CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

The role of domain-specific languages for cyber-physical systems

Jeronimo Castrillon

Chair for Compiler Construction (CCC)

TU Dresden, Germany

Seminar series: Design and Programming Cyber-Physical Systems and IoT applications

Virtual from Dresden, Germany. October 2020

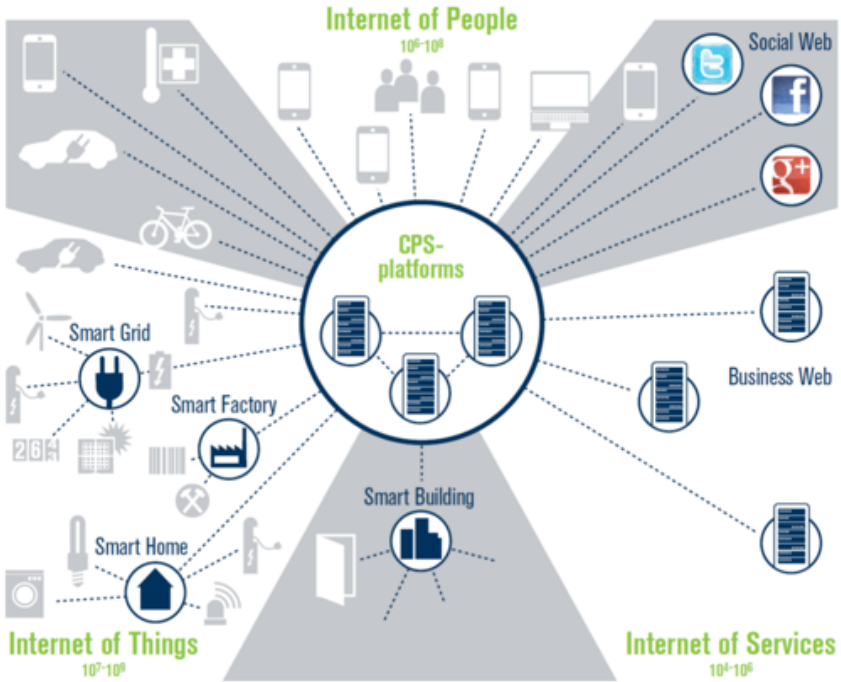
TECHNISCHE UNIVERSITÄT DRESDEN DRESDEN concept DFG WR WISSENSCHAFTSRAT

Cyber-physical systems

- Cyber-Physical Systems (CPS): Integration of computing with physical processes. Embedded computers monitor and control physical processes, usually with feedback loops (physical processes affect computations and vice versa)

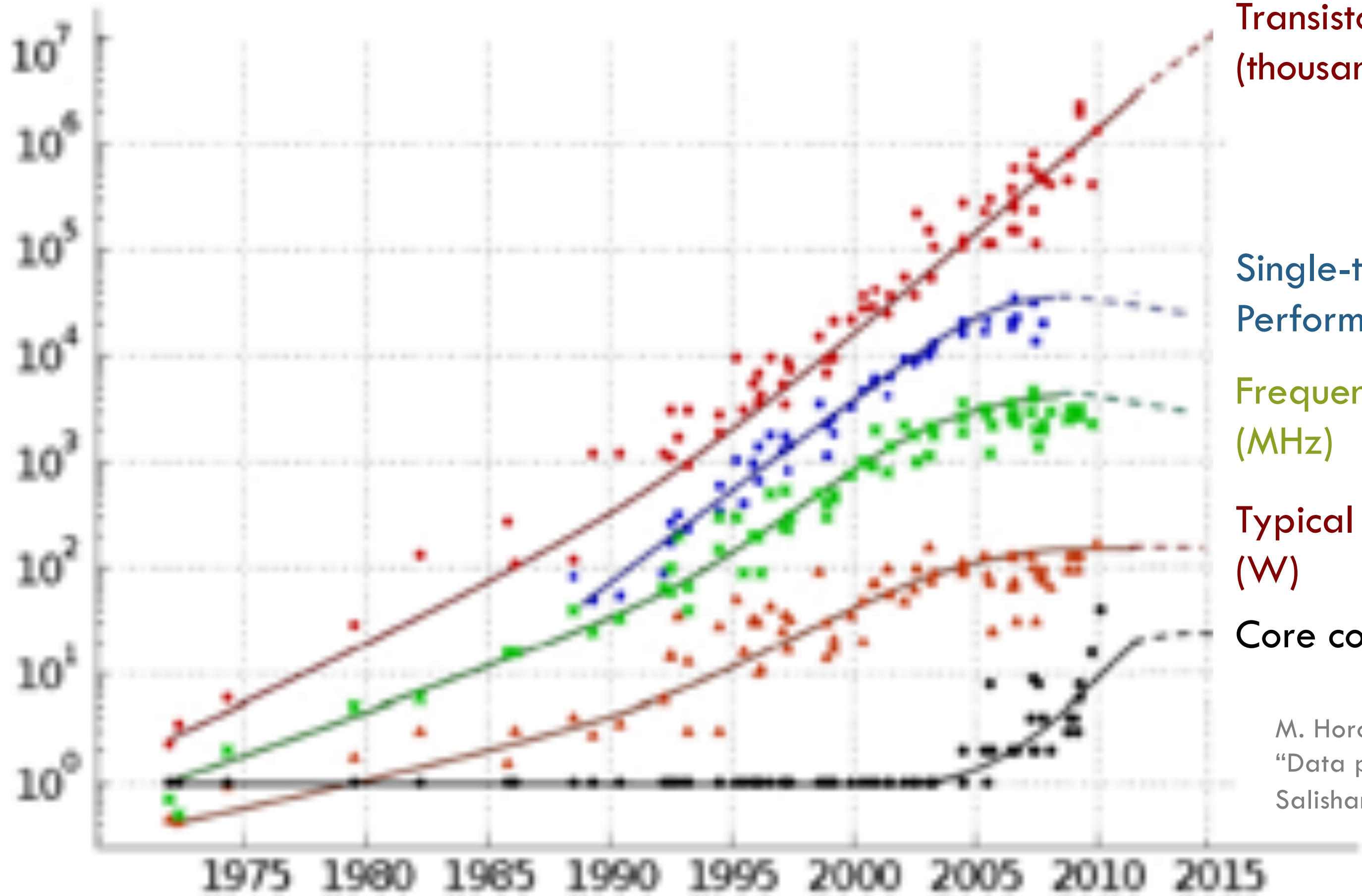
Edward A. Lee, “Cyber physical systems: Design challenge”. ISORC’08

- Special requirements
  - Reactivity
  - Adaptivity
  - Time Sensitivity
  - Safety Criticality
- Even more demanding computational power (inference, data processing, …)



http://www.imm.dtu.dk/~jbjo/cps.html

Evolution of computing



Transistors (thousands)

Single-thread Performance

Frequency (MHz)

Typical power (W)

Core count

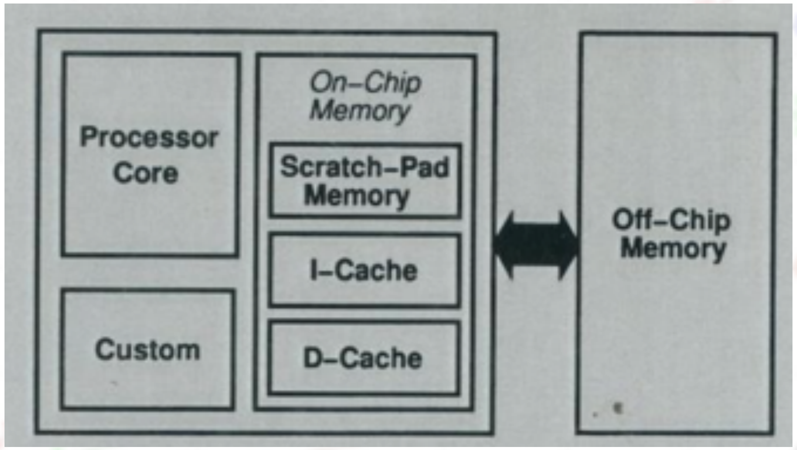
M. Horowitz, F. Labonte, et al. Dotted-line by C. Moore, "Data processing in exascale-class computer systems," The Salishan Conference on High Speed Computing, 2011

CHAIR FOR COMPILER CONSTRUCTION

Evolution of computing: Heterogeneity is mainstream

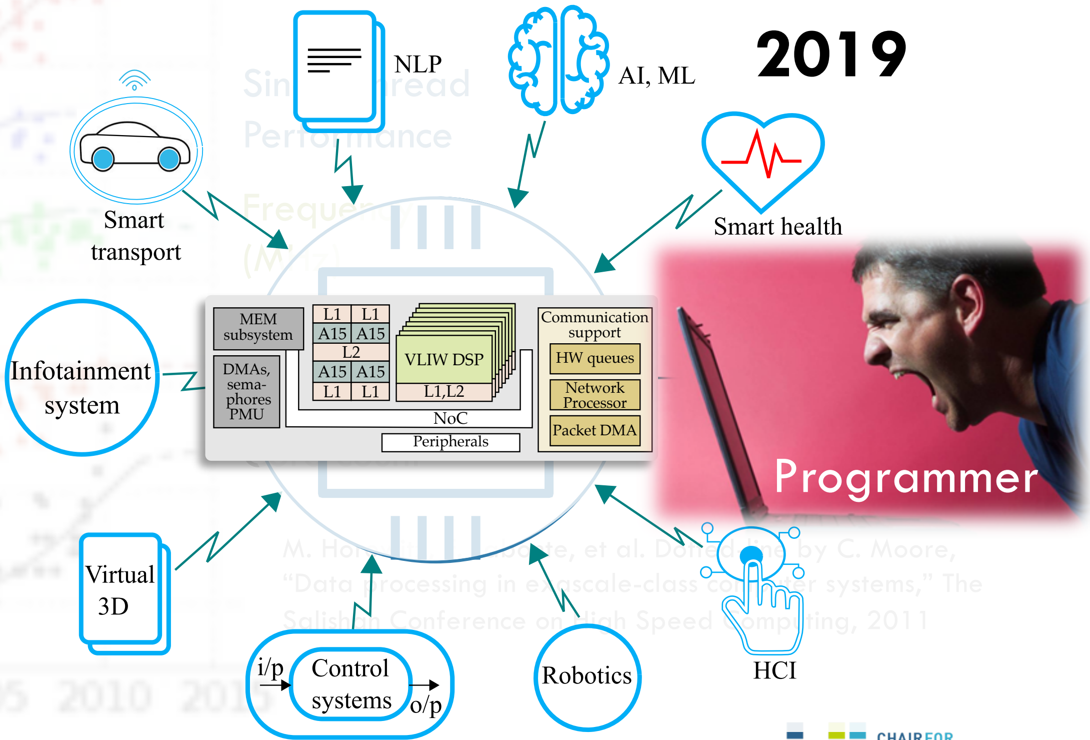
- Heterogeneous many-cores, scalable platforms, complex memory hierarchies, domain-specific accelerators, emerging technologies…
- Plus: Stringent application constraints

