Scheduling languages:

Past, Present & Future

Prof. Anne C. Elster, PhD

Dept. of Computer Science (IDI)
Norwegian Univ. of Science & Technology
(NTNU Trondheim, Norway)
&
Univ. of Texas at Austin (visiting faculty)

Where you will find me..

NTNU Gløshaugen

(formerly Norwegian Institute of Technology)

U of Texas at Austin

2023/24 Sabbatical
(forskningstermin)
& summers ++

Thank yous to:

My Post Docs and graduate students!

06/07:Spring 2007

07/08:@ SC 07

08/09:Spring 2009

09/10:Spring 2010

10/11: @ SC 10

11/12:Spring 2012

12/13: 8 Master students!

13/14 Spring 2014

14/15 Spring 2015 & 15/16 (UT Sabbatical), 16/17:

Anders Wenhaug + PhDs & 2 Post Docs

17/18 Spring 2018

And all our collaborators:

HPC-Lab members & friends
with Tucker Taft, Spring 2014

Colleagues at

CGF SFI Centre for Geophysical Forecasting

Based on several slides by Ingo Müller, Google MCSR Group:

Google

Scheduling Languages: A Past, Present, and Future Taxonomy

Mary Hall (1), Cosmin Oancea (2), Anne C. Elster (3), Ari Rasch (4), Sameeran Joshi (1), Amir Mohammad Tavakkoli, Richard Schulze (4).

(1) University of Utah, (2) University of Copenhagen, (3) Norwegian Institute of Science and Technology, (4) University of Muenster.

https://arxiv.org/abs/2410.19927

MCSR Reading Group, Nov. 14, 2024

Final paper sent to TACO
(**ACM Transactions on Architecture and Code Optimization**):

Scheduling Language Chronology: Past, Present, and Future

MARY HALL, University of Utah, USA
COSMIN OANCEA, University of Copenhagen, Denmark
ANNE C. ELSTER, NTNU, Norway; Univ. of Texas at Austin
ARI RASCH, University of Muenster, Germany
SAMEERAN JOSHI, University of Utah, USA
AMIR MOHAMMAD TAVAKKOLI, University of Utah, USA
RICHARD SCHULZE, University of Muenster, Germany

https://arxiv.org/abs/2410.19927

Outline

- **Overview (Section 1)**
- Past (Section 2)
- Present (Section 3)
- Future (Section 4)
- Related Work (was Section 5, now merged into text)
- Discussion

Overview

1990s, before scheduling languages:

- Compilers are black boxes
- Difficult to predict best sequence of optimizations

1997-2012, work leading up to scheduling languages (aka. “the past”)

- Idea: measure the compiled code

  → auto-tuning, iterative compilation, code variants

- Subsequent work: expose these knobs to expert users

Overview – continued:

2013-2023, today's scheduling languages (aka. "the present", aka. "the Halide decade")

Idea: separate specification from schedule pioneered by Halide

Interface for human experts or for automated search

2024 and beyond, the authors' vision (aka. "the future")

Higher level of abstraction, based on common infrastructure

Overview – continued:

Table 1. Motivations and resulting technology related to scheduling languages in the past, present, and expected future.

Timeline	Motivation	Focus	Approaches
PAST 1997 - 2012 *Explore*	Heuristic-based code optimization decisions ineffective Improve efficiency of expert users	Primarily loop nest computations Embedded & scientific applications & libraries	Self-tuning libraries Exploratory compilers and code generators Rewriting rules & lang. support for code variants
PRESENT 2013 - 2023 *Specialize*	Efficiency of expert users in specific domains	Domain specific languages and compilers	Separate high-level specification and schedule Narrow the search space to utilize autotuning and ML
FUTURE 2024 - *Popularize*	Increase user accessibility Raise abstraction	Broaden to more general applications Unify and incorporate into common infrastructures	Data layout/movement integration Runtime support Expand search space while maintaining practicality

Outline

- Overview (Section 1)
- **Past (Section 2)**
- Present (Section 3)
- Future (Section 4)
- Related Work (was Section 5, now merged into text)
- Discussion

Past: Autotuning

- **Main idea: parametrize program, then find best parameters empirically**
- **Early influential work: ATLAS (1998/2001) for tuning BLAS routines**
  - Parameter search for sub-problems
  - Comparison of code variants
  - Clever composition of the above
  - == manual (and less powerful) equivalent of scheduling language?
  - Modern vendor libraries still do autotuning (plus other things, including Intel MKL and CuBLAS)

Similar work on bitreversal, sparse linear algebra, FFT, signal processing, …

All of the above: not really compiler techniques nor languages…

Past: Iterative Compilation

- **Idea: explore a parameter/decision space in the compiler**

- Early work: OCEANS compiler for matrix multiplication
  - Applies tiling, unrolling, and padding
  - Then searches good parameters
  - High search cost → ~exhaustive search?

- **Subsequent work includes:**
  - Limiting the search space
  - *Optimization space exploration*
  - *Tournament predictor*

- All of the above: search is driven by compiler and not exposed to the outside

Past: Autotuning Compilers and Code Generators

- **Idea: give expert users more control**

- Several approaches with aspects of today's scheduling languages

- Examples: **XLang** (pragmas) and **CHiLL** (external script)

- Related approach: code generators including **ORIO** and **POET**

Past: Autotuning Compilers & Code Generators

```
// code, named loops, xforms
#pragma xlang name iloop
for (i=0; i<NB; i++)
  #pragma xlang name jloop
  for (j=0; j< NB; j++)
    #pragma xlang name kloop
    for (k=0; k<NB; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll MUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.stores after kloop
```

(a) Xlang

```
// code
DO J=1,N
  DO K=1,N
    DO I=1,N
      C(I,J)=C(I,J)+
          A(I,K)*B(K, J)
```

```
// CHiLL
// transformation recipe
permute([3,1,2])
tile(0,2,TJ)
tile(0,2,TI)
tile(0,5,TK)
datacopy(0,3,2,[1])
datacopy(0,4,3)
unroll(0,4,UI)
unroll(0,5,UJ)
```

(b) CHiLL

Fig. 1. Examples of expressing locality optimizations for matrix multiply in Xlang[DBR+06] and CHILL[CCH07].

