CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Programming abstractions: When domain-specific goes mainstream

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

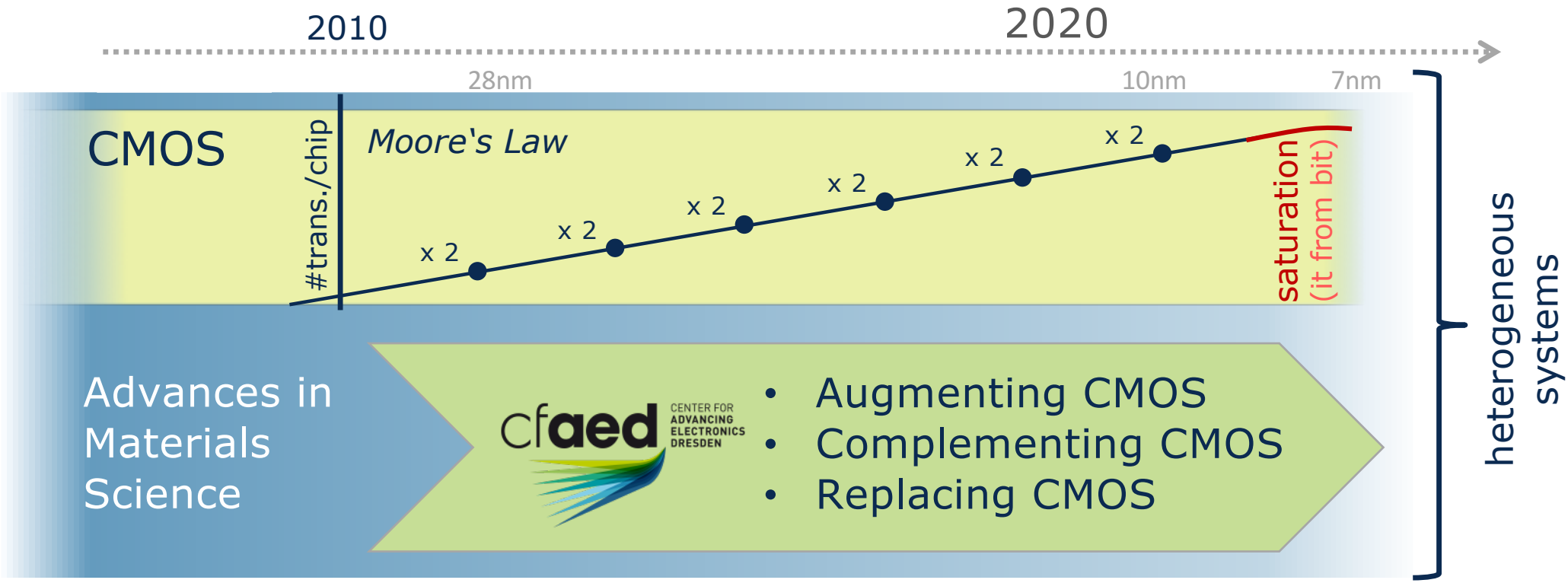
TU Dresden, Germany

28th GI PARS Workshop

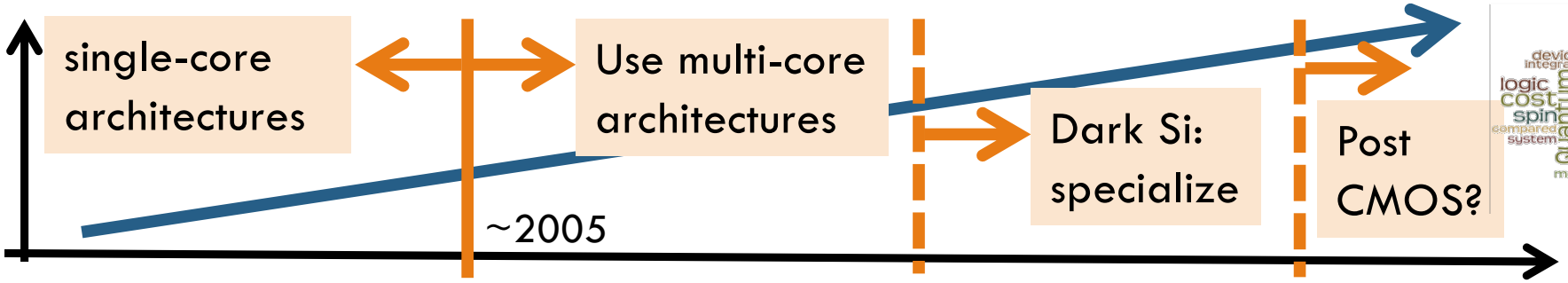
Berlin, Germany. March 21, 2019

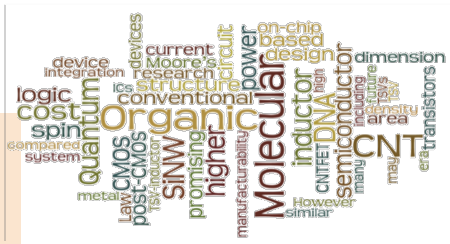
Cfaed: Center for Advancing Electronics Dresden

- New technologies for information processing that overcome the limits of CMOS

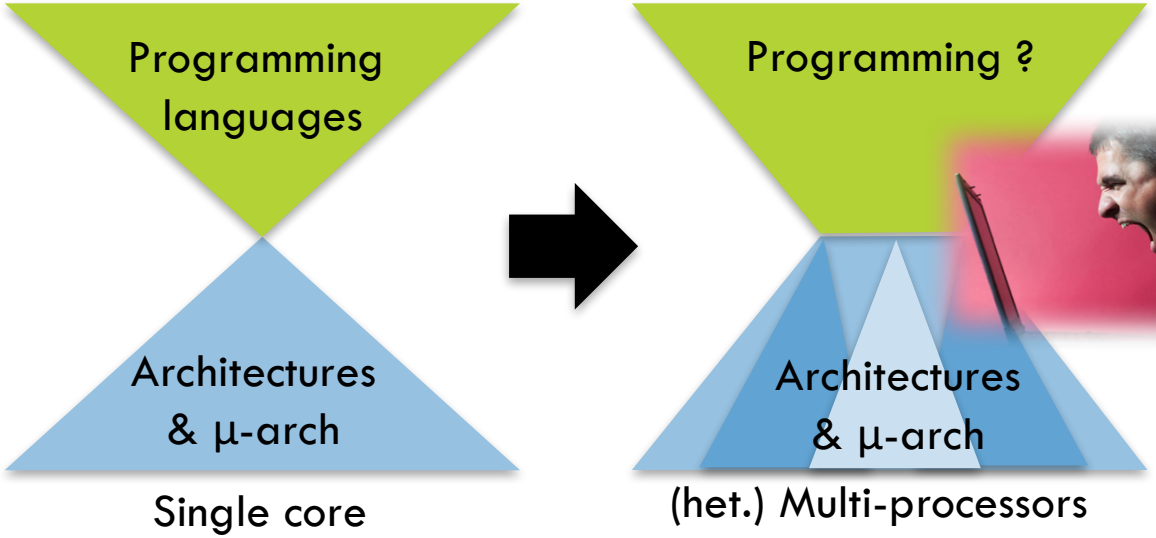


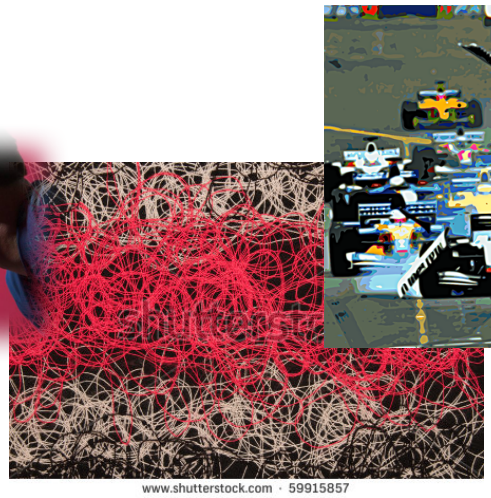
Research: Inflection points and programming





- The programming interface continues to broadens as hardware evolves





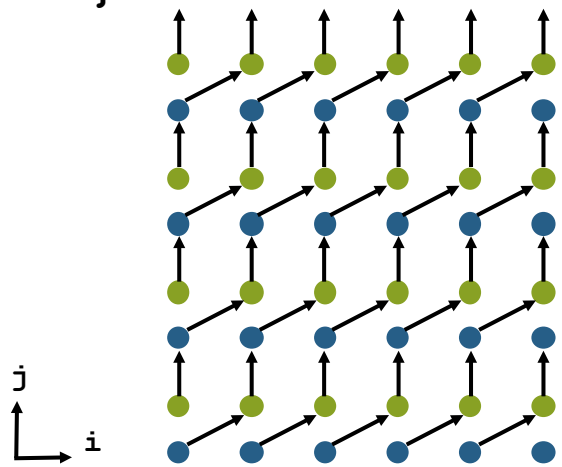
http://nerds-central.blogspot.de/2011/11/atomicinteger-volatile-synchronized-and.html

Auto-parallelization?