1999



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

2019



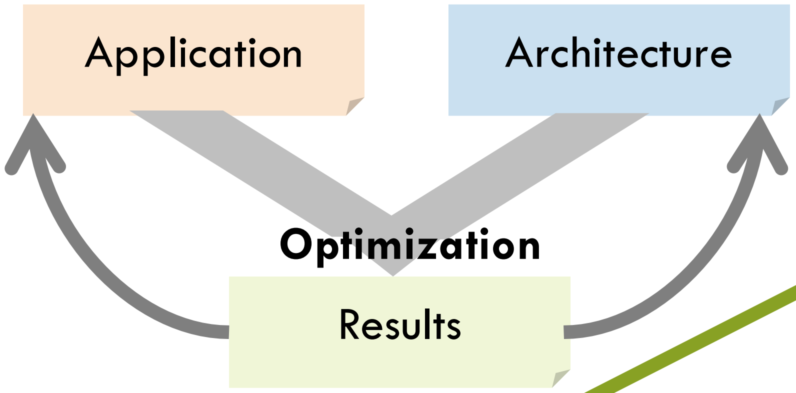
SoC programming: Evolution (required)

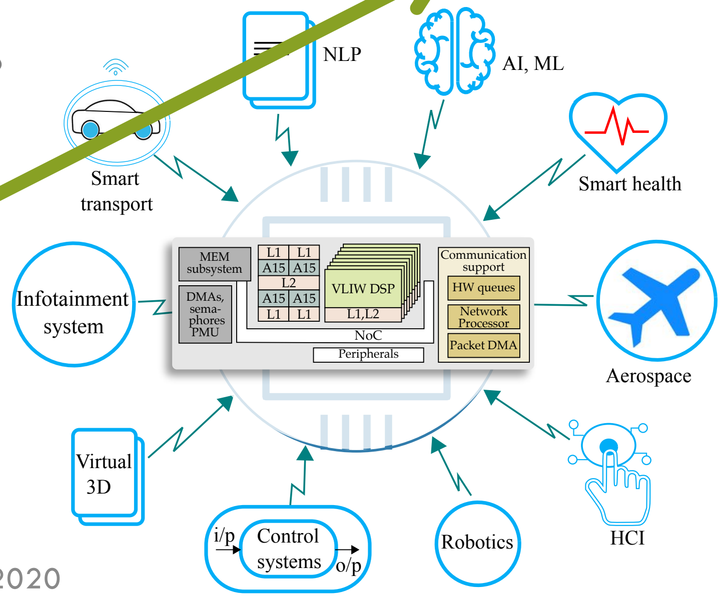
- Sequential: Auto-parallelization, pragmas, …
- Formal model-based code/HW generation
- Higher-level programming abstractions

```
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))
```

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

```
PnTransformSdfToKpn(D, S);
 PnTransformToArrayAccess(D, S);
 CollectChannelAccessRanges(D, S);
 PropagateChannelAccessRanges(D, S);
PnStreamFactory streamFactory(BasePath);
 switch (transTarget) {
 case TransMVP:
   PnTransformTemplateInstantiate(D, S);
   ErasePnProcessTemplates(D);
   PnPrintForMVP(D, S);
   break;
 case TransPthread:
   PnTransformPthreads(D, S, traces);
   ErasePnDefs(D);
   break;
 case TransSystemC:
   PrintForSystemC(D, S, traces, streamFactory);
   ErasePnDefs(D);
   break;
 case TransVPUtg:
   PrintForVPUtg(D, S, streamFactory);
   ErasePnDefs(D);
   break;
 case TransVPUmap:
   PrintForVPUmap(D, S, strMappingFileName, streamFactory);
   ErasePnDefs(D);
   break;
 case TransInvalid:
   assert(false);
   break;
 }
}
```

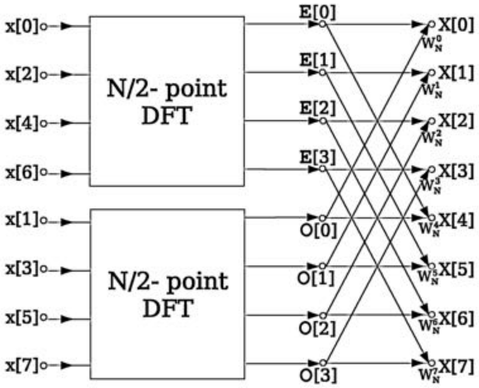




Languages, tools & frameworks

- Heterogeneity not for nices
  - Embedded expert wouldn't expect compiler to recognize an FFT written in C!

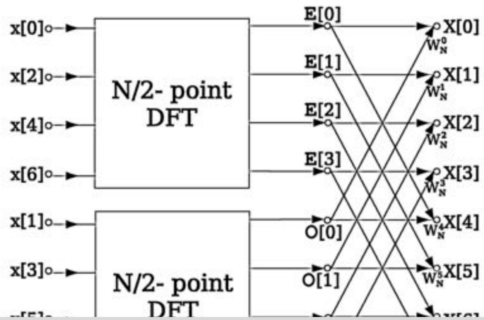
```
void fft(CArray &x)
{
        // DFT
        unsigned int N = x.size(), k = N, n;
        double thetaT = 3.14159265358979323846264338328L / N;
        Complex phiT = Complex(cos(thetaT), -sin(thetaT)), T;
        while (k > 1)
        {
                n = k;
                k >>= 1;
                phiT = phiT * phiT;
                T = 1.0L;
                for (unsigned int l = 0; l < k; l++)
                {
                        for (unsigned int a = l; a < N; a += n)
                        {
                                unsigned int b = a + k;
                                Complex t = x[a] - x[b];
                                x[a] += x[b];
                                x[b] = t * T;
                        }
                        T *= phiT;
                }
        }
        // Decimate
        unsigned int m = (unsigned int)log2(N);
        for (unsigned int a = 0; a < N; a++)
        {
                unsigned int b = a;
                // Reverse bits
                b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
                b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
                b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
                b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
```



Wikipedia

Languages, tools & frameworks

- Heterogeneity not for nices
  - Embedded expert wouldn't expect compiler to recognize an FFT written in C!
- In CPS and computing in general
  - Changing HW substrate
  - Wider range of programmer backgrounds



- **Tools, methodologies and frameworks more important than ever!**
- **High-level tools:** Select the right abstraction when possible
  - More optimization, stronger semantics
  - Domain-specific SW for domain-specific HW
- **Low-level tools:** Legacy, expert coders & target of high-level flows

Background

Languages as abstractions

- Languages evolve, formalizing powerful design patterns (abstractions)
  - Some of them too common, so we do not notice it

- Examples
  - From calling conventions to procedures

Source: https://en.wikipedia.org/wiki/Calling_convention

```
calc:
  push EBP ; save old frame pointer
  mov  EBP,ESP ; get new frame pointer
  sub ESP,localsize ; reserve place for locals
  ... ; perform calculations, leave result in EAX
  mov ESP,EBP ; free space for locals
  pop EBP ; restore old frame pointer
  ret paramsize ; free parameter space and return
```

```
f(x) {...}
```

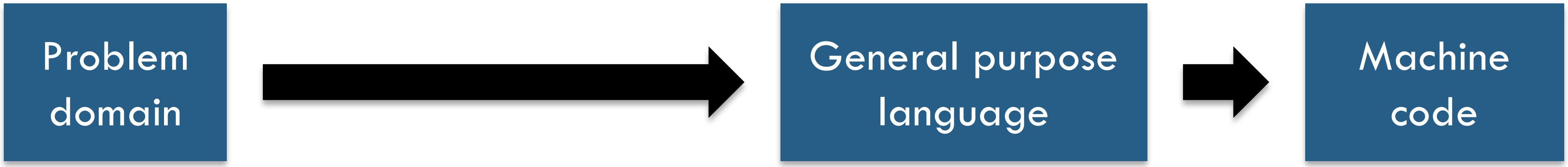
Languages as abstractions

- Languages evolve, formalizing powerful design patterns (abstractions)
  - Some of them too common, so we do not notice it

- Examples
  - From instructions to expressions
  - From calling conventions to procedures
  - From label-goto to structured control flow
  - From memory layout to data types (arrays, structs, …)
  - Memory allocation/deallocation (new/delete, garbage collection)
  - From function pointers and tables to dynamic dispatch
  - …

Domain specific languages (DSLs)

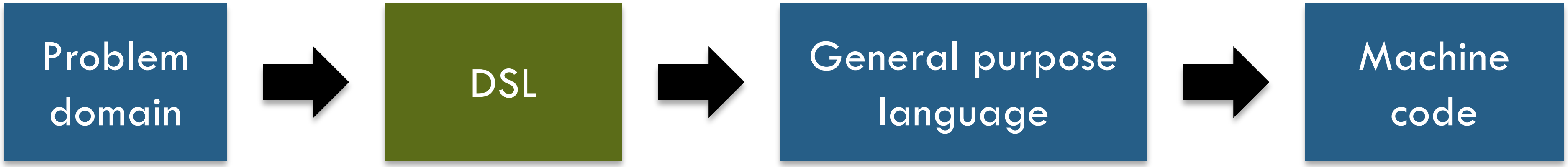
- DSLs help bridge the gap between problem domain and general purpose languages



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

Domain specific languages (DSLs)