Past: Language-Compiler Co-design

- **Idea: expose aspects of compilation to the programming language**

- **Rewrite-rule systems in purely functional languages:**
  - Allow users to write rewrite rules/patterns in the source language
  - "Democratize compiler design"
  - Influential work: **Haskell GHC compiler**

- **Decomposition and algorithmic variants**
  - Idea: task decomposition and variant selection are made explicit
  - Example work: Sequoia (next slide)
  - Similar work: **PetaBricks** and work on **FFT**

Sequoia: Specs (left), Schedule (right)

```
1  void task matmul::inner( in    float A[M][P]
2                         , in    float B[P][N]
3                         , inout float C[M][N] ){
4    // tunable parameters specify the
5    // size of subblocks of A, B, C
6    tunable int U, X, V;
7
8    // Partition matries into sets of blocks
9    blkset Ablks = rchop(A, U, X);
10   blkset Bblks = rchop(B, X, V);
11   blkset Cblks = rchop(C, U, V);
12
13   // Compute all blocks of C in parallel
14   mappar(int i=0 to M/U, int j = 0 to N/V) {
15     mapreduce (int k=0 to P/X) {
16         matmul( Ablks[i][k]    // recursive
17               , Bblks[k][j]    // invocation
18               , Cblks[i][j] ); // on subblocks
19     }
20   }
21 }
22
23 void task matmul::leaf( in    float A[M][P]
24                       , in    float B[P][N]
25                       , inout float C[M][N] ){
26   for (int i=0; i<M; i++)
27     for (int j=0; j<N; j++)
28       for (int k=0; k<P; k++)
29         C[i][j] += A[i][k] * B[k][j];
30 }
```

(a) Algorithmic Specification

```
1  instance {
2    name    = matmul_cluster_inst
3    task    = matmul
4    variant = inner
5    run_at  = cluster_level
6    calls   = matmul_node_inst
7    tunable U = 1024, X = 1024, V = 1024
8    A distribution = 2D block-block
9              (blocksize 1024x1024) ... }
10 instance {
11   name    = matmul_node_inst
12   task    = matmul
13   variant = inner
14   run_at  = node_level
15   calls   = matmul_L2_inst
16   tunable U = 128, X = 128, V = 128   }
17 instance {
18   name    = matmul_L2_inst
19   task    = matmul
20   variant = inner
21   run_at  = L2_cache_level
22   calls   = matmul_L1_inst
23   tunable U = 32, X = 32, V = 32
24   subtask arg A = copy
25   subtask arg B = copy                }
26 instance {
27   name    = matmul_L1_inst
28   task    = matmul
29   variant = leaf
30   run_at  = L1_cache_level            }
```

(b) Instantiation to a Cluster Hardware.

Outline

- Overview (Section 1)
- Past (Section 2)
- **Present: What is in a Schedule? (Section 3)**
- Future (Section 4)
- Related Work (was Section 5, now merged into text)
- Discussion

Present: Image Processing

- **Halide** as the most influential work on separating schedules from algorithms
- **Observation (industry-wide):** specialization gives rise to efficiency
- **Halide: Image processing**
  - Express programs as (1) high-level algorithm and (2) optimization recipe
  - Mainly used to express stencil operations
  - Optimization needs to decide:
    - Granularity of compute
    - Granularity of load/store
    - Order of traversal
  - Array computations → implicitly parallel, transformations are always legal
  - Recipes are written by human expert or found in autotuning (days to weeks)

Present: Image Processing

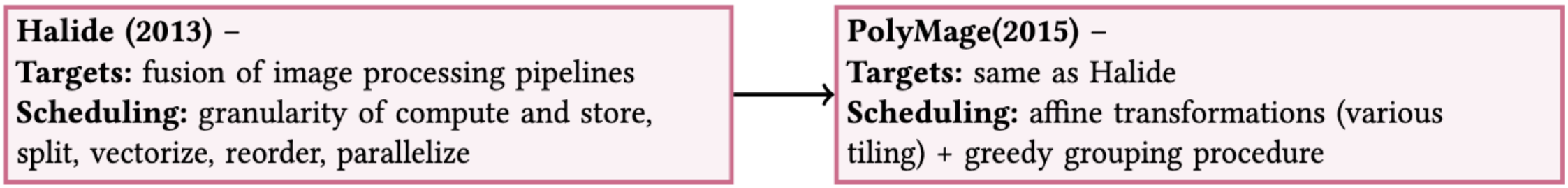


Fig. 5. Overview of two of the central works related to DSL scheduling languages targeting image processing.

- Follow-up work: **PolyMage**
  - Constraints search space to most relevant parts (plus other things)
  - Much faster (one hour)

- More follow-up work improves Halide or it or its ideas to other domains:

Present: Image Processing

- **Modesto** (CPU+GPU), **Absinthe** (CPU) and **Stencil-Gen** [114] (GPU) rely on analytical models to determine the optimal schedule using various tiling strategies (including overlapped tiling with streaming), storage optimizations and (greedy) fusion heuristics.

Works applying dynamic analysis to optimize stencils:

- **OPS** uses a combination of delayed execution and dependence analysis to resolve at runtime hindrances to static analysis typically for large applications,

- **ARTEMIS** uses bottleneck analysis via runtime profiling to guide the application of optimizations, and the tuning of various code generation parameters.

- **ImageCL /AUMA** autotunes by using ML for performance optimizations on CPUs/GPUs, as well as for load-balancing several GPUs on a node or GPUs on a cluster

Present: Tensor Algebra

TCE(2002) –
Targets: dense tensors
Scheduling: compiler exploration instead of scheduling

Barracuda (2015) –
Targets: GPU + domain-specific optimizations
Scheduling: CUDA-CHiLL and Orio

Distal (2022) –
Targets: CPU + GPU clusters
Scheduling: loop transformation + specification of data distribution and communication

CHiLL-I/E (2014) –
Targets: sparse tensors
Layouts: CSR, ELL, DIA, BCSR
Scheduling: inspector/executor+sparse polyhedral
e.g. coalesced, parallel, reduction, make-dense, tile, skew

TACO (2017) –
Targets: co-iteration over multiple tensors
Layouts: per dimension sparse/dense,
COO, ELL, DIA, hashmap
Scheduling: + pos, coord, precompute, bound

Mosaic (2023) –
Targets: integration of optimized library kernels
Scheduling: Bind, Map

TeALL(2023) –
Targets: concise specification for accelerators
Scheduling: rank-order, partitioning, loop-order, space-time

Fig. 7. Overview of some central works related to scheduling languages for dense (blue) and sparse tensor algebra (green).

