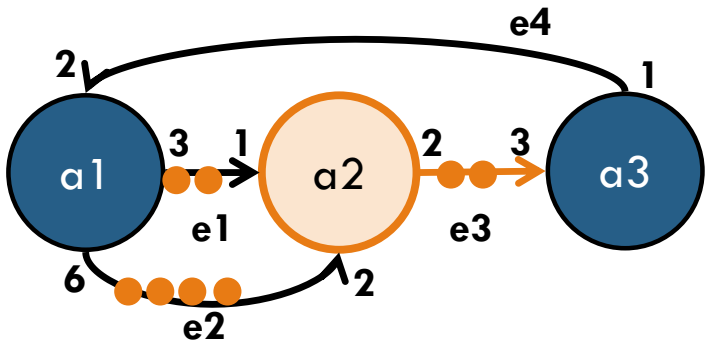
$$\Gamma = \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix}$$

$$s_1 = s_0 + \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 6 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 0 \end{pmatrix}$$

Topology matrix (3)

- **State** of an SDF: tokens in the queues
- The topology matrix can be used to update the state after a *firing*





$$\Gamma = \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix}$$

$$s_2 = s_1 + \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 \\ -2 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 2 \\ 0 \end{pmatrix}$$

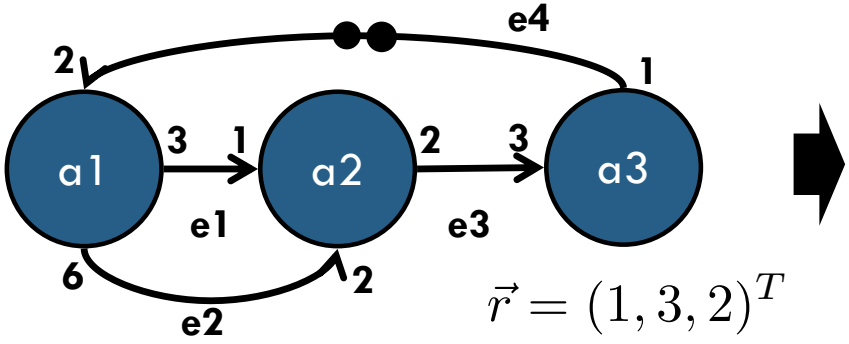
Topology matrix and complete cycles

- Complete cycle: sequence of firing that brings the SDF to the original state

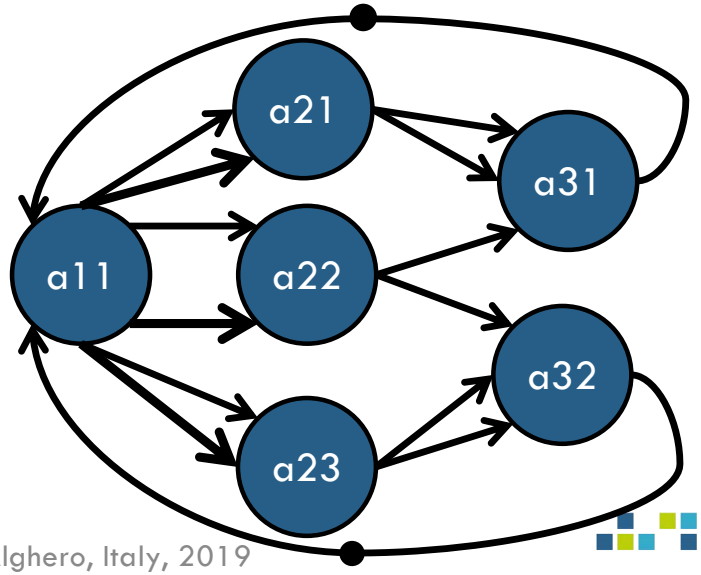
$$s_n = s_0 + \Gamma \cdot (v_1 + v_2 + \cdots + v_n) = s_0$$

$$\Gamma \cdot (v_1 + v_2 + \cdots + v_n) = \Gamma \cdot \vec{r} = 0$$

- What for: Automatic unrolling to express parallelism (preserve semantics)



$$\vec{r} = (1, 3, 2)^T$$



Maximum cycle mean (MCM)

- Unroll SDF is an HSDF
- The MCM relates to the maximum throughput of an HSDF
- Given an HSDF G=(V,E)
  - Let O(G) be the set of all cycles
  - A cycle $C=((v_i,v_{i+1}),(v_{i+2},v_{i+3}),\ldots,(v_{i+n-1}, v_i))$

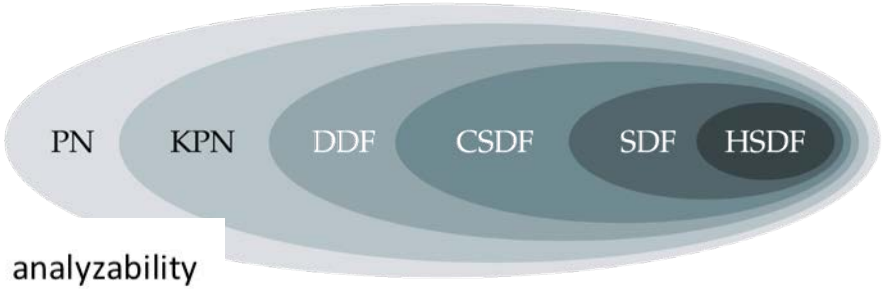
$$MCM(G) = \max_{C \in O(G)} \left(\frac{\sum_{v:(u,v) \vee (v,u) \in C} \rho(v)}{\sum_{e \in C} w(e)} \right)$$

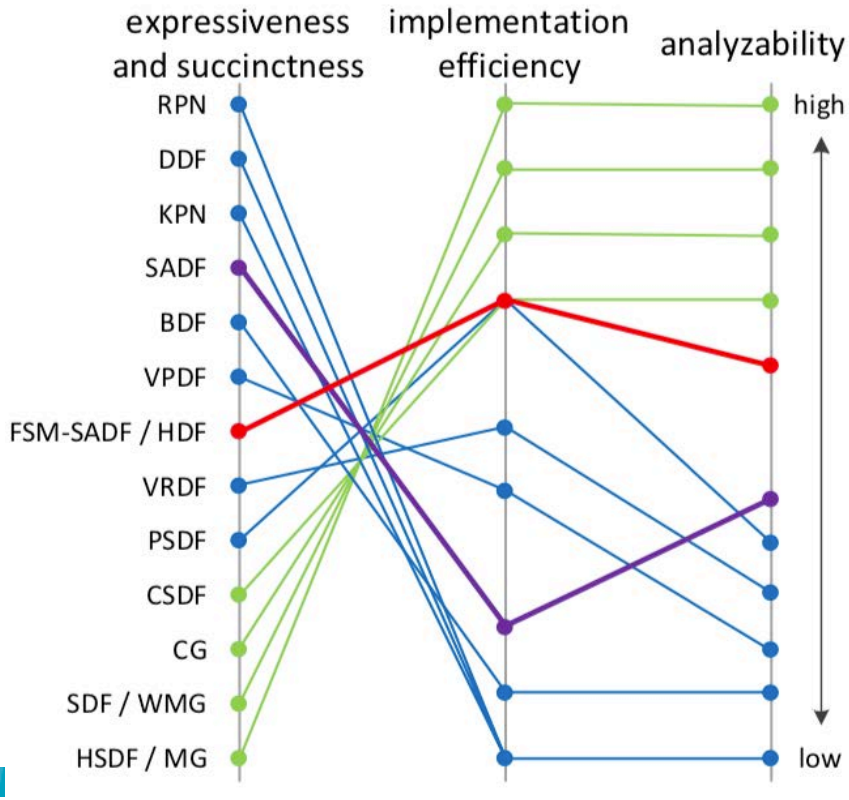
WCET

x delay tokens

Other models

- Trade-off: Expressivity vs analyzability
  - Dynamics: Compromise on analyzability

