CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

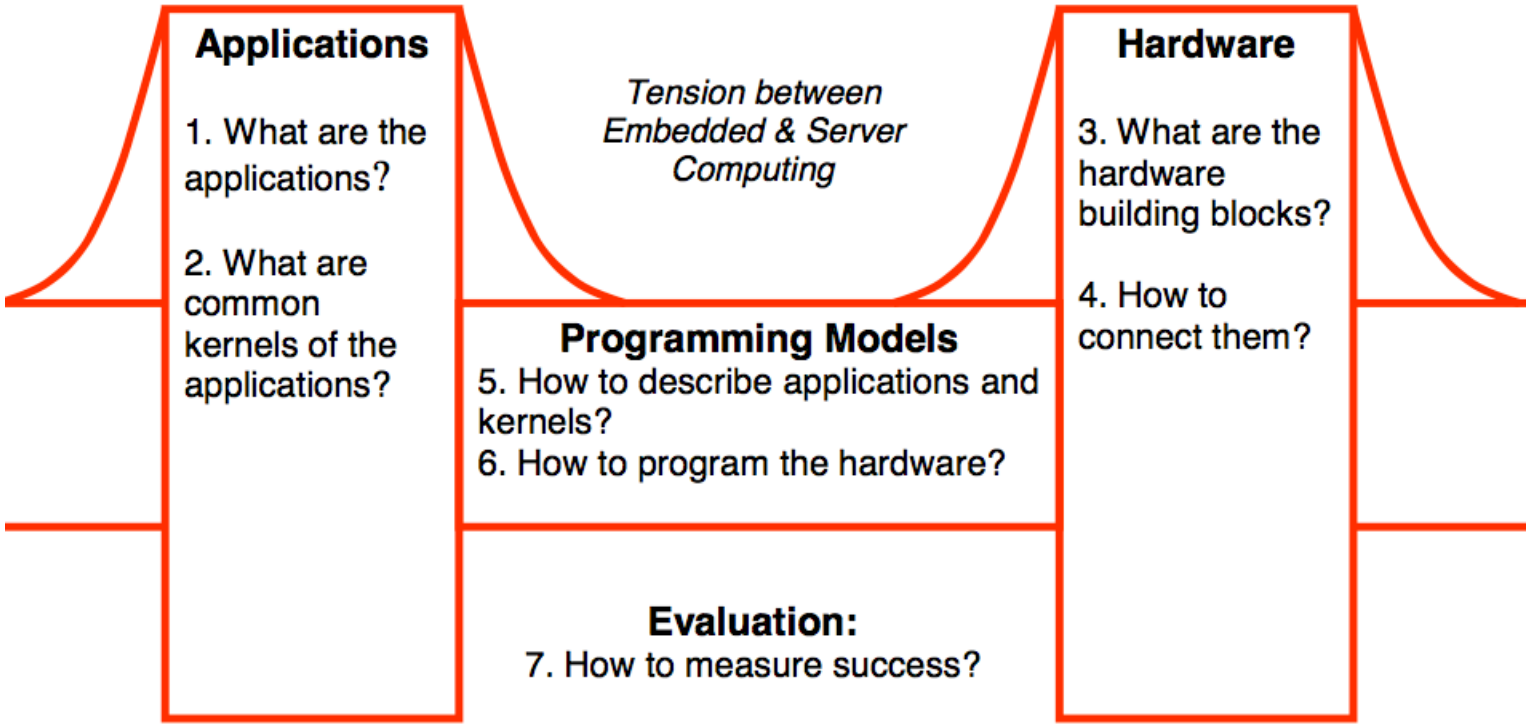
Parallel programming: Current and future systems

Jeronimo Castrillon

Chair for Compiler Construction, TU Dresden
jeronimo.castrillon@tu-dresden.de

50-year Celebration: Department of Electronics, Universidad de Antioquia
IEEE Colombian Conference on Communications and Computing (COLCOM)
Medellin, Colombia, May 16 2018

Programming models



Asanovic, K.; Bodik, R.; Catanzaro, B. C.; Gebis, J. J.; Husbands, P.; Keutzer, K.; Patterson, D. A.; Plishker, W. L.; Shalf, J.; Williams, S. W. & Yelick, K. A. The Landscape of Parallel Computing Research: A View from Berkeley. EECS Department, University of California, Berkeley, 2006