Present: Tensor Algebra (dense)

- **Tensor Contraction Engine (TCE)**
  - Contractions of 4-dimensional tensors with hundreds of terms
  - Order of operations matters
  - Enumerate feasible solutions (DRAM), minimize total ops, fusion heuristics + search and search space pruning

- **Barracuda:** 4-dimensional tensor contractions
  - Exploratory compilation: mathematical transformations + autotuning using random forest search
  - Uses scheduling languages internally (CUDA-CHiLL and Orio)

- **Fireiron:** dense matrix multiplication on NVIDIA GPUs
  - Introduces scheduling primitives to target HW extensions (tensor cores)

- **DISTAL:** scheduling language for tensor algebra on CPU and GPU clusters
  - Loop transformations + communication across nodes + data format

Present: Tensor Algebra (sparse)

- **Sparse Linear Algebra:**
  - Data format stores only non-zero elements
  - Computations only involve non-zero elements
  - Mostly memory-bound instead of compute-bound

- **CHiLL-I/E:**
  - Extends CHiLL to sparse linear algebra
  - Uses inspector/executor paradigm
  - Scheduling language can express
    - standard transformations (coalesce, parallel, reduction) and
    - new ones (make-dense, compact, compact-and-pad)

Present: Tensor Algebra (sparse) continued:

- **TACO:** co-iteration over sparse tensors
  - User specifies layout, compiler generates co-iteration code
  - Can express the most common sparse layouts declaratively
  - Reimplemented in the MLIR sparsifier
  - Scheduling language: split, collapse, reorder, unroll, parallelize, precompute

- **Mosaic:** extend TACO with integration of library kernels

Present: Data parallel approaches

Lift (2015) : purely-functional, map-reduce
Targets: mainly GPU hardware
Scheduling: based on functional rewrite rules

Tiramisu (2019) : C++ embedded DSL
Targets: multiple platforms including multicore, GPU, FPGA, distributed systems
Scheduling: polyhedral transformations

MDH (2023) : purely-functional, map-reduce
Targets: correctness checks, autotuning, visualization. **Hwd**: multicore and GPU
Scheduling: single primitive for systematically expressing (de/re)-composition of computations, based on MDH formalism [109]

DaCe (2019) : Python embedded DSL
Targets: framework for defining own scheduling primitives based on a data-centric IR called *SDFG*. **Hwd**: multicore, GPU, FPGA
Scheduling: affine + user-defined transformations, based on SDFG

Fig. 8. Overview of some of the central works focusing on data parallel approaches and how they are related.

Present: Data parallel approaches

Data-Parallel Computations

- **Lift:** purely-functional programming language with rewrite rules with search and scheduling language

- **Tiramisu:** Similar to Halide but using four-level IR

- **Locus:** scheduling framework for optimizing legacy code (C, C++, Fortran, …)

- **DaCe:** framework for implementing DSLs and optimization pipelines

- **MDH:** express composition at each level of memory and core hierarchy

Present: Deep Learning & Graphs

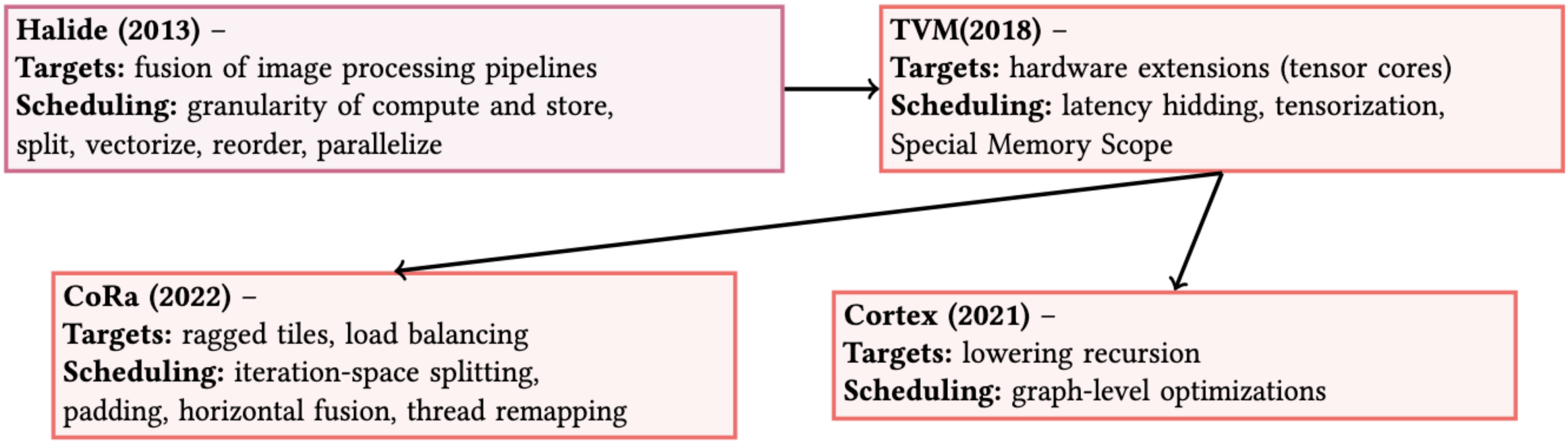


Fig. 9. Overview of some of the central works related to scheduling languages targeting Deep Learning.

Present: Deep Learning & Graphs

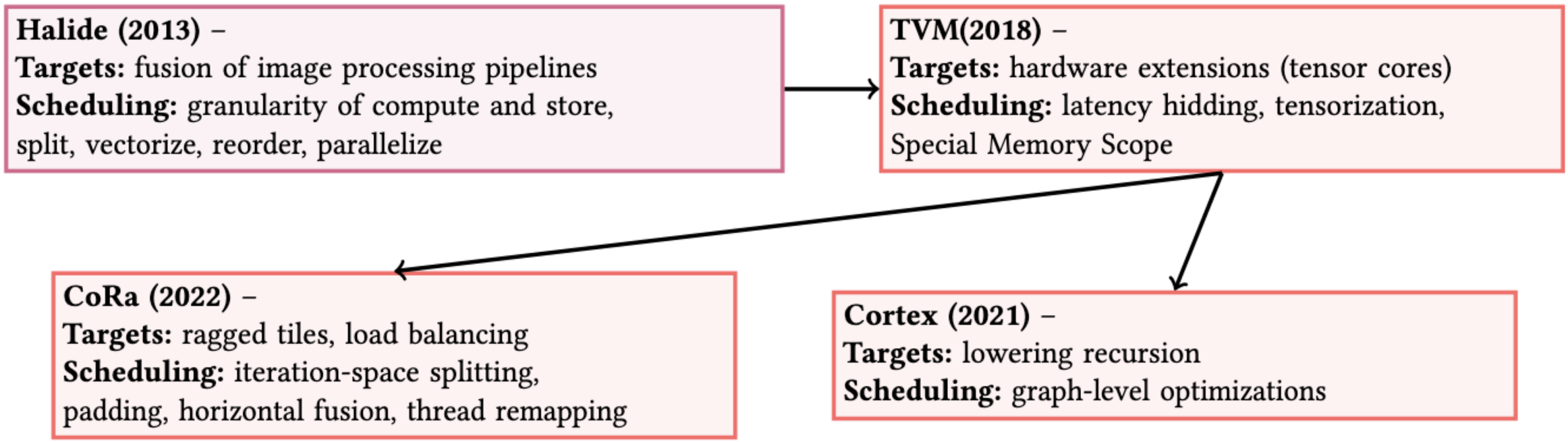


Fig. 9. Overview of some of the central works related to scheduling languages targeting Deep Learning.