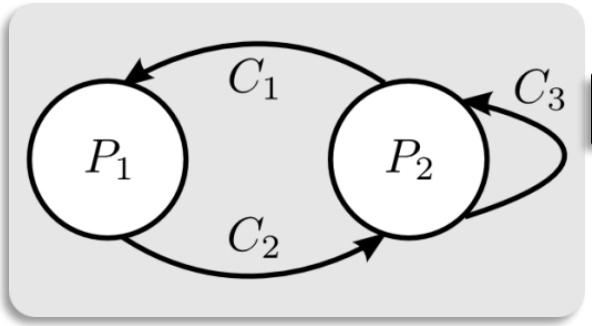


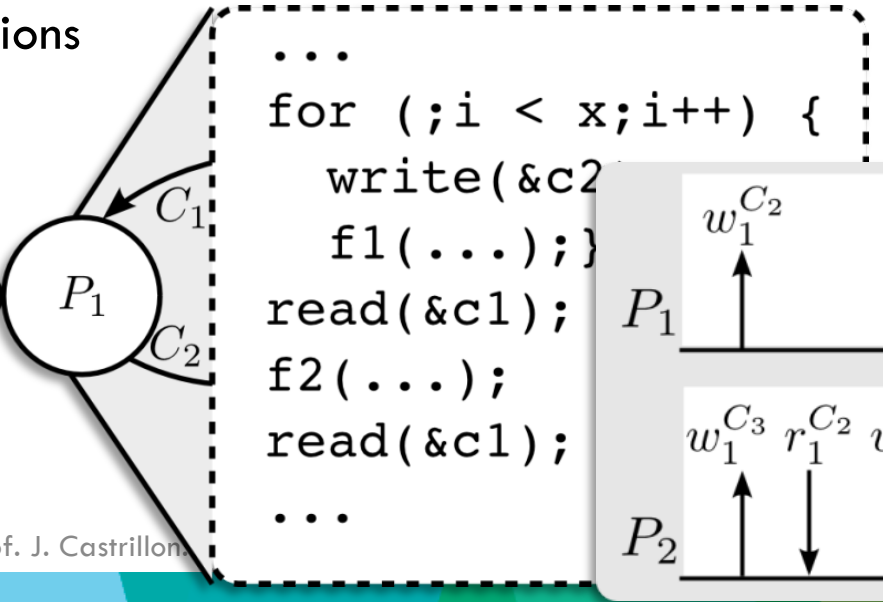


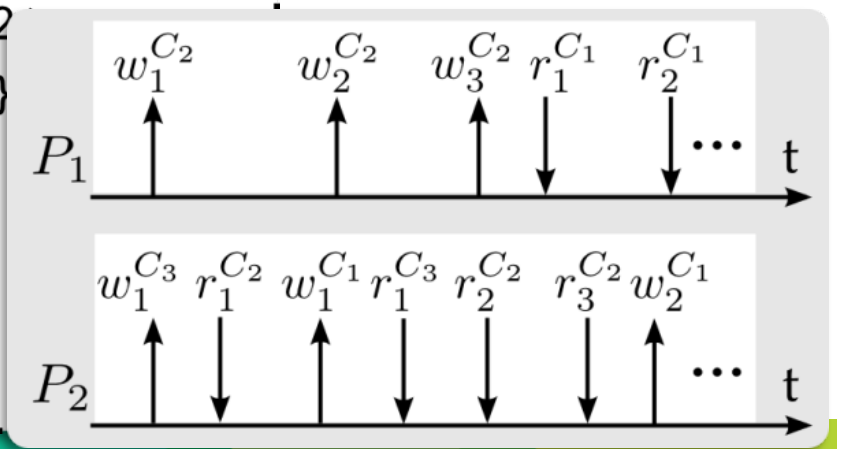
[Stuijk'11]

Dynamic models and language support

- Kahn Process Networks (KPNs)
  - Simple syntax for programming
  - Less model information: more analysis effort
- Via tracing compilers can
  - Identify general communication patterns
  - Exploit event correlations ("dependencies")

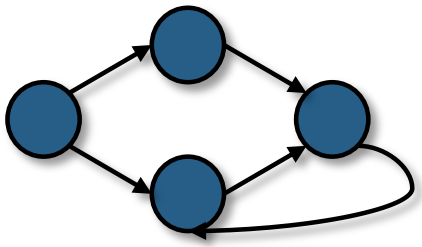






Programming flow: Overview

Dynamic Dataflow Application (CPN)



Non-functional specification

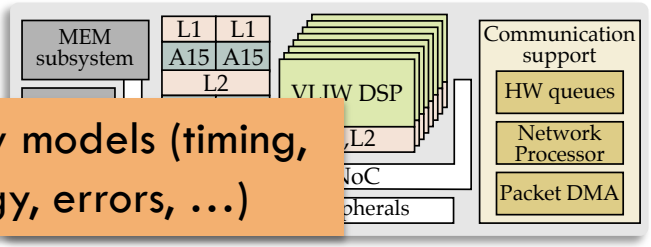
[Springer'14]

Analysis

Synthesis

Code generation

Architecture model



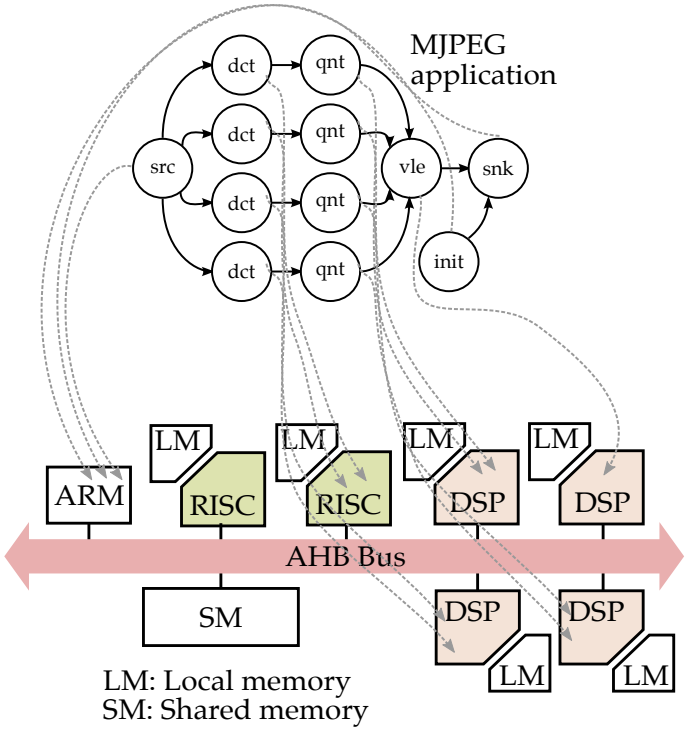
Property models (timing, energy, errors, …)

```
 PNargs_ifft_r.ID = 6U;
    PNargs_ifft_r.PNchannel_freq_coef = f
    PNargs_ifft_r.PNnum_freq_coef = 0U;
    PNargs_ifft_r.PNchannel_time_coef = s
    PNargs_ifft_r.channel = 1;
    sink_left = IPCllmrf_open(3, 1, 1);
    sink_right = IPCllmrf_open(7, 1, 1);
    PNargs_sink.ID = 7U;
    PNargs_sink.PNchannel_in_left = sink_
    PNargs_sink.PNnum_in_left = 0U;
    PNargs_sink.PNchannel_in_right = sink_
    PNargs_sink.PNnum_in_right = 0U;
    taskParams.arg0 = (xdc_UArg)&PNargs_s
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kn
&taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_f
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kn
ft_Templ, &taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_i
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kn
fft_Templ, &taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_s
    taskParams.priority = 1;
```

SILEXICA

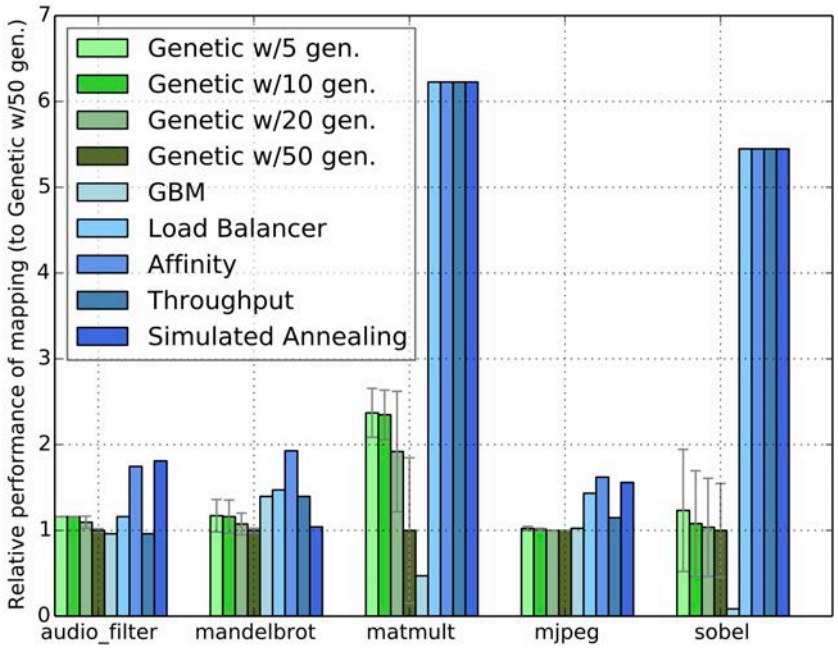
Programming flows

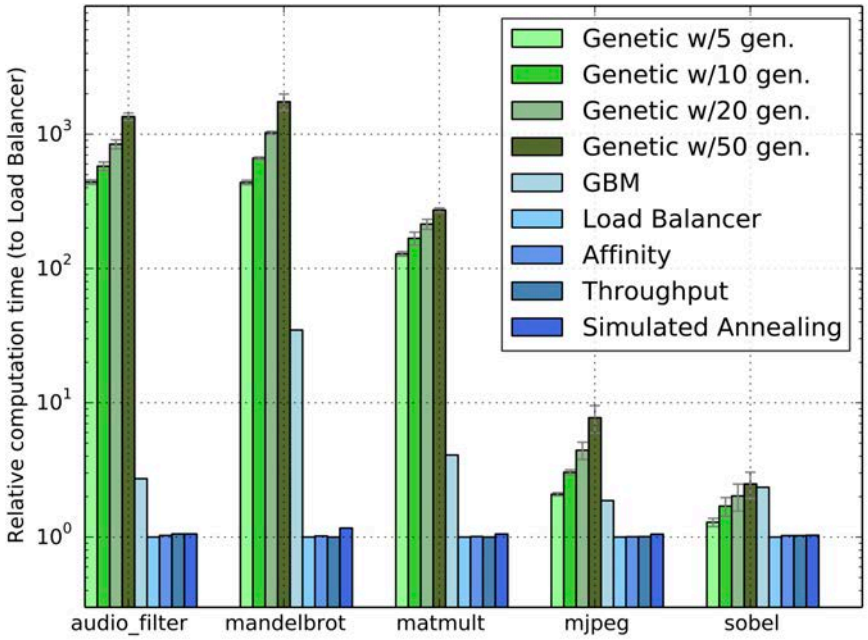
- Many: CAL, DOL, CPN, Daedalus, Ptolemy, CAPH, …
- Information
  - Dataflow model: Rates, states, actions, traces, …
  - Architecture model: Resources, interconnect, costs, …
- Optimization: Mapping application to hardware
  - Multi-objective optimization
  - High-level simulation / cost models