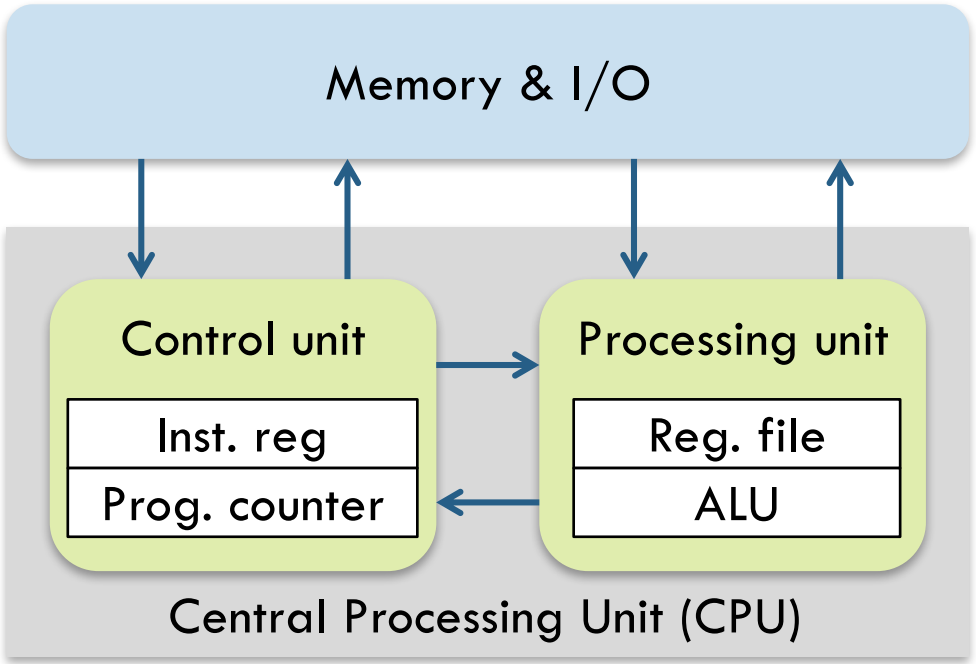
Von Neumann and Sequential Programming

- Program: sequence of instructions in memory

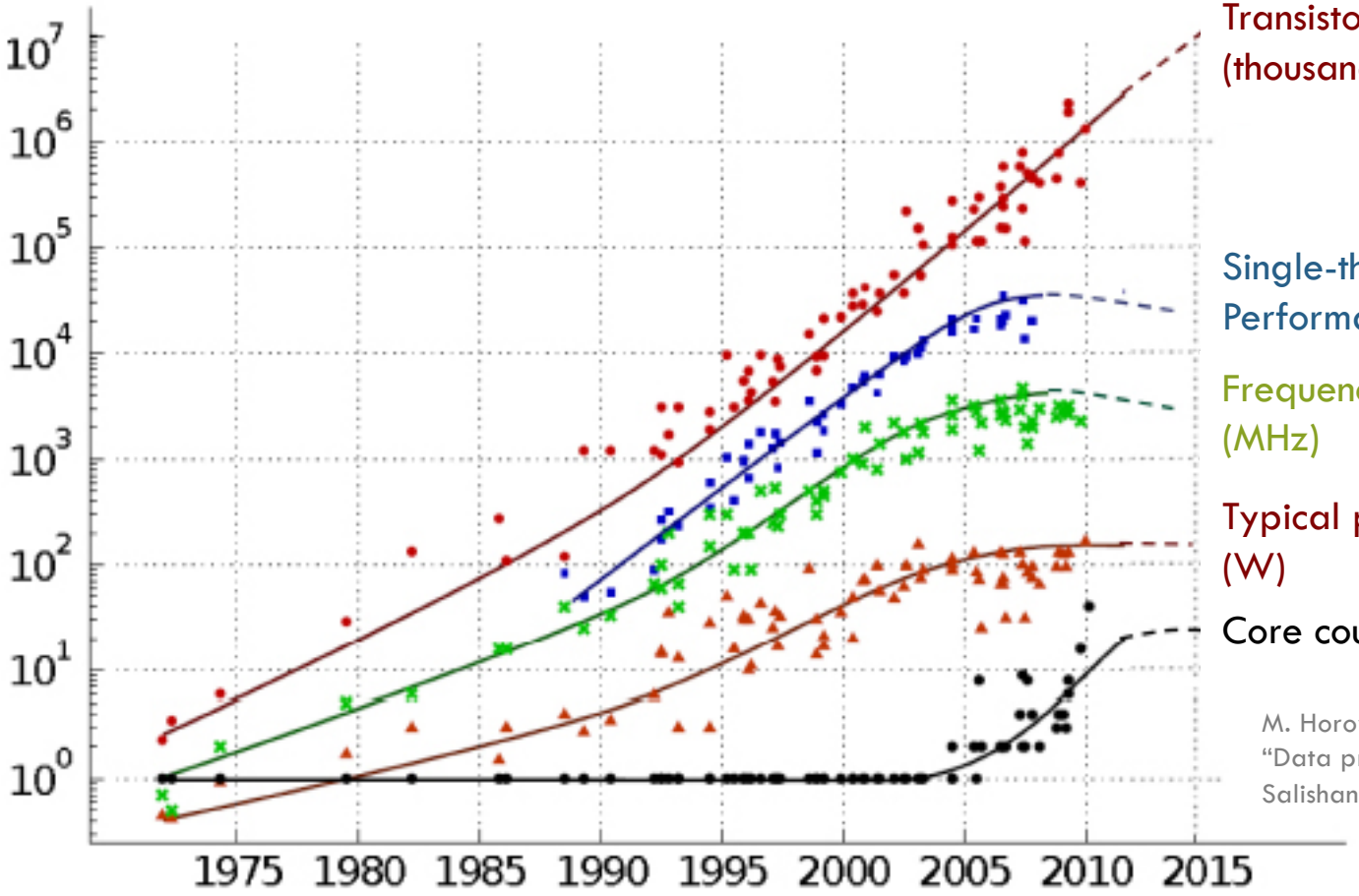
```
double t7[12];
for (unsigned i7 = 0; i7 < 3; i7++) {
  for (unsigned i6 = 0; i6 < 2; i6++) {
    for (unsigned i5 = 0; i5 < 2; i5++) {
      t7[(i5 + 2*(i6 + 2*(i7)))] = 0.0;
      for (unsigned i8_contr = 0; i8_contr < 3; i8_contr++) {
        t7[(i5 + 2*(i6 + 2*(i7)))] += A[(i5 + 2*(i8_contr))]
            * t6[(i6 + 2*(i7 + 3*(i8_contr)))];
      }
    }
  }
}
double t8[1];
double t9[1];
for (unsigned i11 = 0; i11 < 2; i11++) {
  for (unsigned i10 = 0; i10 < 2; i10++) {
    for (unsig  i9 = 0; i9 < 2; i9++) {
      t9[0] = 0.
      for (unsigne                - 0; i12_contr < 3; i12_contr++
          )
        t9
      }
      t8[
      doub
      t10
      v[(
    }
  }
}
```

Good abstractions:

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



Why parallel programing?



Transistors (thousands)

Single-thread Performance

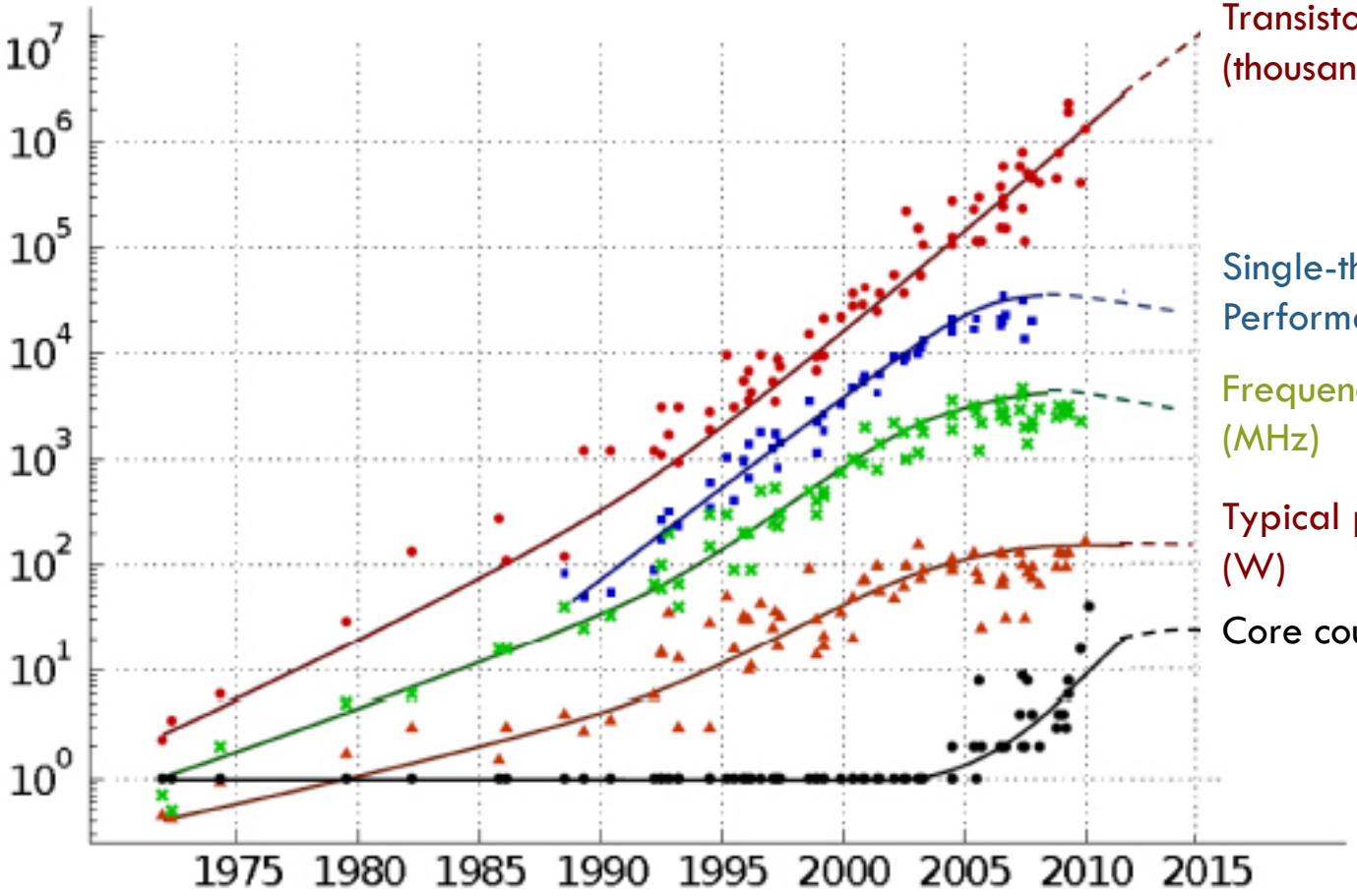
Frequency (MHz)

Typical power (W)

Core count

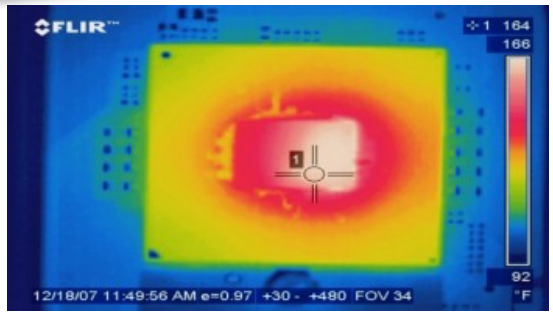
M. Horowitz, F. Labonte, et al. Dotted-line by C. Moore, "Data processing in exascale-class computer systems," The Salishan Conference on High Speed Computing, 2011

Why parallel programing?



