CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Parallel programming methodologies for manycores

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

NeXtream Solution Seminar & Silexica Technology Workshop

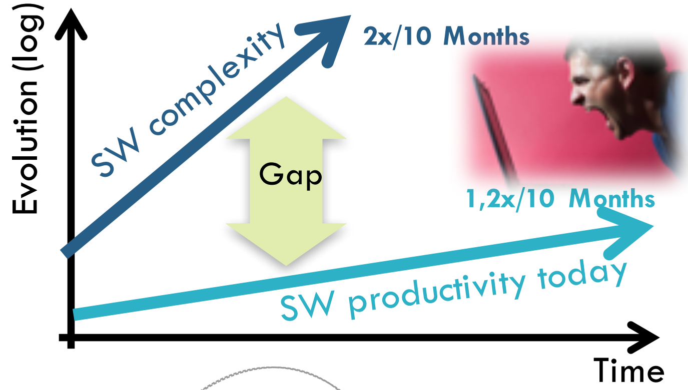
October 17, 2018

TECHNISCHE UNIVERSITÄT DRESDEN DRESDEN concept DFG WR WISSENSCHAFTSRAT

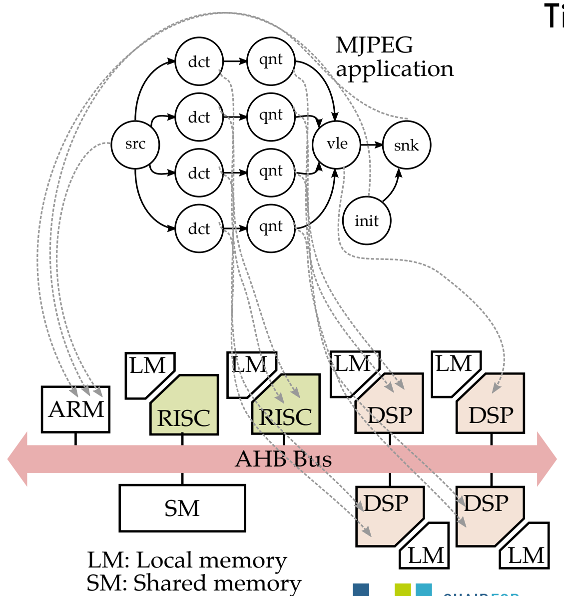
History and context

The software productivity gap

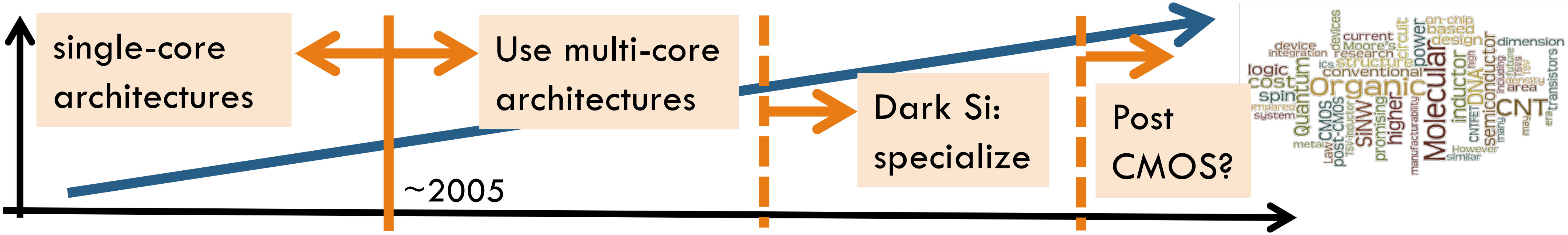
- Bring complex SW to ever-increasing complex HW
  - Important: Inflection point in comp-arch (2005)



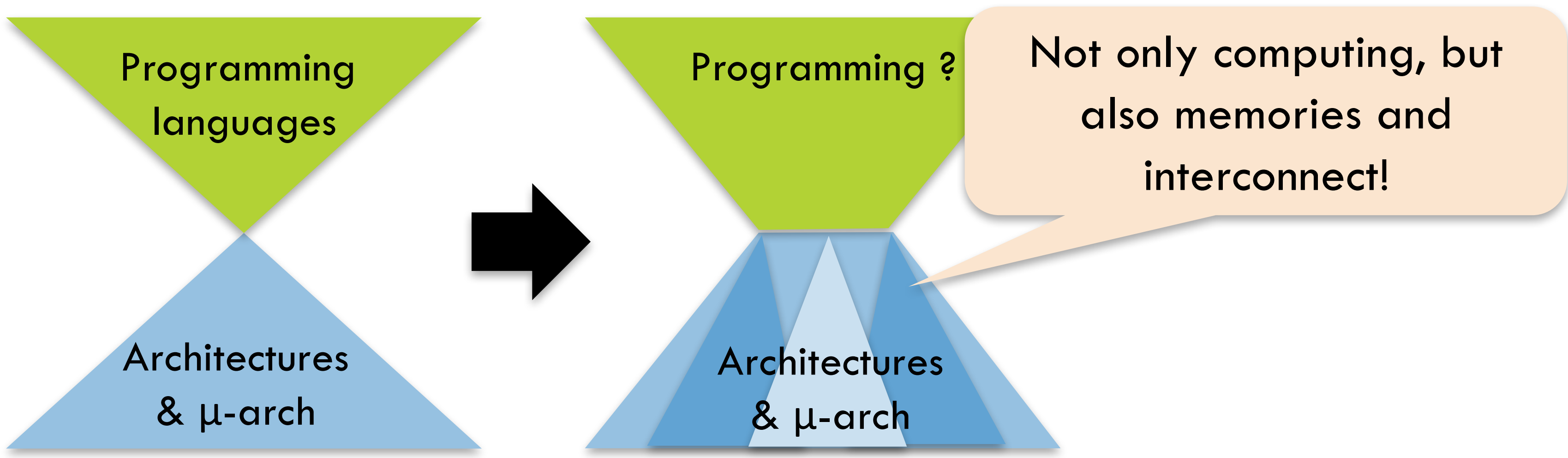
- Pioneering work that led to **SLX**
  - Auto-parallelization [Ceng08]
  - General software synthesis approach [Castrill11]
  - Mapping and scheduling [Castrill13-13a]
  - Debugging [Castrill11a, Murillo14]
  - Cost modeling [Oden13, Eusse16]



Inflection points and programming



- The programming interface continues to broadens as hardware evolves



Single core

(het.) Multi-processors

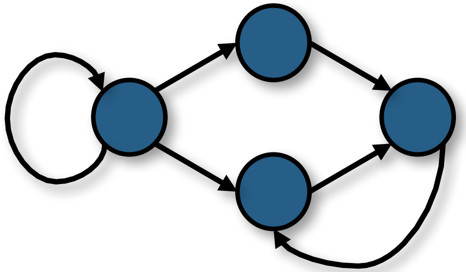
Inflection points and programming

- Programming parallel heterogeneous systems
- Domain-specific languages
- Tools and methodologies for Post-CMOS systems
- Optimization: Performance, energy efficiency & resilience

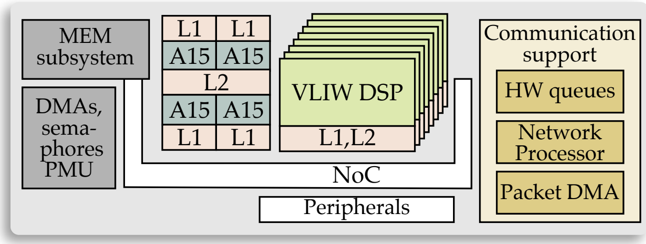
Parallel programming

Programming flow: Overview

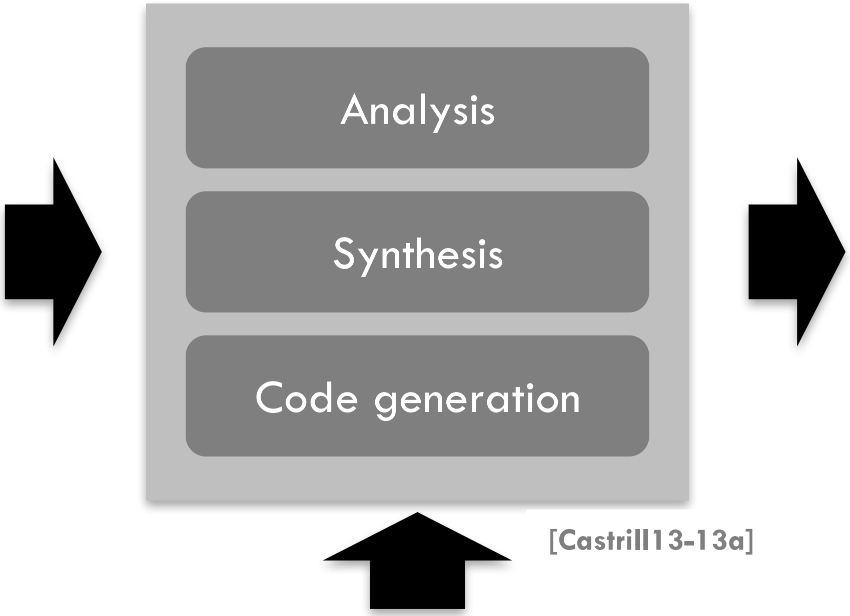
Application



Architecture model



Non-functional specification



[Castrill13-13a]

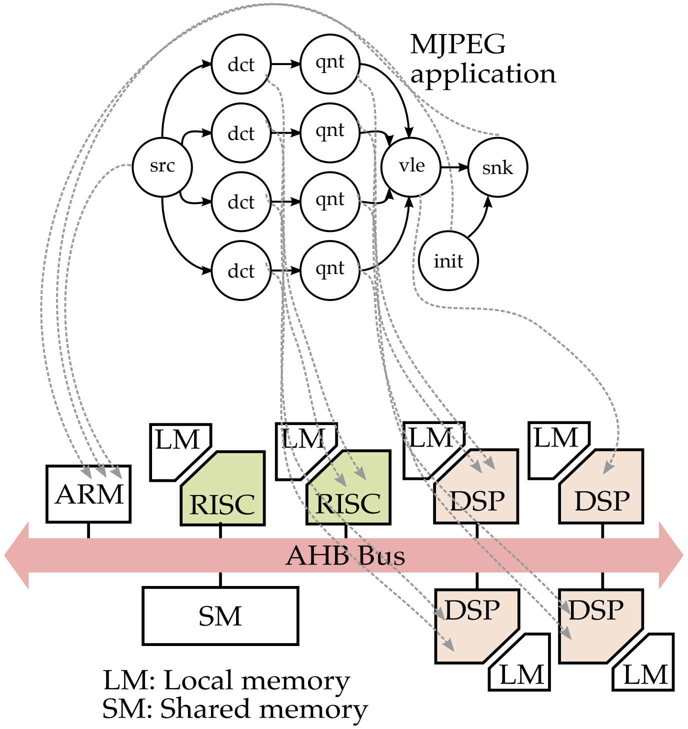
Property models (timing, energy, error, …)

```
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;
```