Automatic mapping to heterogeneous many-cores

- Lots of research: Heuristics and meta-heuristics
- Sample results: Effort-quality trade-off





Multimedia apps on TI Keystone II

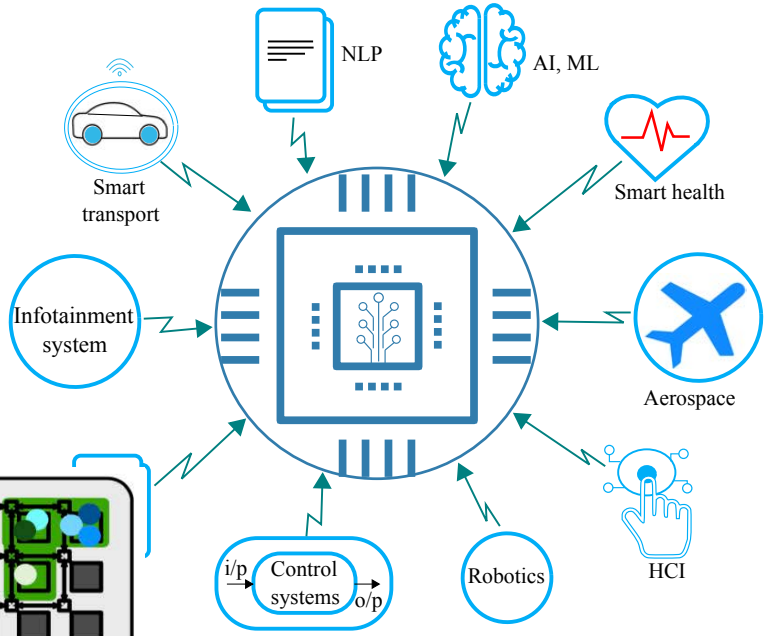
[MCSOC'16]

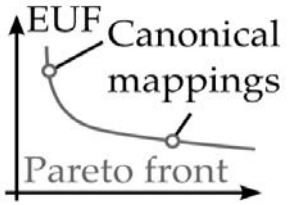
Dataflow and CPSs

- Adaptivity, predictable, multi-app, reactive, …

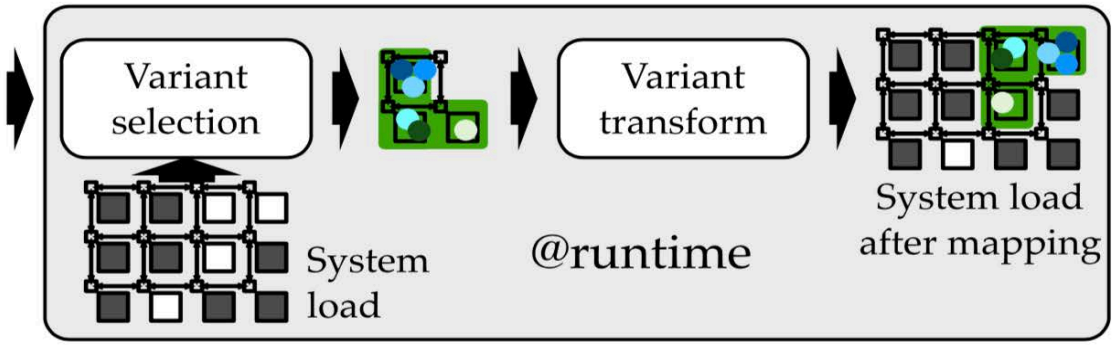
System dynamics: Hybrid mappings

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



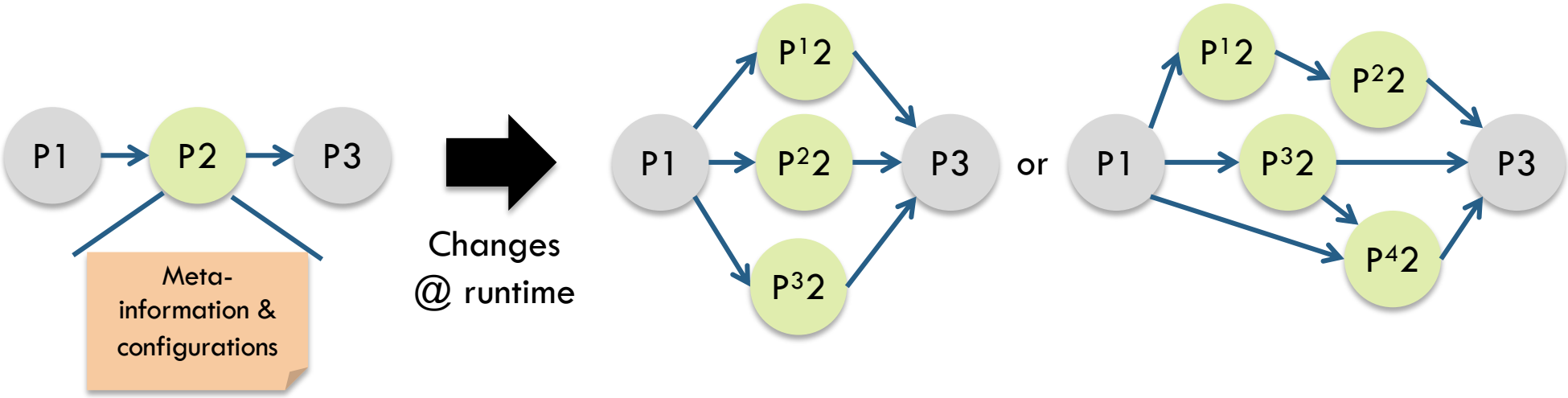


EUF Pareto front @ compile-time



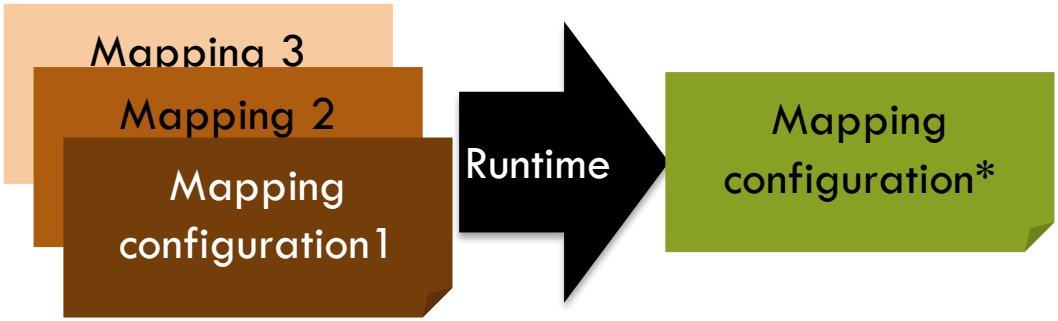
Data-level parallelism: Scalable and adaptive

- Change parallelism from the application specification
- Static code analysis to identify possible transformations (or via annotations)
- Implementation in FIFO library (semantics preserving)
- Similar to AdaPNet or parameterized SDFs [Schor'14, Desnos'13]