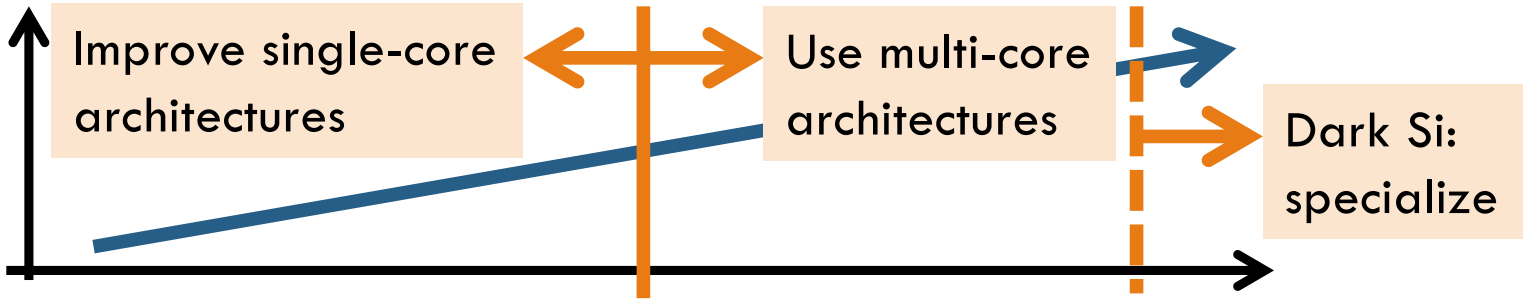
Diminishing returns for single core

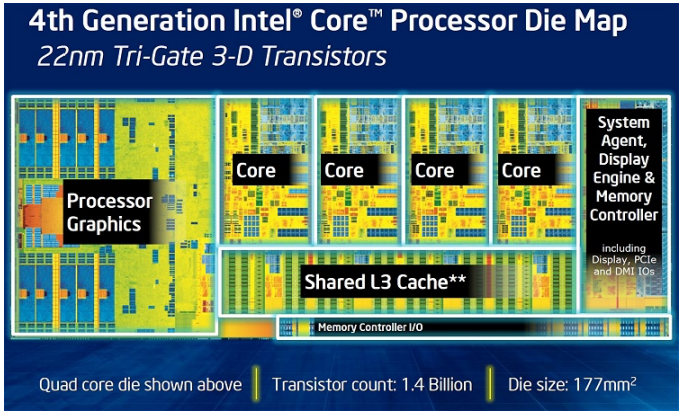
Power density!



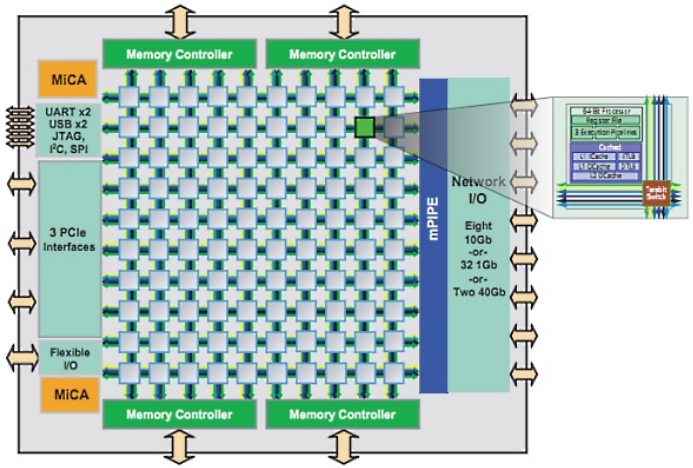
http://www.extremetech.com/computing/112147-hot-chips-laptop-manufacturer-sues-amd-over-allegedly-defective-chips

Inflection points

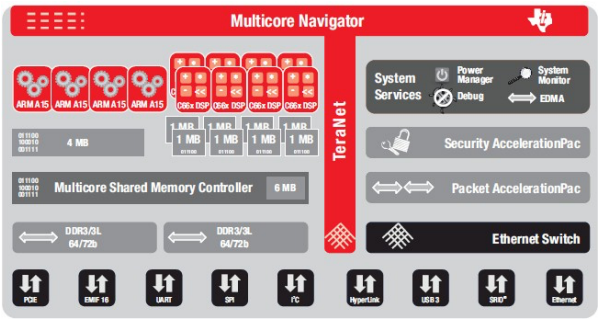




http://www.techspot.com/photos/article/679-intel-haswell-core-i7-4770k/#Slide_01

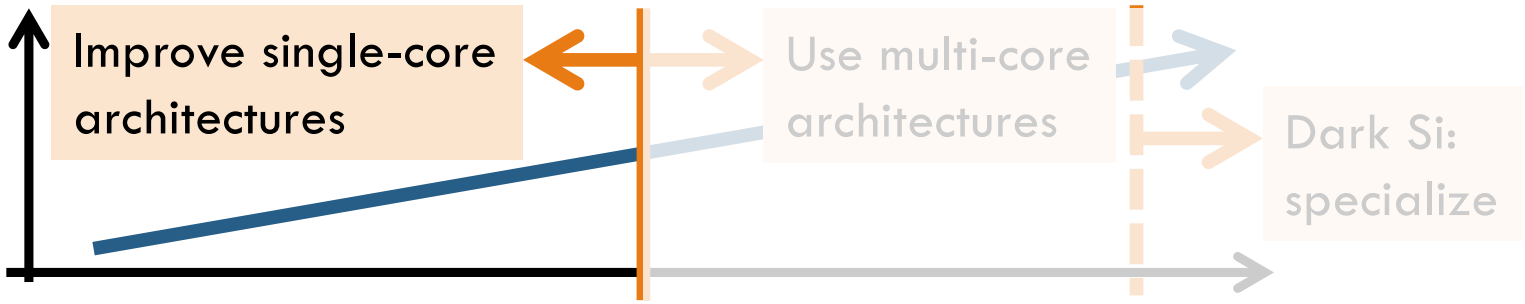


Tilera.com

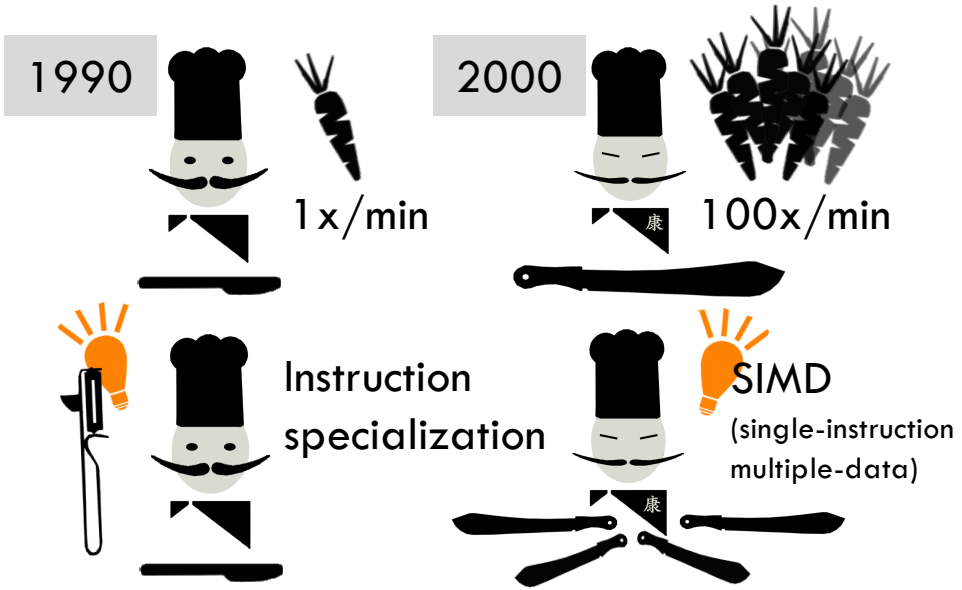


http://www.theregister.co.uk/2013/03/05/hp_moonshot_server_ti_keystone_arm_chip/

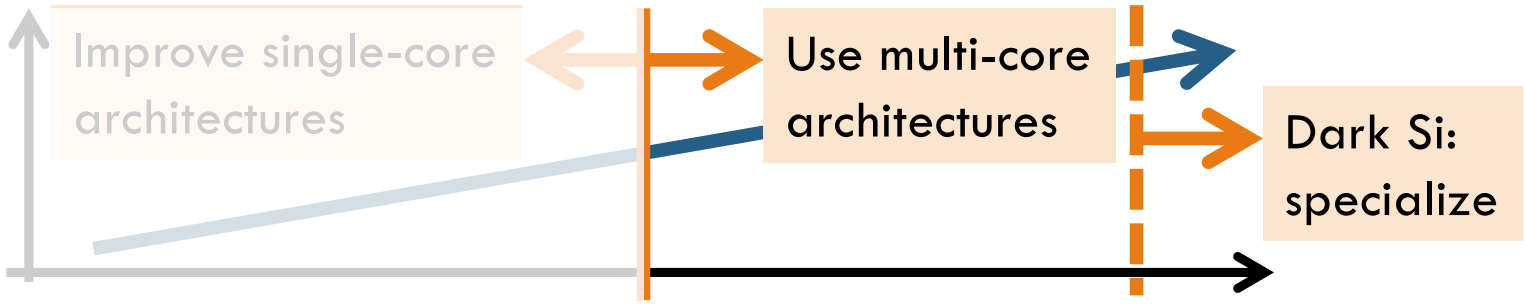
What to do with the transistors?