1) Find all dependencies?

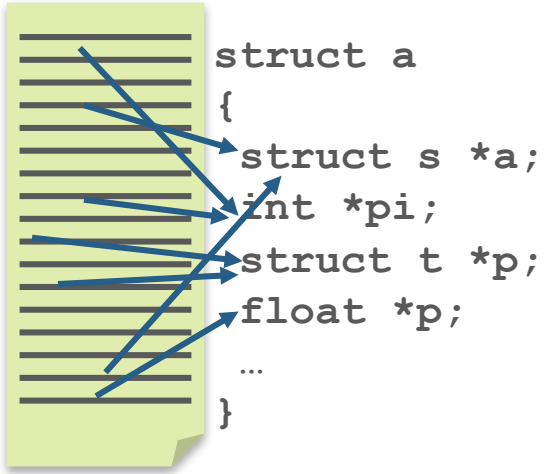
More often than not, impossible!

```
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



Auto-parallelization? (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)

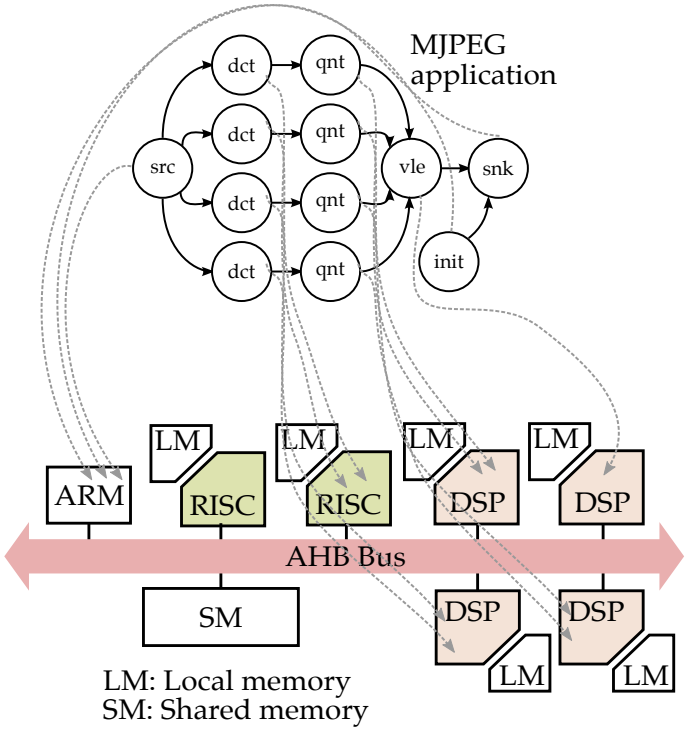
[Edler-CC'18]