Present: Deep Learning & Graphs

Machine learning (as a target application domain)

- **TVM:** deep learning workloads on CPUs, GPUs, and TPUs
  - Extends Halide's scheduling language with architecture-specific constructs

- Extensions: **CoRa, Cortex**
- Autotuning without scheduling language: **Triton**

Graphs

- **Galois** (relaxed graph traversal order) + **GraphIt** (scheduling language)

Outline

- Overview (Section 1)
- Past (Section 2)
- Present (Section 3)
- **Future: Call to Action (Section 4)**
- Related Work (was Section 5, now merged into text)
- Discussion

Future: Call to Action

- **Past:** performance with general-purpose compilers
- **Present:** performance with lang (restricted) . + scheduling lang. for various domains
- **Future:**
  - Lift restrictions
  - Open to domain experts
  - Automate schedule generation + selection

Future:
Unifying Scheduling Languages

- **OpenMP**: high quality but slow
- **Alternative idea:** integrate scheduling language into popular compiler framework
- Example: **MLIR** and transform dialect
  - Optional compiler pass
  - Not really a scheduling language
  - Layers on top could improve this
- **Alternative idea:**
  - Co-design high-level languages with transformations on them
  - Example: functional language with inference rules

    → democratize language and compiler design

Future: Data Movement/Layout, Modern HW, …

- **Observation: data movement is key**

  → should be universal part of future work

- Recent work already does this:
  - **CHiLL, CUDA-CHILL, Fireiron, MDH, Graphene**

- Future work should integrate this into high-level compilers (e.g. **MLIR**)

Future: Data Movement/Layout, Modern HW, … contin.

Other themes:

- Need to be able to **exploit architecture-specific HW** features
  - Some recent work does this (Fireiron, Graphene, Exo, …)

- Need to **integrate with runtime** to support inspector/executor or take data-dependent decisions (parameter, variant, …) at runtime
  - Current scheduling languages have weak support for this.

- **Raise the level of abstraction to accommodate for domain experts**

Future: Raising Abstraction Level

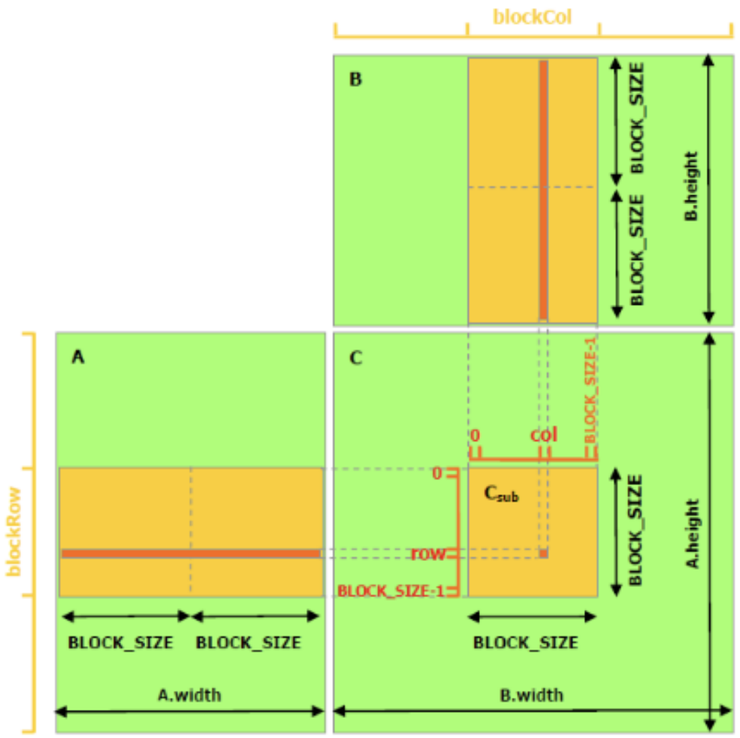
Approach 1: Pair original program with a specification of the high-level (loop) structure of the target (optimized) code.

- E.g., the original program could be annotated with
- (comment) labels identifying loops and code fragments and the labeled names can be used in the specification
- to declare the target code structure.

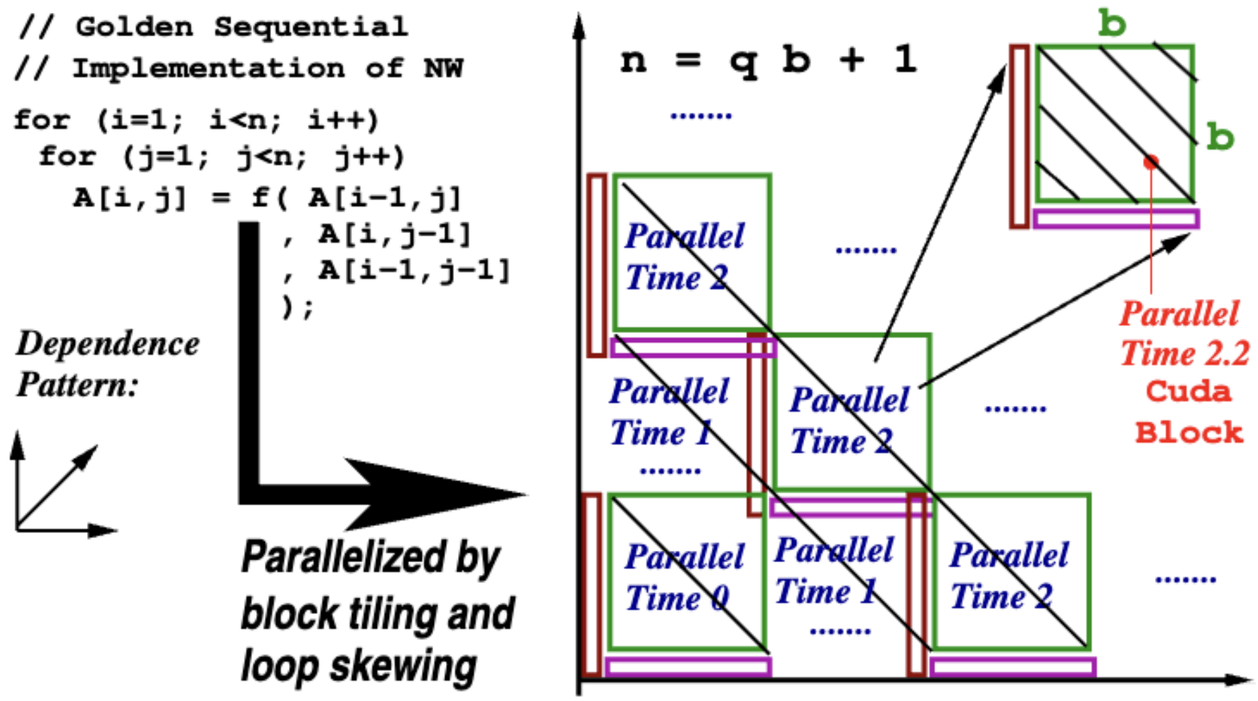
- Cjahallenge: inferring the sequence of (loop) code transformations whose application conforms with the target code structure and in designing a clear semantics of the specification language that is intuitive and promotes ease of use.