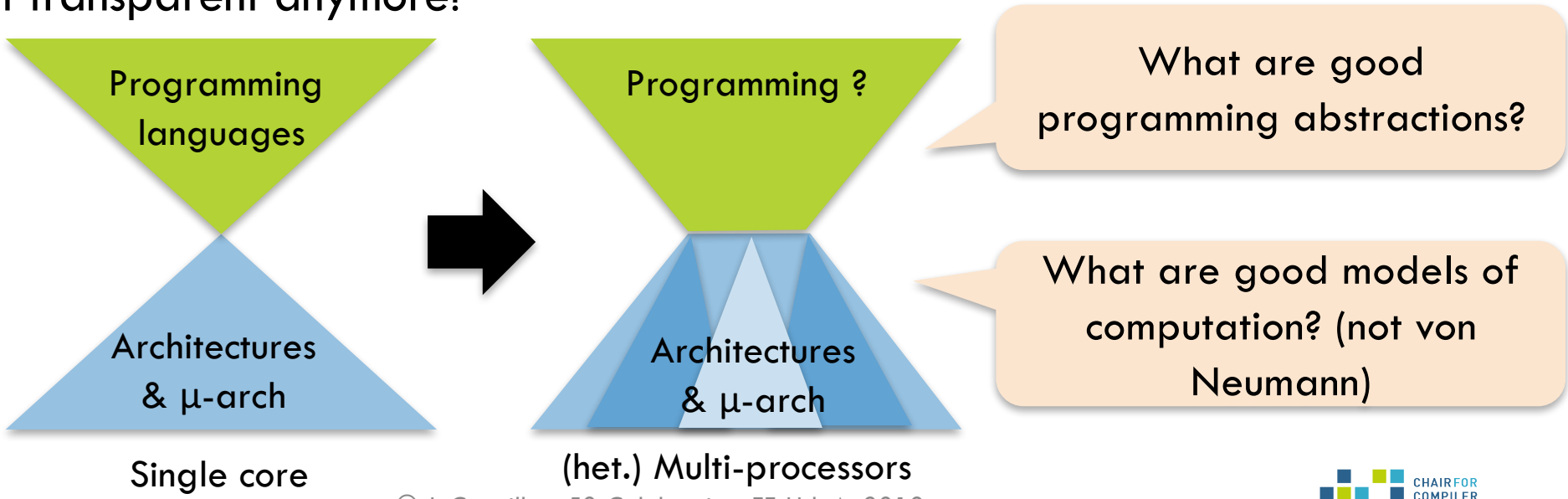
- Mostly transparent for programmers
  - Faster, specialized ISA
  - Incremental compiler innovation



What to do with the transistors?



- Not transparent anymore!



Agenda

- Introduction
- Programming models: Overview
- Systematic parallel programming
- Future electronics
- Wrap-up

Agenda

- Introduction
- **Programming models: Overview**
- Systematic parallel programming
- Future electronics
- Wrap-up

Keep it sequential

- Parallel programming is hard (with most abstractions) – more on this later
  - Let the compiler do the work

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

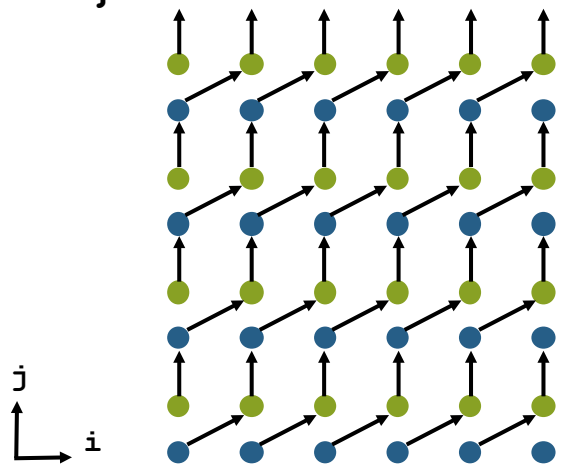
"Optimizing Compilers for Modern Architectures: A Dependence-Based Approach", Allen and Kennedy, 2002, Ch. 2.

Problems for auto-parallelizing compilers

1) Find all dependencies?

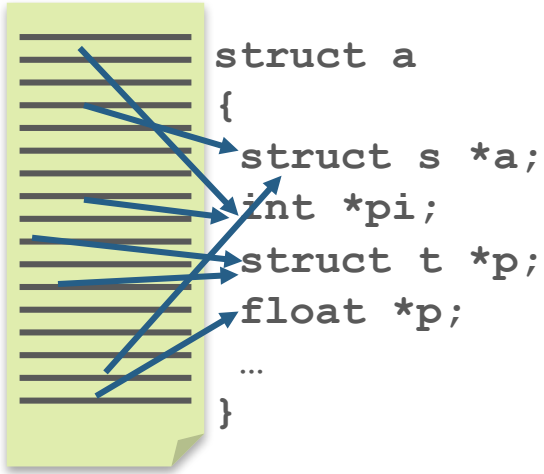
More often than not, impossible!

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



Example: **Polyhedral compilation**

2) Coding style and the illusion of infinite shared memory



Problems for auto-parallelizing compilers

1) Find all dependencies?

2) Coding style and the illusion of infinite shared memory

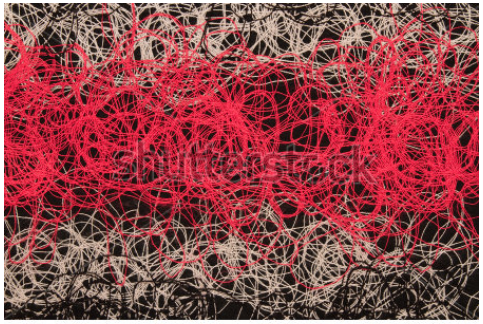
3) Dependencies can sometimes be violated! (they are artifacts of style)