Compilation for parallel & heterogeneous systems

- Understood
  - Language, compiler and mapping algorithms
  - Hardware modeling, performance estimation
  - Code generation, runtime HW/SW for heterogeneous multicores

- Current work
  - Symmetries and language extensions for **scalability**
  - Symmetries and runtimes for **adaptivity**
  - Design centering for **robustness**



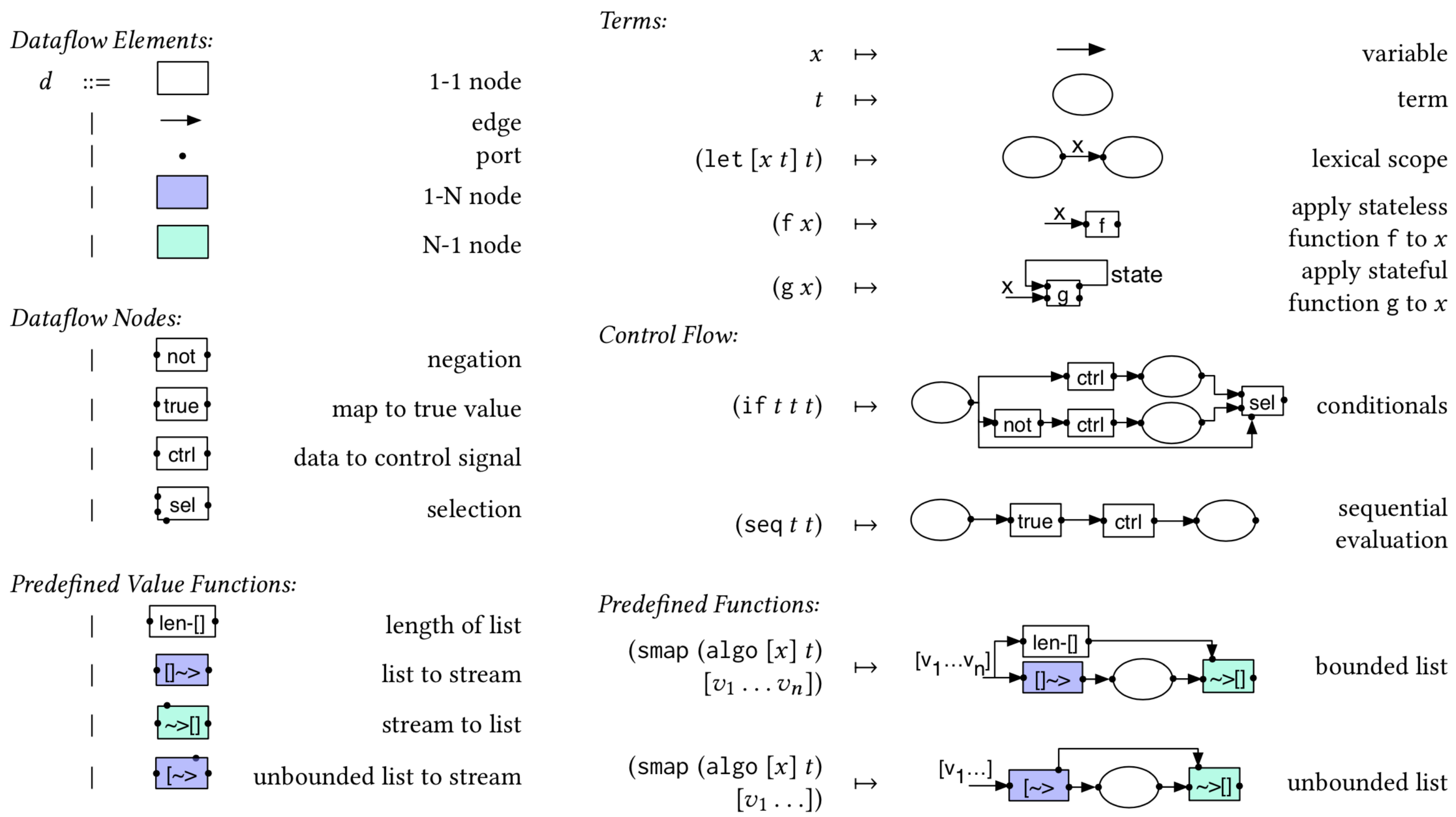
Higher-level abstraction for dataflow

- Functional abstraction for implicitly describing the graph
  - Not so much about syntax: Clojure, Haskell, Rust, Java, …

```
(ohua :import [web.translation]) ; import the namespace where the used
                                 ; functions are defined
(defn translate [server-port]
  (ohua (let [[cnn req] (read-socket (accept (open server-port)))
              [_ file-name _ lang] (parse-request req)
              [^List content length] (if (exists? file-name)
                                         (load-file-from-disk file-name)
                                         (generate-reply "No such file."))
              ^String word (decompose content) ; poor man's translation
              _ (log "translating word")
              updated-content (collect length (translate word lang))]
          (reply cnn (compose length updated-content))))
```

[Ertel18a]

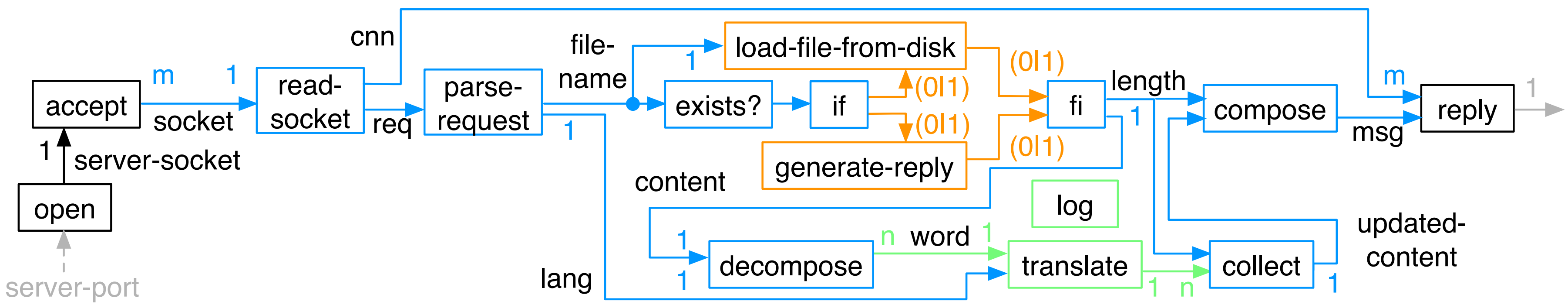
Functional to dataflow



[Ertel18a]

Functional to dataflow

```
(ohua :import [web.translation]) ; import the namespace where the used
                                 ; functions are defined
(defn translate [server-port]
  (ohua (let [[cnn req] (read-socket (accept (open server-port)))
              [_ file-name _ lang] (parse-request req)
              [^List content length] (if (exists? file-name)
                                        (load-file-from-disk file-name)
                                        (generate-reply "No such file."))
              ^String word (decompose content) ; poor man's translation
              _ (log "translating word")
              updated-content (collect length (translate word lang))]
          (reply cnn (compose length updated-content))))
```



[Ertel18a]

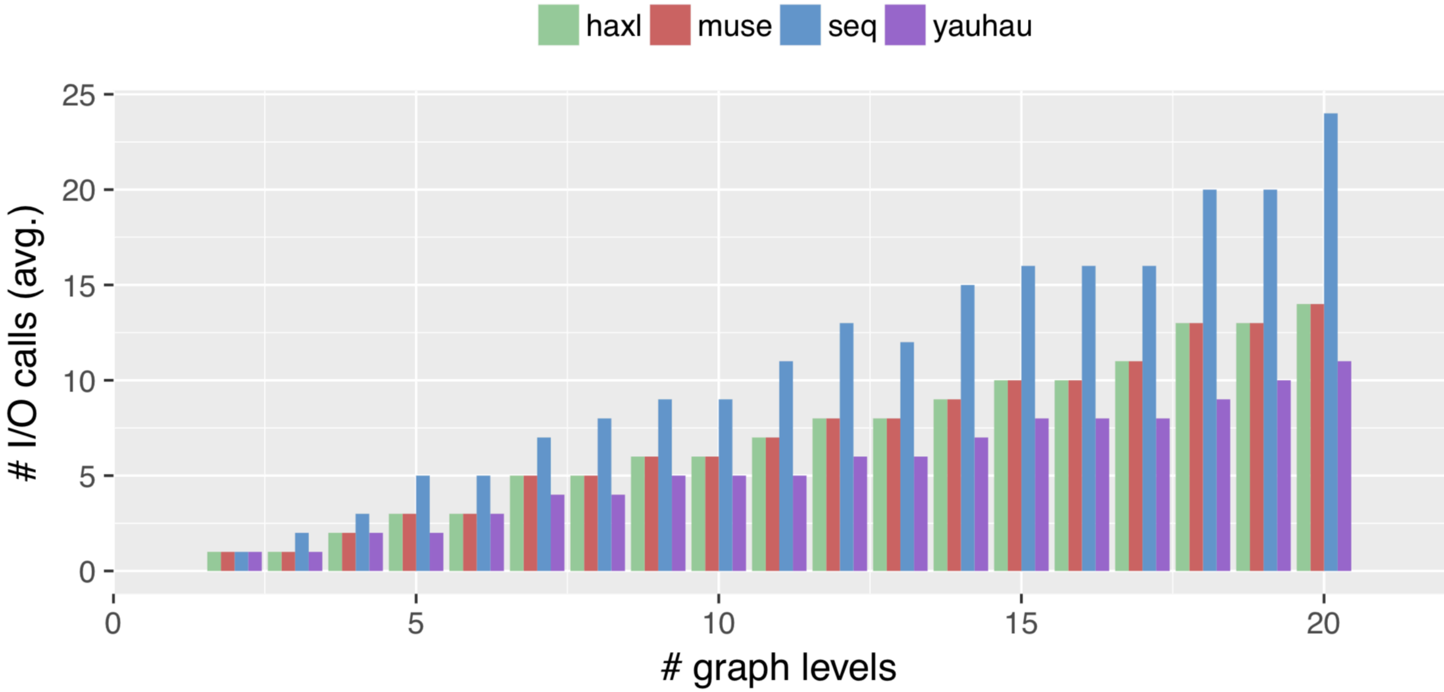
I/O optimization in uServices

- Functional abstraction: amenable for micro-service architectures
- Problem
  - Modularity at odds with performance due to repeated I/O calls
  - Currently solved via complex **applicative functors** (Facebook)
- Develop simple dataflow rewrites to optimize I/O batching