- DSLs help bridge the gap between problem domain and general purpose languages



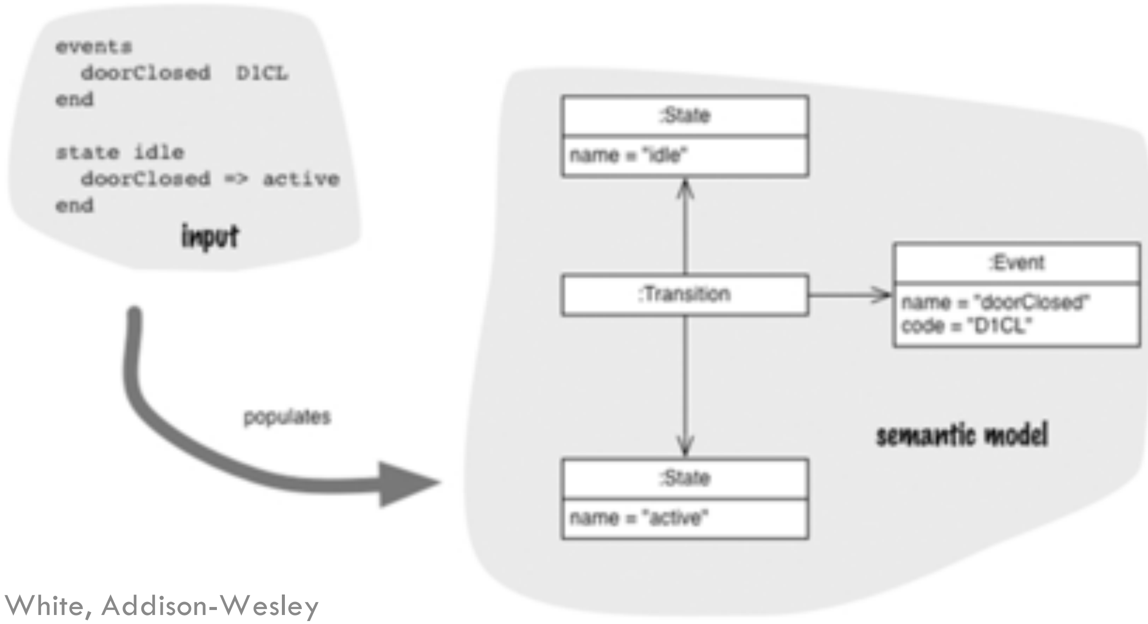
Adapted from lecture: "Concepts of Programming Languages", Eelco Visser, TU Delft

  - Natural vocabulary for concepts are fundamental to problem domain
  - Faster way to write common concepts (**concrete syntax**)
  - Optimization potential due to domain-specific information
  - A DSL can be disguised as library or framework

DSLs and semantic model

- DSLs offer a way of manipulating an abstraction (or **semantic model**)
  - Other example of abstraction: APIs
  - A DSL can be evolve from an API: more flexible manipulation

When defining a DSL, the hardest and most important part is the definition of the semantic model (the rest is engineering)



Source: "Domain-Specific Languages" by Martin Fowler, Terry White, Addison-Wesley Professional. September 23, 2010. Print ISBN-10: 0-321-71294-3

Basics of PL: Formal (dynamic) semantics

- Relation between syntax (e.g., as context-free grammar), states and values
  - States: Can be seen as the state of the machine it runs on
  - Values: Actual values of symbols during execution

Quite complex (useless?) for C++, but great asset for DSLs!

- **Operational** semantics
  - Set of inference rules: Describe how syntactic constructs update the state
  - Small (detailed via transitions systems) vs Big (fewer transitions in **derivation tree**)
  - Commonly based on lambda-calculus
- **Denotational** semantics
  - Direct notation: meaning provided by functional style (aka state transformers)
  - Compositionality: Program execution as the composition of functions

Operational semantics (nutshell)

- General form
  - If I can prove that expression **E** in state **s** evaluates to value **V**
  - Then program **L := E** in that state will update the state, giving **L** the value **V**
- Simple, though complex notation, enables analysis in DSLs

$$\frac{\langle E, s\rangle \Rightarrow V}{\langle L := E\,,\ s\rangle \longrightarrow (s \uplus (L \mapsto V))}$$

$[\text{ass}_{\text{ns}}]$ $$\langle x := a,\ s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$[\text{skip}_{\text{ns}}]$ $$\langle \texttt{skip},\ s\rangle \rightarrow s$$

$[\text{comp}_{\text{ns}}]$ $$\frac{\langle S_1,\ s\rangle \rightarrow s',\ \langle S_2,\ s'\rangle \rightarrow s''}{\langle S_1;S_2,\ s\rangle \rightarrow s''}$$

$[\text{if}^{\text{tt}}_{\text{ns}}]$ $$\frac{\langle S_1,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$[\text{if}^{\text{ff}}_{\text{ns}}]$ $$\frac{\langle S_2,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$[\text{while}^{\text{tt}}_{\text{ns}}]$ $$\frac{\langle S,\ s\rangle \rightarrow s',\ \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \rightarrow s''}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s''} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$[\text{while}^{\text{ff}}_{\text{ns}}]$ $$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s \ \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

http://softlang.wikidot.com/rlaemmel:home

Big-step operational semantics: Derivation tree

- Using *asign* and *compose*
- Program:
  - `z:=x; x:=y; y:=z`
  - Initial state: x=5, y=7, z=0

$$[\text{ass}_{\text{ns}}] \quad \langle x := a,\ s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{ns}}] \quad \langle \textbf{skip},\ s\rangle \rightarrow s$$

$$[\text{comp}_{\text{ns}}] \quad \frac{\langle S_1,\ s\rangle \rightarrow s',\ \langle S_2,\ s'\rangle \rightarrow s''}{\langle S_1; S_2,\ s\rangle \rightarrow s''}$$

$$\cfrac{\cfrac{\langle \text{z:=x},\ s_0\rangle \rightarrow s_1 \qquad \langle \text{x:=y},\ s_1\rangle \rightarrow s_2}{\langle \text{z:=x; x:=y},\ s_0\rangle \rightarrow s_2} \qquad \langle \text{y:=z},\ s_2\rangle \rightarrow s_3}{\langle \text{z:=x; x:=y; y:=z},\ s_0\rangle \rightarrow s_3}$$

Derivation tree

Transition = big step

$$\begin{aligned}
s_0 &= [\text{x}\mapsto 5,\ \text{y}\mapsto 7,\ \text{z}\mapsto 0]\\
s_1 &= [\text{x}\mapsto 5,\ \text{y}\mapsto 7,\ \text{z}\mapsto 5]\\
s_2 &= [\text{x}\mapsto 7,\ \text{y}\mapsto 7,\ \text{z}\mapsto 5]\\
s_3 &= [\text{x}\mapsto 7,\ \text{y}\mapsto 5,\ \text{z}\mapsto 5]
\end{aligned}$$

http://softlang.wikidot.com/rlaemmel:home

Denotational semantics

- (recall) Direct notation: meaning provided by functional style
  - Semantic domains
    - Store transformation: $storeT = store \nrightarrow store$
    - Store observation: $storeO = store \nrightarrow value$
  - Semantic functions: mapping from syntax to semantics
    - Semantics of statements $\mathcal{S} : stmt \rightarrow storeT$
    - Semantics of expressions $\mathcal{E} : expr \rightarrow storeO$

Denotational semantics (2)

- Semantic functions

$$\mathcal{S} : stmt \rightarrow storeT$$
$$\mathcal{E} : expr \rightarrow storeO$$

- Example
  - Underlined functions are *semantic combinators:* combine meanings, not syntax

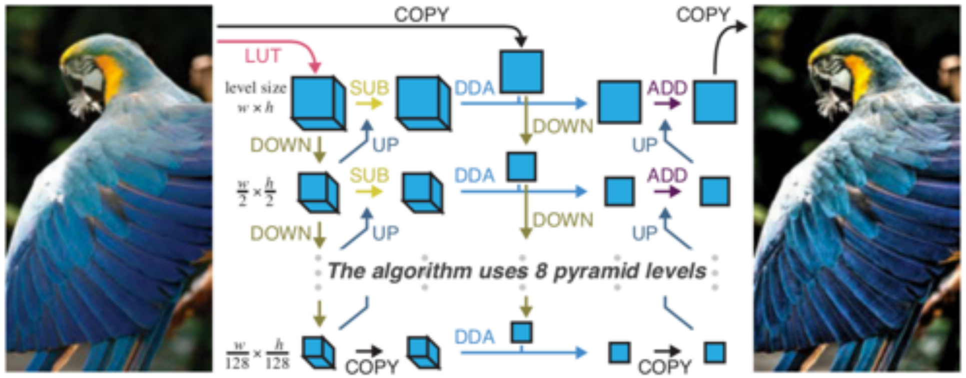
$$
\begin{aligned}
\mathcal{S}[\![\mathsf{skip}]\!] &= \underline{skip} \\
\mathcal{S}[\![\mathsf{assign}(x,e)]\!] &= \underline{assign}\ x\ (\mathcal{E}[\![e]\!]) \\
\mathcal{S}[\![\mathsf{seq}(s_1,s_2)]\!] &= \underline{seq}\ (\mathcal{S}[\![s_1]\!])\ (\mathcal{S}[\![s_2]\!]) \\
\mathcal{S}[\![\mathsf{if}(e,s_1,s_2)]\!] &= \underline{if}\ (\mathcal{E}[\![e]\!])\ (\mathcal{S}[\![s_1]\!])\ (\mathcal{S}[\![s_2]\!]) \\
\mathcal{S}[\![\mathsf{while}(e,s)]\!] &= \underline{while}\ (\mathcal{E}[\![e]\!])\ (\mathcal{S}[\![s]\!])
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[\![\mathsf{intconst}(i)]\!] &= \underline{intconst}\ i \\
\mathcal{E}[\![\mathsf{var}(x)]\!] &= \underline{var}\ x \\
\mathcal{E}[\![\mathsf{unary}(o,e)]\!] &= \underline{unary}\ o\ (\mathcal{E}[\![e]\!]) \\
\mathcal{E}[\![\mathsf{binary}(o,e_1,e_2)]\!] &= \underline{binary}\ o\ (\mathcal{E}[\![e_1]\!])\ (\mathcal{E}[\![e_2]\!])
\end{aligned}
$$

http://softlang.wikidot.com/rlaemmel:home

Example 1: Halide

- DSL for image processing pipelines
  - Composition of multiple stensils
- Abstraction
  - No explicit loops
  - Declarative approach to define filters as operation between functions
  - Functions: map coordinates to pixels, i.e., f(i,j) returns the pixel at position i,j




```
UniformImage in(UInt(8), 2)
Var x, y
Func blurx(x,y) =  in(x-1,y) +    in(x,y) +    in(x+1,y)
Func out(x,y) = blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)
```

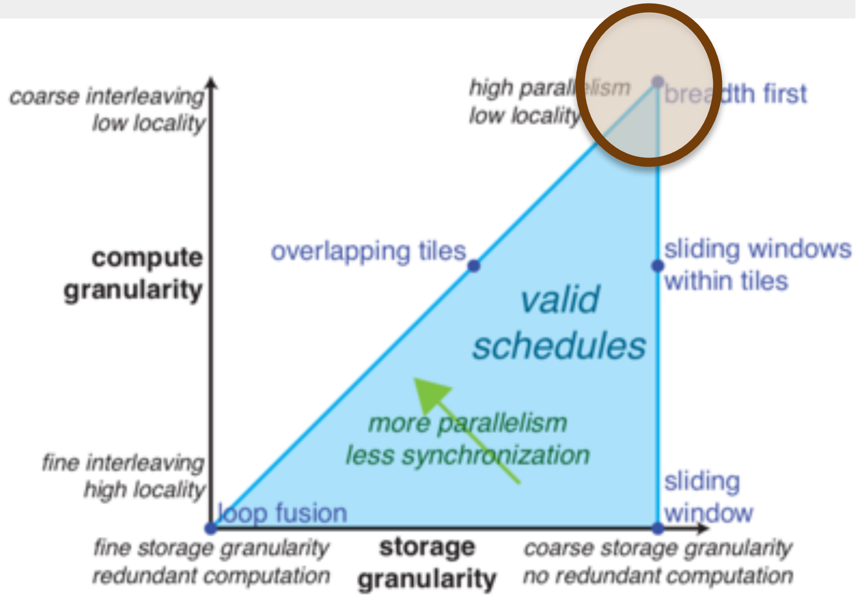
Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. SIGPLAN Not. 48, 6 (June 2013), 519-530.