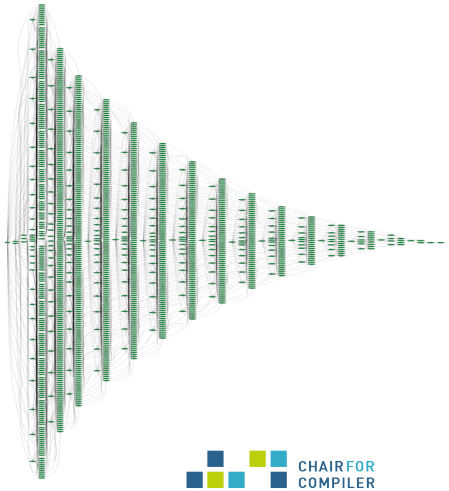
```
19  while(!queue.empty())
20  {
21    // Dequeue a vertex from queue
22    s = queue.front();
23    queue.pop_front();
24
25    // Apply function f to s, accumulate values
26    result += f(s);
27
28    // Get all adjacent vertices of s.
29    // If an adjacent node hasn't been visited,
30    //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

T. J.K. Edler von Koch, S. Manilov, C. Vasiladiotis, M. Cole, and B.Franke. Towards a Compiler Analysis for Parallel Algorithmic Skeletons. In: Proceedings of the 2018 International Conference on Compiler Construction (CC'18), Vienna, 2018.

Options for parallel programming

- Myriads of options: languages, libraries, run-time systems, …
- Shared-memory programming
  - Pthreads, OpenMP, …
  - Task-based programing (Intel TBB, Cilk)
- Distributed-memory programming
  - Message Passing Interface (MPI)
  - Map-Reduce





Options for parallel programming

- Myriads of options: languages, libraries, run-time systems, …
- Shared-memory programming
  - Pthreads, OpenMP, …
  - Task-based programing (Intel TBB, Cilk
- Distributed-memory programming
  - Message Passing Interface (MPI)
  - Map-Reduce

Threads: assembly of parallel programming, hard to get it right

OpenMP: Convenient abstraction of threads

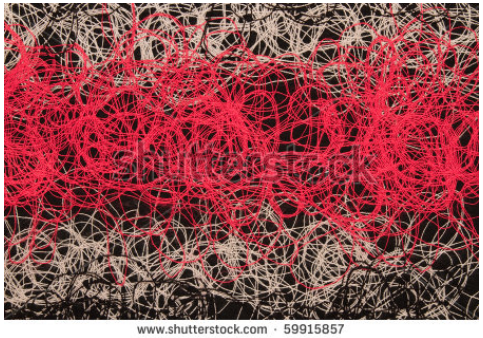
Task-based: More abstract – focus on application parallelism

MPI: 400+ APIs – Need a PhD

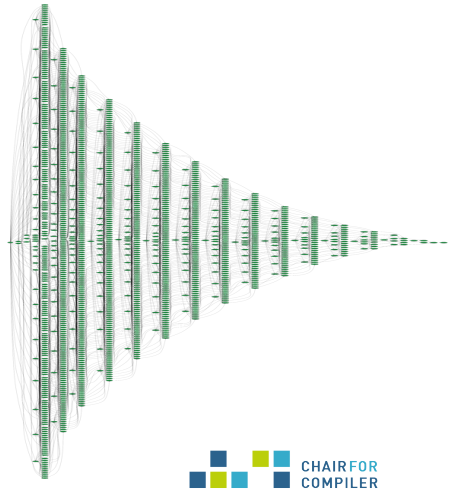
M&R: More abstract – works great if application adheres to skeleton

Options for parallel programming

- Myriads of options: languages, libraries, run-time systems, …
  - Pthreads, OpenMP, …
  - Task-based programing (Intel TBB, Cilk)
  - Message Passing Interface (MPI)
  - Map-Reduce



Some models better than others, but still too much of the architecture percolates to programmers (i.e. false sharing, load imbalances, …)



Agenda

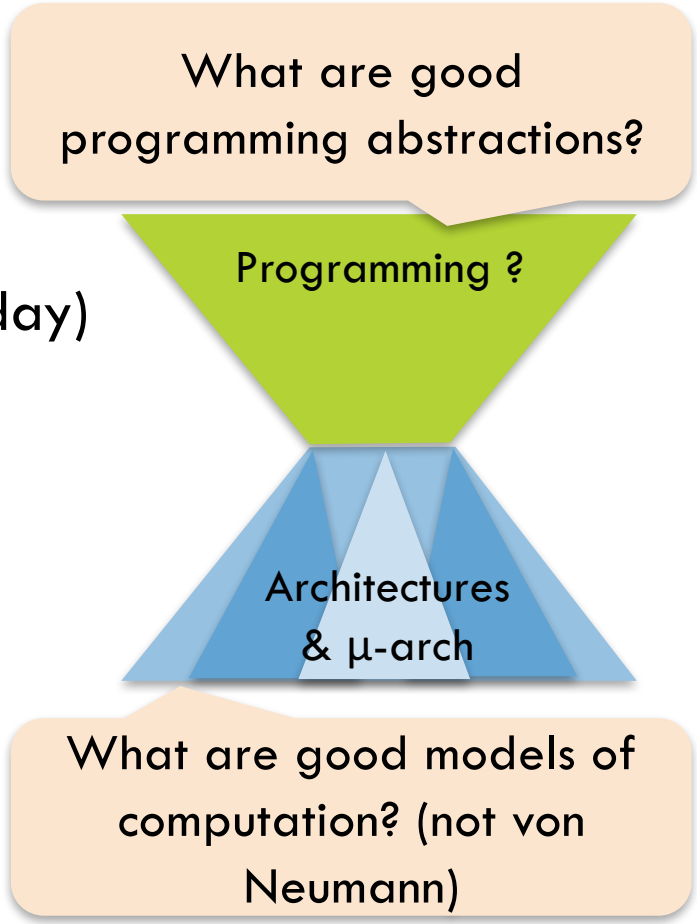
- Introduction
- Programming models: Overview
- **Systematic parallel programming**
- Future electronics
- Wrap-up

Systematic programming

- Better programming abstractions: No silver bullet
  - Broad spectrum (not GPP): Dataflow
  - Narrow spectrum: Domain-specific languages
- Models of architectures: Whole topic in itself (not today)

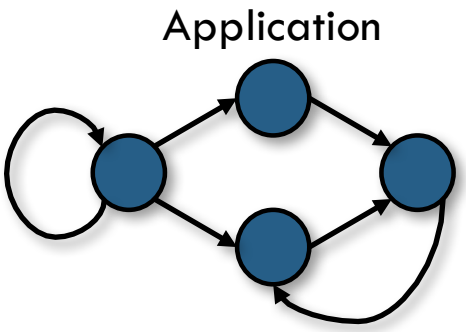
```
-<Platform>
   <Processors List="dsp0 dsp1 dsp2 dsp3 dsp4 dsp5 dsp6 dsp7"/>
   <Memories List="local_mem_dsp0_L2 local_mem_dsp1_L2 local_mem_dsp2_L2 local_mem_dsp
   local_smem_dsp1_L2 local_smem_dsp2_L2 local_smem_dsp3_L2 local_smem_dsp4_L2 local_smem
   local_mem_dsp3_DDR local_mem_dsp4_DDR local_mem_dsp5_DDR local_mem_dsp6_DDR local_m
   <CommPrimitives List="IPCll_SL2 IPCll_DDR EDMA3_SL2 EDMA3_DDR EDMA3_LL2"/>
 </Platform>
 <Processor Name="dsp0" CoreRef="DSPC66"/>
 <Processor Name="dsp1" CoreRef="DSPC66"/>
     ...
 <Processor Name="dsp7" CoreRef="DSPC66"/>
-<Memory>
   <LocalMemory Name="local_mem_dsp0_L2" Size="524288" BaseAddress_hex="00800000"
 </Memory>
     ...
```

- Systematic: Use automation (SW tools) to **lower the high-level spec to the hardware**



Broad: Dataflow programming

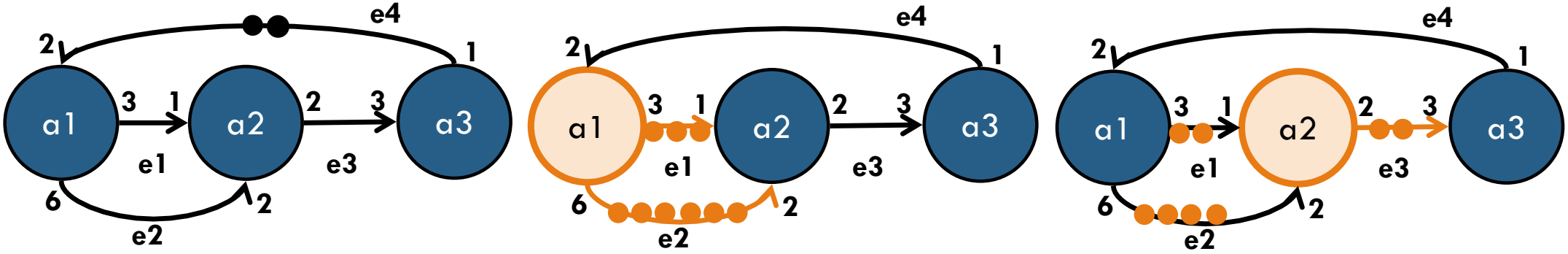
- Graph representation of applications
  - Nodes represent computation
  - Edges represent communication
  - Multiple variations of the precise semantics proposed



- Popular abstraction in many domains
  - Embedded signal processing
  - High-performance computing (OmpSs → OpenMP 4.5)
  - Databases
  - Big-data processing pipelines

Dataflow programming: Systematic

- Intuition for simple case: Synchronous Dataflow (SDF)



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

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

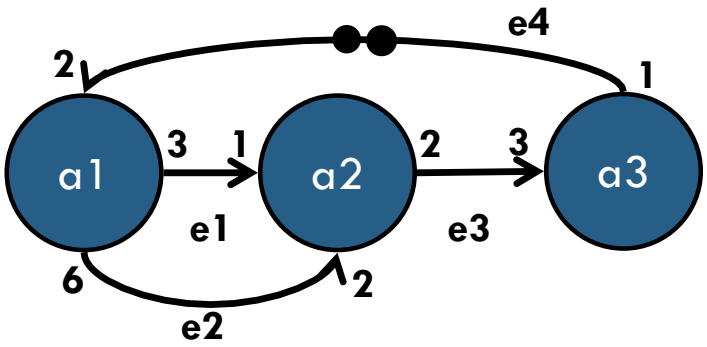
$$s_1 = s_0 + \begin{pmatrix} 3 & -1 & 0 \\ 6 & -2 & 0 \\ 0 & 2 & -3 \\ -2 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 6 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 0 \end{pmatrix}$$

Lee, E. A. & Messerschmitt, D. G. Synchronous Data Flow. Proceedings of the IEEE, 1987, 75, 1235-1245

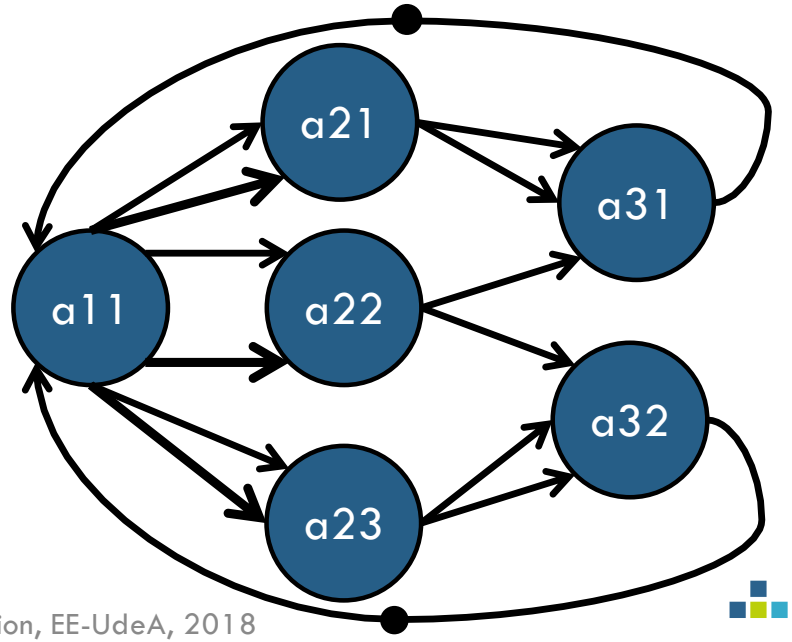
Dataflow programming: Systematic (2)

- Intuition for simple case: Synchronous Dataflow (SDF)
- Consequence
  - Possible to guarantee deadlock-freedom and execution on bounded memory
  - Possible to compute static schedule and optimize for throughput, energy, …

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

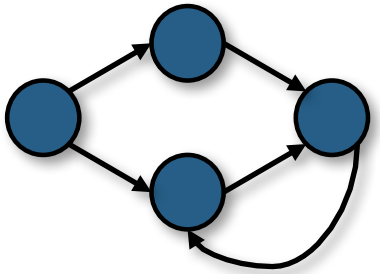


$$\vec{r} = (1, 3, 2)^T$$

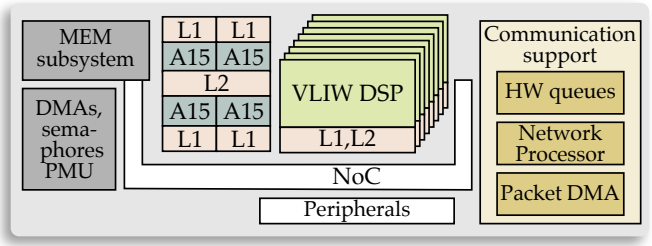


Dataflow programming flow

Application



Architecture model



Non-functional specification

[Castrill11, Castrill13, Castrill14]

Analysis

Synthesis

Property models (timing, energy, error, …)

Configurations

Source-to-source compilation