[Ertel18b]

Second use-case: I/O optimization in uServices

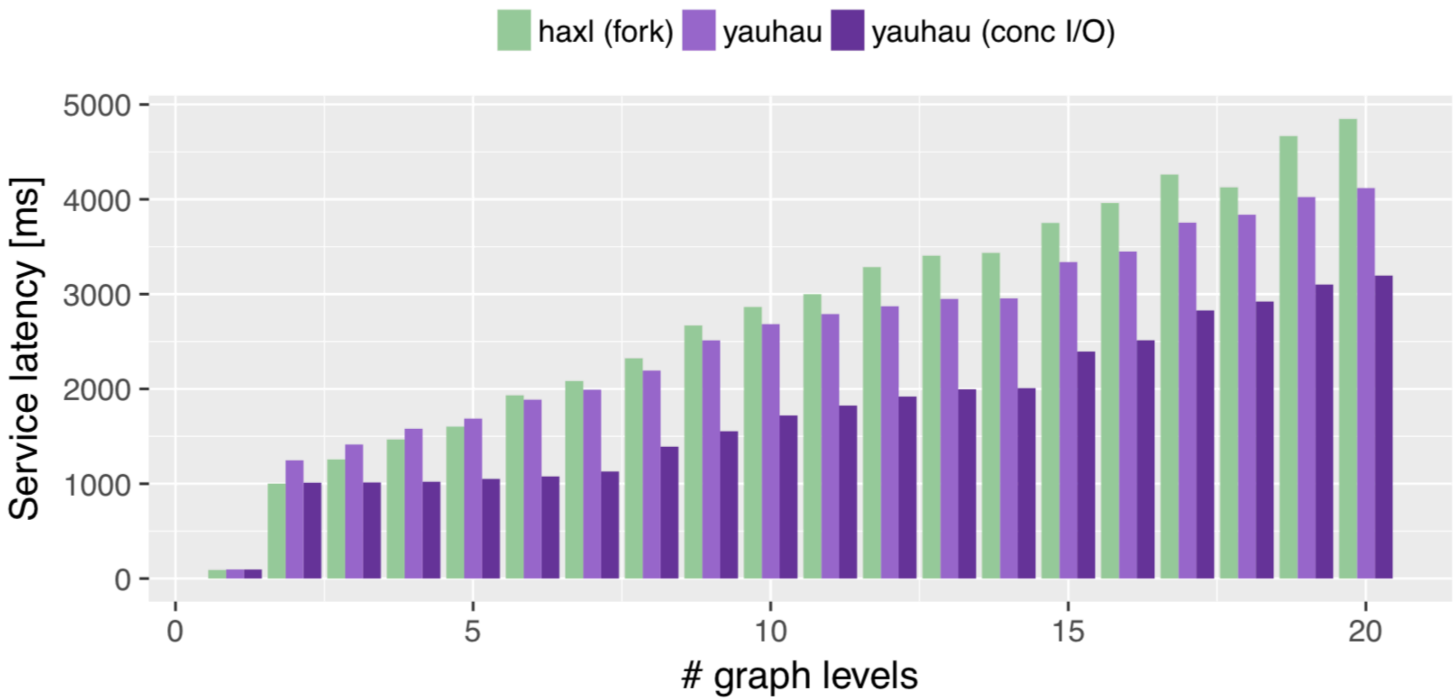
- Functional abstraction: amenable for micro-service architectures
- Develop simple dataflow rewrites to optimize I/O batching



[Ertel18b]

Second use-case: I/O optimization in uServices

- Functional abstraction: amenable for micro-service architectures
- Develop simple dataflow rewrites to optimize I/O batching



[Ertel18b]

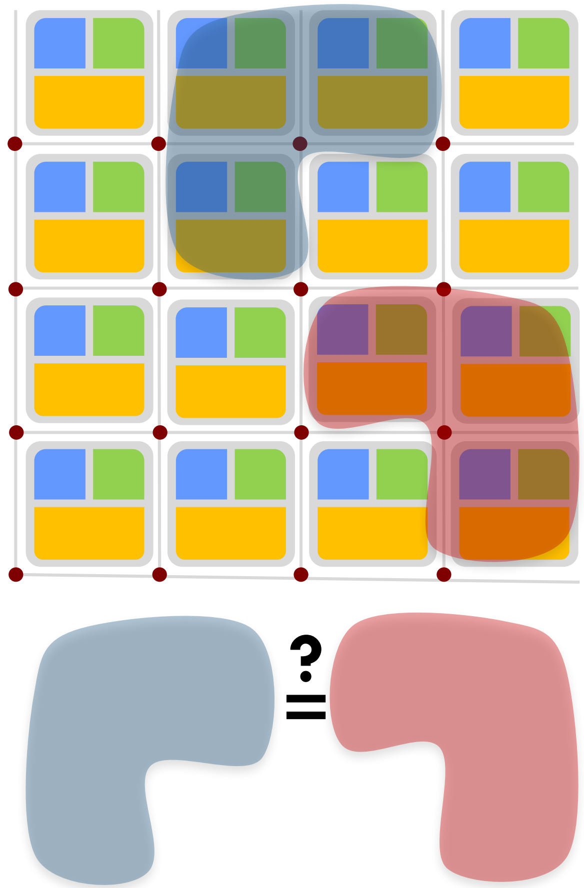
Adaptivity

- Originally in embedded domain: Applications meant to execute alone
- Today
  - Multiple applications sharing resources
  - Available resources unpredictable at load time
  - Design space to large for exploration at running time
  - But: You still want **time-predictability**
- Strategy
  - Generate multiple (**canonical**) variants
  - Select and perform cheap transformations at running time

Source: Chen, NTU, MPSoC 2008

Exploiting symmetries

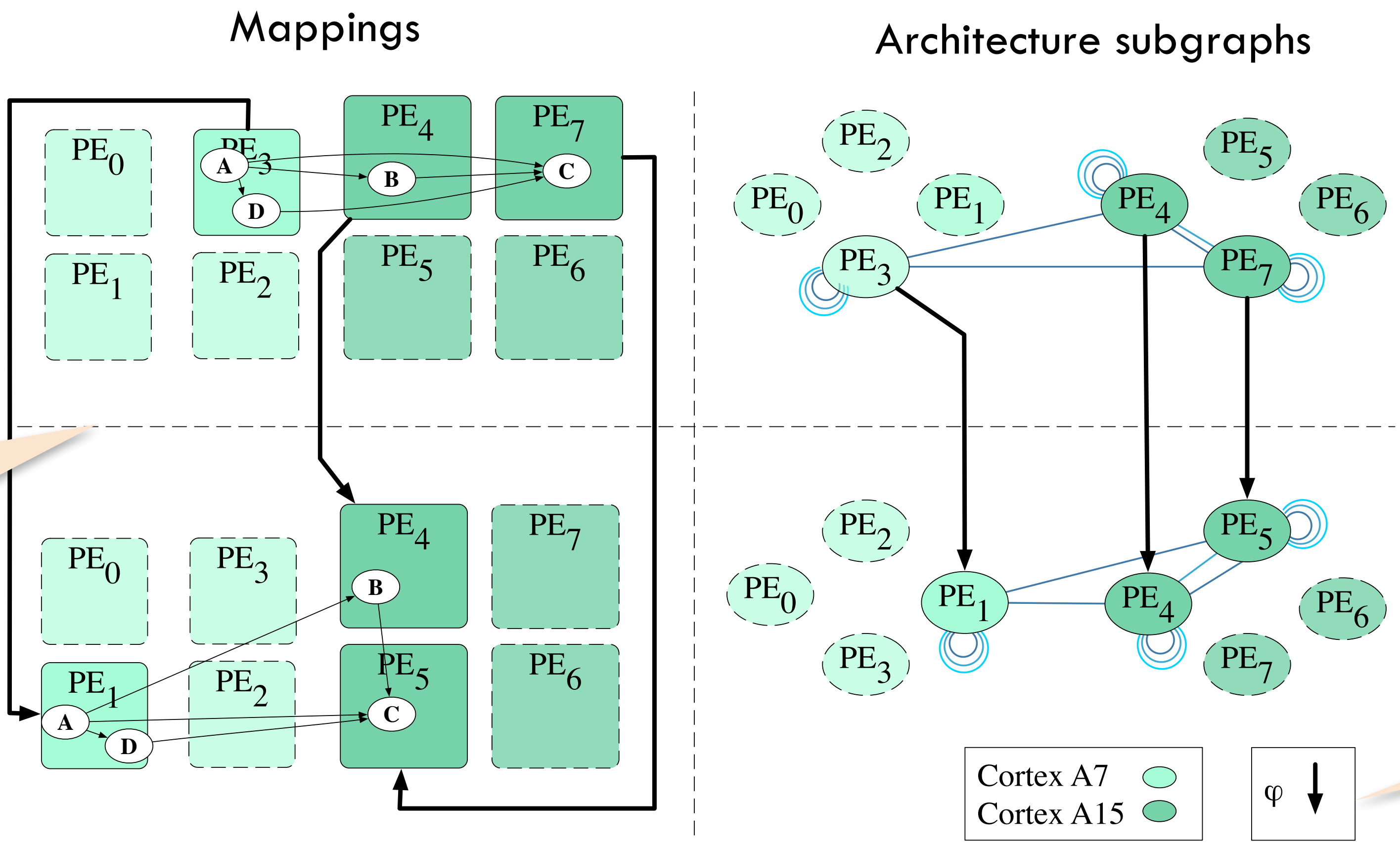
- Intuition
  - SW: Some tasks/processes/actors may do the same
  - HW: Symmetric latencies (CoreX ←→ CoreY)
  - Symmetry: Allows **transformations** w/o changing the **outcome**



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

(Symmetries have been implicitly exploited in the past)