Future: Raising Abstraction Level

Approach 2: Describe via pictures the array slices produced/used at each level of the hardware, respectively.



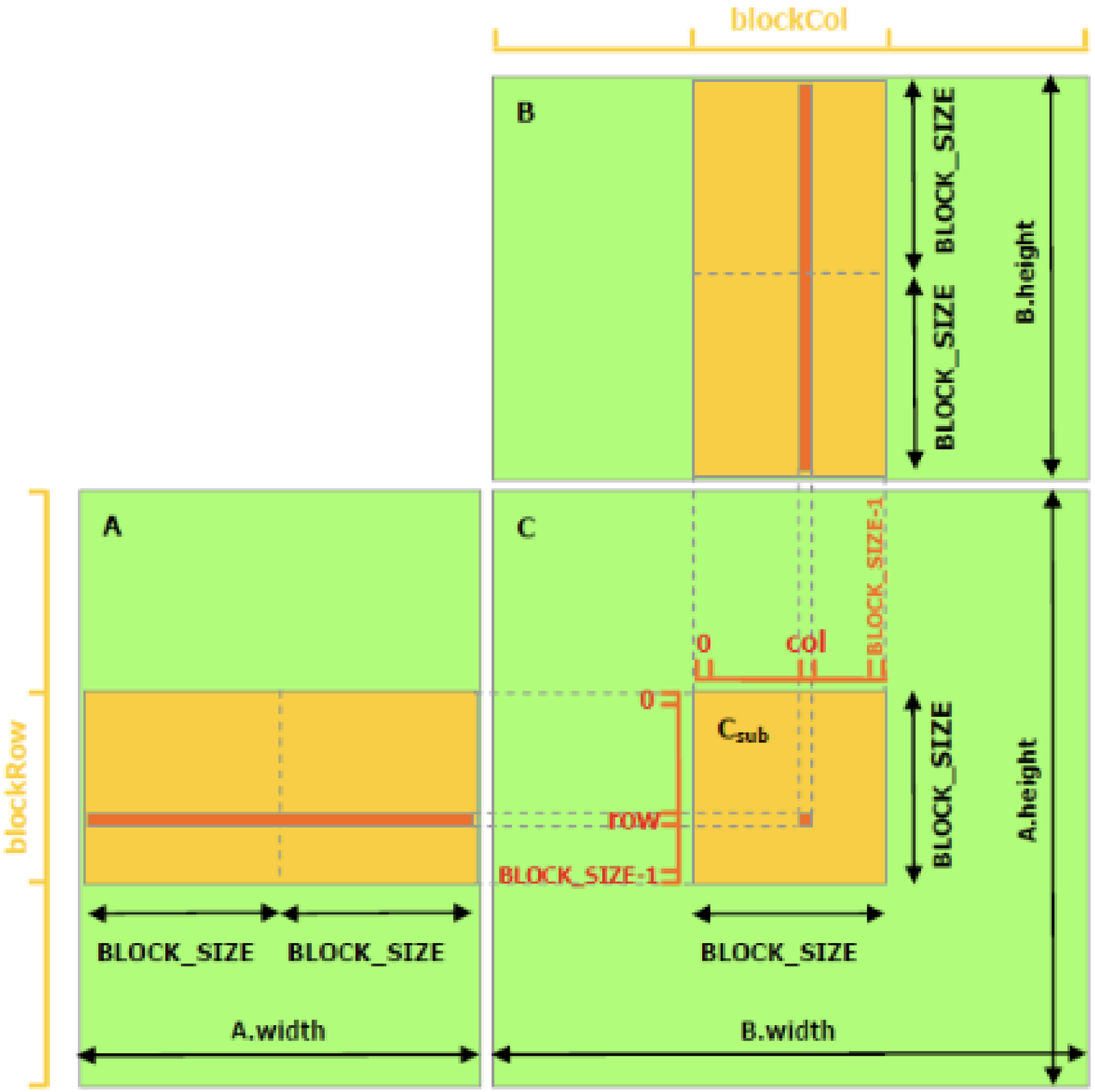
(a) Picture schedule for Matrix Multiplication (MM)



(b) Picture Schedule for Needleman-Wunsch (NW).

The former requires tiling at each hardware level, and the latter requires tiling and two loop-skewing transformations.