```
PNargs_ifft_r.ID = 6U;
   PNargs_ifft_r.PNchannel_freq_coef = filtered_coef_r
   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;
```

Narrow: Domain-specific languages

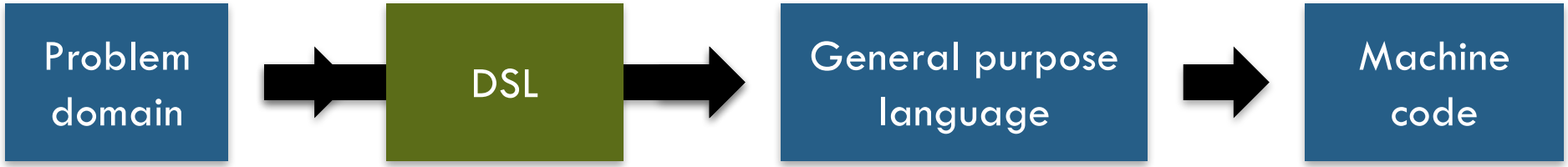
- Languages as abstractions

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

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
```

- DSLs: bridge gap between problem domain and general purpose languages



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

  - Why today? SW becoming pervasive, systems more complex, …

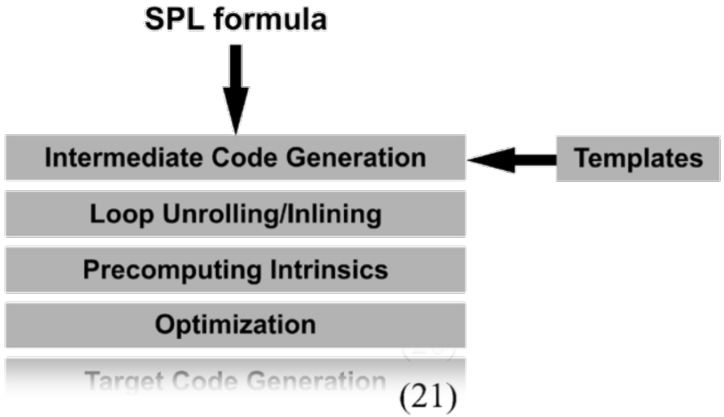
DSLs examples

- There are many examples, already for many years
  - Regexp in linux
  - SQL for databases
- Closer to my topics
  - Spiral (CMU/ETHZ): for linear transformations
  - TensorFlow (Google): for machine learning
  - Firedrake (Imperial College): for finite elements methods

Spiral

- Exploit **structure of known matrixes** to perform automatic algebraic transformations (e.g., factorization)

$$\mathbf{DCT\text{-}2}_n \rightarrow L^n_m(\mathbf{DCT\text{-}2}_m \oplus \mathbf{DCT\text{-}4}_m) \cdot (F_2 \otimes I_m)(I_m \oplus J_m)$$



- Rewrite rules to produce different variants

$$\mathbf{DFT}_n \rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \gcd(k,m) = 1 \tag{21}$$

$$\mathbf{DFT}_p \rightarrow R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime} \tag{22}$$

$$\mathbf{DCT\text{-}3}_n \rightarrow (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4))(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1} \\ \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \quad n = 2m \tag{23}$$

$$\mathbf{DCT\text{-}4}_n \rightarrow S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0\le k<n}\left(1/(2\cos\tfrac{(2k+1)\pi}{4n})\right) \tag{24}$$

$$\mathbf{IMDCT}_{2m} \rightarrow (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right) \mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m} \tag{25}$$

Püschel, M.; Moura, J.; Johnson, J.; Padua, D.; Veloso, M.; Singer, B.; Xiong, J.; Franchetti, F.; Gacic, A.; Voronenko, Y.; Chen, K.; Johnson, R. & Rizzolo, N. SPIRAL: Code Generation for DSP Transforms. Proceedings of the IEEE, 2005, 93, 232 -275

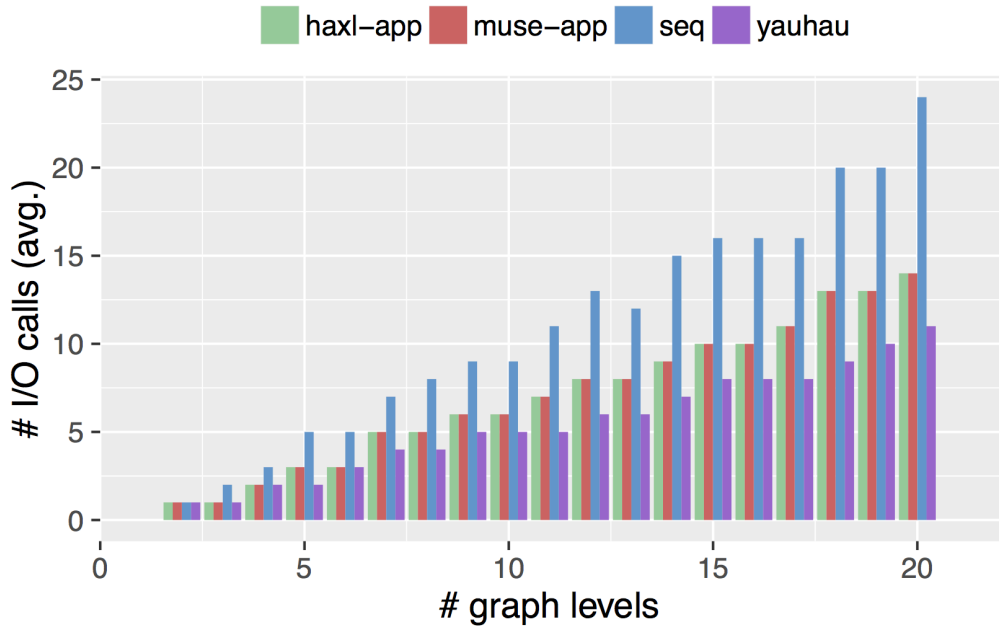
Ohua

- Ohua: Functional programming abstraction to implicitly create dataflow graphs
  - Functional program: High-level algorithm (clojure)
  - State-full functions: actual application logic (clojure, java, scala, …)

```
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)))
```