Symmetries in Odroid: Example



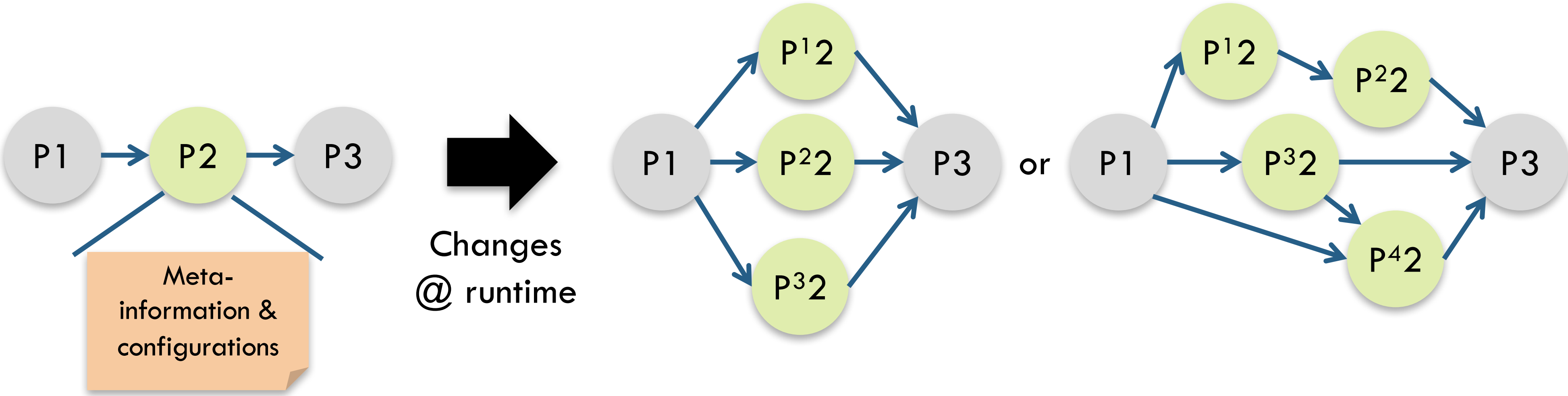
Equivalent mappings

Graph isomorphism

[Goens15, Goens17a]

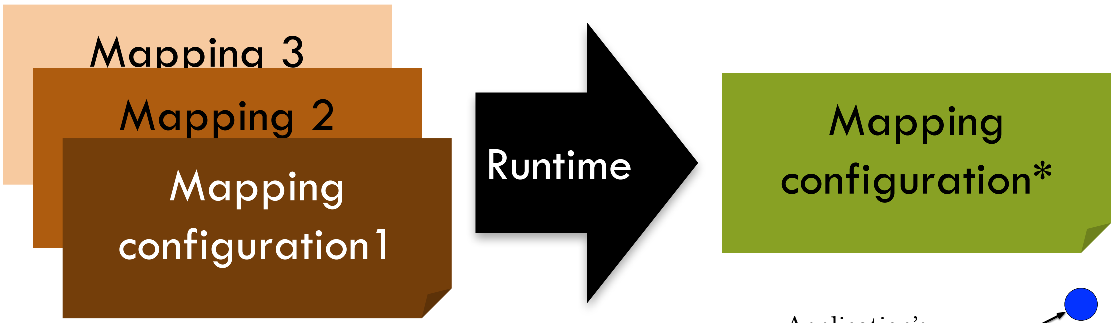
Data-level parallelism: Scalable and adaptive

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

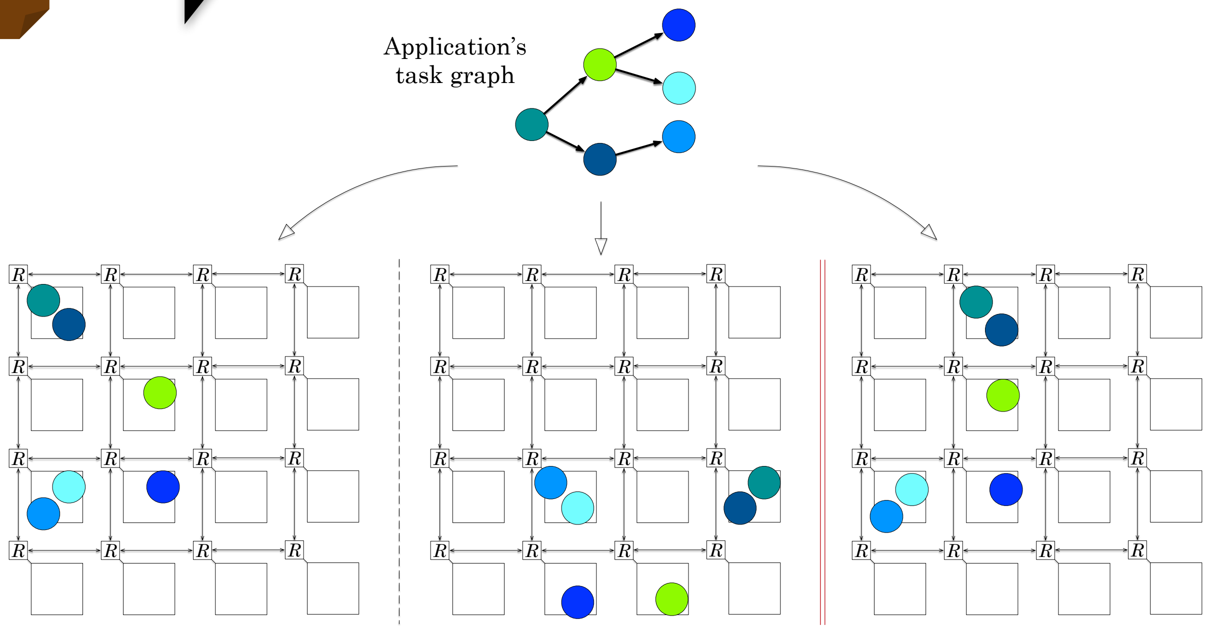


[Khasanov18]

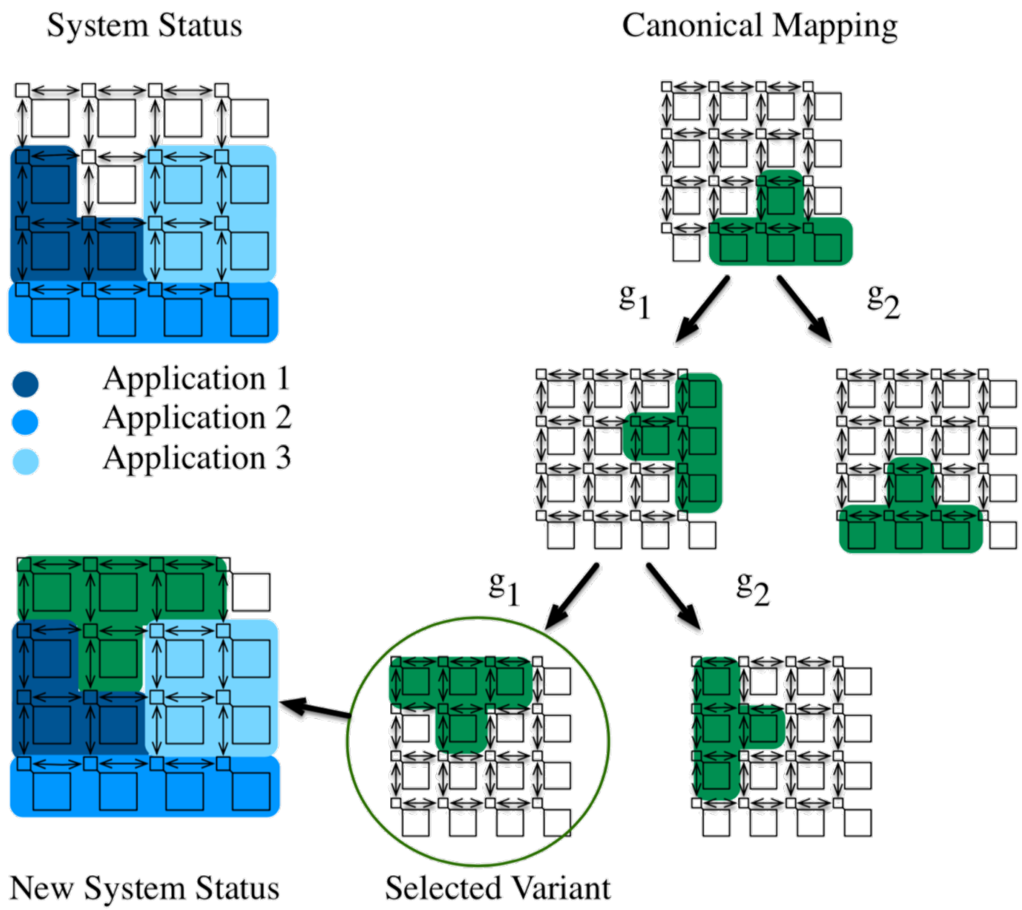
Flexible mappings



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



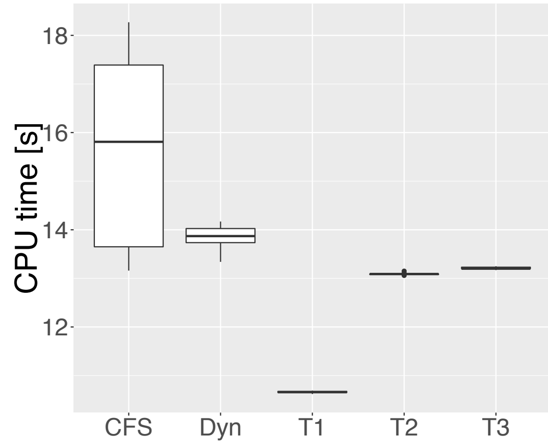
Flexible mappings: Tetris with extra rotations



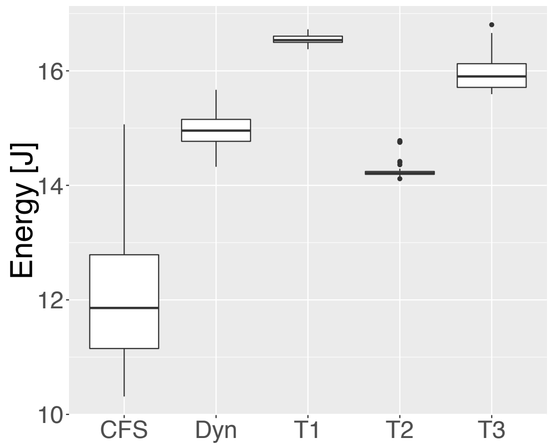
[Goens17b]