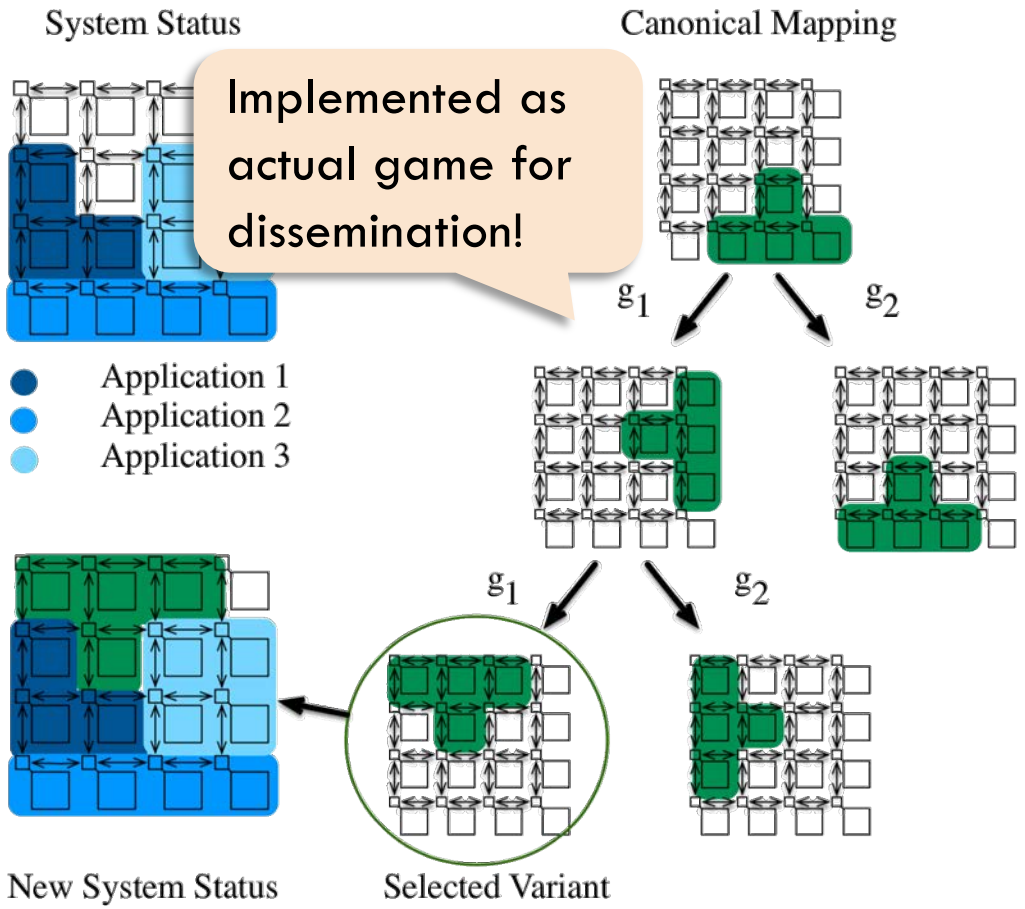


[PARMA-DITAM'18]

Flexible mappings: Generalized Tetris



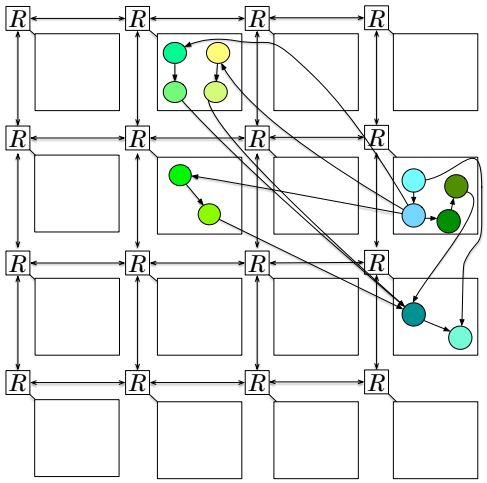
- Given multiple **canonical** configs by compiler, select one at run-time
- Exploit mapping **equivalences** and **similarities**
- Related approach: Produce mapping **constraints at compile-time** [Schwarzer'17]



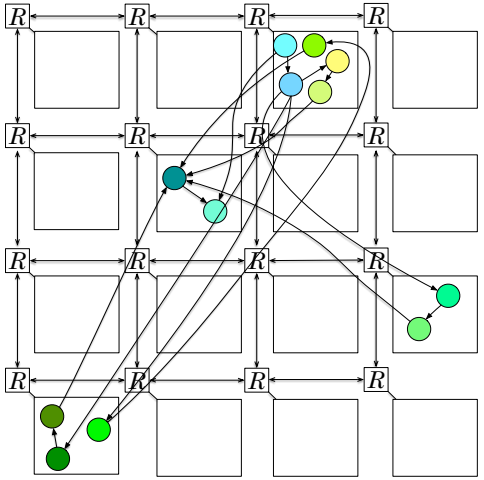
[ACM TACO'17, SCOPES'17b]

Sample symmetries for MJPEG

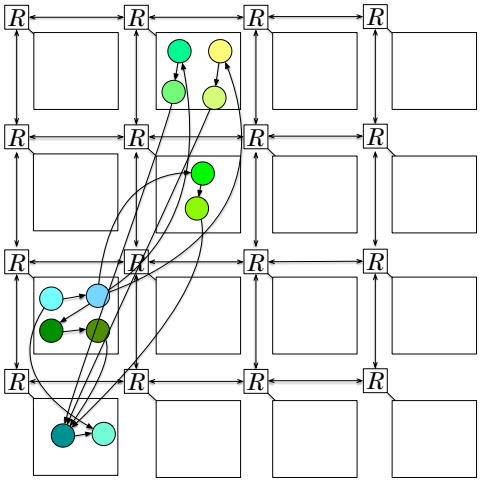
- Multimedia benchmarks on Adapteva (16 Epiphany cores)
- Not always very intuitive



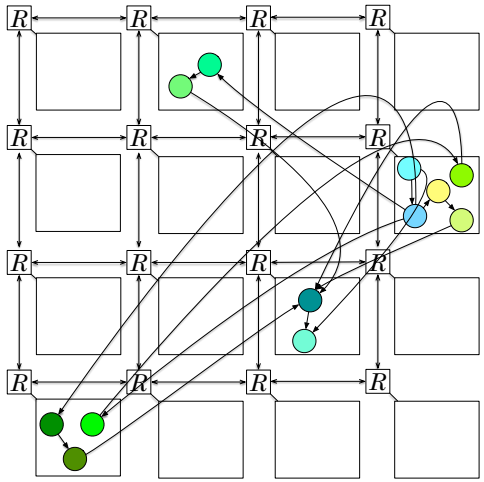
Result “simple”



Best result



Pruned (equiv. to “simple”)

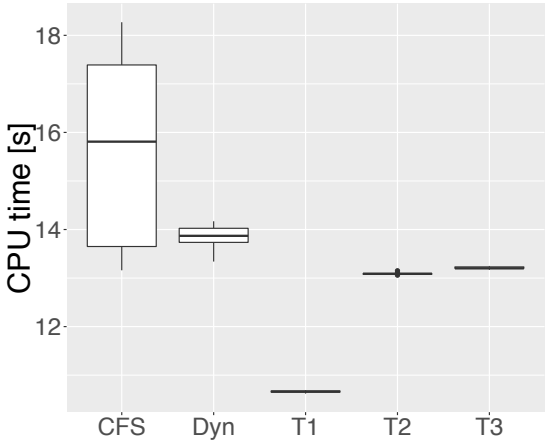


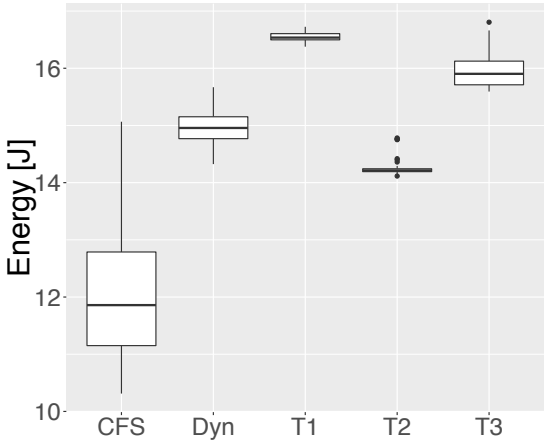
Pruned (equiv. to best)

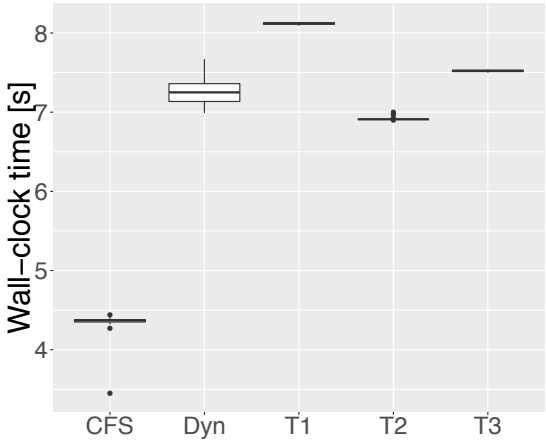
[ACM TACO’17]

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]