Halide: The power of abstraction

- Automatically play tradeoffs: storage+compute

```
UniformImage in(UInt(8), 2)
Var x, y
Func blurx(x,y) =  in(x-1,y) +    in(x,y) +    in(x+1,y)
Func out(x,y) = blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)
```

```
alloc blurx[2048][3072]
for each y in 0..2048:
  for each x in 0..3072:
    blurx[y][x] = in[y][x-1] + in[y][x] + in[y][x+1]
alloc out[2046][3072]
for each y in 1..2047:
  for each x in 0..3072:
    out[y][x]=blurx[y-1][x] + blurx[y][x] + blurx[y+1][x]
```



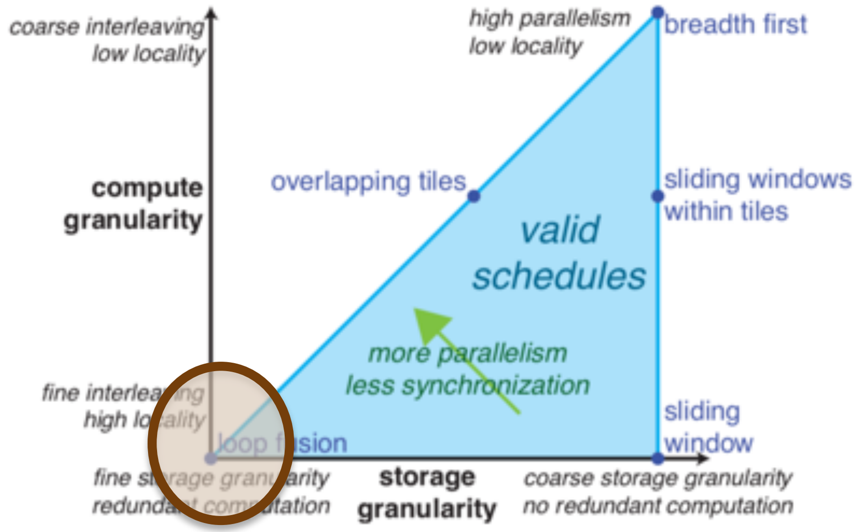
Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. SIGPLAN Not. 48, 6 (June 2013), 519-530.

Halide: The power of abstraction

- Automatically play tradeoffs: storage+compute

```
UniformImage in(UInt(8), 2)
Var x, y
Func blurx(x,y) =  in(x-1,y) +    in(x,y) +    in(x+1,y)
Func out(x,y) = blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)
```

```
alloc out[2046][3072]
for each y in 1..2047:
  for each x in 0..3072:
    alloc blurx[-1..1]
    for each i in -1..1:
      blurx[i]= in[y-1+i][x-1]+in[y-1+i][x]+in[y-1+i][x+1]
    out[y][x] = blurx[0] + blurx[1] + blurx[2]
```

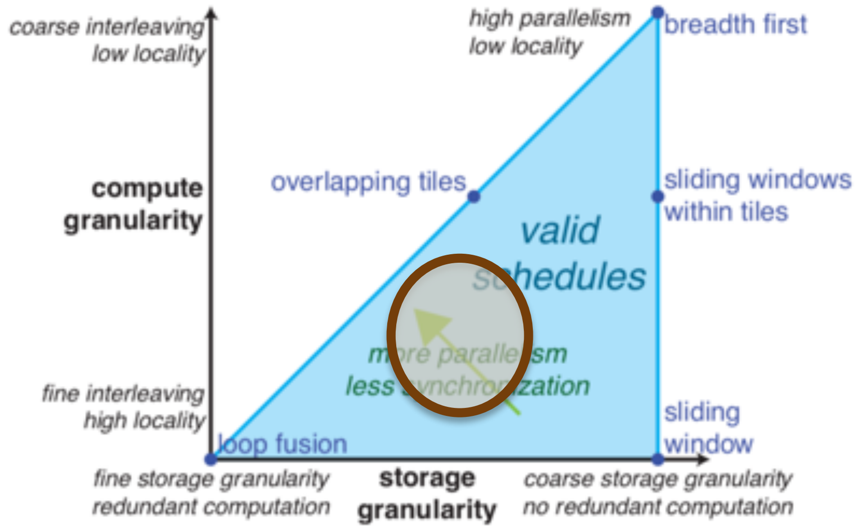


Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. SIGPLAN Not. 48, 6 (June 2013), 519-530.

Halide: The power of abstraction

- Automatically play tradeoffs: storage+compute

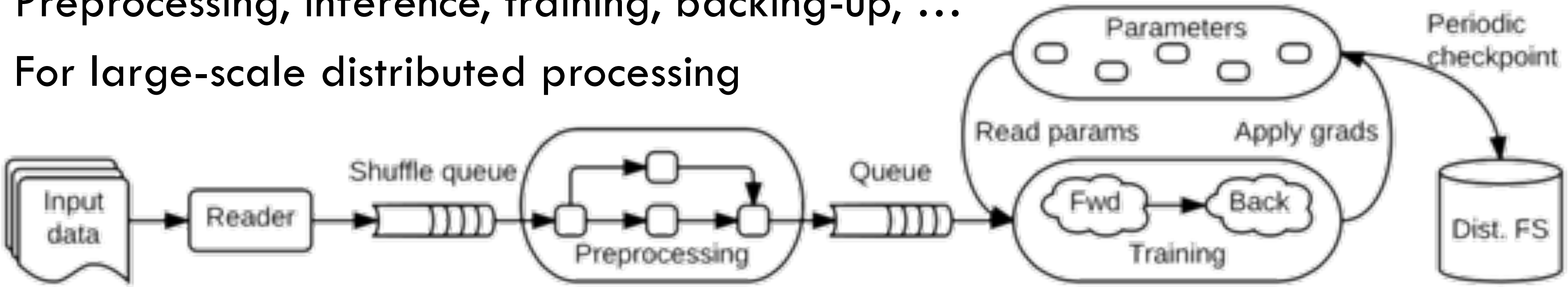
```
alloc out[2046][3072]
for each ty in 0..2048/32:
  for each tx in 0..3072/32:
    alloc blurx[-1..33][32]
    for y in -1..33:
      for x in 0..32:
        blurx[y][x] = in[ty*32+y][tx*32+x-1]
                    + in[ty*32+y][tx*32+x]
                    + in[ty*32+y][tx*32+x+1]
    for y in 0..32:
      for x in 0..32:
        out[ty*32+y][tx*32+x] = blurx[y-1][x]
                              + blurx[y  ][x]
                              + blurx[y+1][x]
```



Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. SIGPLAN Not. 48, 6 (June 2013), 519-530.

Example 2: Tensorflow

- Dataflow representation for the entire machine learning process
  - Preprocessing, inference, training, backing-up, …
  - For large-scale distributed processing



- On top of pure dataflow
  - Allow controlled global state (parameters) → vertices can modified shared state
  - Explicit special queues (with known access patterns)
  - Edges are tensors (multi-dimensional arrays)
  - Include symbolic differentiation for training

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

Tensorflow: API

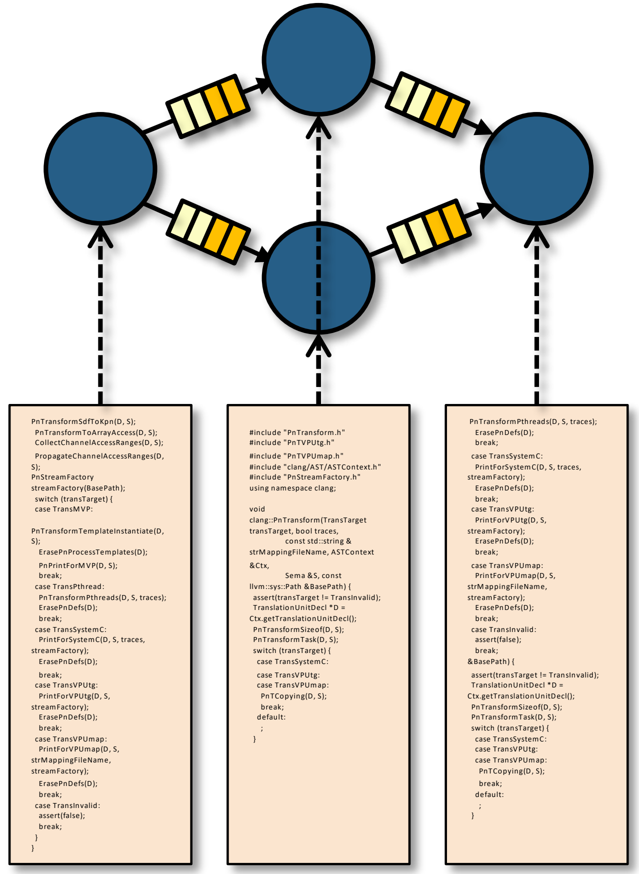
```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x1
b_1 = tf.Variable(tf.zeros([100]))
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10
b_2 = tf.Variable(tf.zeros([10]))                  # 10-ele
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

High-level known tensor operations

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

MoC-based programming languages

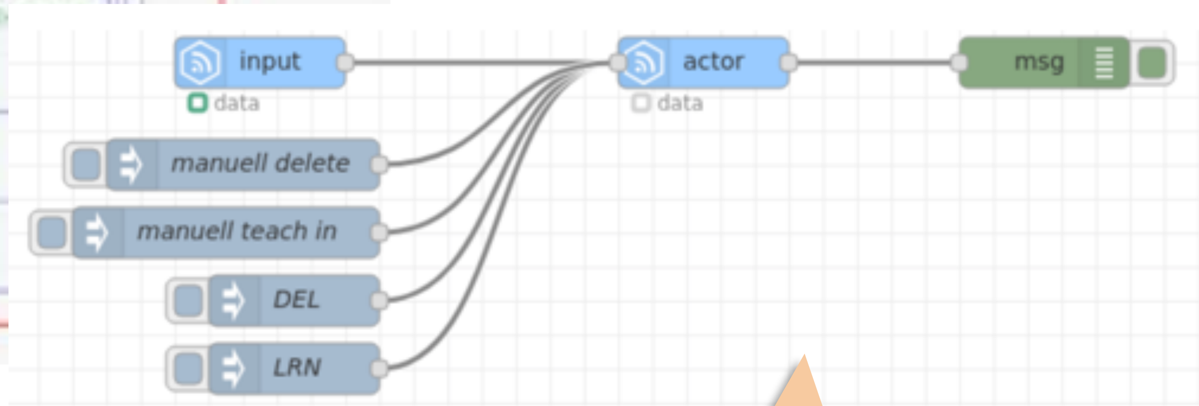
- Models of computation (MoCs) define components and rules of how they interact
- For programming, MoCs also define possible transitions a system may follow
- Many examples: Synchronous dataflow, Khan Process Networks, Reactors, … (see lecture by **Prof. Edward Lee**)



Graphical programmign environments

LabVIEW multi-rate diagram

M. Danneberg, N. Michailow, I. Gaspar, D. Zhang and G. Fettweis, "Flexible GFDM Implementation in FPGA with Support to Run-Time Reconfiguration," (VTC2015-Fall



Node-RED (actors, devices, …)

https://flows.nodered.org

For ENSI students: Talk to Chadlia about this

Example 3: C for process networks

- FIFO Channels