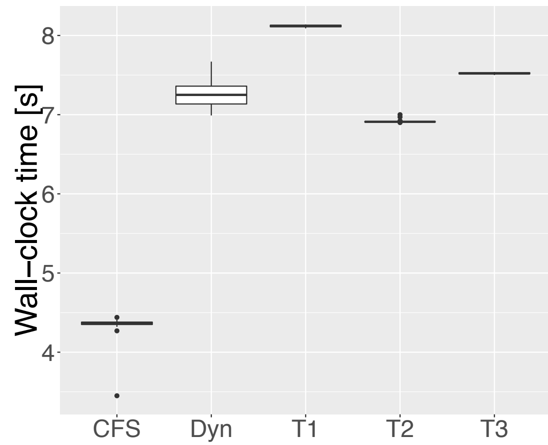
Flexible mappings: Run-time analysis

- Modified 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



Single AF

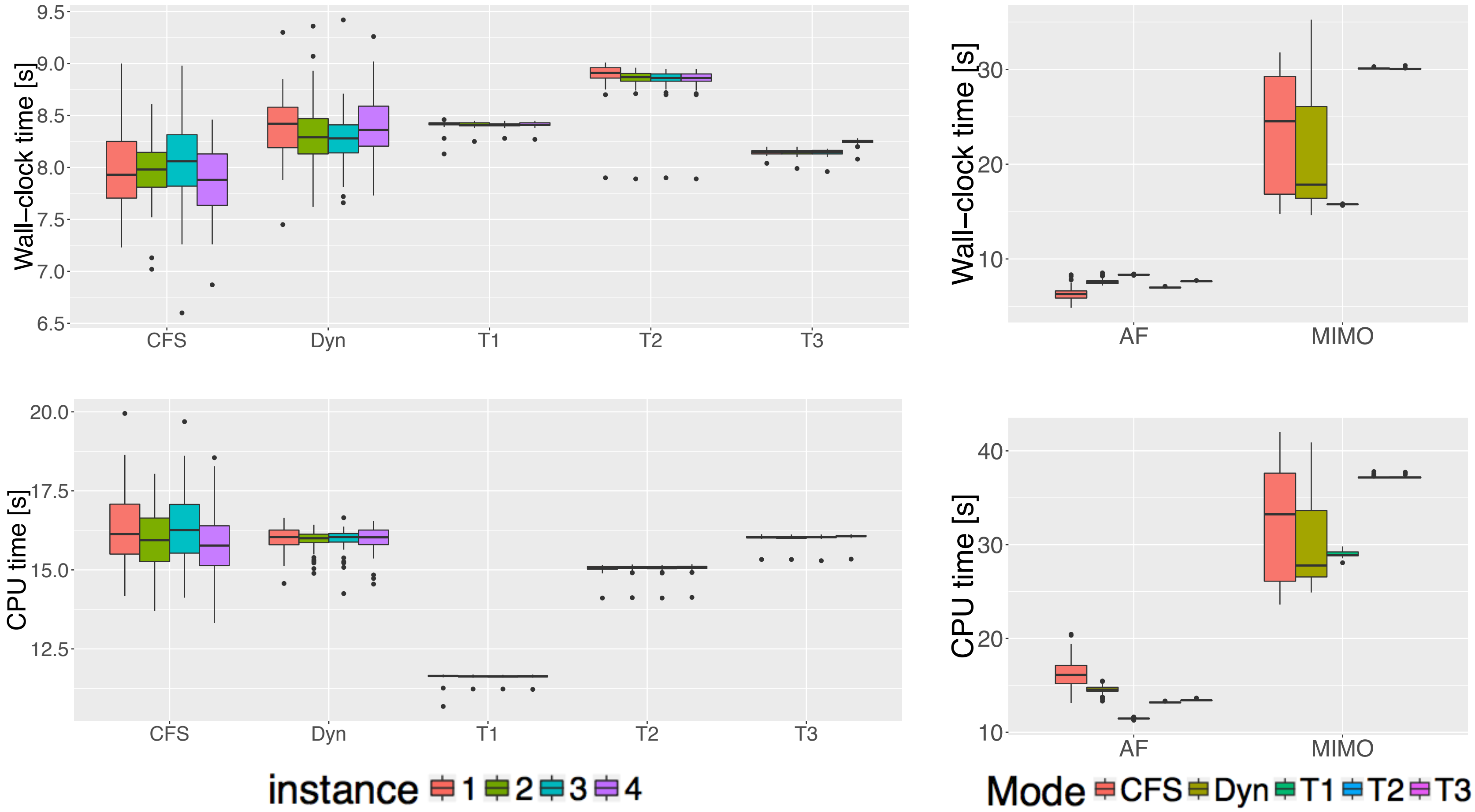




[Goens17b]

CHAIR FOR COMPILER CONSTRUCTION

Flexible mappings: Multi-application results (1)

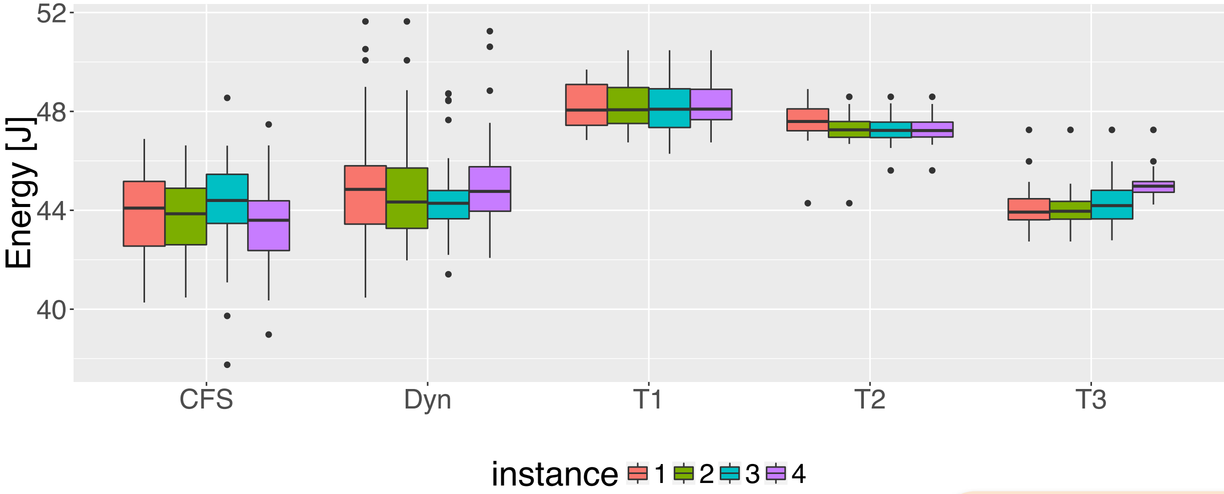


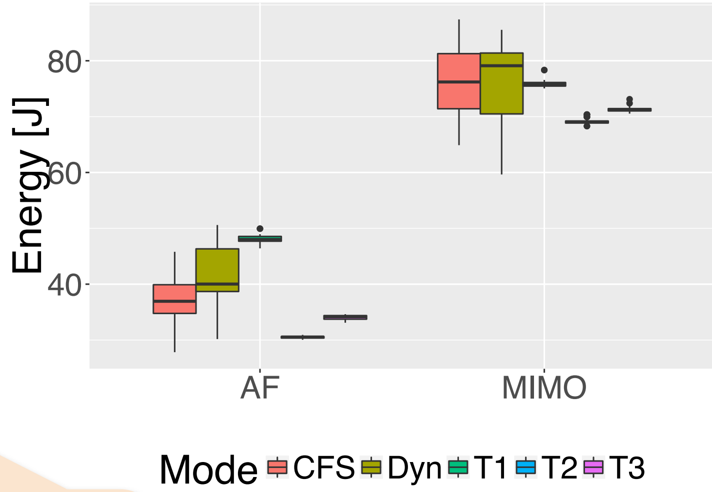
More predictable performance

Comparable performance to dynamic mapping

[Goens17b]

Flexible mappings: Multi-application results (2)





Better energy predictability as well

[Goens17b]

Robustness

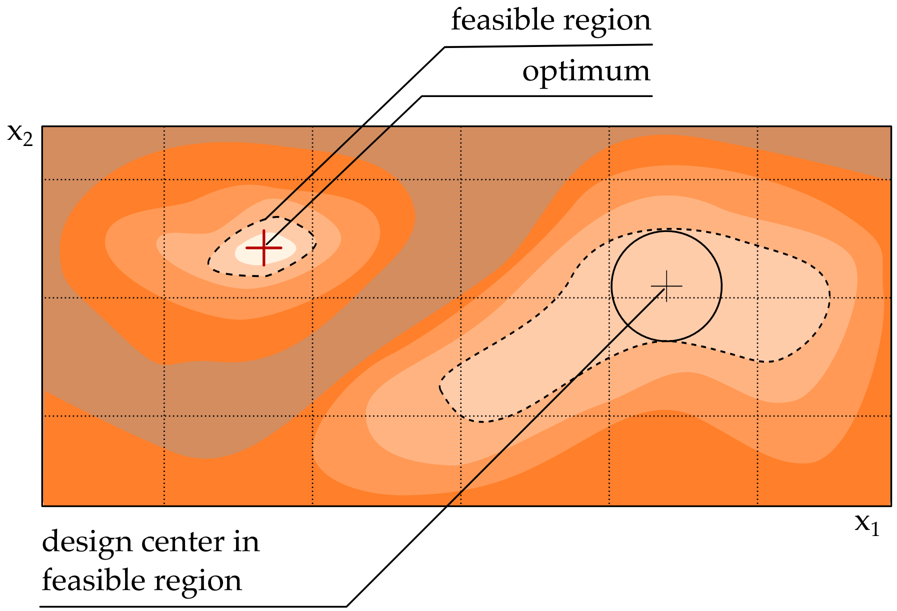
- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions