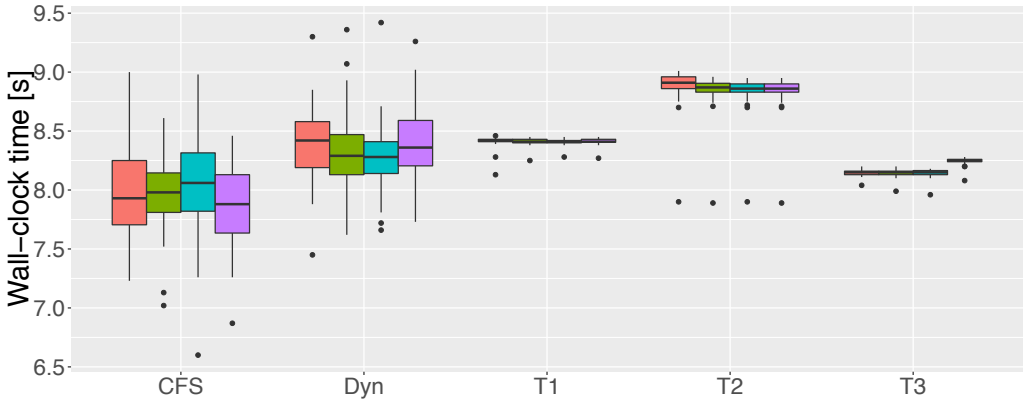


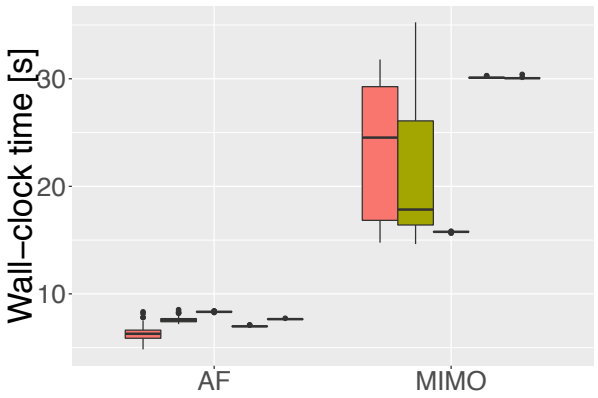


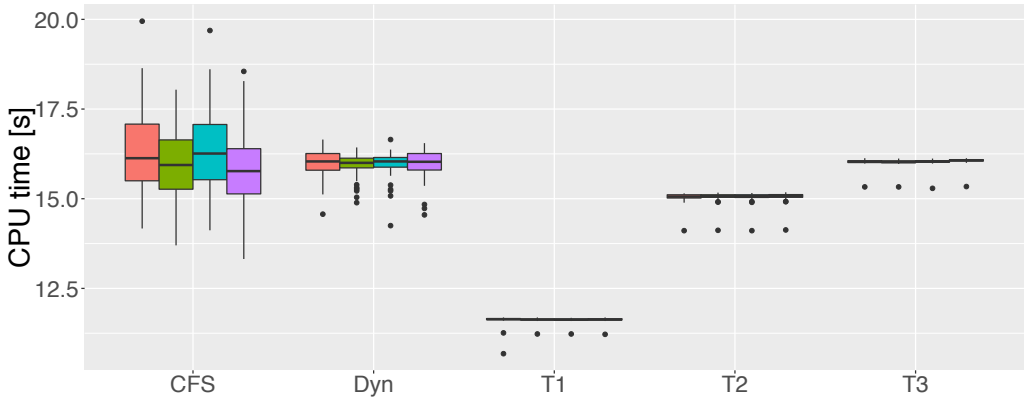
Single AF

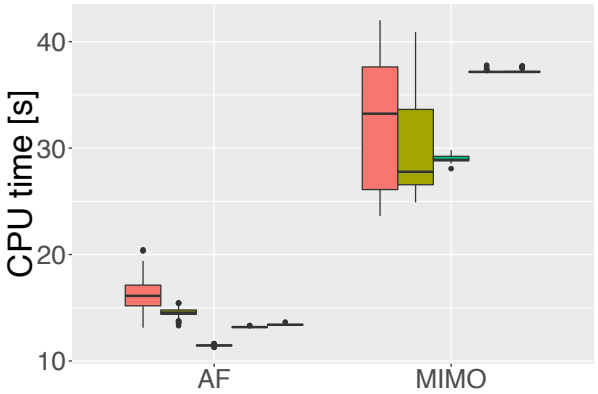
[SCOPES'17b]

Flexible mappings: Multi-application results (1)









[SCOPES'17b]

More predictable performance

Comparable performance to dynamic mapping

Robustness

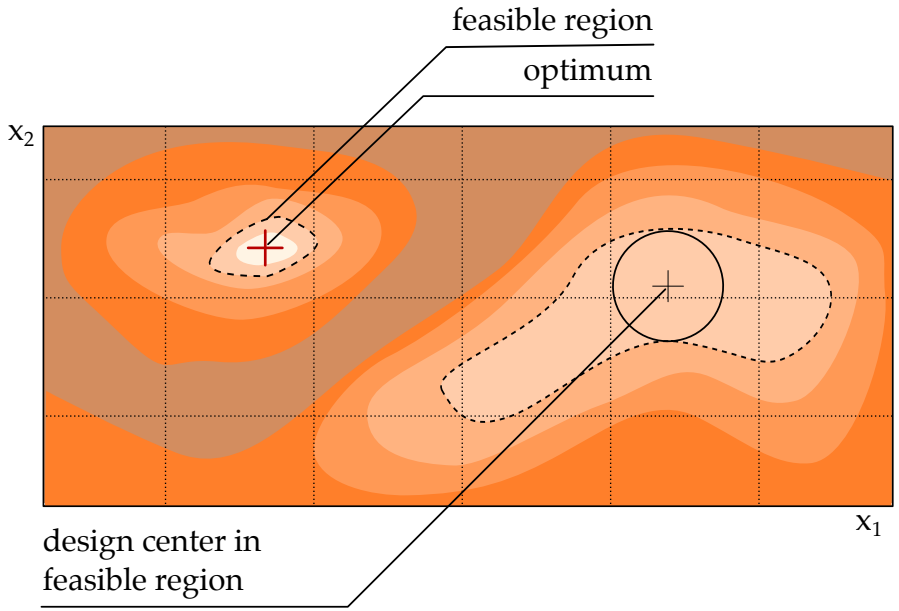
- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions

➔ 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]

Design centering: Algorithmic

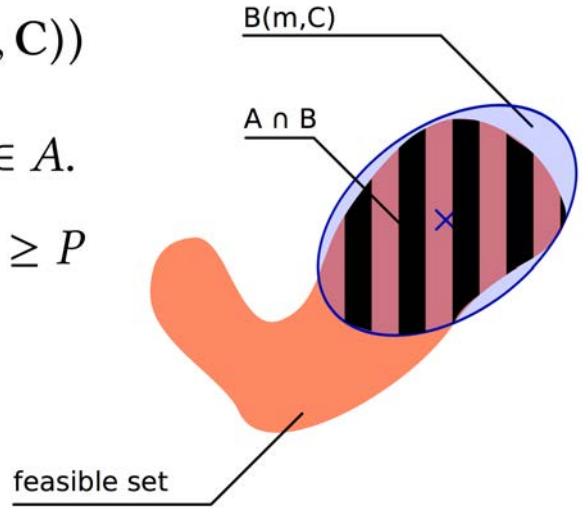
- Intuition: Find the **center** and the **form** of a region, in which parameters deliver a **correct solution**

- Formally
  - A: Set of correct solutions
  - P: Hitting probability
  - L: Generic metric space

$$\max_{B=B(\mathbf{m},\mathrm{C})\in\mathcal{L}_p^n} \mathrm{vol}(B(\mathbf{m},\mathrm{C}))$$

$$\text{s.t.} \quad \mathbf{m} \in A.$$

$$\frac{\mathrm{vol}(A \cap B(\mathbf{m},\mathrm{C}))}{\mathrm{vol}(B(\mathbf{m},\mathrm{C}))} \geq P$$

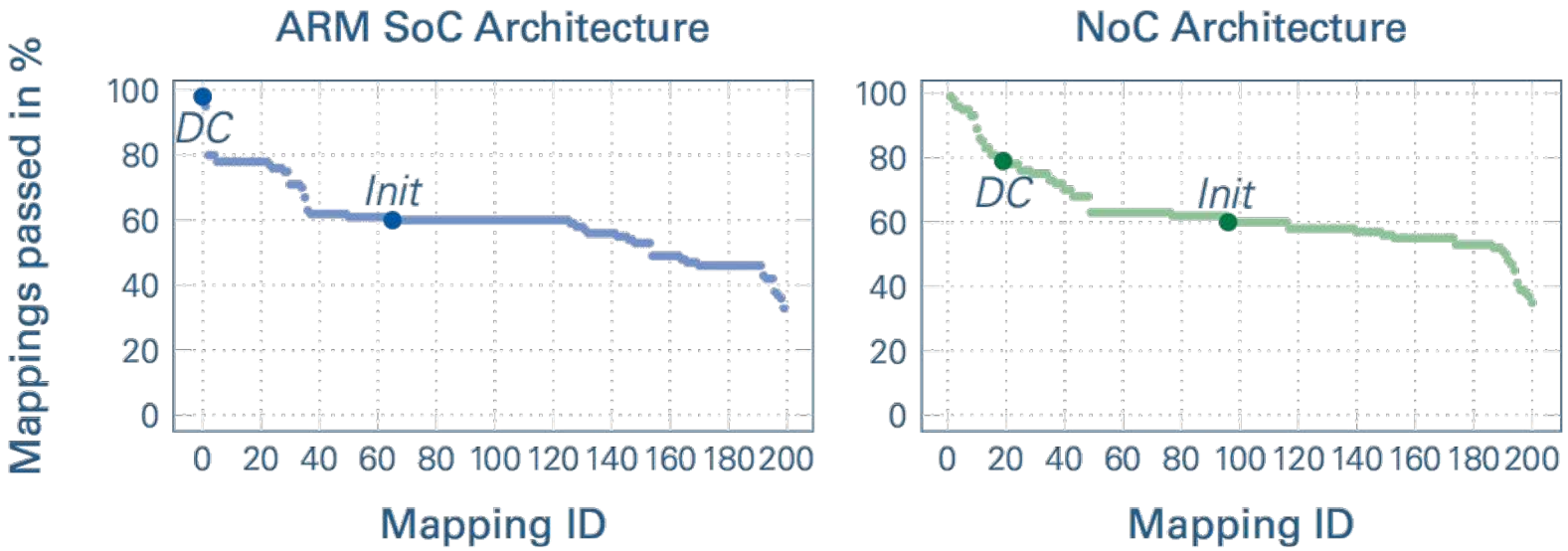


- Searching: Allow annealing (dynamically change P)