```
typedef struct { int i; double d; } my_struct_t;
__PNchannel my_struct_t S;
__PNchannel int A = {1, 2, 3}; /* Initialization */
__PNchannel short C[2], D[2], F[2], G[2];
```

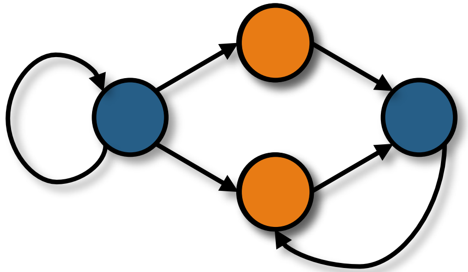
- Processes & networks

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

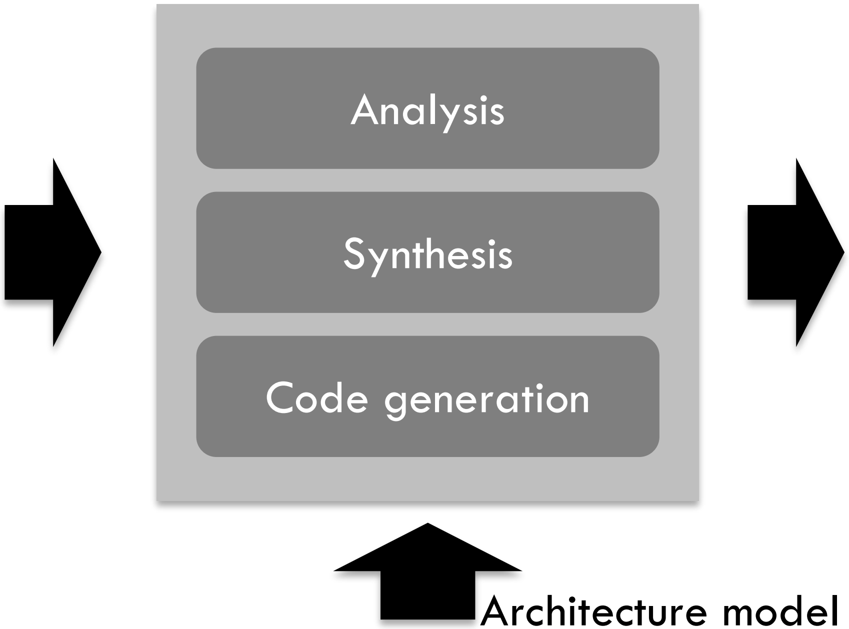
```
__PNkpn AudioAmp __PNin(short A[2]) __PNout(short B[2])
                 __PNparam(short boost){
  while (1)
    __PNin(A) __PNout(B) {
      for (int i = 0; i < 2; i++)
        B[i] = A[i]*boost;
    }}
__PNprocess Amp1 = AudioAmp __PNin(C) __PNout(F) __PNparam(3);
__PNprocess Amp2 = AudioAmp __PNin(D) __PNout(G) __PNparam(10);
```

CPN compiler and tool flow

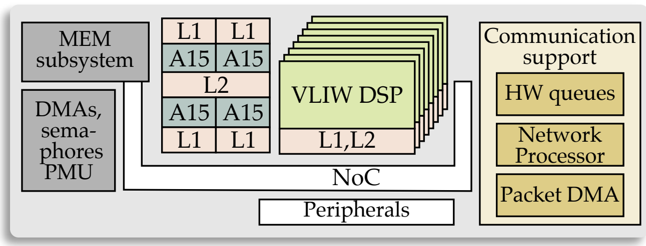
KPN Application



Non-functional specification



Architecture model

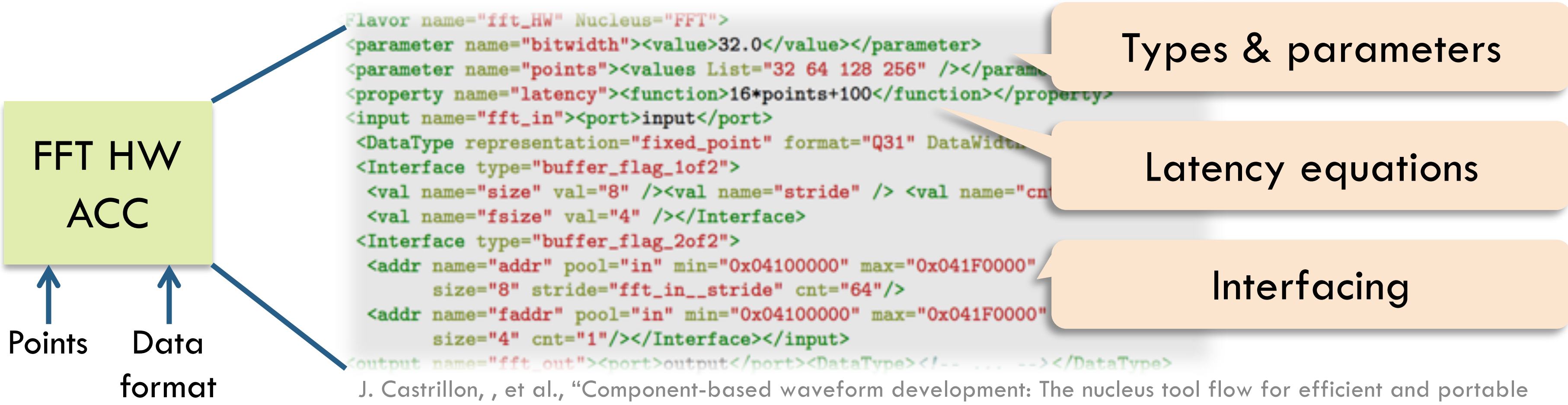


J. Castrillon, R. Leupers, "Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap" , Springer, pp. 258, 2014

```
 PNargs_ifft_r.ID = 6U;
    PNargs_ifft_r.PNchannel_freq_coef = filtered_coe
    PNargs_ifft_r.PNnum_freq_coef = 0U;
    PNargs_ifft_r.PNchannel_time_coef = sink_right;
    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_left;
    PNargs_sink.PNnum_in_left = 0U;
    PNargs_sink.PNchannel_in_right = sink_right;
    PNargs_sink.PNnum_in_right = 0U;
    taskParams.arg0 = (xdc_UArg)&PNargs_src;
    taskParams.priority = 1;

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

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

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

Algorithmic description

- Extended application specification
  - Selected processes are algorithmic kernels with **algorithmic parameters**
- Extended platform model
  - SW/HW accelerated kernels and their **implementation parameters**

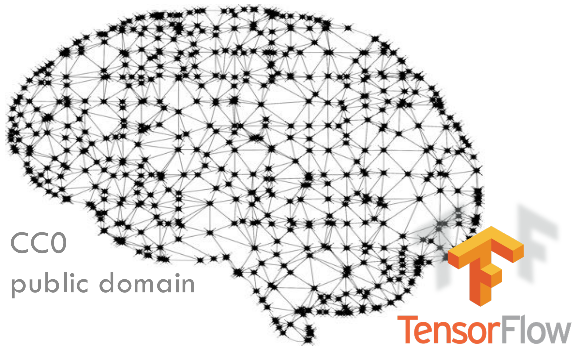


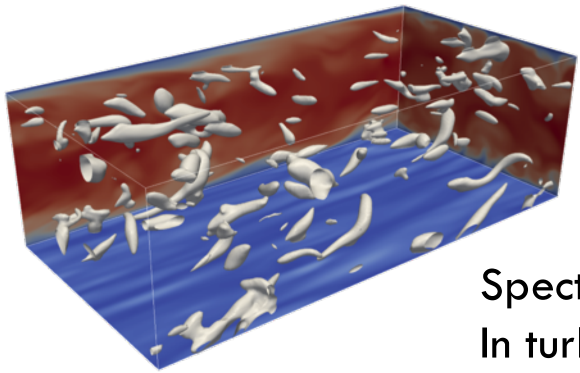
J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable SDR," Wireless Innovation Conference and Product Exposition (SDR), 2010

J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable software defined radio", Analog Integrated Circuits and Signal Processing, 173–190, 2011