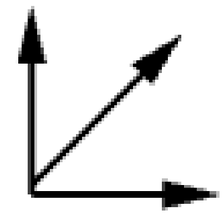
Future: Raising Abstraction Level



(a) Picture schedule for Matrix Multiplication (MM)

```
// Golden Sequentia
// Implementation o
for (i=1; i<n; i++)
  for (j=1; j<n; j++
    A[i,j] = f( A[i-
                , A[i,
                , A[i-
                );
```

Dependence Pattern:



Parallelize block tili loop ske

(b) Picture S

Outline

- Overview (Section 1)
- Past (Section 2)
- Present (Section 3)
- Future (Section 4)
- **Related Work (was Section 5, now merged into text)**
- Discussion

Future: Related work

Ingo Müller told us he didn't have time to read this section

But pointed out some promising sections:

. Integrating Scheduling Languages with Autotuning (Section 5.2)

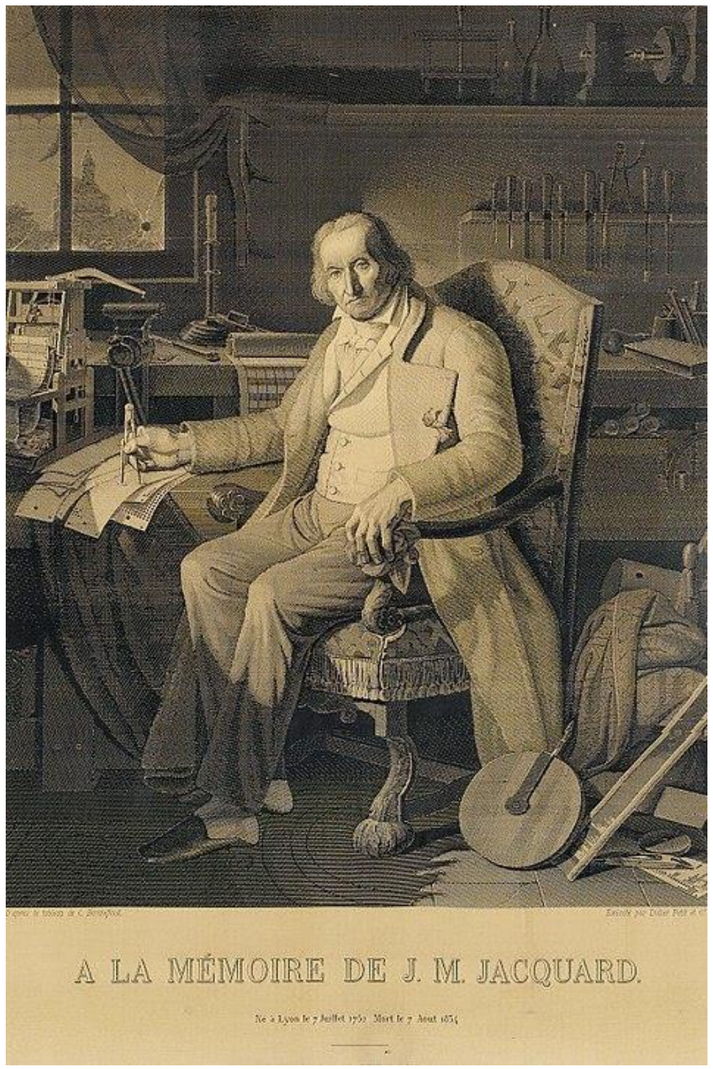
. Tuning using Machine Learning (Section 5.3)

The final paper submitted cites 162 papers

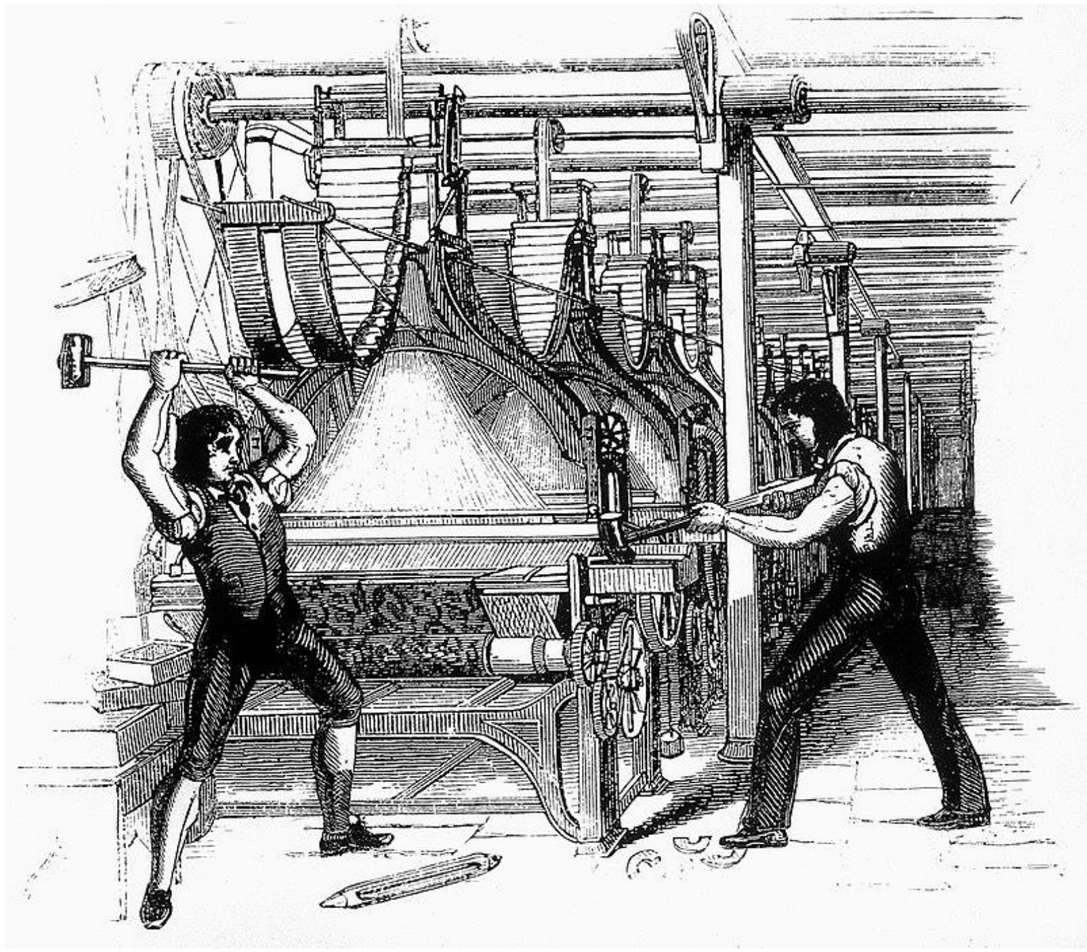
Outline

- **Overview (Section 1)**
- Past (Section 2)
- Present (Section 3)
- Future (Section 4)
- Related Work (was Section 5, now merged into text)
- **Discussions/ Questions?**

AI not the first technology threat



Silk weave from 1829 of Jaquard made using 24 000 punch cards!