➔ 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

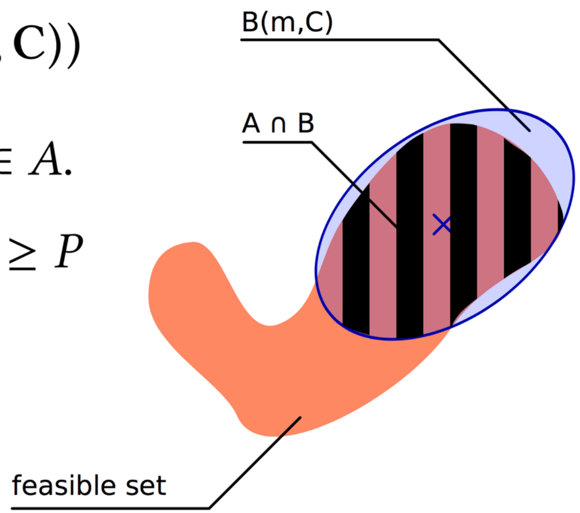


Design centering: Algorithmic

- 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

$$\max_{B=B(\mathbf{m},\mathrm{C})\in\mathcal{L}_p^n} \mathrm{vol}(B(\mathbf{m},\mathrm{C}))$$
$$\text{s.t.} \quad \mathbf{m} \in A,$$
$$\frac{\mathrm{vol}(A \cap B(\mathbf{m},\mathrm{C}))}{\mathrm{vol}(B(\mathbf{m},\mathrm{C}))} \geq P$$

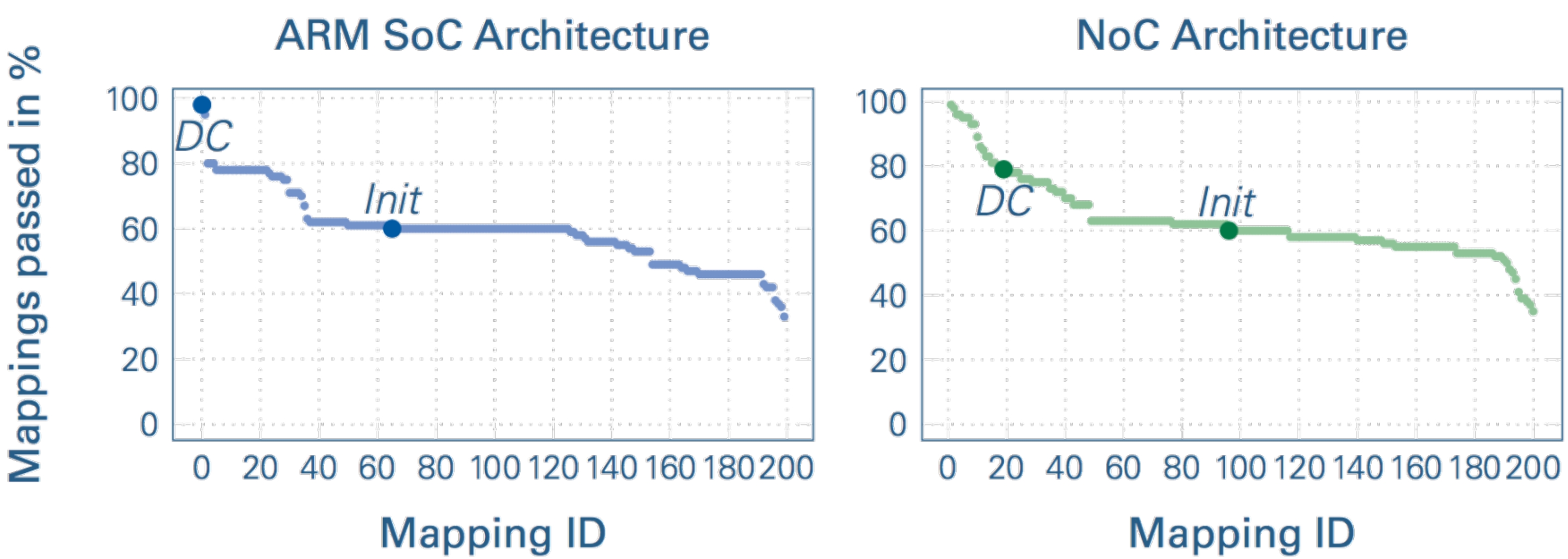


- Searching: Allow annealing (dynamically change P)

[Hempel17]

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)

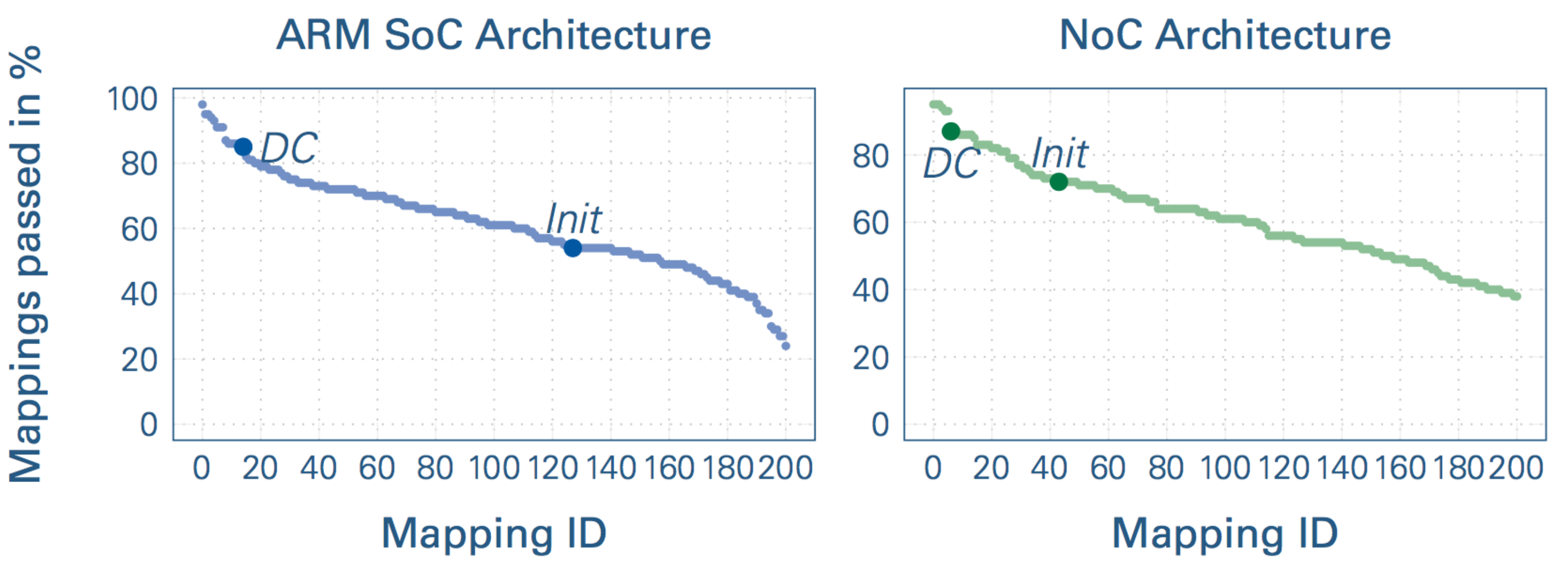


MIMO-OFDM

[Hempel17]

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)

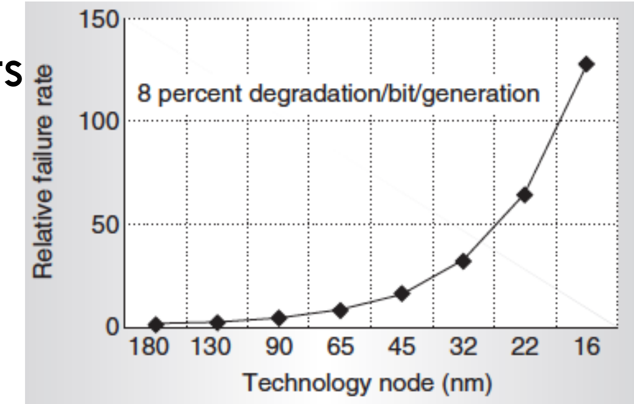


Audio-filter

[Hempel17]

Robustness: SW-based error correction

- Today and future technologies feature hardware faults and soft-errors
  - Need to protect against them at different levels



Source: [Borkar05]

- Typical approach: Compiler duplicates dataflow and insert checks

Original code

```
%3 = add i64 %0, %1
%4 = mul i64 %3, %2
```

transformation

Fault-tolerant code

```
%3  = add i64 %0,  %1
%r3 = add i64 %r0, %r1
%4  = mul i64 %3,  %2
%r4 = mul i64 %r3, %r2

%f0 = icmp eq i64 %4, %r4
br i1 %f0, label continue,
           label recover
```

duplicated dataflow (first four lines)

error check (last lines)

[Oh12]

Robustness: AN Encoding

- Arithmetic codes: One can still do meaningful arithmetic on encoded data
- AN encoding: **Make integer values multiples of a fixed constant A**
  - Check for errors like this:

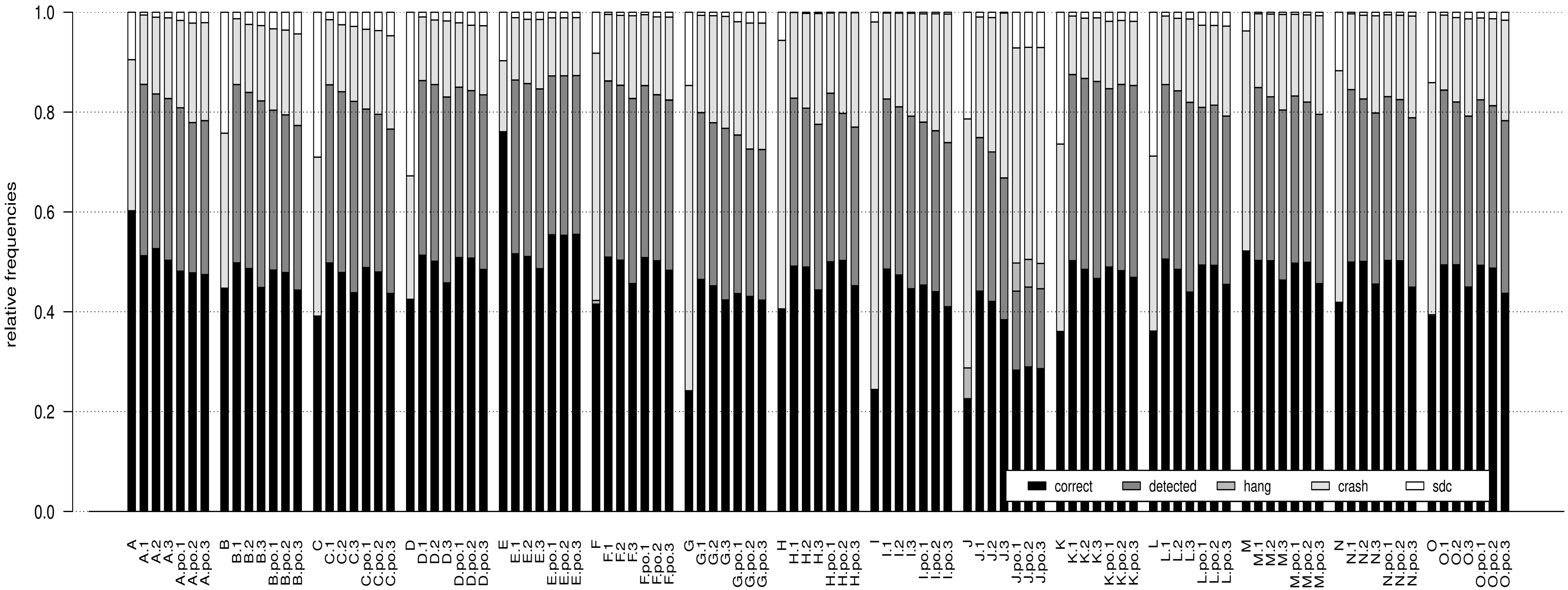
```
if (n % A != 0) { error_handler(); }
```

- Can be automated by compiler!