Deep dive: TensorDSLs

Tensors: Multi-dimensional arrays





Spectral-elements
In turbulent flow

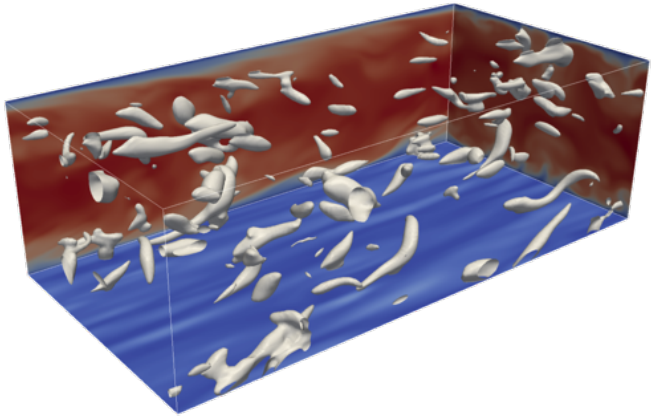
- Tensors are common to different areas: ML, quantum chemistry, physics sims.
- Algorithmic kernels concisely expressed with tensors turn into deep loop nests
    - Challenging to optimize for embedded devices (inference in CPS)

Starting point: CFDlang physics simulations

- Tensor expressions typically occur in numerical codes

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

$$v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'e}$$



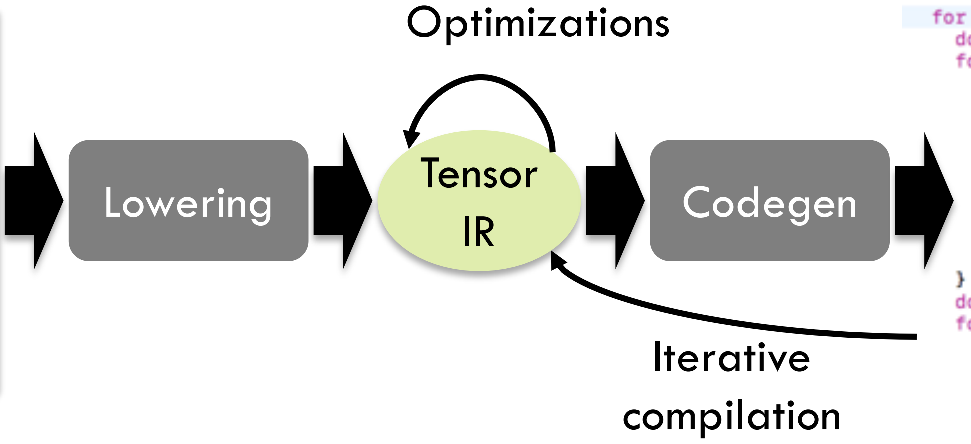
  - Matrixes are small, so libraries like BLAS do not always help
  - Expressions result in deeply nested for-loops
  - Performance highly depends on the *shape* of the loop nests

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];
      }
    }
  }
}
```

N. A. Rink, I. Huismann, A. Susungi, J. Castrillon, et al. "CFDlang: High-level Code Generation for High-order Methods in Fluid Dynamics", RWDSL 2018

Example: Interpolation operator

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

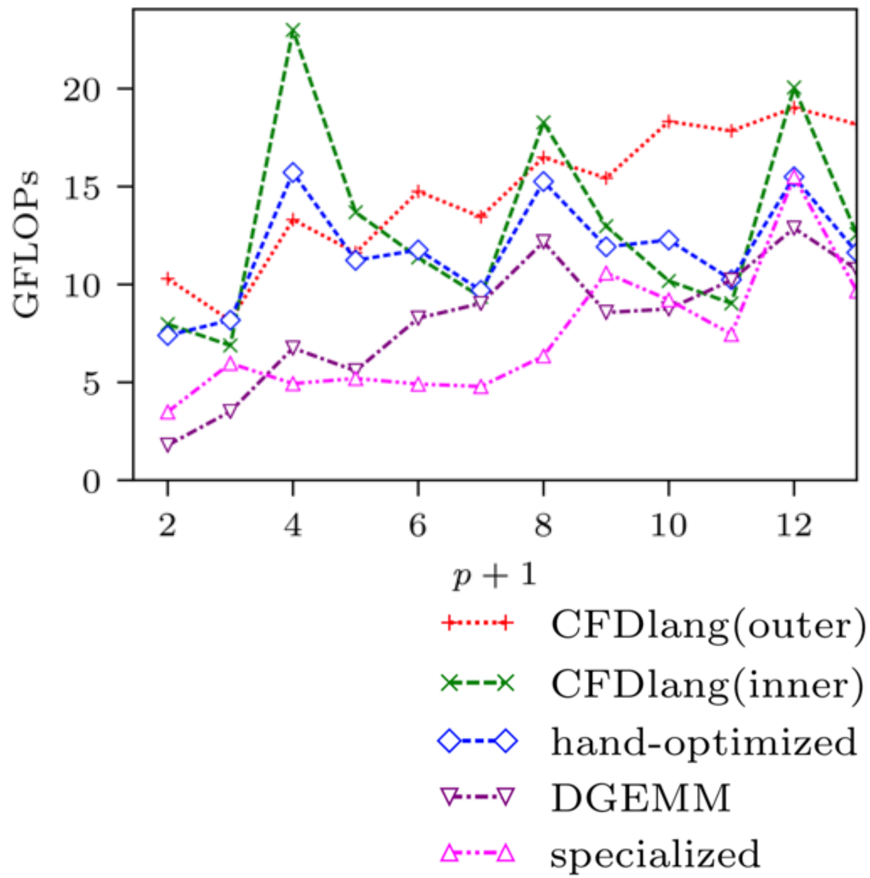
$$v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}$$

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



A. Susungi, N. A. Rink, J. Castrillon, et al "Towards Compositional and Generative Tensor Optimizations". GPCE 17

TeML: Meta-programming for Tensor Optimizations

- Generalization across domains for tensor expressions
- Clean expression language: **Index-free notation**
- **Formal semantics** for correctness and transformations
  - An expression is a Tree (T)
  - Expressions can be implemented as loop-nests (L)
  - The state maps identifiers to either T or L

$$S = \mathit{identifier} \rightarrow (T + L)$$

```
⟨program⟩      ::= ⟨stmt⟩ ⟨program⟩
                |  ϵ
⟨stmt⟩         ::= ⟨id⟩ = ⟨expression⟩
                |  ⟨id⟩ = @⟨id⟩ : ⟨expression⟩
                |  codegen (⟨ids⟩)
                |  init (...)
⟨expression⟩   ::= ⟨Texpression⟩
                |  ⟨Lexpression⟩
⟨Texpression⟩  ::= scalar ()
                |  tensor ([⟨ints⟩])
                |  eq (⟨id⟩, ⟨iters⟩? → ⟨iters⟩)
                |  vop (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?])
                |  op (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?] → ⟨iters⟩)
⟨Lexpression⟩  ::= build (⟨id⟩)
                |  stripmine (⟨id⟩, ⟨int⟩, ⟨int⟩)
                |  interchange (⟨id⟩, ⟨int⟩, ⟨int⟩)
                |  fuse_outer (⟨id⟩, ⟨id⟩, ⟨int⟩)
                |  fuse_inner (⟨id⟩, ⟨int⟩)
                |  unroll (⟨id⟩, ⟨int⟩)
⟨iters⟩        ::= [⟨ids⟩]
⟨ids⟩          ::= ⟨id⟩ (, ⟨id⟩)*
⟨ints⟩         ::= ⟨int⟩ (, ⟨int⟩)*
```

A Susungi, N. A. Rink, A. Cohen, J. Castrillon, C. Tadonki, "Meta-programming for Cross-Domain Tensor Optimizations". GPCE 18,

TeML: Meta-programming for Tensor Optimizations

$$S = identifier \to (T + L)$$

- Recall

$$\begin{aligned} storeT &= store \nrightarrow store \\ storeO &= store \nrightarrow value \end{aligned} \qquad \begin{aligned} \mathcal{S} &: stmt \to storeT \\ \mathcal{E} &: expr \to storeO \end{aligned}$$

- In TeML

$$\begin{aligned} \mathcal{P}_{prog} &: program \to (S \to S)\,, \\ \mathcal{P}_{stmt} &: stmt \to (S \to S)\,, \\ \mathcal{E}_t &: Texpression \to (S \to T)\,, \\ \mathcal{E}_l &: Lexpression \to (S \to L)\,. \end{aligned}$$

$$\begin{aligned} \mathcal{P}_{stmt}[\![id = e_t]\!] = \lambda\sigma.\text{let } x &= \mathcal{E}_t[\![e_t]\!]\sigma \\ x' &= x[id/\square] \\ \text{in } \sigma&\{id \mapsto x'\} \end{aligned}$$

```
⟨program⟩      ::= ⟨stmt⟩ ⟨program⟩
                |  ϵ
⟨stmt⟩         ::= ⟨id⟩ = ⟨expression⟩
                |  ⟨id⟩ = @⟨id⟩ : ⟨expression⟩
                |  codegen (⟨ids⟩)
                |  init (...)
⟨expression⟩   ::= ⟨Texpression⟩
                |  ⟨Lexpression⟩
⟨Texpression⟩  ::= scalar ()
                |  tensor ([⟨ints⟩])
                |  eq (⟨id⟩, ⟨iters⟩? → ⟨iters⟩)
                   ⟨iters⟩?, ⟨iters⟩?])
                   ters⟩?, ⟨iters⟩?] → ⟨iters⟩)
                   , ⟨int⟩, ⟨int⟩)
                   d⟩, ⟨int⟩, ⟨int⟩)
                   ), ⟨id⟩, ⟨int⟩)
                   ), ⟨int⟩)
                |  unroll (⟨id⟩, ⟨int⟩)
⟨iters⟩        ::= [⟨ids⟩]
⟨ids⟩          ::= ⟨id⟩ (, ⟨id⟩)*
⟨ints⟩         ::= ⟨int⟩ (, ⟨int⟩)*
```

A Susungi, N. A. Rink, A. Cohen, J. Castrillon, C. Tadonki, "Meta-programming for Cross-Domain Tensor Optimizations". GPCE 18,

TeML: Meta-programming

$$\mathcal{E}_l[\![\texttt{stripmine}(l, r, v)]\!] =$$
$$\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_r, xs\rangle \ldots\rangle = \sigma(l)$$
$$(b, e, 1) = i_r$$
$$i'_r = (0, (e-b)/v - 1, 1)$$
$$i'_{r+1} = (b + v \cdot i'_r, b + v \cdot i'_r + (v-1), 1)$$
$$\text{in } \langle i_1, \ldots \langle i'_r, [\langle i'_{r+1}, xs\rangle]\rangle \ldots\rangle$$

$$\mathcal{E}_l[\![\texttt{interchange}(l, r_1, r_2)]\!] =$$
$$\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_{r_1}, \ldots \langle i_{r_2}, xs\rangle \ldots\rangle \ldots\rangle = \sigma(l)$$
$$\text{in } \langle i_1, \ldots \langle i_{r_2}, \ldots \langle i_{r_1}, xs\rangle \ldots\rangle \ldots\rangle$$

$$\mathcal{P}_{stmt}[\![l' = \texttt{tile}(l, v)]\!] =$$
$$\mathcal{P}_{prog}\left[\!\!\left[\begin{array}{l} l_0 = \texttt{stripmine\_n}(l, d, v) \\ l_1 = \texttt{interchange\_n}(l_0, 2, 2d-2) \\ l_2 = \texttt{interchange\_n}(l_1, 3, 2d-3) \\ \ldots \\ l_{d-1} = \texttt{interchange\_n}(l_{d-2}, d, d)) \\ l' = \texttt{interchange\_n}(l_{d-1}, d+1, d-1) \end{array}\right]\!\!\right]$$

Formally defined transformation primitives

Higher-level transformations via composition