```
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 }
```

Dataflow abstraction

Dataflow programming abstraction

- Possibly cyclic graphs represent stream processing
- **Large history in niches** (embedded het. multi-cores)
  - Deterministic, race free, restricted expressiveness, analyzability (memory, schedules)
- Plenty of work on: Languages, compilers, mapping algorithms, HW models, performance estimation, code generation and runtime systems [Springer'14]



- Going more general purpose
  - More implicit programming
  - More adaptivity (applications do not run alone)

Higher-level abstraction for dataflow

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

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

Functional to dataflow

Dataflow Elements:

d ::=

	1-1 node
	edge
	port
	1-N node
	N-1 node

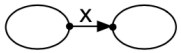
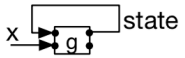
Dataflow Nodes:

	negation
	map to true value
	data to control signal
	selection

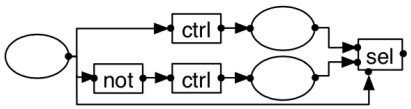
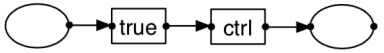
Predefined Value Functions:

	length of list
	list to stream
	stream to list
	unbounded list to stream

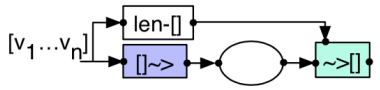
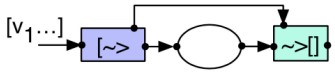
Terms:

x	$\mapsto$		variable
t	$\mapsto$		term
(let [x t] t)	$\mapsto$		lexical scope
(f x)	$\mapsto$		apply stateless function f to x
(g x)	$\mapsto$		apply stateful function g to x

Control Flow:

(if t t t)	$\mapsto$		conditionals
(seq t t)	$\mapsto$		sequential evaluation

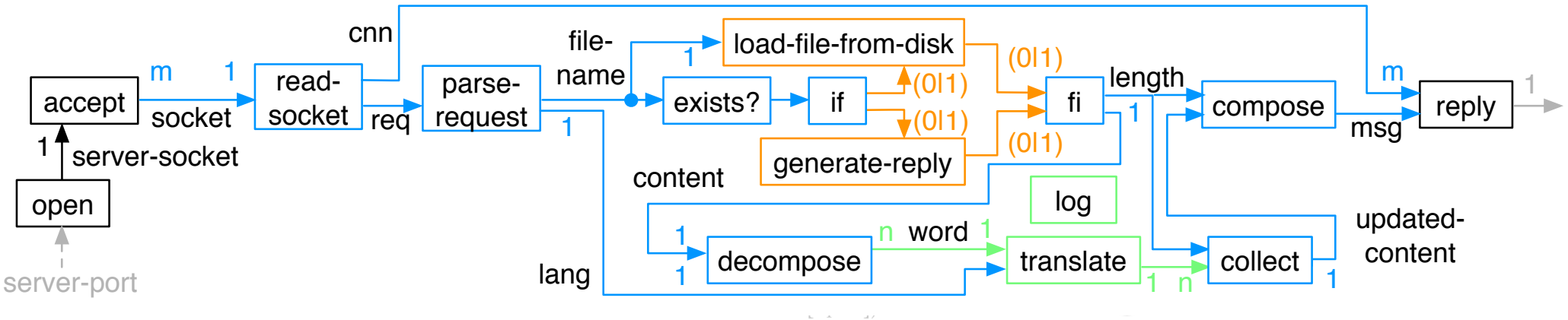
Predefined Functions:

(smap (algo [x] t) [$v_1 \ldots v_n$])	$\mapsto$		bounded list
(smap (algo [x] t) [$v_1 \ldots$])	$\mapsto$		unbounded list

[PMAM'18]

Functional to dataflow

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



I/O optimization in uServices

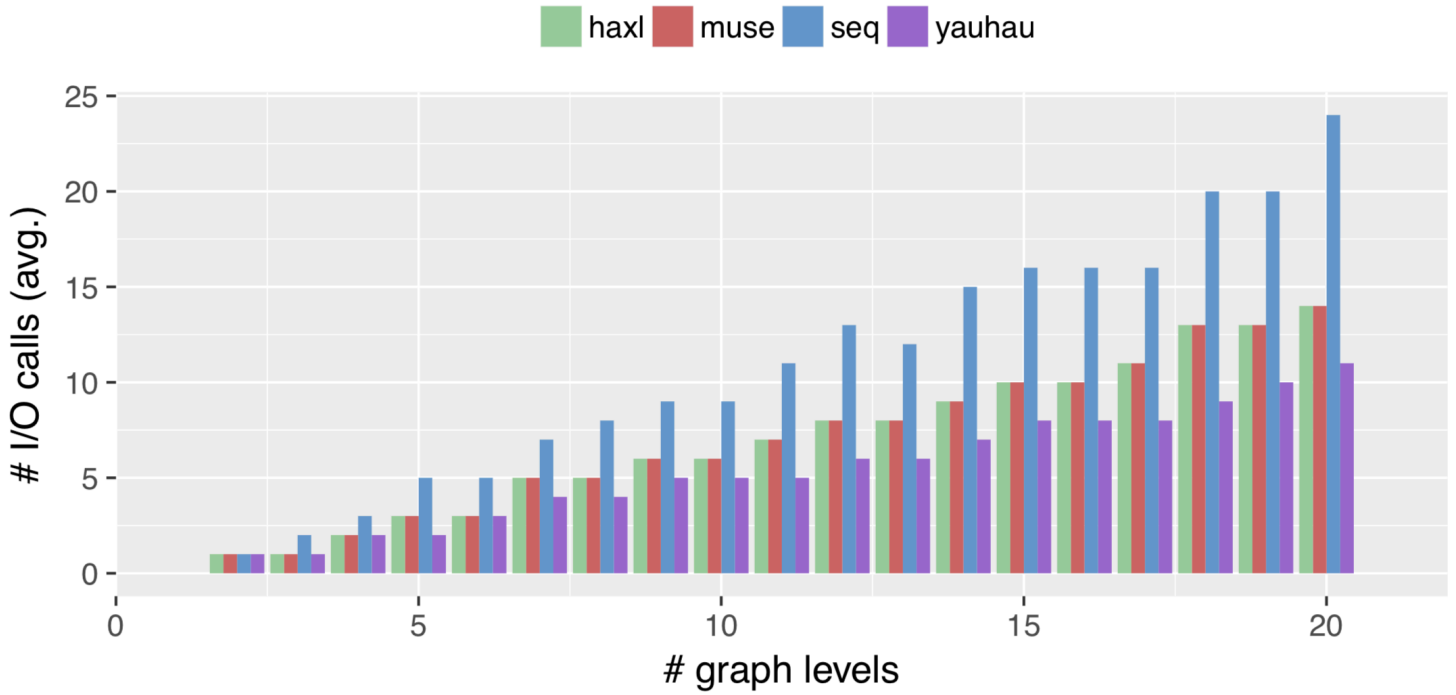
- Functional abstraction: amenable for micro-service architectures
- Problem
  - Modularity at odds with performance due to repeated I/O calls
  - Currently solved via **applicative functors** (Facebook)

- Develop simple dataflow rewrites to optimize I/O batching

[CC'18]

Second use-case: I/O optimization in uServices