Luddites smashing powerlooms in 1812
from https://en.wikipedia.org/wiki/Power-loom_riots

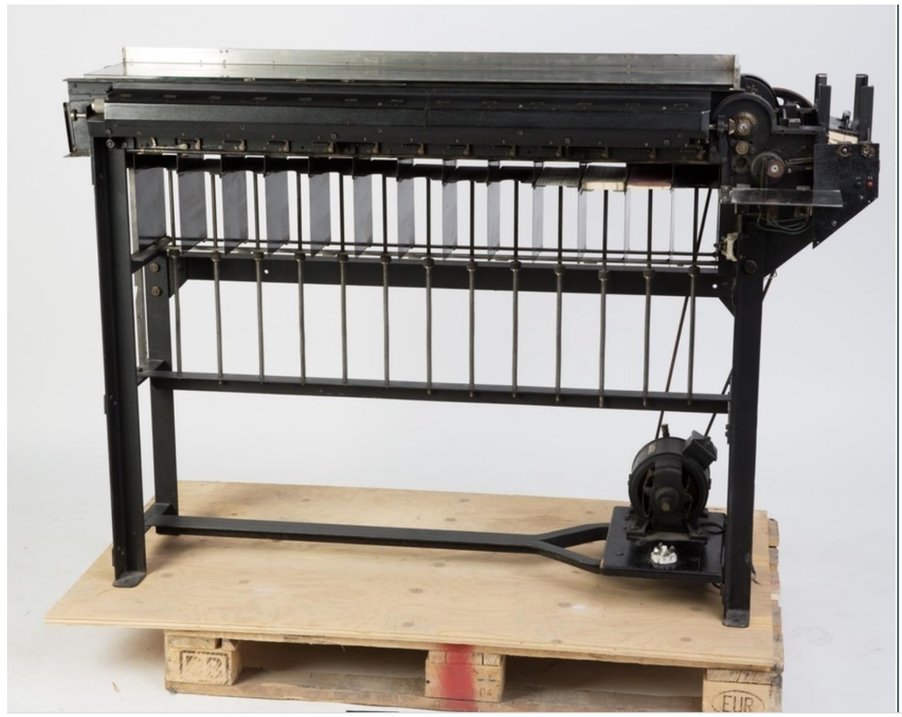
Replaced thousands of skilled weavers..

https://spectrum.ieee.org/the-jacquard-loom-a-driver-of-the-industrial-revolution

https://www.scienceandindustrymuseum.org.uk/objects-and-stories/jacquard-loom

Fredrik Rosinge Bull & A/S OKI

1922-25: Bull sold 14 machines through contract with
A/S OKA lead by classmate
Reidar Knutsen, mainly to insurance companies



A/S OKA Bull Machine from ca 1921, Horizontal model.
(From Teknisk museum, Oslo)

1924: Norway´s Statistisk Sentralbyrå replaced their Hollerith with improved Bull model
–> Hollerith´s first real competitor.

1924: Hollerith´s company renamed
«International Business Machines»
a.k.a. **IBM**

Source: snl.no/Fredrik_Rosing_Bull

Bull → Honeywell Bull

1925: Bull dies of cancer.
Company taken over by Reidar´s brother
Knut Andreas Knutsen who moved production
to Switzerland and later France.

1936-38:
Konrad Zuse
First elektro-
Mechanical
Computer, Z1

1930: Accolade @ Paris Fair for office machines

1931: **Egli-Bull** established

1933: Renamed **CMB** (Compagnie des Machines Bull)

1936-45 War efforts – see separate slide

1964: 66% taken over by **General Electric,**
the rest **Bull SA** owned by French government
for **military research**

1970: GE Computer division → Honeywell Inc
→ **Honeywell Bull**

Atos **Bull**Sequena rack

Jülich Supercompter JUWEL

ATOS

2014: **Controlling stakes in Bull SA**

Aug 2014: Controlling stake of
Xerox IT outsourcing business
for USD 1 050 000 000

Oct 2018 Aquired Syntel($3.4 Billion)

2020: **Jülich Supercompter JUWEL**

2021: Operates in 73 countries,
110 000 employees
Revenue: €11.58 billion (2019)

Jan 2021: Proposal to acquire **DXC Technologies,**
but bailed out Feb 2021

https://exanode.eu/partners/bull/

www.bull.com (now maps to atos.net)

ARM

ARM IP in many processors, including **Apple´s M1** & **Nvidia´s new Grace CPU**

ARM has acquired dozens of smaller IT companies including

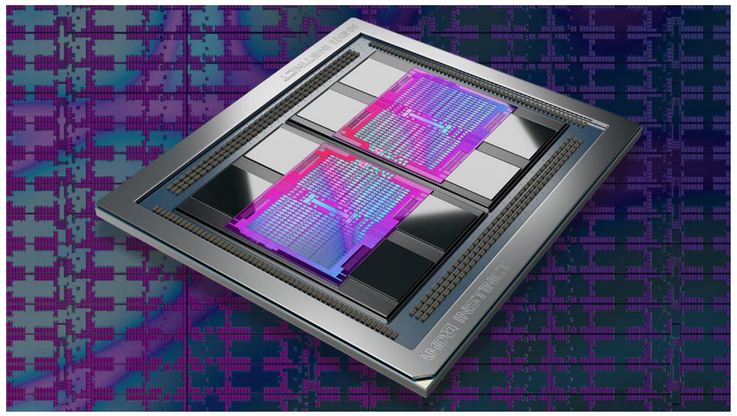
Falanx Microsystems, an NTNU spin-off in Trondheim, Norway

→ARM Norway

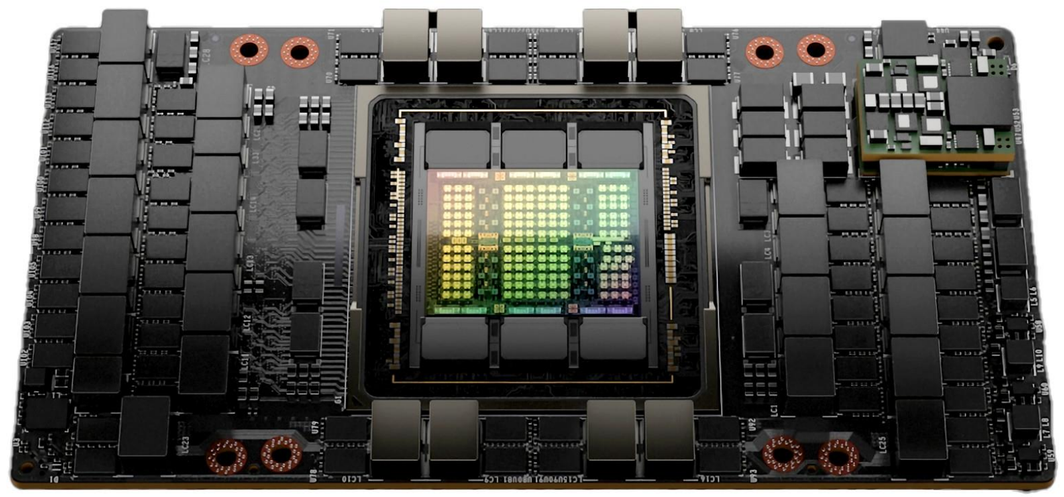
→ARM Mali (GPU) processors

(Fredrik Kjølstad used to work at ARM Norway. Aksel Hjerpbakk from HPC-Lab joined 2022)