```
⟨program⟩      ::= ⟨stmt⟩ ⟨program⟩
                 | ϵ
⟨stmt⟩         ::= ⟨id⟩ = ⟨expression⟩
                 | ⟨id⟩ = @⟨id⟩ : ⟨expression⟩
                 | codegen (⟨ids⟩)
                 | init (...)
⟨expression⟩   ::= ⟨Texpression⟩
                 | ⟨Lexpression⟩
⟨Texpression⟩  ::= scalar ()
                 | tensor ([⟨ints⟩])
                 | eq (⟨id⟩, ⟨iters⟩? → ⟨iters⟩)
                 | vop (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?])
                 | op (⟨id⟩, ⟨id⟩, [⟨iters⟩?, ⟨iters⟩?] → ⟨iters⟩)
⟨Lexpression⟩  ::= build (⟨id⟩)
                 | stripmine (⟨id⟩, ⟨int⟩, ⟨int⟩)
                 | interchange (⟨id⟩, ⟨int⟩, ⟨int⟩)
                 | fuse_outer (⟨id⟩, ⟨id⟩, ⟨int⟩)
                 | fuse_inner (⟨id⟩, ⟨int⟩)
                 | unroll (⟨id⟩, ⟨int⟩)
⟨iters⟩        ::= [⟨ids⟩]
⟨ids⟩          ::= ⟨id⟩ (, ⟨id⟩)*
⟨ints⟩         ::= ⟨int⟩ (, ⟨int⟩)*
```

A Susungi, N. A. Rink, A. Cohen, J. Castrillon, C. Tadonki, "Meta-programming for Cross-Domain Tensor Optimizations". GPCE 18,

TeIL: Safe code generation

- Formalized core tensor primitives
- Showed flaws in widespread languages
- Proved no out of bound accesses (using Coq)