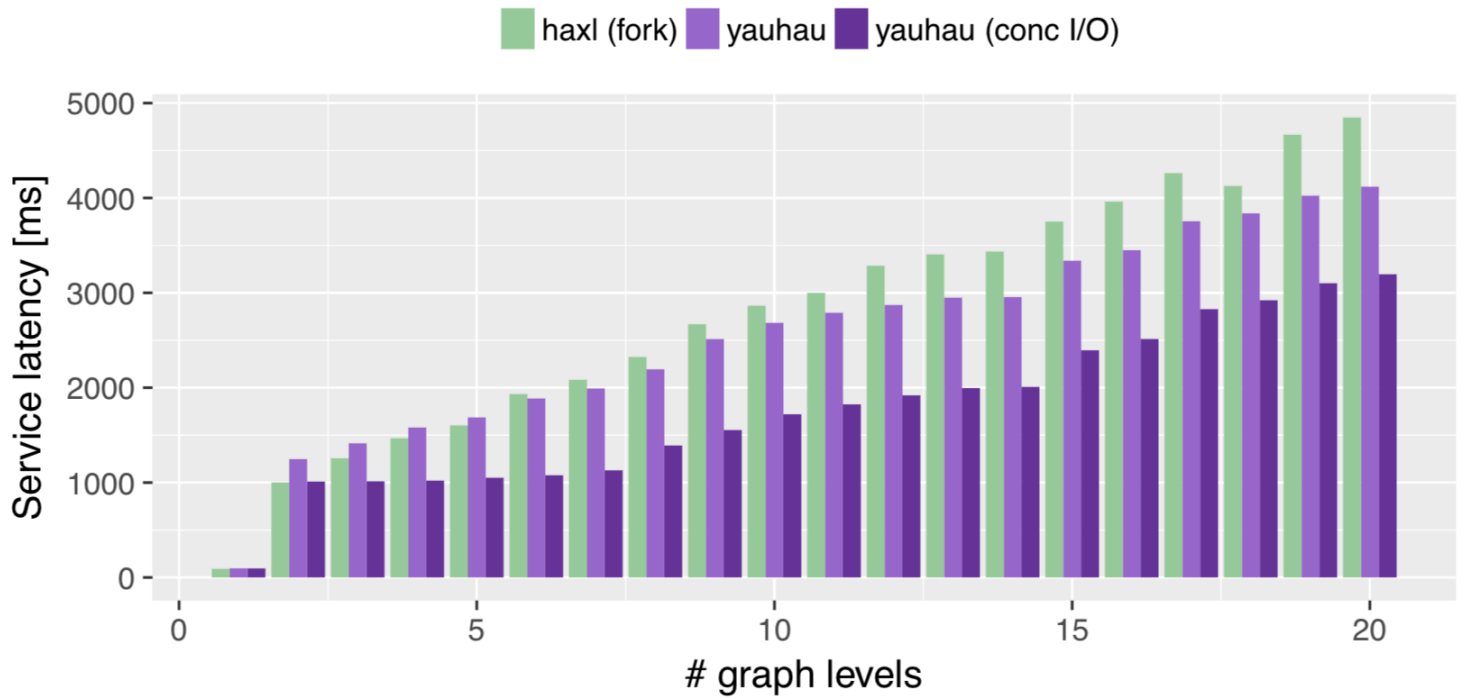
- Functional abstraction: amenable for micro-service architectures
- Develop simple dataflow rewrites to optimize I/O batching



[CC'18]

Second use-case: I/O optimization in uServices

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



[CC'18]

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

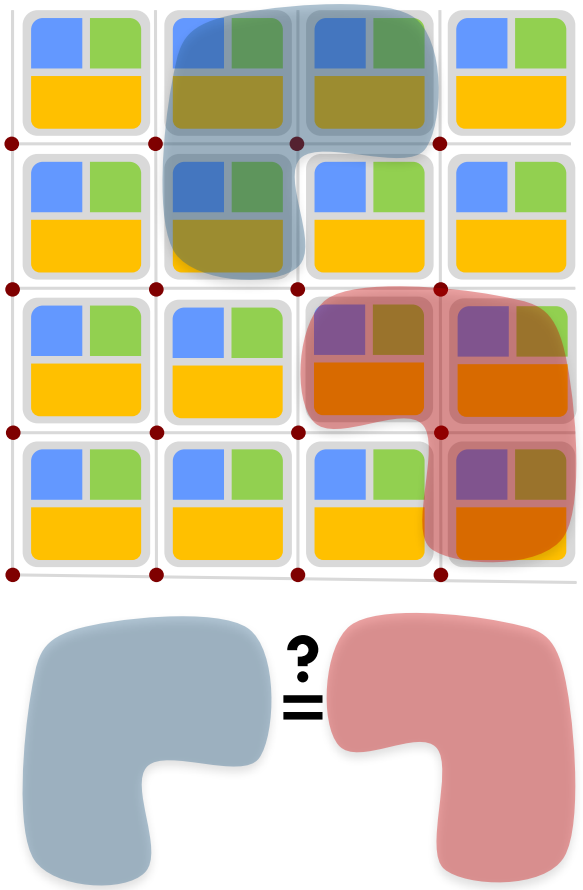
Source: Chen, NTU, MPSoC 2008

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)

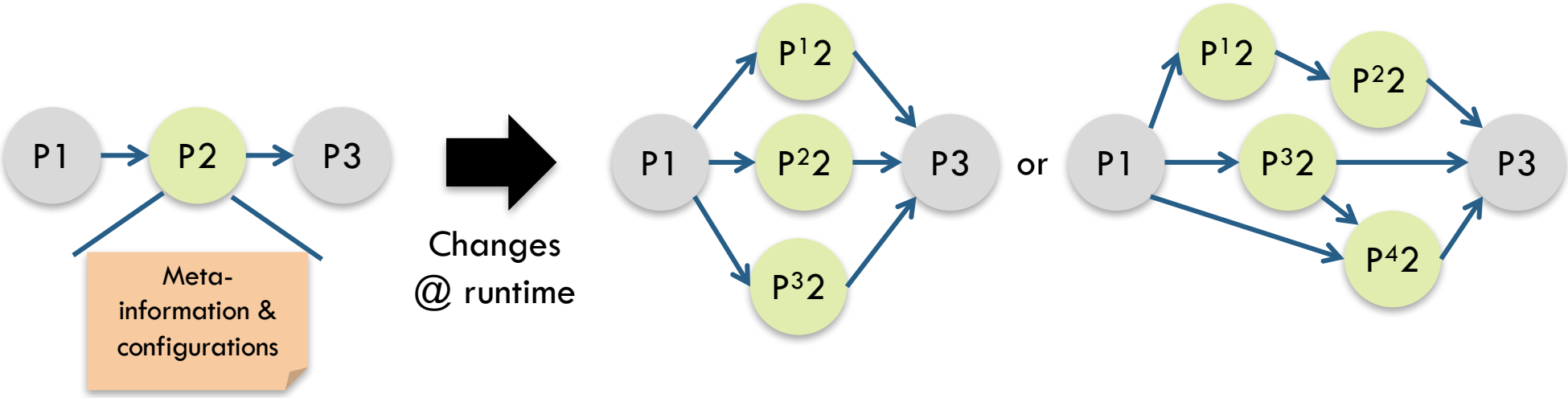
➔ Formal and generic definition of symmetries in the mapping problem



[IESS'15, ACM TACO'17]

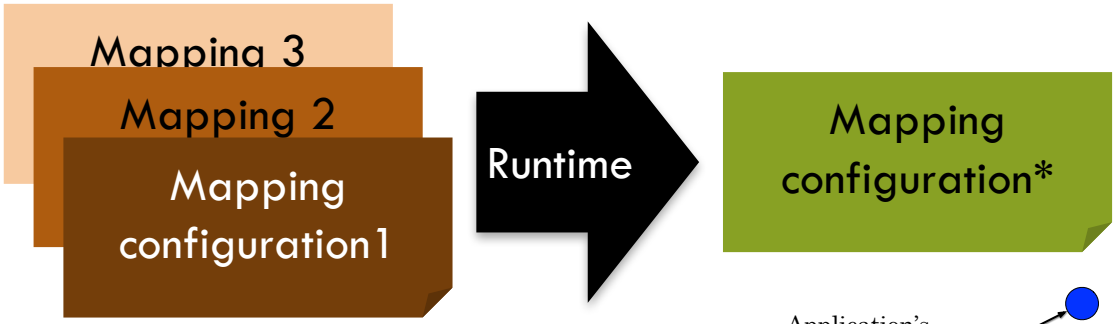
Data-level parallelism: Scalable and adaptive

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

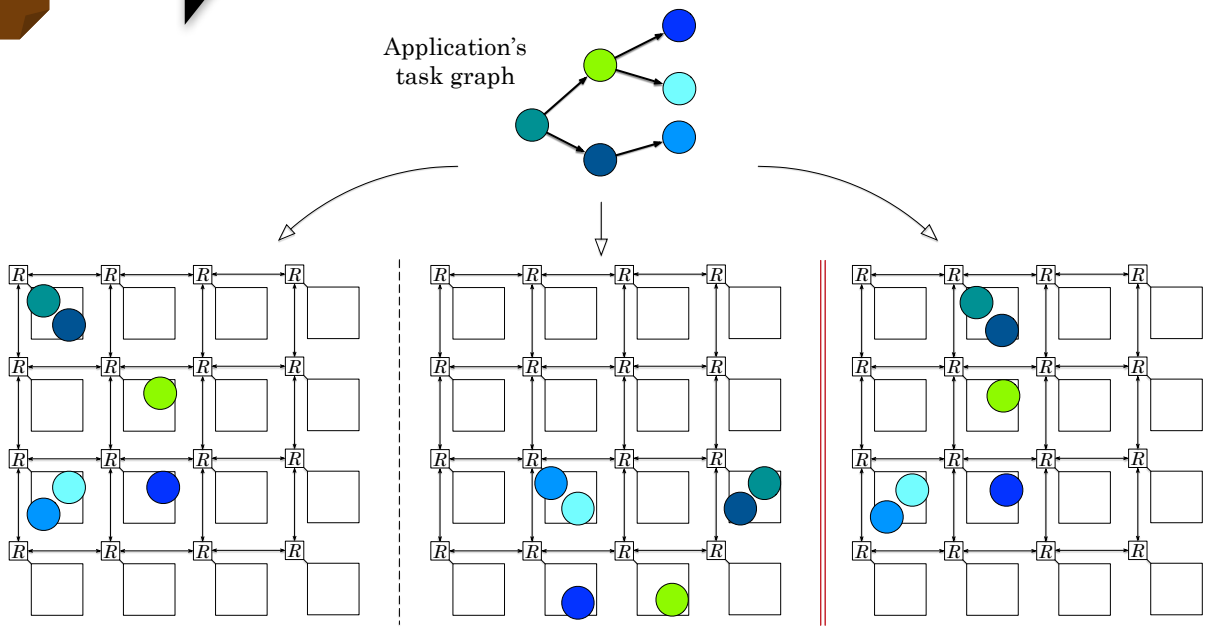


[PARMA-DITAM'18]

Flexible mappings

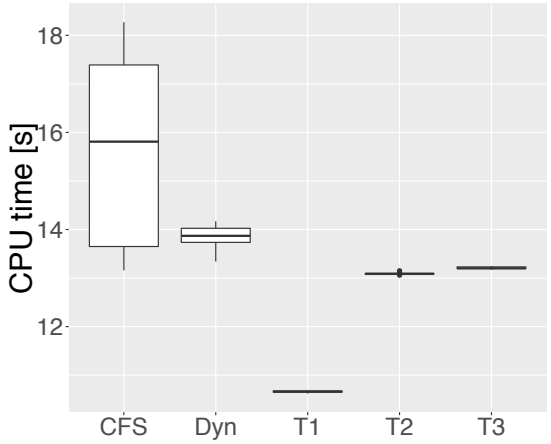


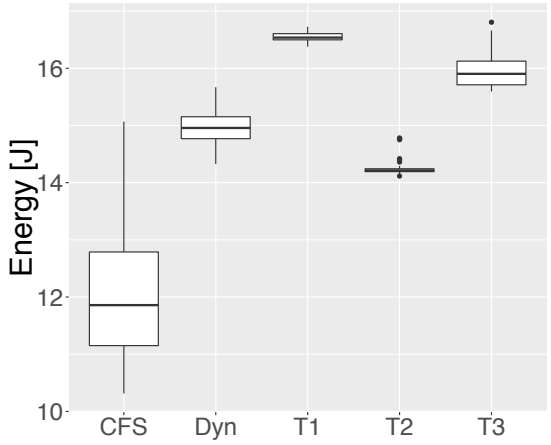
- Given multiple **canonical** configs by compiler, select one at run-time
- Exploit mapping **equivalences** and **similarities**



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

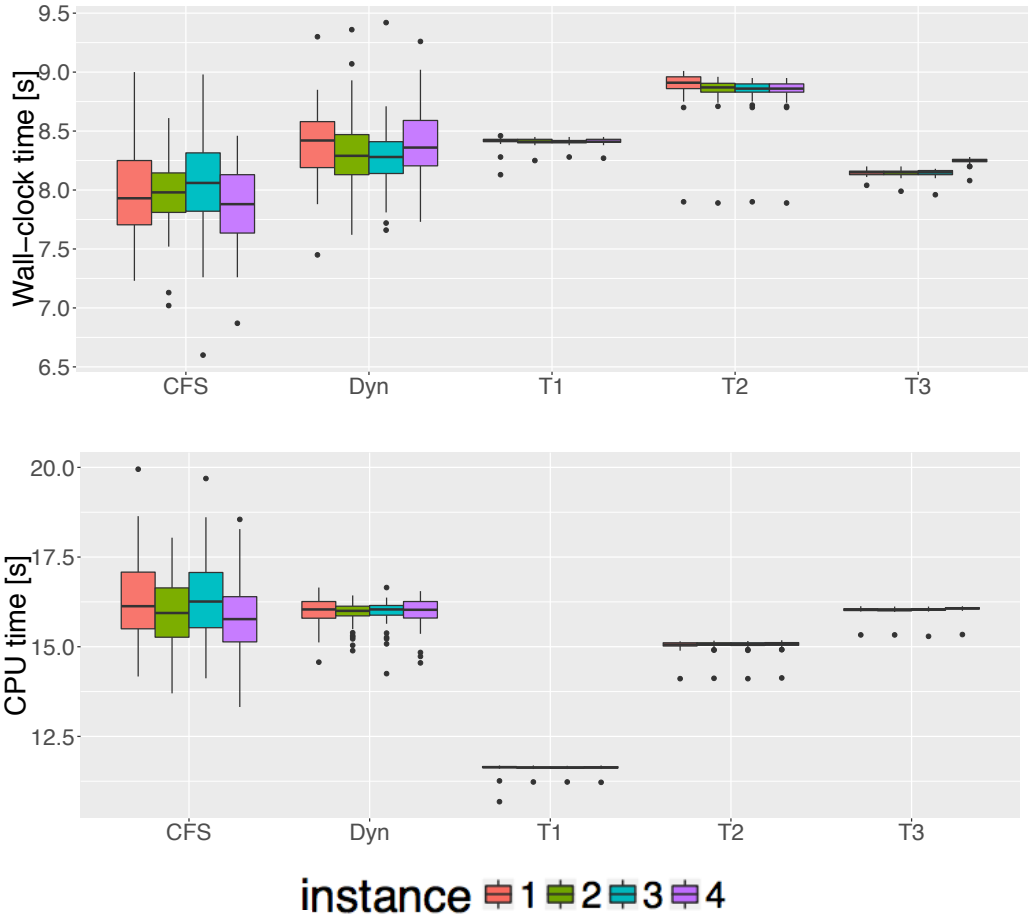


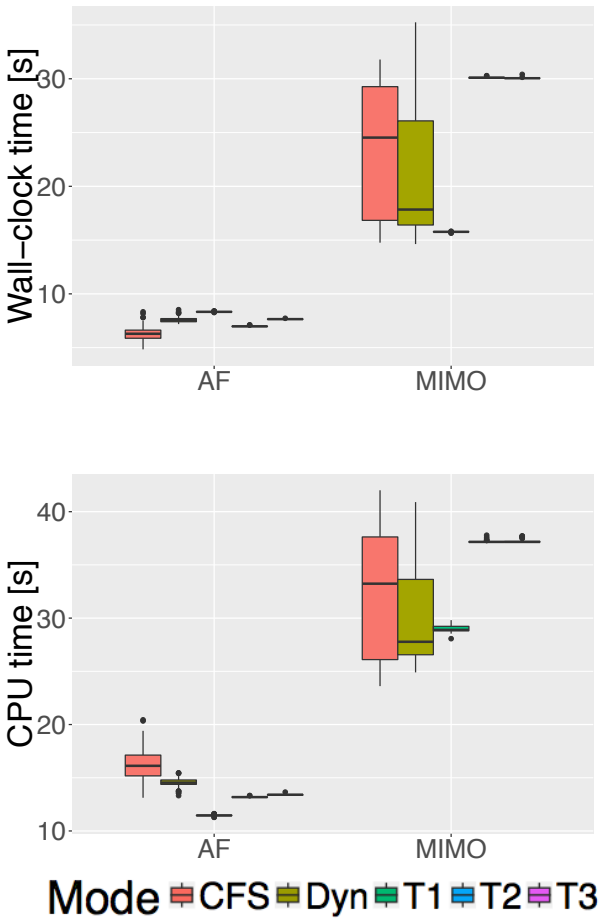


Single AF

[SCOPES'17b]

Flexible mappings: Multi-application results (1)



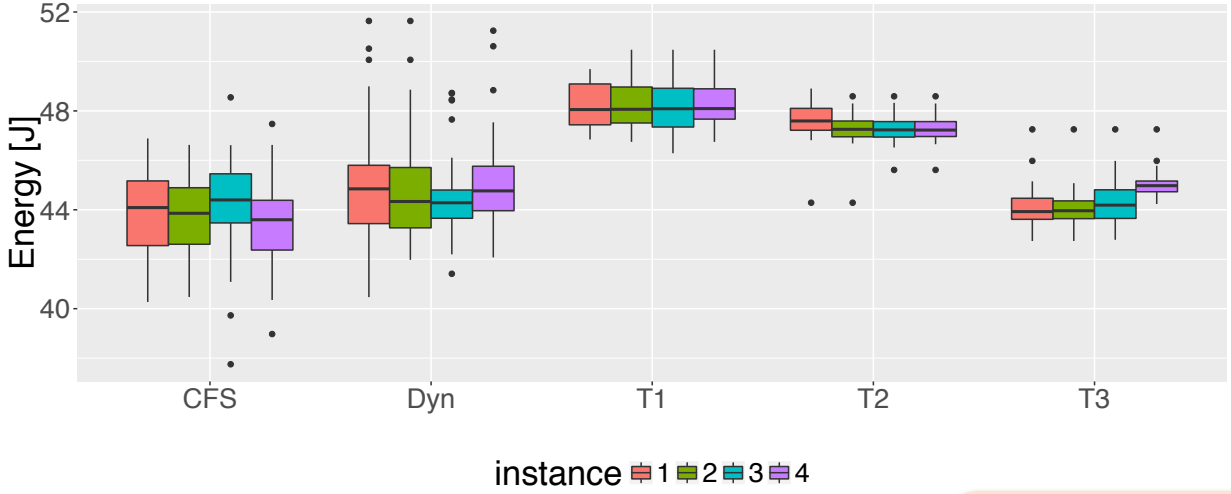


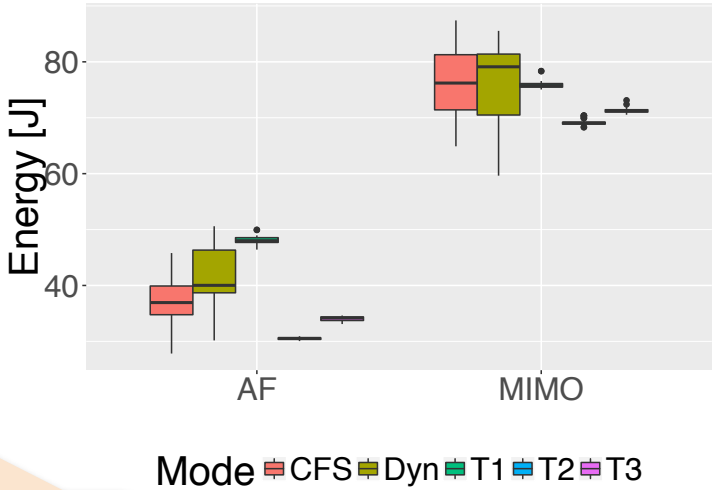
More predictable performance

Comparable performance to dynamic mapping

[SCOPES'17b]

Flexible mappings: Multi-application results (2)





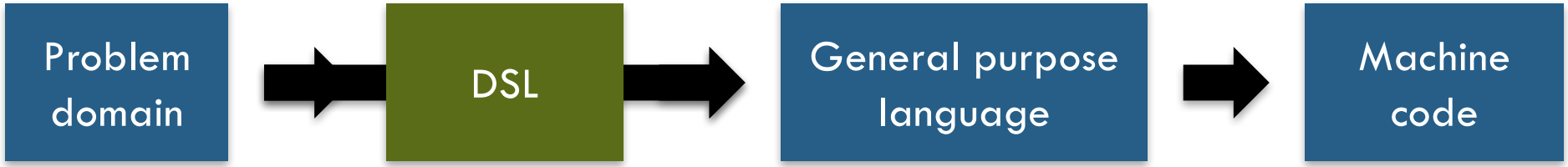
Better energy predictability as well

[SCOPES'17b]

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



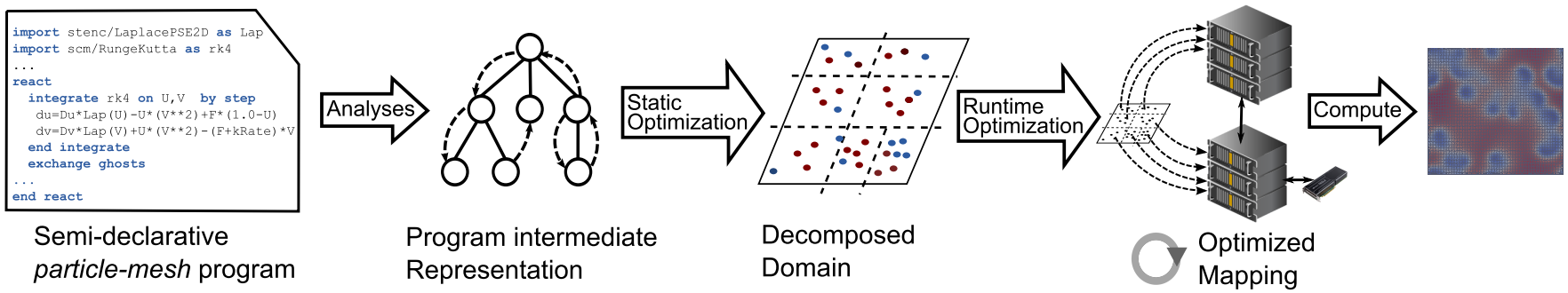
Adapted from lecture: "Concepts of Programming Languages", Eelco Visser, TU Delft

- Many quite successful DSLs today (Halide, Spiral, TVM, TensorFlow, …)

PPME: Parallel particle-mesh environment

- Domain: Python-based DSL for Particle-Mesh simulations
  - Parallel particle-mesh library (PPM): abstractions for **computational biology**
  - Collaboration with the mosaic group (http://mosaic.mpi-cbg.de/)

- Provide more information about structure, data-usage and computation patterns to the compiler for parallel execution



PPML: DSL Example