[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)

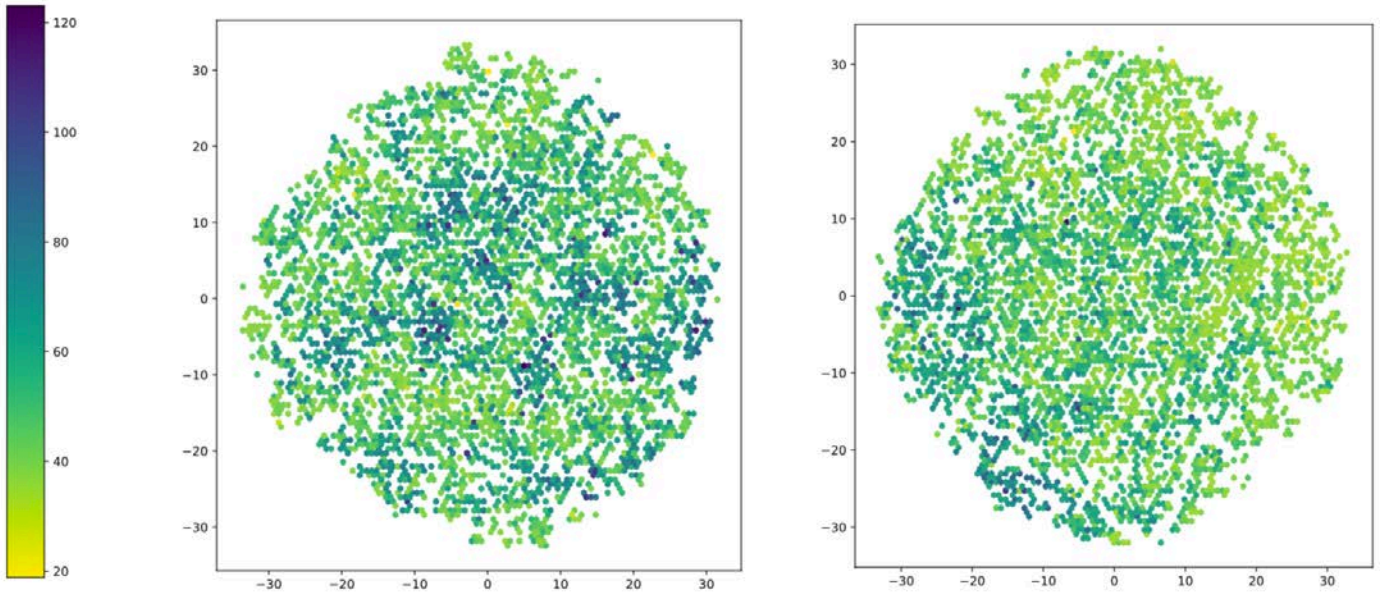


MIMO-OFDM

[SCOPES'17a]

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]

Domain-specific abstractions

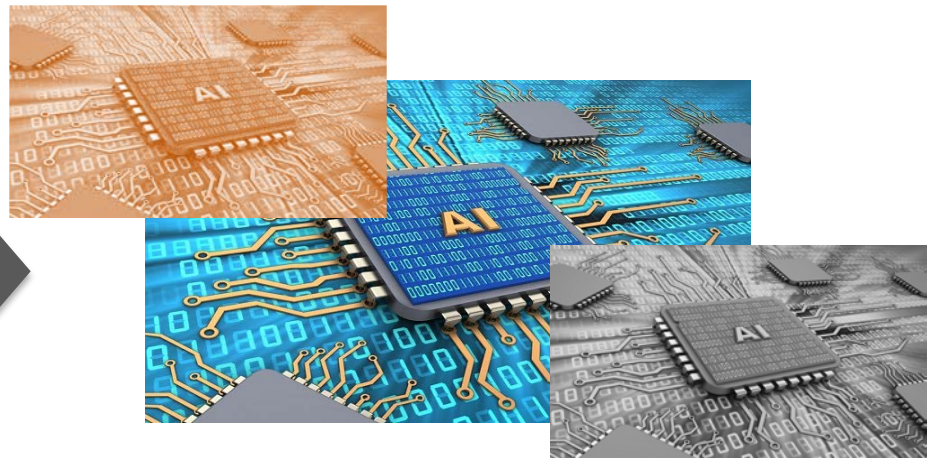
Higher-level Abstractions

- Dataflow: Nice abstraction across domains (maybe too overloaded)
- Need to match domain-specific accelerators with domain-specific abstractions

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

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

[RWDSL'18]

Domain-specific language



https://www.hpcwire.com/2017/04/10/nvidia-responds-google-tpu-benchmarking/

Example: CFDlang

- Origin: Spectral element methods in Computational Fluid Dynamics

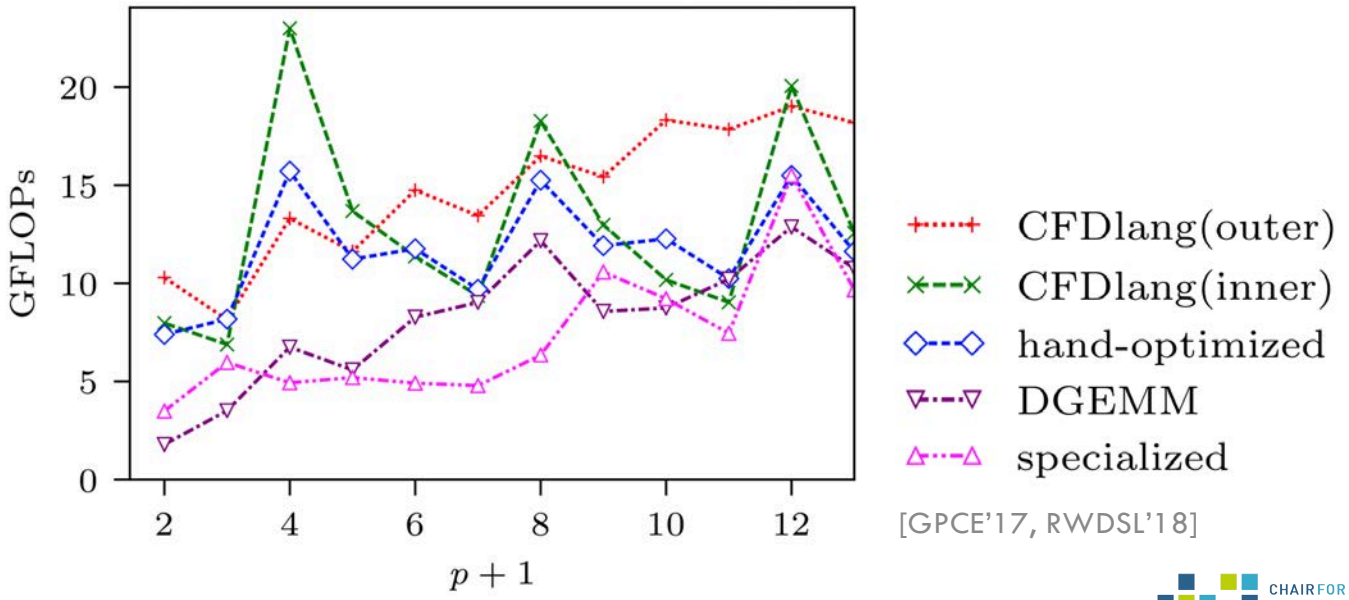
$$\mathbf{v}_e = (\mathbf{A} \otimes \mathbf{A} \otimes \mathbf{A})\, \mathbf{u}_e$$

- Simple expression language, formal semantics, **domain expert optimizations**

```
source =
type matrix    : [mp np]          &
type tensorIN  : [np np np ne]    &
type tensorOUT : [mp mp mp me]    &
                                  &
var input A    : matrix           &
var input u    : tensorIN         &
var input output v  : tensorOUT   &
var input alpha : []              &
var input beta  : []              &
                                  &
v = alpha * (A # A # A # u .
    [[5 8] [3 7] [1 6]]) + beta * v
```