[Rink15, Rink16]

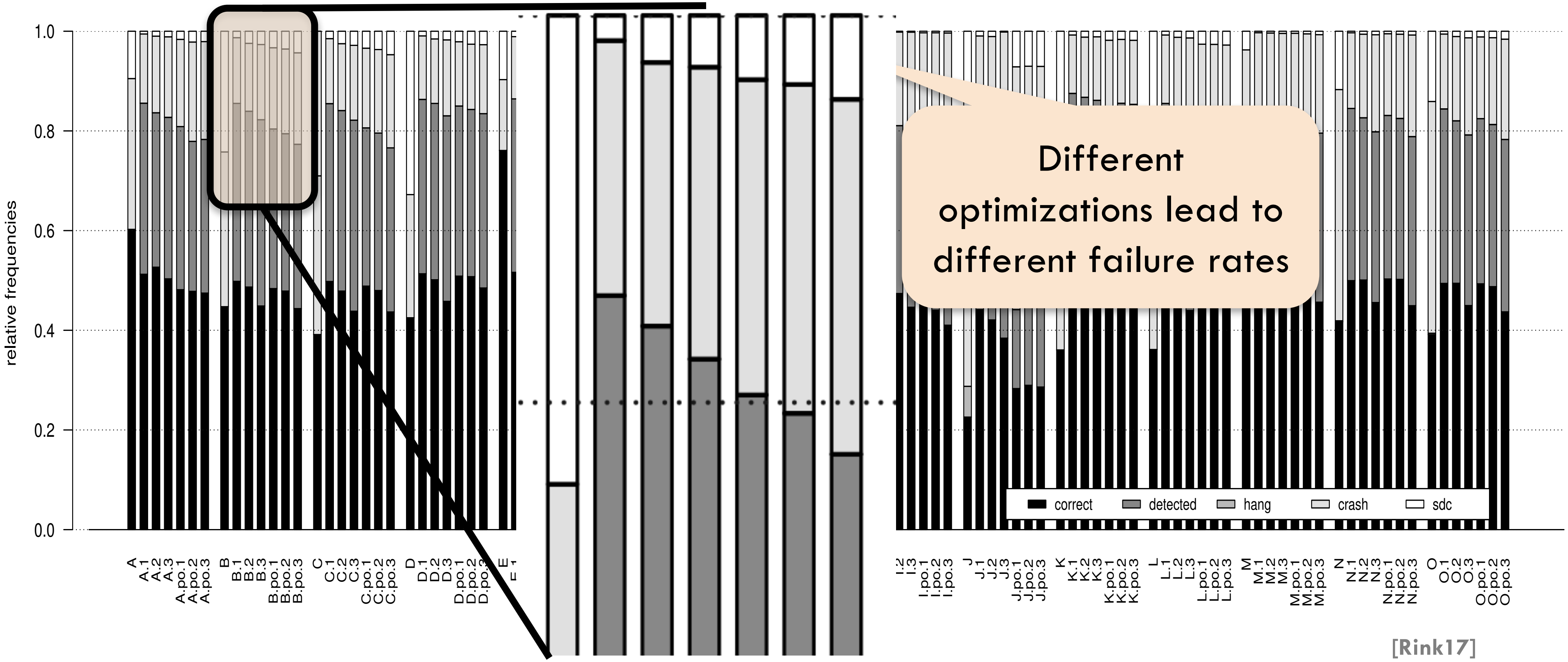
  - Some operations require non-trivial transformations
  - Integer division: $m/n \longmapsto (A*A*m)/(A*n) = A*(m/n)$
- Advantages over code duplication
  - Data in memory is encoded
  - Good for multithreading and shared memory!
- Disadvantages: Large **runtime overheads** (up to and over several 10x)

Compiler for resilience: Results – Failure rates



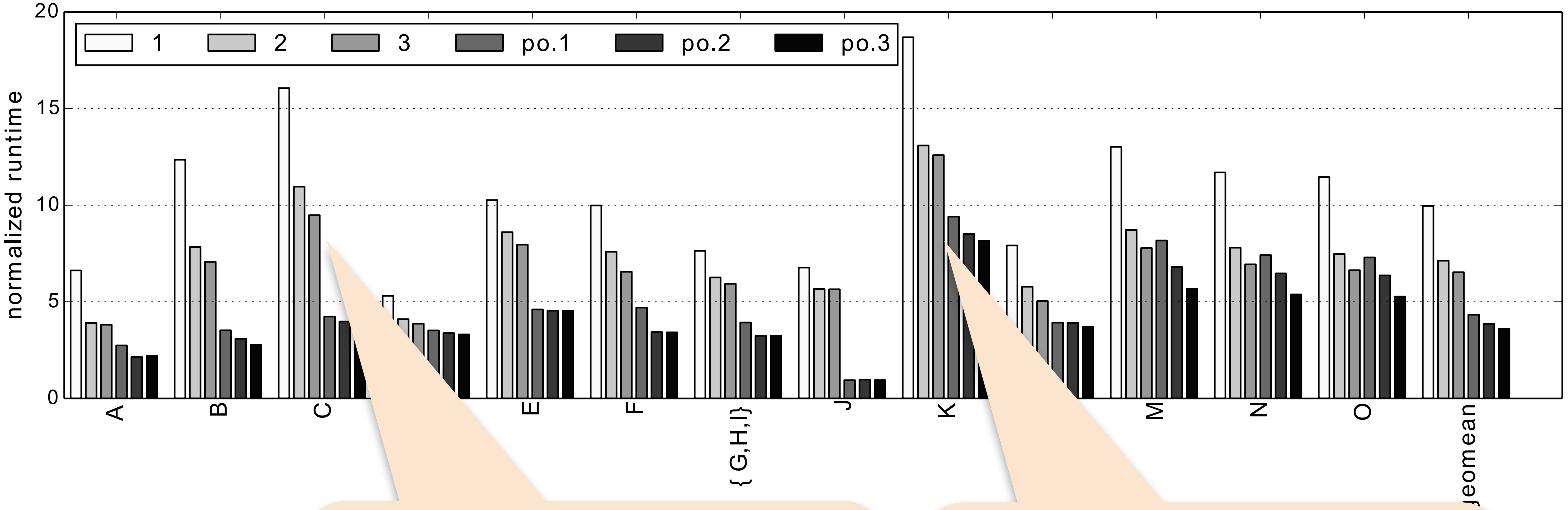
[Rink17]

Compiler for resilience: Results – Failure rates



[Rink17]

Compiler for resilience: Results – Overhead



variant	overhead
1	9.9
2	7.2
3	6.5
po.1	4.3
po.2	3.8
po.3	3.6

Different optimizations lead to different runtime overheads

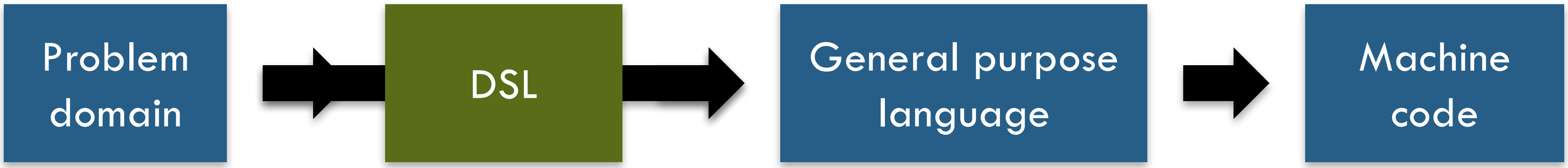
Possible direction: Raise the level of abstraction

[Rink17]

Domain-specific languages (DSLs)

Domain-specific languages

- Languages evolve, formalizing powerful design patterns (abstractions)
  - Some of them too common, so we do not notice it (goto → structured control, calling conventions → procedures, …)
- DSLs: bridge gap between problem domain and general purpose languages



Adapted from lecture: "Concepts of Programming Languages", Eelco Visser, TU Delft

- Many quite successful DSLs today (dataflow above, also a DSL)

Example: Tensors (Physics and Machine learning)

- Tensor expressions typically occur in numerical codes

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

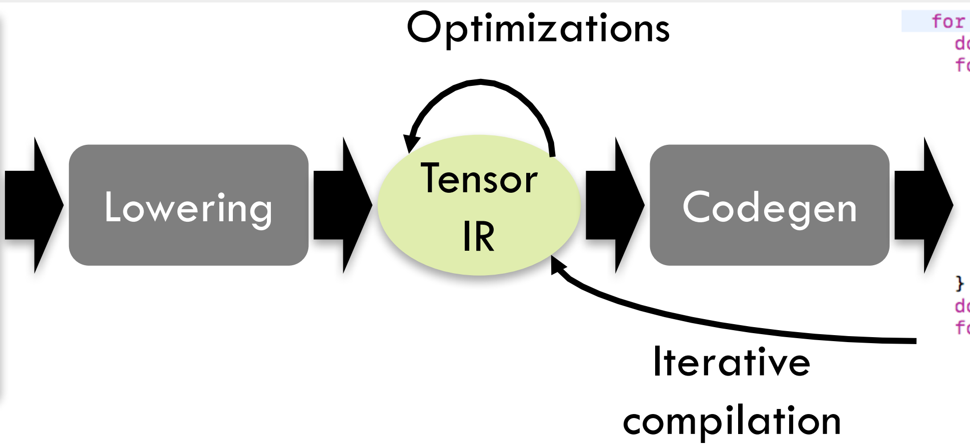
  - Tensor product: common in computational fluid dynamics
- On performance
  - Matrixes are small, so libraries like BLAS don't always help
  - Expressions result in deeply nested for-loops
  - Performance highly depends on the *shape* of the loop nests
- Higher-level expressions: No need for complex polyhedral analysis