```
client grayscott
  integer, dimensior
  real(..), dimensic
  integer

  add_arg(k_rate,<#
  add_arg(F,<#real(r
  add_arg(D_u,<#real
  add_arg(D_v,<#real

  ppm_init(1)

  U = create_field(1
  V = create_field(1
  topo = create_topo
  c = create_particl

  allocate(displace(
  call random_numbe
  call c%move(displ
  call c%apply_bc(ir

  global_mapping(c,
  discretize(U,c)
  discretize(V,c)
  ghost_mapping(c)

  foreach p in parti
    U_p = 1.0_mk
    V_p = 0.0_mk
    if (((x_p(1)-0.5

      U_p = 0.5_mk
      V_p = 0.25_mk
    end if
  end foreach
```

ArithmeticOperation × | typeof_BinaryExpression × | typeof_AssignmentExpression ×

```
import LaplacePSE2D from stencils as Lap
import BcdefPeriodic from topologies as bcdef
import RungeKutta from schemes as rk4

module GrayScott
  external real kRate = 1.0;  "reaction rate"
  external real F;            "reaction parameter"
  external real Du = 1.0;     "diffusion constant of U"
  external real Dv = 1.0;     "diffusion constant of V"

  phase initialize
    << ... >>
  end initialize

  phase solve
    field <real , 1> U;
    field <real , 1> V;
    integer t;