Fortran embedding

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

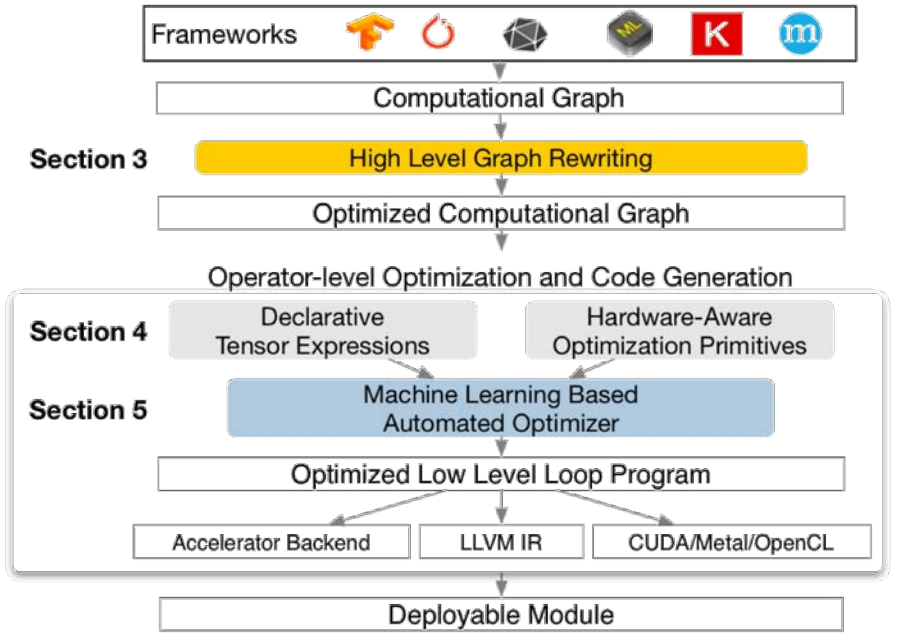


[GPCE'17, RWDSL'18]

Tensor languages (hype): Machine learning

- High-level dataflow graph, low-level tensor operators
- Many existing frameworks, e.g., TVM, Tensor Comprehensions, TensorFlow, …

- Recent work
  - Cross-domain tensor transformations [GPCE'18]
  - Formal semantics for safety checks [Array'19]
    (particularly important for CPS!)
  - Optimization for emerging memories [LCTES'19]



Example flow: TVM [Chen'18]

Correct by construction

- Similar to formal MoCs
  - 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

$[\![\cdot]\!] : \mathit{Context} \to \mathit{Memory} \to (\mathit{list\ of}\ \mathsf{Nat}) \to \mathbb{D}$

$[\![x]\!]\,\Gamma\,\mu\,\bar{\imath} = \mu\,x\,\bar{\imath}$

$[\![(e)]\!]\,\Gamma\,\mu\,\bar{\imath} = [\![e]\!]\,\Gamma\,\mu\,\bar{\imath}$

$[\![\mathsf{add}\ e_0\ e_1]\!]\,\Gamma\,\mu\,\bar{\imath} = [\![e_0]\!]\,\Gamma\,\mu\,\bar{\imath} + [\![e_1]\!]\,\Gamma\,\mu\,\bar{\imath}$

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

$[\![\mathsf{prod}\ e_0\ e_1]\!]\,\Gamma\,\mu\,(\bar{\imath}_0 \# \bar{\imath}_1) = [\![e_0]\!]\,\Gamma\,\mu\,\bar{\imath}_0 \cdot [\![e_1]\!]\,\Gamma\,\mu\,\bar{\imath}_1\,,$
if $\mathit{rank}_\Gamma(e_0) = \mathit{length}(\bar{\imath}_0)$ and $\mathit{rank}_\Gamma(e_1) = \mathit{length}(\bar{\imath}_1)$

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

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

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

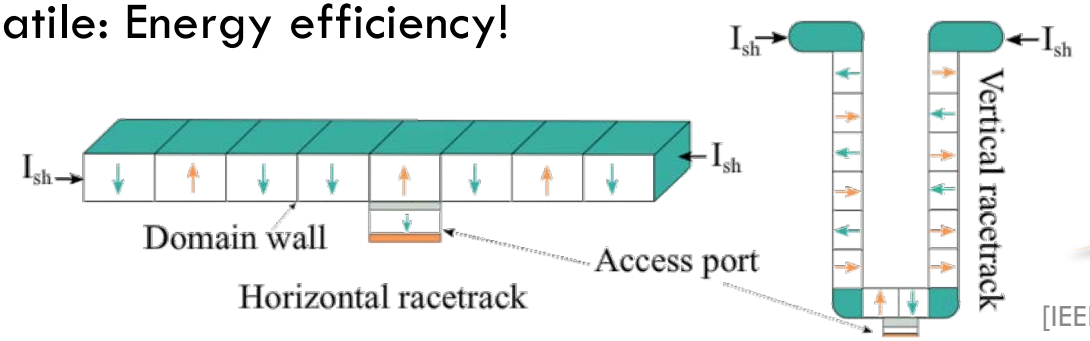
$[\![\mathsf{proj}\ i\ m\ e]\!]\,\Gamma\,\mu\,[j_1,\ldots,j_{i-1},j_i\ldots,j_k] =$
$[\![e]\!]\,\Gamma\,\mu\,[j_1,\ldots,j_{i-1},m,j_i,\ldots,j_k]$

[Array'19] $[\![\mathsf{red}_+\ i\ e]\!]\,\Gamma\,\mu\,[j_1,\ldots,j_{i-1},j_i,\ldots,j_k] = \sum_{m=1}^{n} [\![e]\!]\,\Gamma\,\mu\,[j_1,\ldots,j_{i-1},m,j_i,\ldots,j_k]$, if $\mathit{type}_\Gamma(e) = [n_1,\ldots,n_{i-1},n,n_{i+1},\ldots,n_{k+1}]$

Emerging architectures

- Lots of research on tensor computations on tensor accelerators! e.g., [Kim'19]
- Recently looking into emerging memory architectures
  - Hybrid architectures STT/Re-RAM [TVLSI'18, TVLSI'19]
  - Racetrack memories

- Racetrack memories: Predicted extreme density at low latency
  - Multiple bits stored in nano-wires with magnetic domains
  - Non-volatile: Energy efficiency!

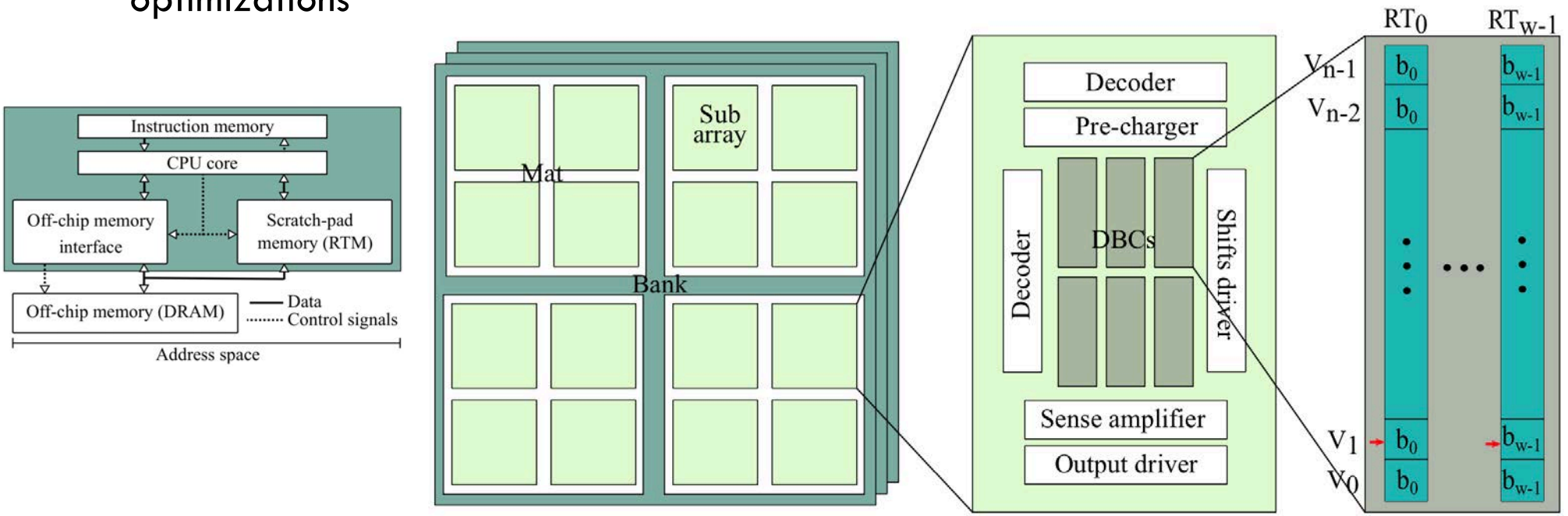


[IEEE CAL'19]