CFDlang and tool flow

```
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

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



Linkable C code

```
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];
      }
    }
  }
}
```

[Rink18]

Example: Interpolation operator

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

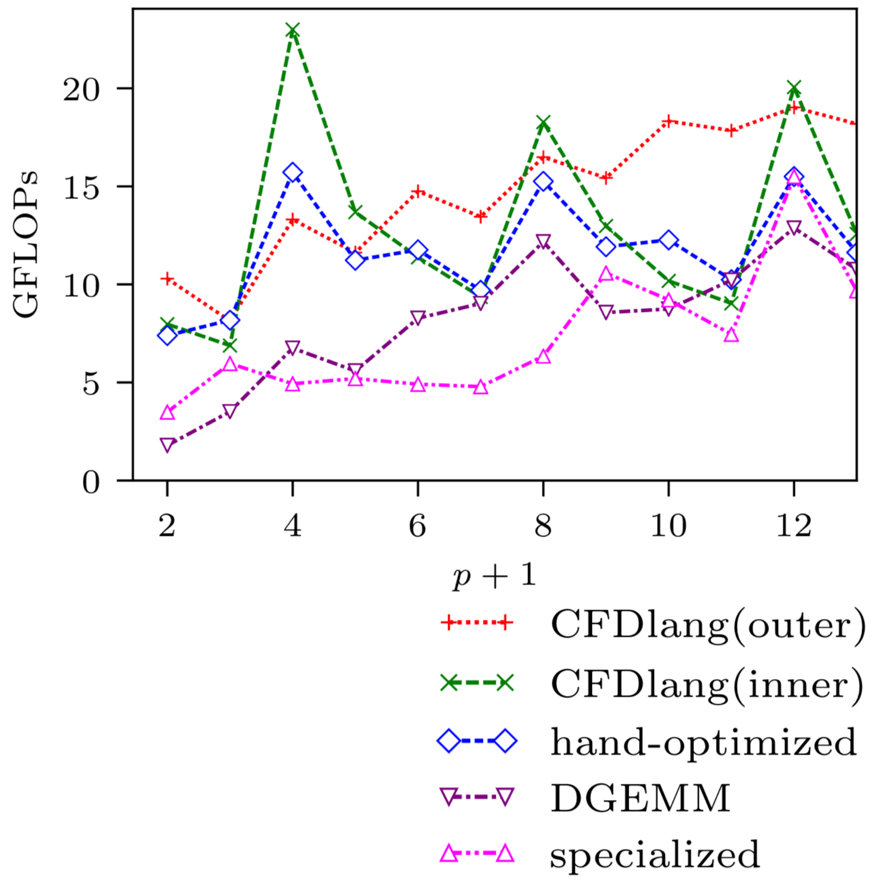
$$v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}$$

- Three alternative orders (besides naïve)

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

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

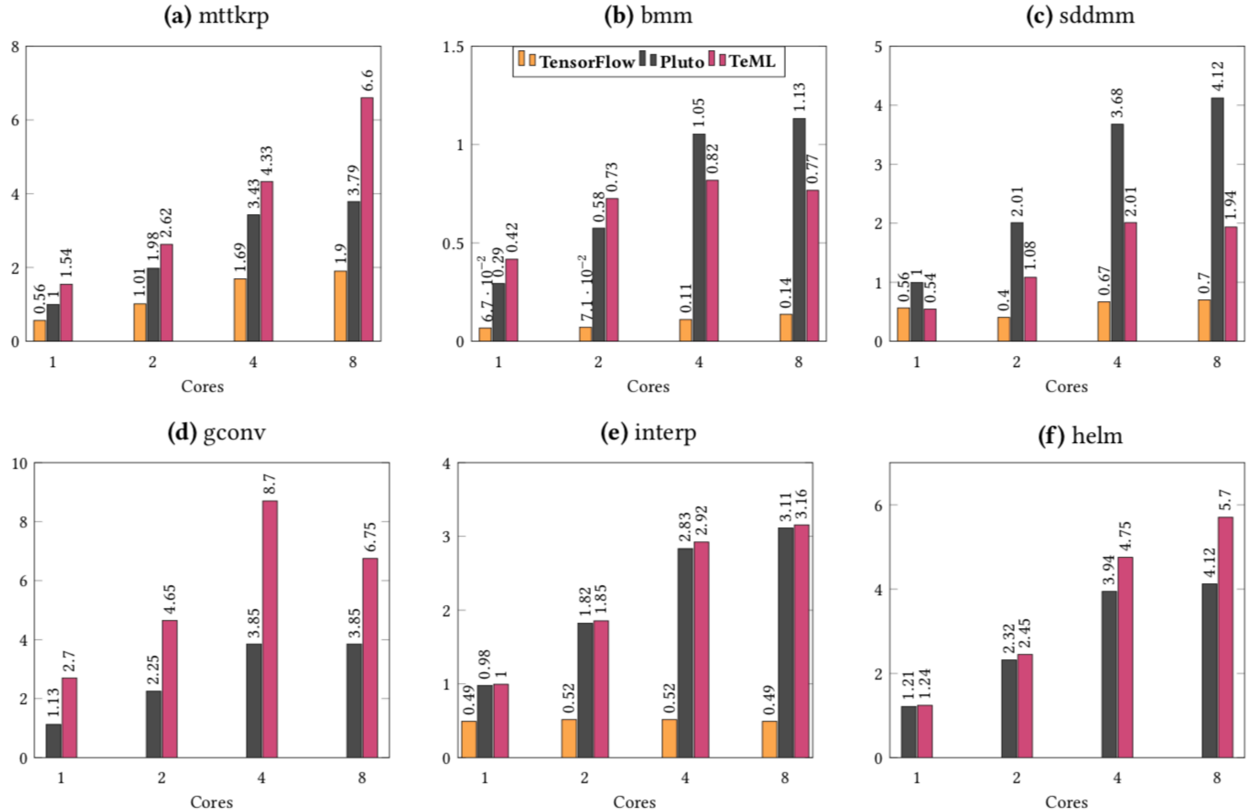
E3: $v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))$



[Susungi17, Rink18]

Meta-programming for optimizations: Results (2)

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



[Susungi18]

Summary

- Current research in tools for heterogeneous manycores
  - High-level abstractions for **language scalability**
  - Exploit symmetries and variable parallelism for runtime **adaptivity**
  - Reason about **robustness** of a mapping and of general code

- Further raise level of abstraction with DSLs
  - Example for tensors (CFD and Machine Learning)
  - Towards more automation on top of adaptive autosar

References

[Ceng08] Ceng, Jianjiang, et al. "MAPS: an integrated framework for MPSoC application parallelization." Proceedings of the 45th annual Design Automation Conference. DAC'18.

[Castrill11] Castrillon, Jeronimo, Weihua Sheng, and Rainer Leupers. "Trends in embedded software synthesis." 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation.

[Castrill13a] Castrillon Mazo, Jeronimo. Programming heterogeneous mpsocs: Tool flows to close the software productivity gap. Dissertation Aachen, Techn. Hochsch., Diss., 2013.

[Castrill13] 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

[Castrill11a] Castrillon, Jeronimo, et al. "Backend for virtual platforms with hardware scheduler in the MAPS framework." Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on. IEEE, 2011.

[Murillo14] Murillo, Luis Gabriel, et al. "Automatic detection of concurrency bugs through event ordering constraints." Proceedings of the conference on Design, Automation & Test in Europe. European Design and Automation Association, 2014.

[Oden13] Odendahl, Maximilian, et al. "Split-cost communication model for improved MPSoC application mapping." System on Chip (SoC), 2013 International Symposium on. IEEE, 2013.

[Eusse15] Eusse, Juan Fernando, et al. "Coex: A novel profiling-based algorithm/ architecture co-exploration for asip design." *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 8.3 (2015): 17.

[Ertel18a] S. Ertel, et al., "Supporting Fine-grained Dataflow Parallelism in Big Data Systems" , PMAM'18, ACM, pp. 41–50, New York, NY, USA, Feb 2018.

[Ertel18b] S. Ertel, et al "Compiling for Concise Code and Efficient I/O" , CC '18

[Hempel17] Hempel, G. et al. "Robust Mapping of Process Networks to Many-Core Systems using Bio-Inspired Design Centering". SCOPES'17

[Goens15] 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

[Goens16] A. Goens, R. Khasanov, J. Castrillon, S. Polstra, A. Pimentel, "Why Comparing System-level MPSoC Mapping Approaches is Difficult: a Case Study" , MCSoC-16

[Goens17a] Goens, A. et al. "Symmetry in Software Synthesis". In: ACM Transactions on Architecture and Code Optimization (TACO) (2017)

[Goens17b] Goens, A. et al. "TETRiS: a Multi-Application Run-Time System for Predictable Execution of Static Mappings", SCOPES'17

[Khasanov18] R. Khasanov, et al, "Implicit Data-Parallelism in Kahn Process Networks: Bridging the MacQueen Gap" , PARMA-DITAM 18, ACM, pp. 20–25, New York, NY, USA, Jan 2018.

[Borkar05] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," IEEE Micro, vol. 25, no. 6, 2005.

[Oh12] N Oh, P P Shirvani, and E J McCluskey. Error detection by duplicated instructions in super-scalar processors. IEEE Transactions on Reliability, 51(1):63–75, March 2002.

[Rink15] Rink, Norman A., et al. "Compiling for Resilience: the Performance Gap." PARCO. 2015.

[Rink16] Rink, Norman A., et al. "Fault Tolerance with Aspects: A Feasibility Study.", MODULARITY'16

[Rink17] N. Rink, et al. "Trading Fault Tolerance for Performance in AN Encoding", ACM International Conference on Computing Frontiers (CF'17), May 2017

[Rink18] N. A. Rink, et al. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL 2018

[Susungi17] A. Susungi, et al. "Towards Compositional and Generative Tensor Optimizations" GPCE 17

[Susungi18] A. Susungi, et al. " "Meta-programming for cross-domain tensor optimizations" GPCE 18