    δU/δt = Du * ∇²U - U * V² + F * (1 - U);
    δV/δt = Dv * ∇²V + U * V² - (F + kRate) * V;
  end solve

  phase finalize
    << ... >>
  end finalize

end module GrayScott
```

```
._mk * c%h_avg#>)
.0_mk, 1.0_mk], "Lap")

am_op_dcpse,
der=>2,c=>1.0_mk])

ott_rhs,[U=>c,V], rk4)

h sca_fields(U,V,dU,dV)
 + F*(1.0_mk-U_p)
 - (F+k_rate)*V_p
```

[ACM TOMS'18]

CFDlang for computational fluid dynamics (CFD)

- Tensor expressions typically occur in numerical codes

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

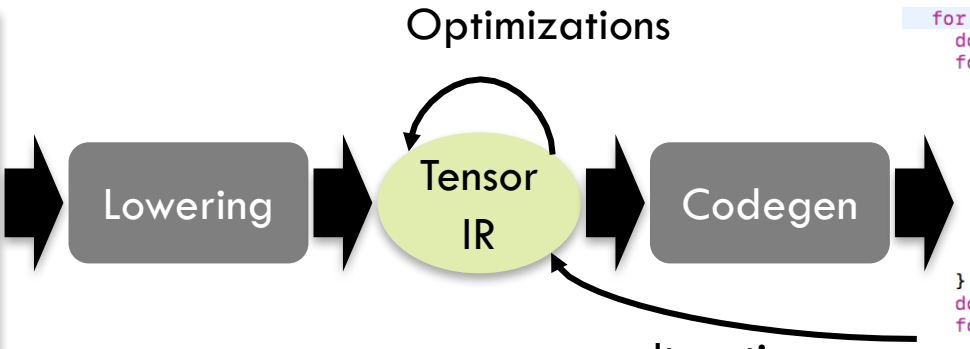
  - Tensor product notation popular in the CFD domain
- On performance
  - Matrixes are small, so libraries like BLAS do not always help
  - Expressions result in deeply nested for-loops
  - Performance highly depends on the *shape* of the loop nests
- Higher-level expressions: No need for complex polyhedral 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];
      }
    }
  }
}
```

[RWDSL'18]

Example: Interpolation operator

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

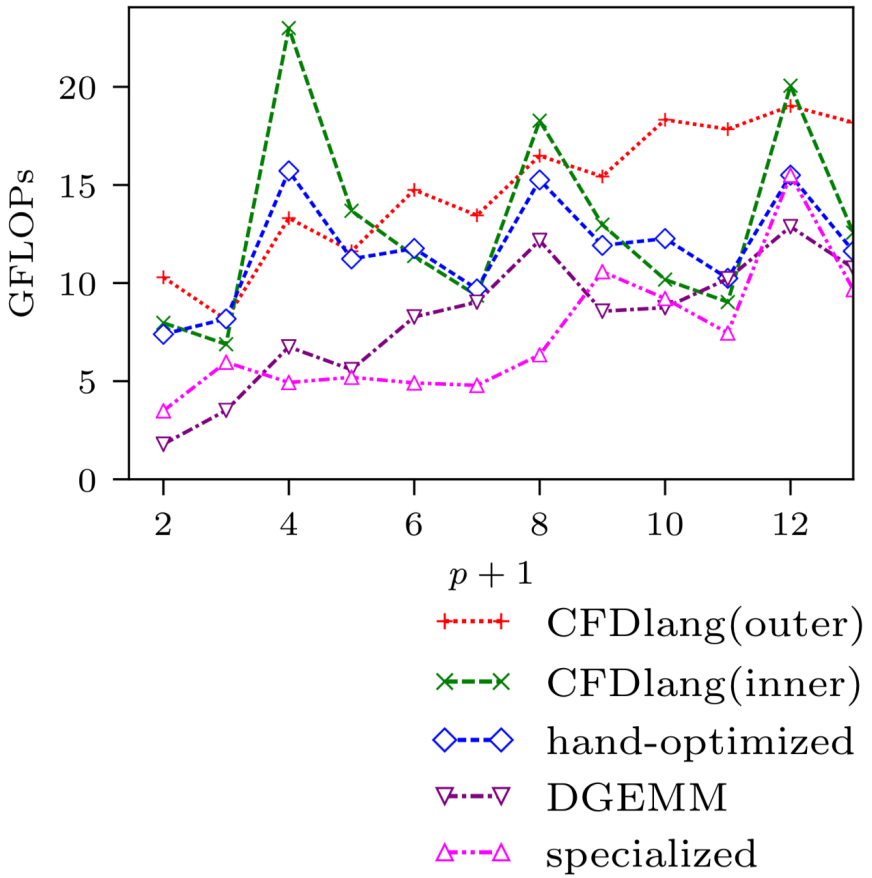
$$v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}$$

- 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: Meta-programming for Tensor Optimizations

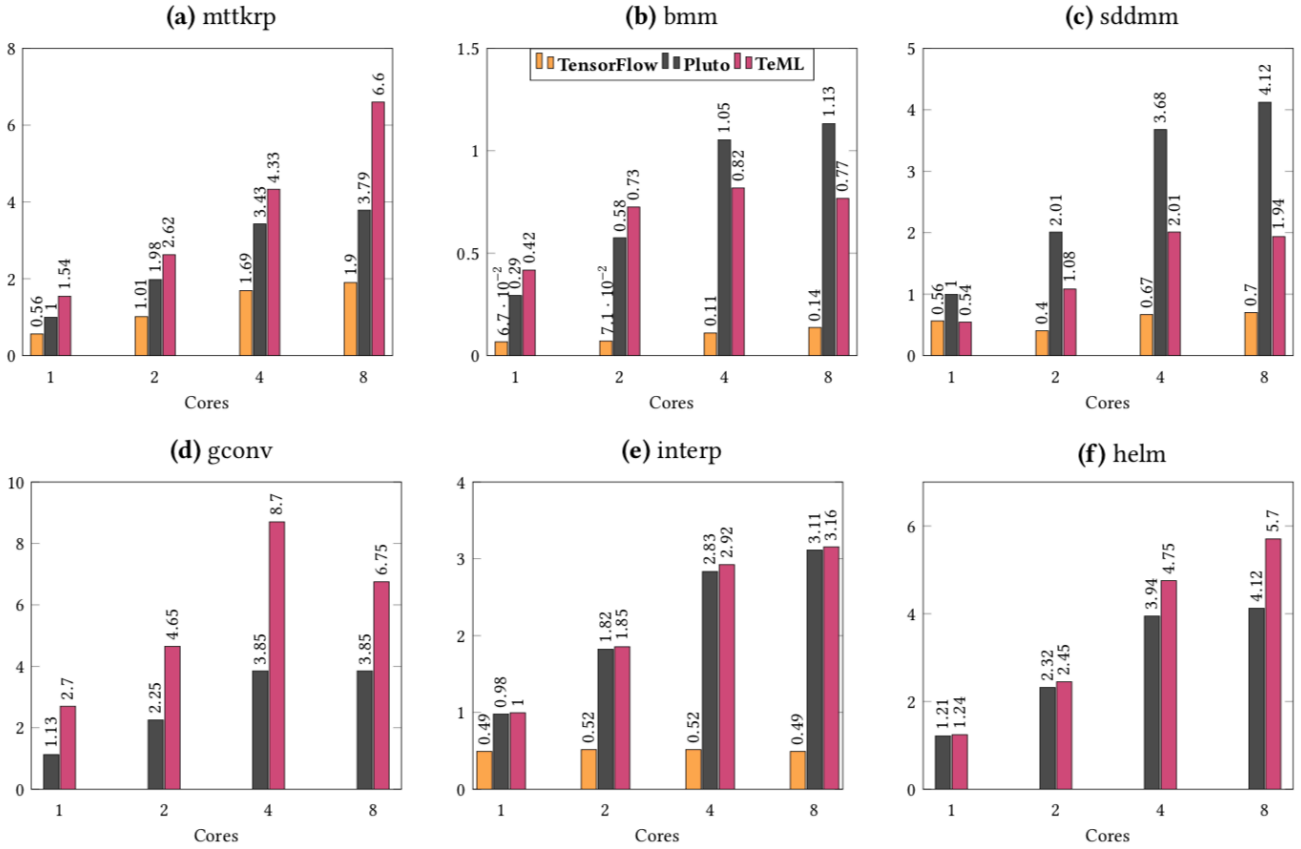
- Extend grammar and semantics with semantic-preserving, composable transformations
  - For optimization expert or for automated search

$$
\begin{aligned}
\langle Lexpression \rangle \quad ::= \quad & \texttt{build}\ (\langle id \rangle) \\
\mid \quad & \texttt{stripmine}\ (\langle id \rangle, \langle int \rangle, \langle int \rangle) \\
\mid \quad & \texttt{interchange}\ (\langle id \rangle, \langle int \rangle, \langle int \rangle) \\
\mid \quad & \texttt{fuse\_outer}\ (\langle id \rangle, \langle id \rangle, \langle int \rangle) \\
\mid \quad & \texttt{fuse\_inner}\ (\langle id \rangle, \langle int \rangle) \\
\mid \quad & \texttt{unroll}\ (\langle id \rangle, \langle int \rangle)
\end{aligned}
$$