Need to exploit sequentially on top of locality

Architecture and data layout optimization

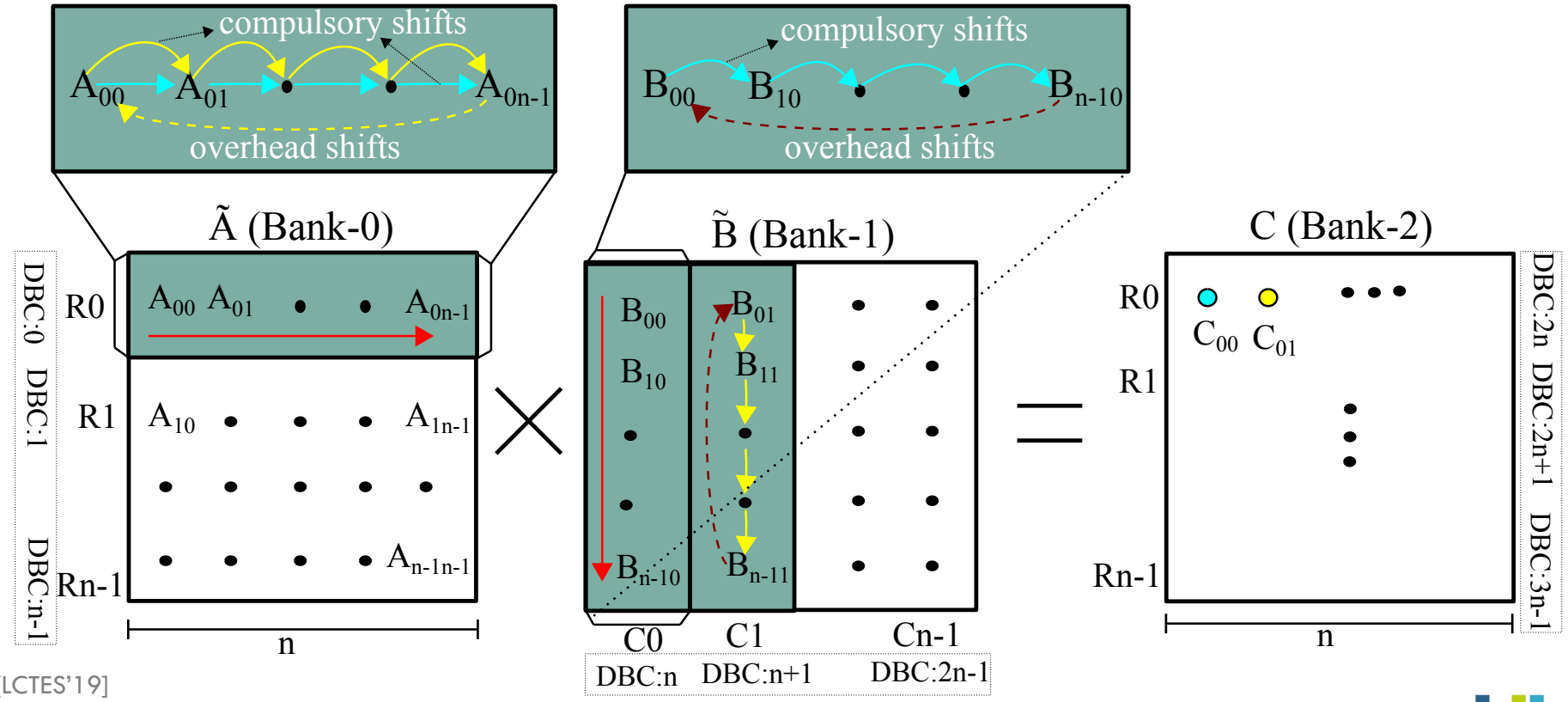
- Architecture – software co-optimization
  - Embedded system for inference: RTM as scratchpad with pre-shifting and other optimizations



[LCTES'19]

Architecture and data layout optimization

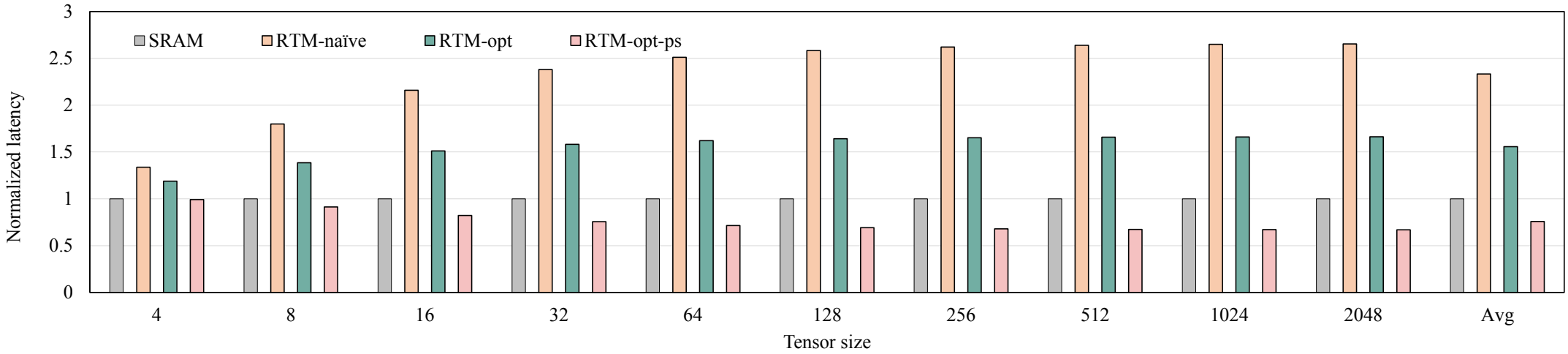
- Data-layout: Reduce the number of shifts



[LCTES'19]

Latency comparison vs SRAM

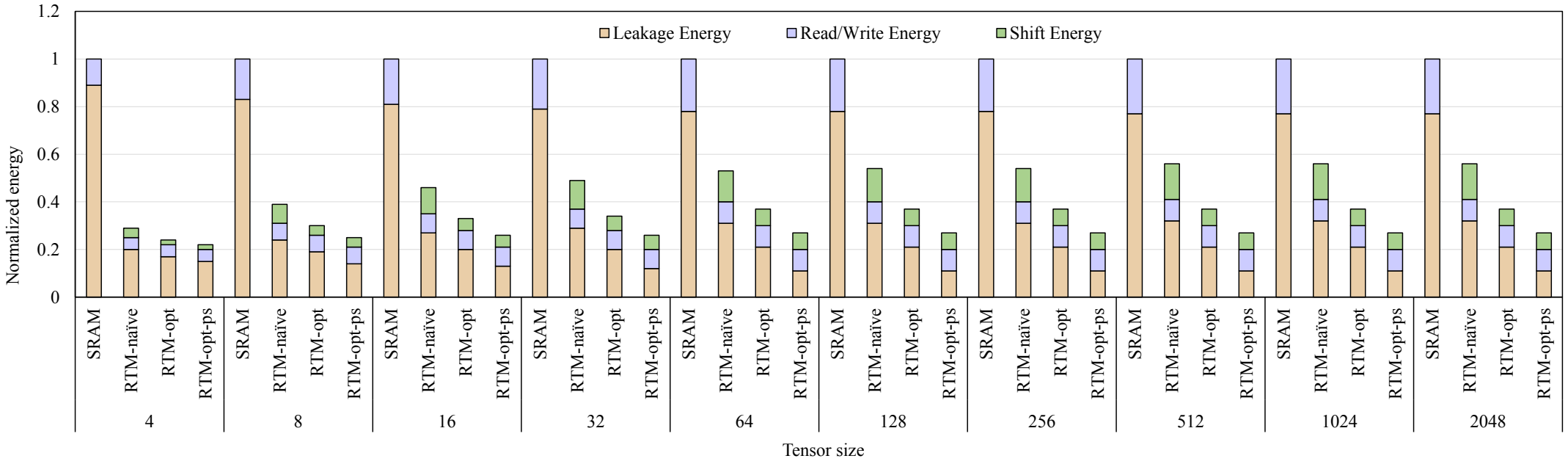
- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)



[LCTES'19]

Energy comparison vs SRAM

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



[LCTES'19]

Summary

Summary

- Principled methodologies for programming are a must! – *preaching to the choir*
  - 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.

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

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

[Lee'87] E.A. Lee and D. G. Messerschmitt. "Synchronous data flow." Proceedings of the IEEE 1987

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

[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.

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

[Stuijk'11] Stuijk, S.; Geilen, M.; Theelen, B. & Basten, T. "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications", SAMOS'11 404-411

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

[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.

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

[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.

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

[Schwarzer'17] T. Schwarzer, et al. "Symmetry-eliminating design space exploration for hybrid application mapping on many-core architectures." IEEE TCAD 37.2 (2017): 297-310.

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

[CyPhy'19] M. Lohstroh, et al. "Reactors: A Deterministic Model for Composable Reactive Systems" Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy 2019) Oct 2019.

[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, N. A. Rink, J. Castrillon, et al. "Towards Compositional and Generative Tensor Optimizations". GPCE'17, Oct. 2017, pp. 169–175.

[Chen'18] Chen, Tianqi, et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning." 13th OSDI, 2018.

[GPCE'18] A. Susungi, N. A. Rink, A. Cohen, J. Castrillon, and C. Tadonki. "Meta-programming for cross-domain tensor optimizations". GPCE'18, Nov. 2018, pp. 79–92.

[Array'19] N.A. Rink, N. A. and J. Castrillon. "TeIL: a type-safe imperative Tensor Intermediate Language", ARRAY, ACM, 2019, pp. 57-68

[Kim'19] J. Kim, et al. "A code generator for high-performance tensor contractions on GPUs." Proceedings of the Symposium on Code Generation and Optimization. IEEE Press, 2019.

[IEEE CAL'19] Asif Ali Khan, Fazal Hameed, Robin Bläsing, Stuart Parkin, Jeronimo 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, Jun 2019, pp. 5-18