```
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))
```

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

$$\llbracket\cdot\rrbracket : Context \rightarrow Memory \rightarrow (list\ of\ \mathrm{Nat}) \rightarrow \mathbb{D}$$

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

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

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

$$\llbracket \mathtt{mul}\; e_0\, e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath} = \begin{cases} \llbracket e_0 \rrbracket\, \Gamma\, \mu\, [\,] \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{if } type_\Gamma(e_0) = [\,] \\ \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath} \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}, & \text{otherwise} \end{cases}$$

$$\llbracket \mathtt{prod}\; e_0\, e_1 \rrbracket\, \Gamma\, \mu\, (\bar{\imath}_0 \# \bar{\imath}_1) = \llbracket e_0 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_0 \cdot \llbracket e_1 \rrbracket\, \Gamma\, \mu\, \bar{\imath}_1, \quad \text{if } rank_\Gamma(e_0) = length(\bar{\imath}_0) \text{ and } rank_\Gamma(e_1) = length(\bar{\imath}_1)$$

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

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

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

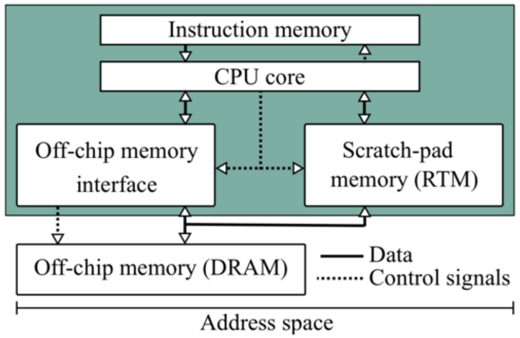
$$\llbracket \mathtt{proj}\; i\, m\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, \ldots, j_k] = \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k]$$

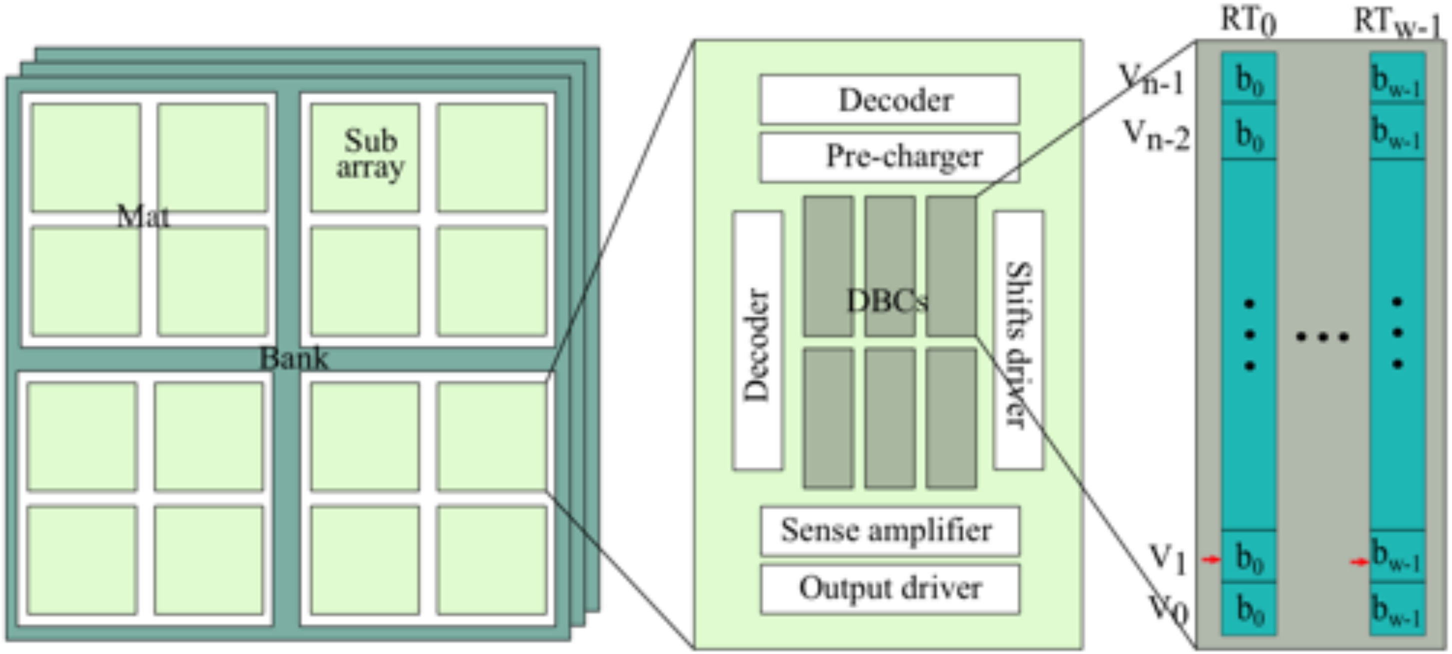
$$\llbracket \mathtt{red}_+\; i\, e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, j_i, \ldots, j_k] = \textstyle\sum_{m=1}^{n} \llbracket e \rrbracket\, \Gamma\, \mu\, [j_1, \ldots, j_{i-1}, m, j_i, \ldots, j_k], \text{ if } type_\Gamma(e) = [n_1, \ldots, n_{i-1}, n, n_{i+1}, \ldots, n_{k+1}]$$

N. A. Rink, J. Castrillon, "TeIL: a type-safe imperative Tensor Intermediate Language". ARRAY'19

Emerging memories

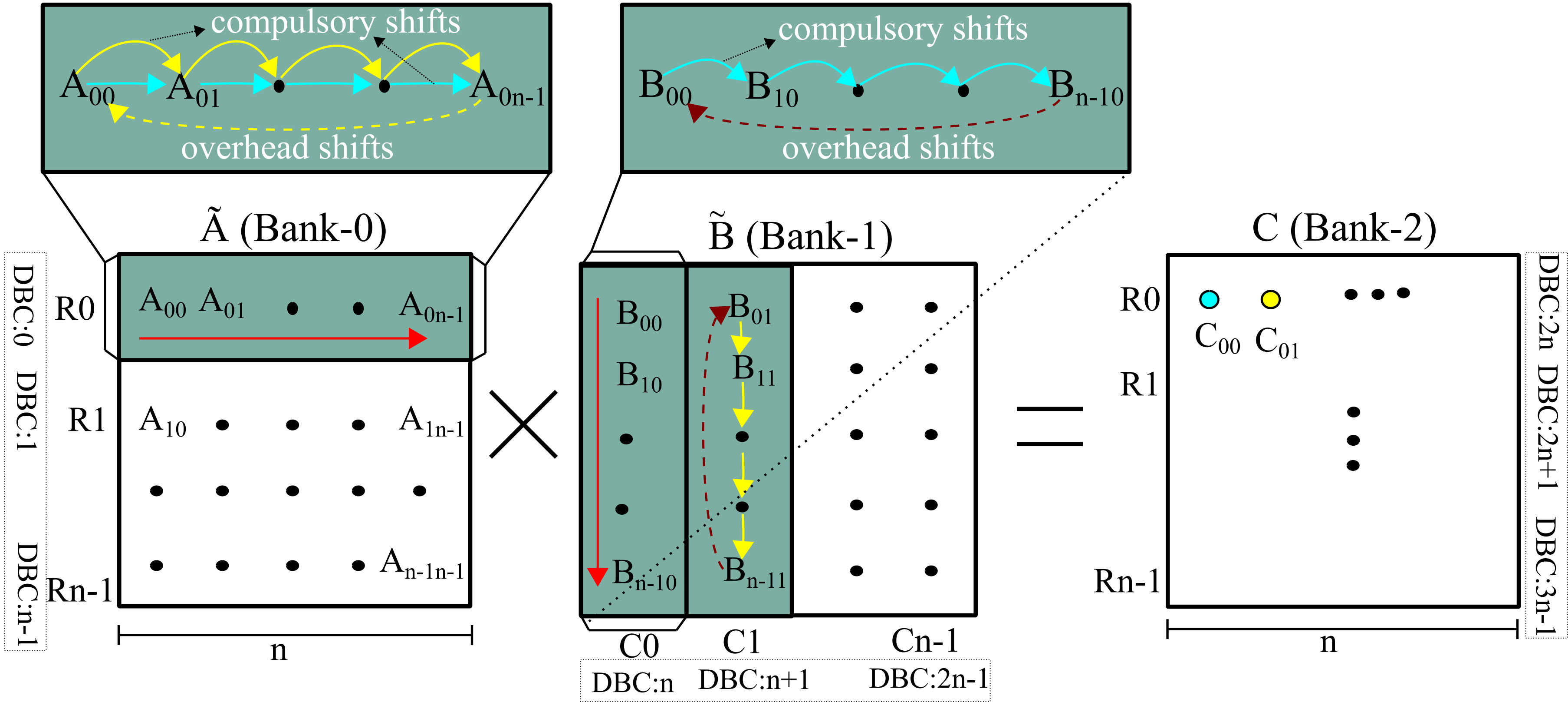
- Non-volatile memories provide high energy efficiency (think embedded)
- Racetrack-memories provide unprecedented density (embedded inference)





A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". In ACM TECS'20

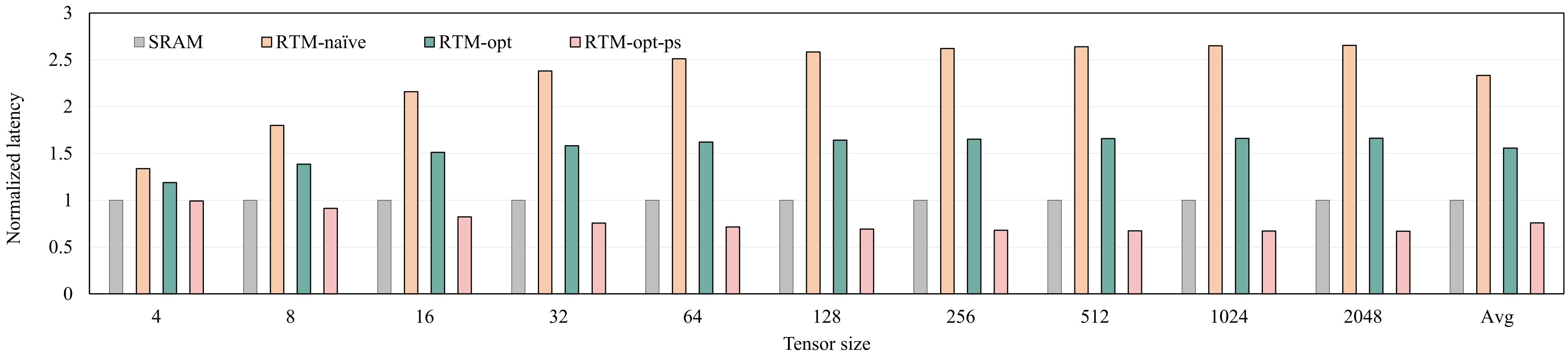
Layout optimization: By hand



A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories". In ACM TECS'20

Latency comparison vs SRAM

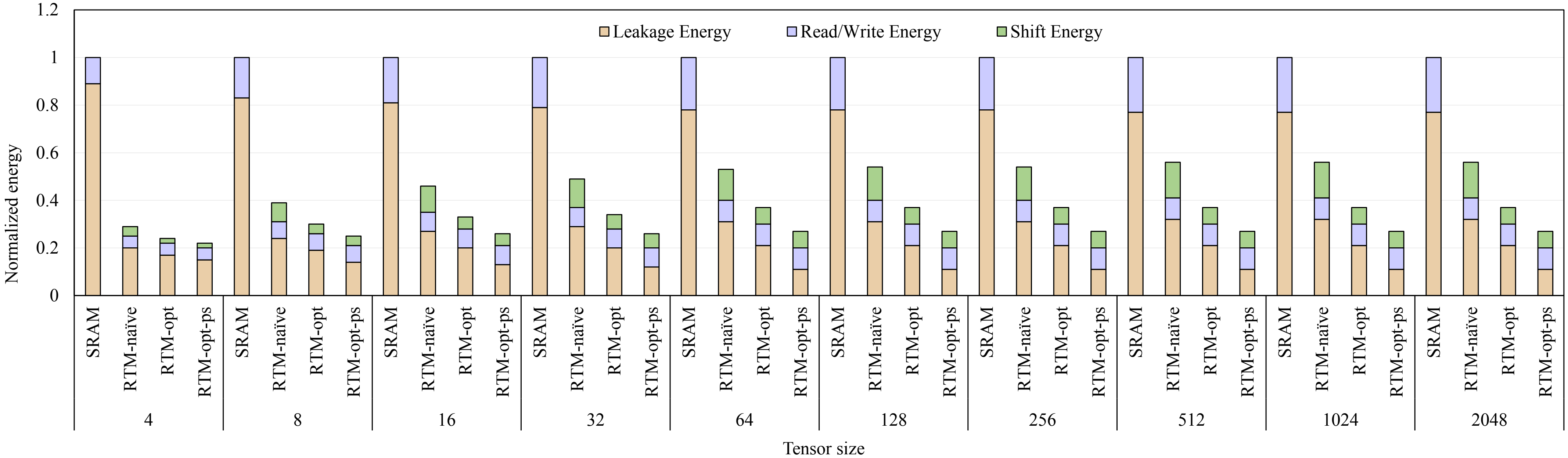
- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)



A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, " "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads". In LCTES'19

Energy comparison vs SRAM

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

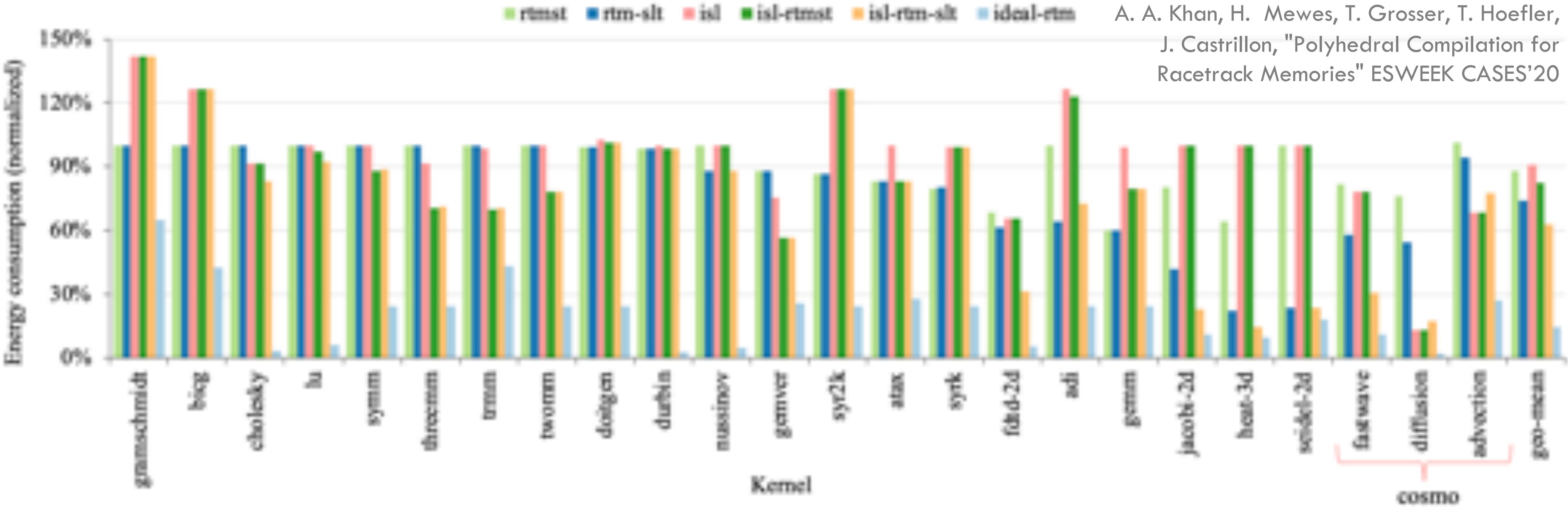


A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, " "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads". In LCTES'19

Compiler optimization

- Recent work on automatic identification of loop patterns

A. A. Khan, H. Mewes, T. Grosser, T. Hoefler, J. Castrillon, "Polyhedral Compilation for Racetrack Memories" ESWEEK CASES'20



- Working on (easier) optimization from Tensor DSLs

Closing remarks

Summary

- CPS challenges: Heterogeneity, changing HW substrate, interconnectivity, …
- Background: DSL principles, formal foundations and examples
- Deep-dive: Tensor DSLs in general and at our lab @ TU Dresden
  - Formalization for transformations (enable search space)
  - Formalization for correctness proofs
  - Current work transformations for emerging NVMs

- Role of DSLs (and tools) in CPS programming
  - Productivity boost (specially as coders' backgrounds widens)
  - Correctness of the specification (try to avoid having to debug!)
  - Enabler of more powerful (higher-level) optimizations

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. SIGPLAN Not. 48, 6 (June 2013), 519-530.

M. Danneberg, N. Michailow, I. Gaspar, D. Zhang and G. Fettweis, "Flexible GFDM Implementation in FPGA with Support to Run-Time Reconfiguration". VTC2015-Fall

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

J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable SDR," Wireless Innovation Conference and Product Exposition (SDR), 2010

J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable software defined radio", Analog Integrated Circuits and Signal Processing, 173–190, 2011

N. A. Rink, I. Huismann, A. Susungi, J. Castrillon, et al. "CFDlang: High-level Code Generation for High-order Methods in Fluid Dynamics" , RWDSL 2018

A. Susungi, N. A. Rink, J. Castrillon, et al "Towards Compositional and Generative Tensor Optimizations" , GPCE 17,

A Susungi, N. A. Rink, A. Cohen, J. Castrillon, C. Tadonki, "Meta-programming for Cross-Domain Tensor Optimizations" , GPCE 18,

N. A. Rink, J. Castrillon, "TeIL: a type-safe imperative Tensor Intermediate Language" , ARRAY'19

A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, "Optimizing Tensor Contractions for Embedded Devices with Racetrack and DRAM Memories", In ACM TECS'20

A. Ali Khan, N. A. Rink, F. Hameed, J. Castrillon, " "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", In LCTES'19

A. A. Khan, H. Mewes, T. Grosser, T. Hoefler, J. Castrillon, "Polyhedral Compilation for Racetrack Memories" ESWEEK CASES'20