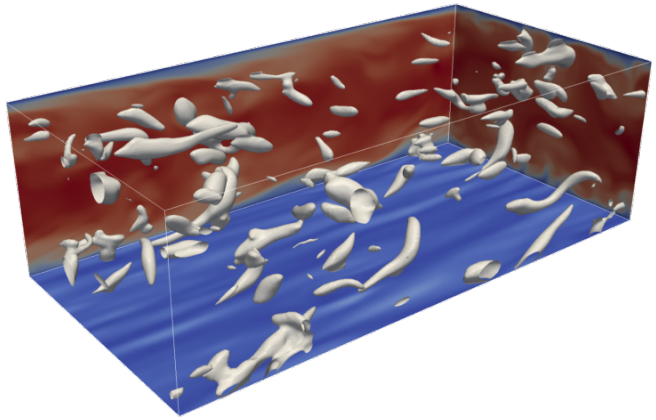
Ohua applications: Micro-services

- Automatic batching of I/O in micro-services: Via graph rewrites
- Similar performance than e.g., Facebook, with better code style



Sebastian Ertel, Andres Goens, Justus Adam, and Jeronimo Castrillon. "Compiling for Con- cise Code and Efficient I/O". In: Proceedings of the 27th International Conference on Compiler Construction (CC 2018)

CFDlang for computational fluid dynamics (CFD)



- Tensor expressions typically occur in numerical codes

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

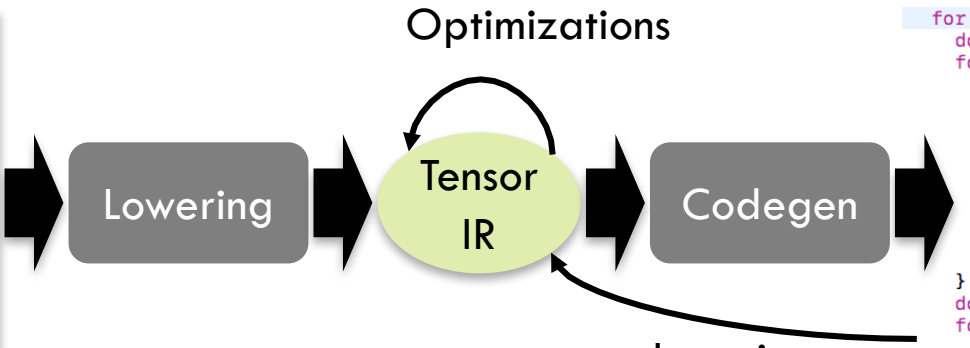
  - Tensor product notation popular in the CFD domain
- On performance
  - Matrixes are small, so libraries like BLAS don't always help
  - Expressions result in deeply nested for-loops
  - Performance highly depend on the *shape* of the loop nests

- **No need to do complex polyhedral analysis if we know the tensors and the semantics of the operators!**

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



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, J. Stiller, J. Fröhlich, and C. Tadonki. “CFDlang: High-level code generation for high-order methods in fluid dynamics”. RWDSL 2018

Example: Interpolation operator

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

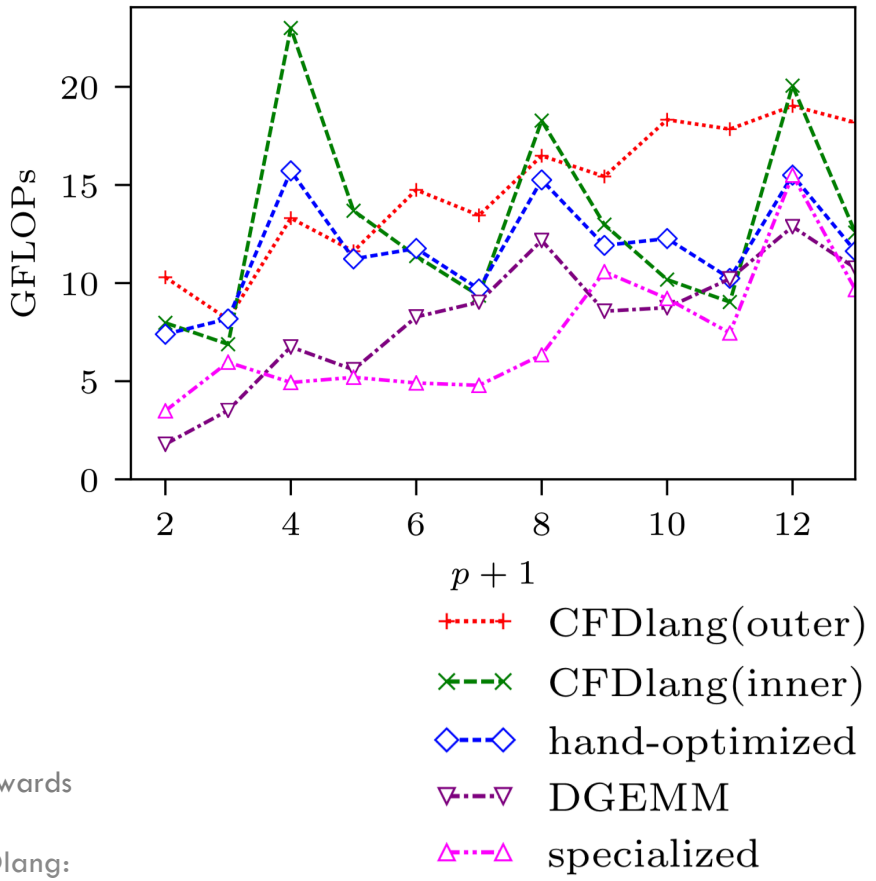
$$v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot u_{lmn}$$

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



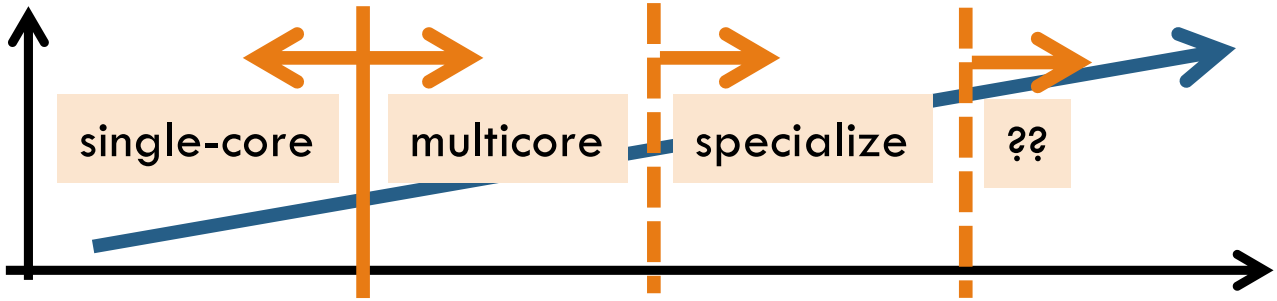
A. Susungi, N. A. Rink, J. Castrillon, I. Huismann, A. Cohen, C. Tadonki, J. Stiller, J. Fröhlich, "Towards Compositional and Generative Tensor Optimizations" GPCE 17

N. A. Rink, I. Huismann, A. Susungi, J. Castrillon, J. Stiller, J. Fröhlich, and C. Tadonki. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL 2018

Agenda

- Introduction
- Programming models: Overview
- Systematic parallel programming
- **Future electronics**
- Wrap-up

Extreme heterogeneity



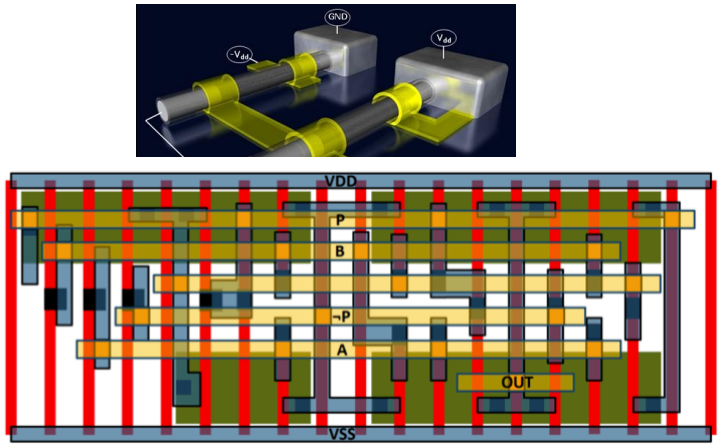
- Inflection points
  - End of frequency scaling: Multicores
  - End of Dennard scaling and power density: heterogeneous systems
  - Physical limits of CMOS: Extreme heterogeneity (new materials, new paradigms)

- Many ideas: quantum, neuromorphic, protein-based, DNA storage, spin-orbit-tronics, organic, …