[GPCE'18]

TeML: Results

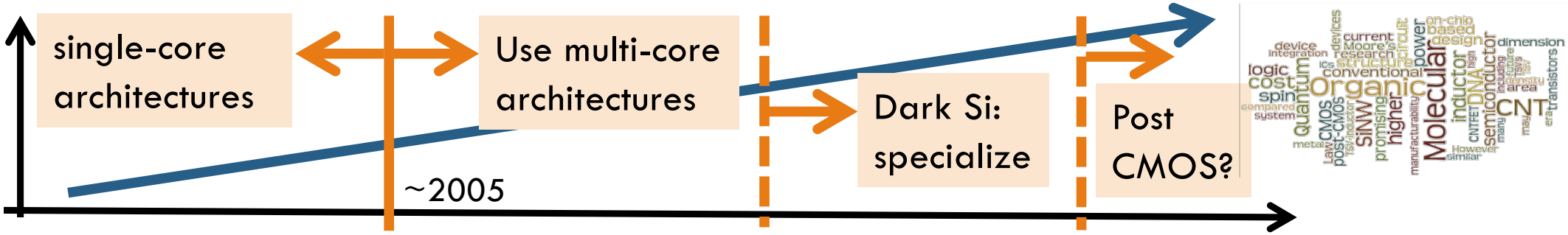
- Extra control allow for new optimization (vs pluto): changing shapes
- General tensor semantics allow covering more benchmarks than TensorFlow



[GPCE' 18]

Emerging technologies

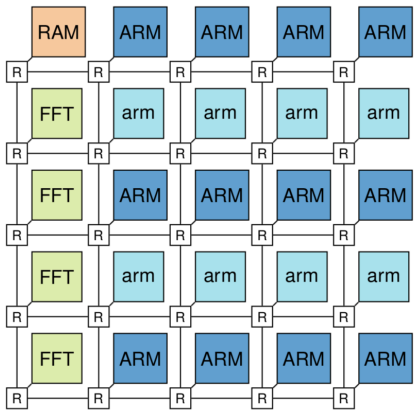
Specialization and post-CMOS

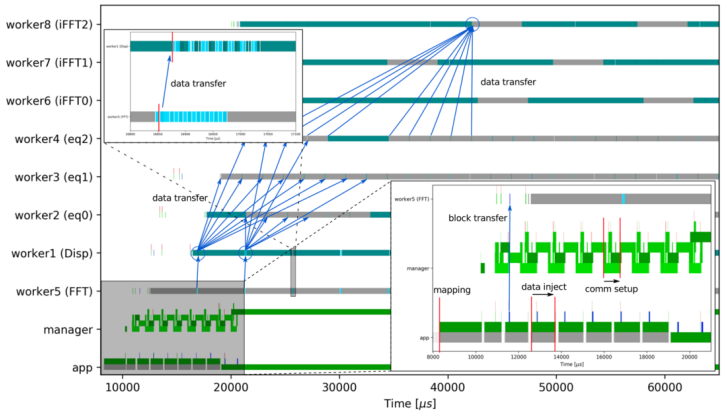


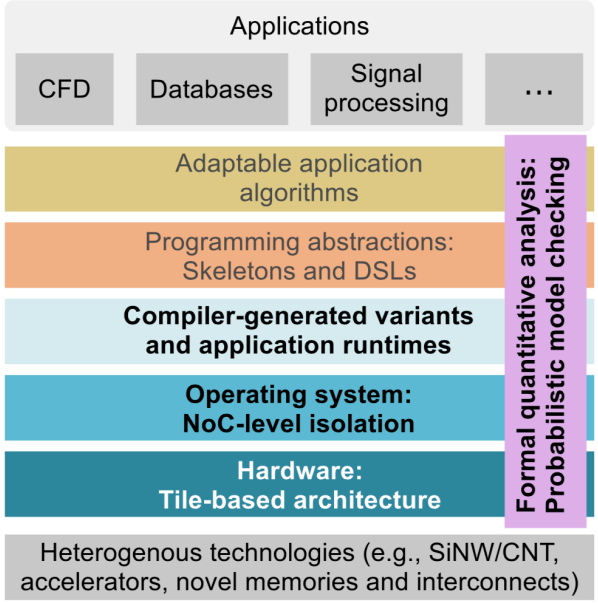
- Power of abstractions
  - Reasoning about schedules (dataflow), high-level type checks (particles) or data-layout and loop schedules (tensors)
  - Key to handle and optimize systems with emerging interconnect, memories and accelerators

Emerging techs. example: cfaed SW stack

- Excellence Cluster Cfaed
  - New devices/paradigms: Silicon nanowires, carbon nanotubes, organics, bio-inspired computing, …
  - Software: Domain-specific languages, micro-kernels, formal analysis methods, …
  - Evaluation: Extensive use of simulators



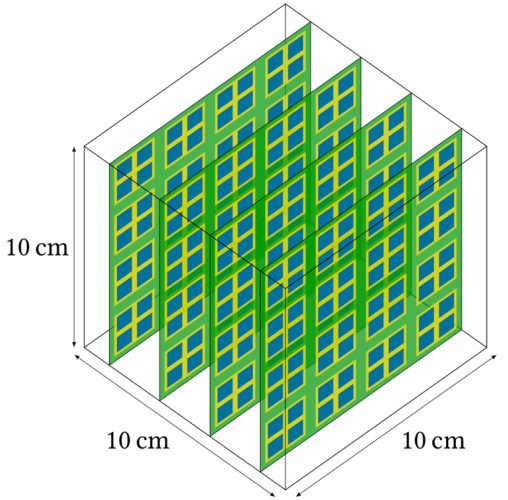


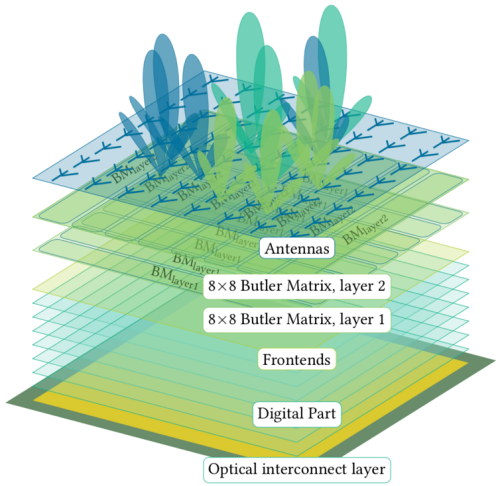


[SAMOS'17, IEEE TMSCS'18]

Emerging techs. example: HAEC Box

- HAEC CRC: Highly adaptive energy efficient computing
  - In board optical interconnect
  - Wireless interconnect in between chip-stacks
- Dataflow-based programming model
  - Rich set of options for data to flow
  - Target system for adapting mappings at runtime

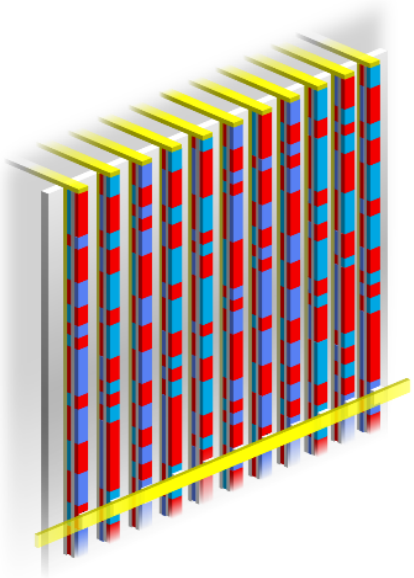




[IEEE Proc.'19]