Context: Center for advancing electronics Dresden (2)

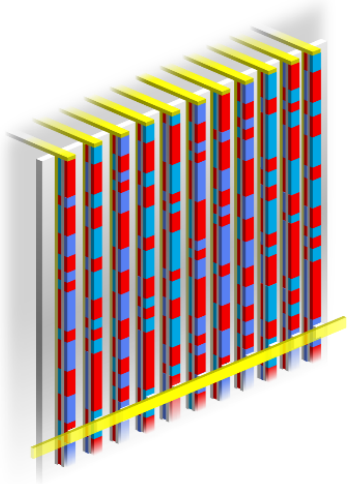
- Sample technologies

Novel transistor reconfigurability



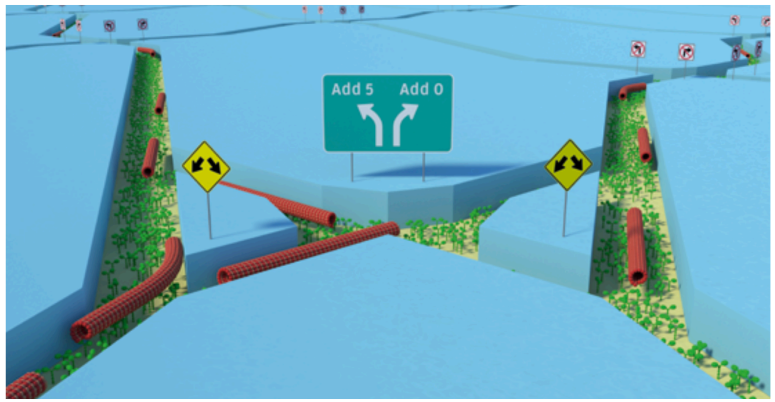
M. Raitza, et al., "Exploiting Transistor-Level Reconfiguration to Optimize Combinational Circuits" , DATE 2017

Spin-orbit Racetracks



Parkin, US patents 6834005, 6898132.
Parkin et al., Science 320, 190 (2008).
Parkin, Scientific American (2009).

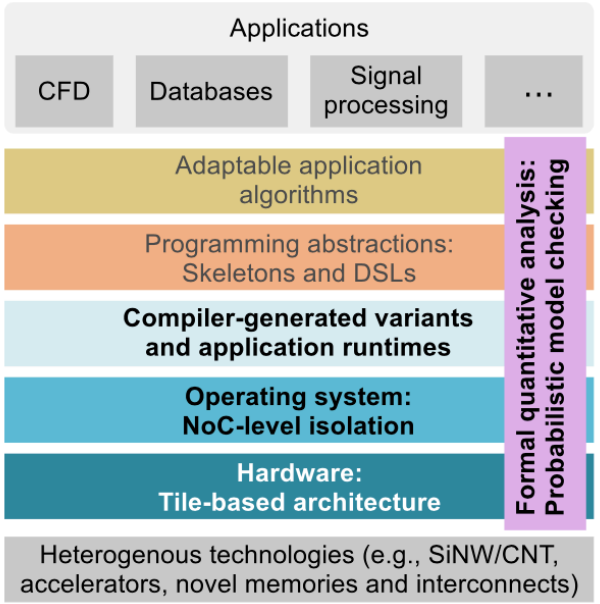
Protein-based computing



Nicolau, Dan V., et al. "Parallel computation with molecular-motor-propelled agents in nanofabricated networks". PNAS 2016.

SW for extreme heterogeneity

- Working on principles for a programming stack
- Programming abstractions
  - High-level: DSLs
  - Lower-level: Dataflow execution models
- Execution abstractions
  - Application runtimes for adapativity
  - Micro-kernel based OSes



J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems". IEEE TMSCS, 2017

- Requires models: SW people require closer communication with technologists!

SW for extreme heterogeneity (2)

- Working on principles for a programming
- Programming abstractions
  - High-level: DSLs
  - Lower-level: Dataflow execution models
- Execution abstractions
  - Application runtimes for adapativity
  - Micro-kernel based OSes

Report from Dagstuhl Seminar 17061

Wildly Heterogeneous Post-CMOS Technologies Meet Software

Edited by

Jerónimo Castrillón-Mazo[1], Tei-Wei Kuo[2], Heike E. Riel[3], and Matthias Lieber[4]

1 TU Dresden, DE, jeronimo.castrillon@tu-dresden.de
2 National Taiwan University – Taipei, TW, ktw@csie.ntu.edu.tw
3 IBM Research Zurich, CH, hei@zurich.ibm.com
4 TU Dresden, DE, matthias.lieber@tu-dresden.de

Abstract

The end of exponential scaling in conve
years by now. While advances in fab
predicted, the so anticipated end seems t
"Wildly Heterogeneous Post-CMOS Tec
material research, hardware componen
systems. By bringing together experts
interdisciplinarily across fields, the sem
challenges of advancing computing bey
visions about a future hardware/softwa

- **Requires models: SW people require closer communication with technologists!**

Agenda

- Introduction
- Programming models: Overview
- Systematic parallel programming
- Future electronics
- **Wrap-up**

Wrap-up

- Discussed facts that continue to lead innovation in electronics
- Need for abstractions and models: Software research
  - Examples: DSLs and dataflow models
- Cfaed: Next inflection point in computing? → makes things even more challenging (and exciting!)
  - Research opportunities on computer architecture (open source simulators)
- What will be talking about at the 100 year's celebration?
  - I believe EE students can play a pivotal role

References

Asanovic, K.; Bodik, R.; Catanzaro, B. C.; Gebis, J. J.; Husbands, P.; Keutzer, K.; Patterson, D. A.; Plishker, W. L.; Shalf, J.; Williams, S. W. & Yelick, K. A. The Landscape of Parallel Computing Research: A View from Berkeley. EECS Department, University of California, Berkeley, 2006

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

"Optimizing Compilers for Modern Architectures: A Dependence-Based Approach", Allen and Kennedy, 2002, Ch. 2.

T. J.K. Edler von Koch, S. Manilov, C. Vasiladiotis, M. Cole, and B.Franke. Towards a Compiler Analysis for Parallel Algorithmic Skeletons. In: Proceedings of the 2018 International Conference on Compiler Construction (CC'18), Vienna, 2018.

Lee, E. A. & Messerschmitt, D. G. Synchronous Data Flow. Proceedings of the IEEE, 1987, 75, 1235-1245

Castrillon, J.; Sheng, W. & Leupers, R. Trends in Embedded Software Synthesis. Proceedings of the International Conference Embedded Computer Systems: Architecture, Modeling and Simulation (SAMOS), 2011, 2011, 347-354.

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

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

Püschel, M.; Moura, J.; Johnson, J.; Padua, D.; Veloso, M.; Singer, B.; Xiong, J.; Franchetti, F.; Gacic, A.; Voronenko, Y.; Chen, K.; Johnson, R. & Rizzolo, N. SPIRAL: Code Generation for DSP Transforms. Proceedings of the IEEE, 2005, 93, 232 -275

Sebastian Ertel, Andres Goens, Justus Adam, and Jeronimo Castrillon. "Compiling for Concise Code and Efficient I/O". In: Proceedings of the 27th International Conference on Compiler Construction (CC 2018)

A. Susungi, N. A. Rink, J. Castrillon, I. Huismann, A. Cohen, C. Tadonki, J. Stiller, J. Fröhlich, "Towards Compositional and Generative Tensor Optimizations" GPCE 17

N. A. Rink, I. Huismann, A. Susungi, J. Castrillon, J. Stiller, J. Fröhlich, and C. Tadonki. "CFDlang: High-level code generation for high-order methods in fluid dynamics". RWDSL 2018

J. Castrillon, et al. "A Hardware/Software Stack for Heterogeneous Systems". IEEE TMSCS, 2017

M. Raitza, et al., "Exploiting Transistor-Level Reconfiguration to Optimize Combinational Circuits" , DATE 2017

Parkin, US patents 6834005, 6898132.

Parkin et al., Science 320, 190 (2008).

Parkin, Scientific American (2009).

Nicolau, Dan V., et al. "Parallel computation with molecular-motor-propelled agents in nanofabricated networks". PNAS 2016.

J. Castrillon, T.-W. Kuo, H. E. Riel, M. Lieber, "Wildly Heterogeneous Post-CMOS Technologies Meet Software (Dagstuhl Seminar 17061)" , In Dagstuhl Reports (Jerónimo Castrillón-Mazo and Tei-Wei Kuo and Heike E. Riel and Matthias Lieber) , Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, vol. 7, no. 2, pp. 1–22, Dagstuhl, Germany, Aug 2017.