Emerging techs. example: Racetrack memories

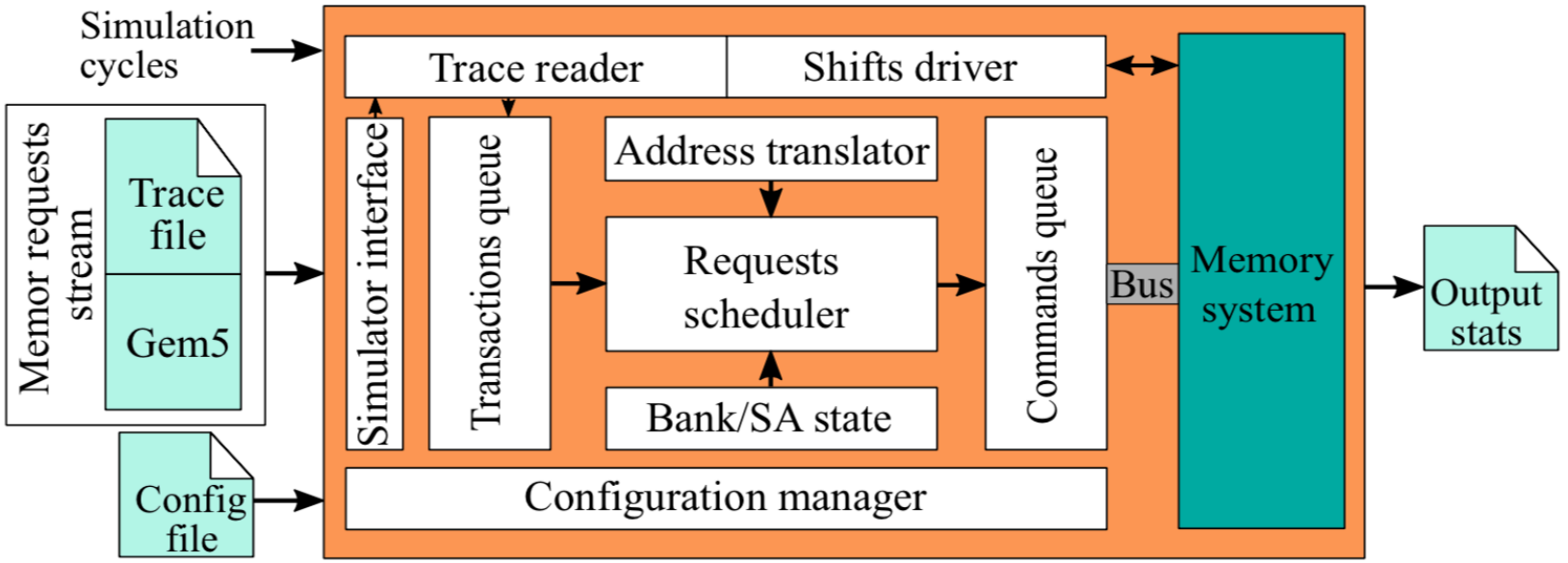
- Racetrack memories: one of many future alternatives
- Predicted extreme density at low latency
  - 3D nano-wires with magnetic domains
  - One port shared for many bits
  - Domains move at high speeds (1000 ms^{-1})
- Sequential: Game changer for current HW/SW stack
  - Memory management
  - Integration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



Parkin, Stuart, and See-Hun Yang. "Memory on the racetrack." Nature nanotechnology 10.3 (2015): 195.

Current research topics: Simulation

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

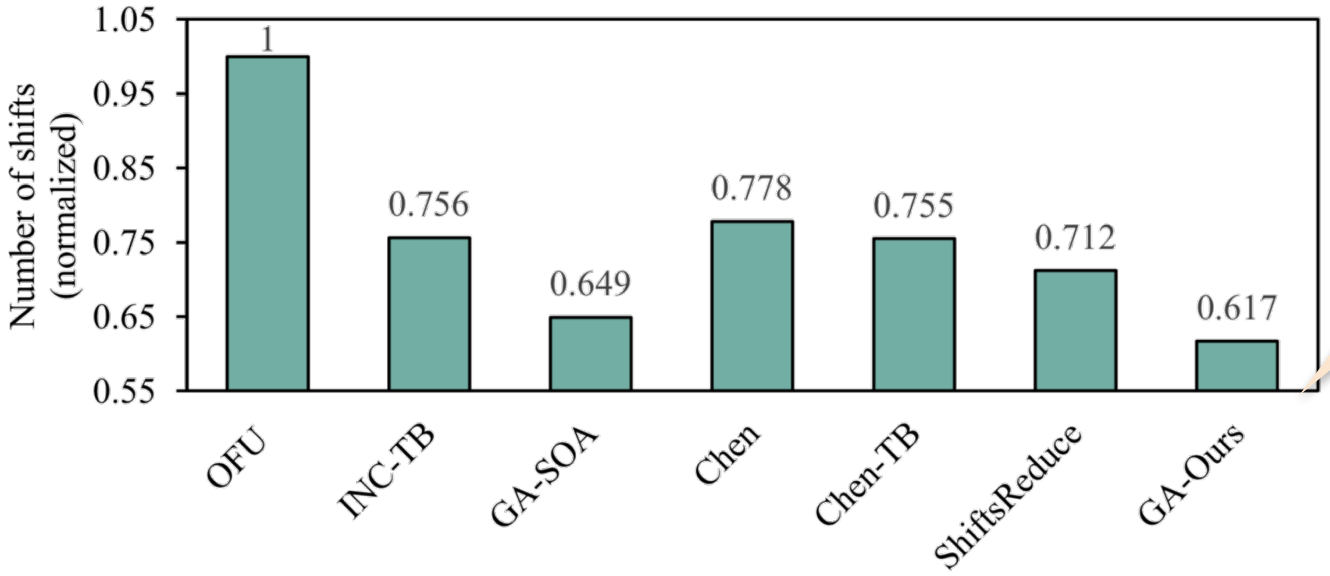


[IEEE CAL'19]

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

Current research topics: Data and instruction layout

- Compiler pass to reorganize data: Reduce the impact of shifting
  - Allocation of instructions, allocation of stack variables, allocation of tensors



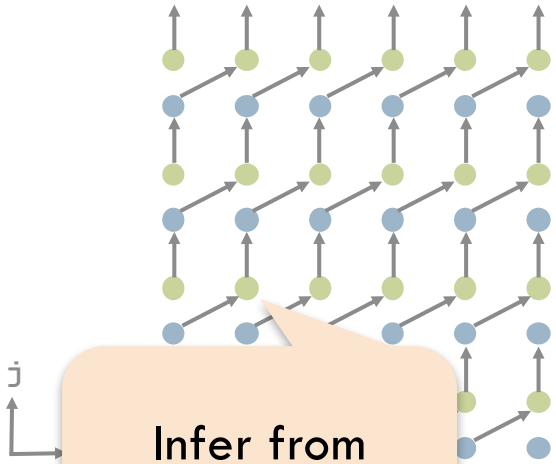
Average shift reduction across 28 benchmark applications

[ArXiv'19]

Discussion

Ease analysis

1) Find all dependencies?



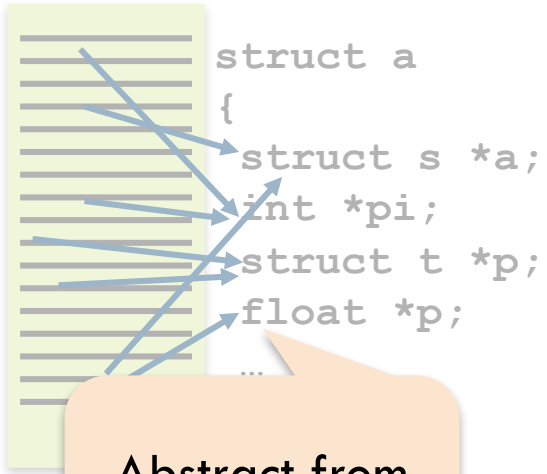
Infer from high-level spec

More abstract algorithms (→ declarative)

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    {
```

2) Coding style and the illusion of infinite shared memory



Abstract from actual layout

Summary

- Evolution of hardware always stressing the programming stack
  - Single, multi-core, many-core, heterogeneity (specialization), post-CMOS, …
- Domain-specific hardware not only for niches: The need for domain-specific programming abstractions
  - Dataflow for parallelism beyond embedded systems
  - DSLs to ease high-level manipulation and mapping of program constructs

References

[Edler-CC'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.

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

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

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

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

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

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

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

[ACM TOMS'18] Karol, S., Nett, T., Castrillon, J., Sbalzarini, I. F. "A Domain-Specific Language and Editor for Parallel Particle Methods", ACM Transactions on Mathematical Software (TOMS), ACM, 2018, 44, 34:1-34:32.

[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.

[SAMOS'17] Menard, C., Jung, M., Castrillon, J., Wehn, N. "System Simulation with gem5 and SystemC: The Keystone for Full Interoperability" SAMO'17, 62-69.

[Nature-nano15] Parkin, Stuart, and See-Hun Yang. "Memory on the racetrack." Nature nanotechnology 10.3 (2015): 195.

[IEEE CAL'19] Asif Ali Khan, Fazal Hameed, Robin Bläsing, Stuart Parkin, Jeronimo Castrillon, "RTSim: A Cycle-accurate Simulator for Racetrack Memories" (to appear), In IEEE Computer Architecture Letters, Feb 2019.

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

[IEEE TMSCS'18] Castrillon, J., Lieber, M., Klüppelholz, S., et al. "A Hardware/Software Stack for Heterogeneous Systems", IEEE Transactions on Multi-Scale Computing Systems, 2018, 4, 243-259.

[IEEE Proc.'19] Fettweis, G., Dörpinghaus, M., Castrillon, et al. "Architecture and Advanced Electronics Pathways Towards Highly Adaptive Energy-Efficient Computing", Proceedings of the IEEE, 2019, 107, 204-231.