Paralell Computing &
HPC (High Performance Computing)



AMD Instinct MI250X



Nvidia H100 (Hopper GPU)

LUMI Supercomputer in Finland

Anne C. Elster NTNU/IDI – 2024

HPC Challenge
-- Programming for Performance:

The challenges of data locality

1. Feed the parallel cores registers fast enough
2. Move compute to data
3. Pick the right algorithm (that addresses with the data distribution)

I.e.

Computing is now like real estate – it is all about:

LOCATION, LOCATION & LOCATION!!

Come il prezzo degli immobili!

Recursion, Autotuning & AI

- Bit-reversal & FFTs
  – utilize recursion e.g. FFTW, cu-FFT, etc

- Parameterization & ML

- How to use technology for AI for HPC
  – Numerical codes
  – Compilation tools

Elster´s Bit-reversal algorithm

https://www.jb101.co.uk/2017/01/08/paper-study-fast-bit-reversal-algorithms-by-elster.html

Paper Study - Fast Bit-Reversal Algorithms by Elster

Jan 8, 2017

Given a sequence of integers in the range $[0..2^t)$, its bit-reversal permutation is given as a sequence of integers whose bits are in reverse order to the original sequence.

For example, for $t = 3$:

```
Input:  0, 1, 2, 3, 4, 5, 6, 7 == 000, 001, 010, 011, 100, 101, 110, 111
Output: 0, 4, 2, 6, 1, 5, 3, 7 == 000, 100, 010, 110, 001, 101, 011, 111
```

The bit-reversal permutation is commonly used in Fast Fourier Transform algorithms, to reorder the input sequence into an order that makes computing recursively smaller DFTs convenient (the necessary elements are adjacent to each other).

Main HPC Tricks of the trade:

USE LIBRARIES!!

- **BLAS (MKL/Atlas/cuBLAS…)**
  - **Basic linear subprograms**
    (Matrix-vector/matrix multiplication,
    rank-k updates, triangular solves ++

- **FFTW**

- **PetSC (Conjugate gradient, SOR, GMRES +++)**

- **NVIDIA CUDA Libraries +++**

AI Challenges

The challenges of data

1. **Quality (avoiding bad data, bias, etc)**
2. **Representative enough training set**
3. **Heterogenous data**

I.e.

AI is a lot about:

DATA, DATA & DATA!!

2015: HPC-Lab´s First ML paper at IPDPS Workshop:

“Machine Learning Based Auto-tuning for Enhanced OpenCL Performance Portability”

Thomas L. Falch, Anne C. Elster

(Submitted on ArXiv 2 Jun 2015)

Test bench: Intel i7 3770 CPU, an Nvidia K40 GPU and an AMD Radeon HD 7970 GPU.

Achievement:

Our model achieved a mean relative error as low as 6.1%,
and was able to find configurations as little as 1.3% worse than the global minimum

Comments: The ArXiv version is a pre-print version of an article later published in the Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). For personal use only
Subjects: Distributed, Parallel, and Cluster Computing (cs.DC)
DOI: 10.1109/IPDPSW.2015.85
Cite as: arXiv:1506.00842 [cs.DC]

Master theses (July 2018):

August L. Bækken (co-adviser Dr. Gavin Taylor, USNA/U Maryland)
"ML-based profile analysis of CUDA programs' compiler flag impact"

- Goal: train a neural network in order to predict how compiler optimization settings impact the execution speed of new untested programs

- Used CUDA STK programs with various flags

- Not as successful as hoped…but good training

NTNU

Autotuning Research at HPC-Lab

— Parameterized benchmark suites [MSc: Sund and Kirkhorn]

— Selecting optimal autotuning techniques [MSc: Tørring]

— Benchmarking techniques for autotuning [1]

— Deep learning and NLP-models for autotuning [2]

[1] Jacob O. Tørring, J. C. Meyer, and A. C. Elster. “Autotuning Benchmarking Techniques: A Roofline Model Case Study”. In: 2021 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW). Apr. 2021.

[2] L. Bjertnes, Jacob O. Tørring, and A. C. Elster. “LS-CAT: A Large-Scale CUDA AutoTuning Dataset”. In: 2021 IEEE International Conference on Applied Artifical Intelligence (ICAPAI). Apr. 2021

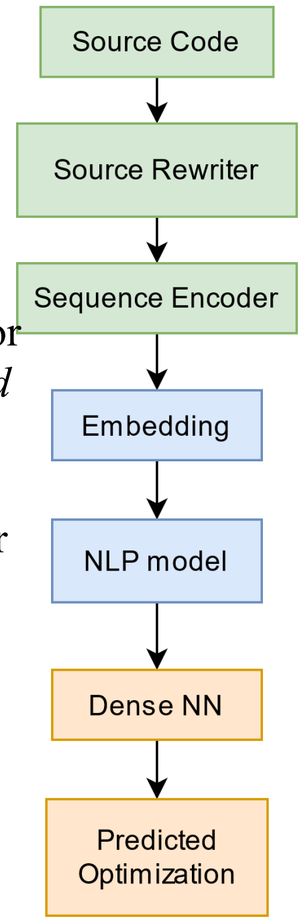
HPC LAB

CGF SFI Centre for Geophysical Forecasting

ML/Deep learning for Autotuning

References

- ML-based Autotuning for OpenCL [1,2]
- DL Autotuning by Cummins et al. [3]
- Large-scale CUDA Dataset for Deep Learning-based autotuning [4]



[1] Thomas L. Falch, Anne C: Elster, Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications, *Concurrency and Computation: Practice and Experience* 29 (8), 2017

[2] TL Falch, AC Elster: ImageCL: Language and source-to-source compiler for performance portability, load balancing, and scalability prediction on heterogeneous systems, *Concurrency and Computation: Practice and Experience* 30 (9), 2018

[3] C. Cummins et al. “End-to-End Deep Learning of Optimization Heuristics”. In: 2017 26th International Conference on Parallel Architecture

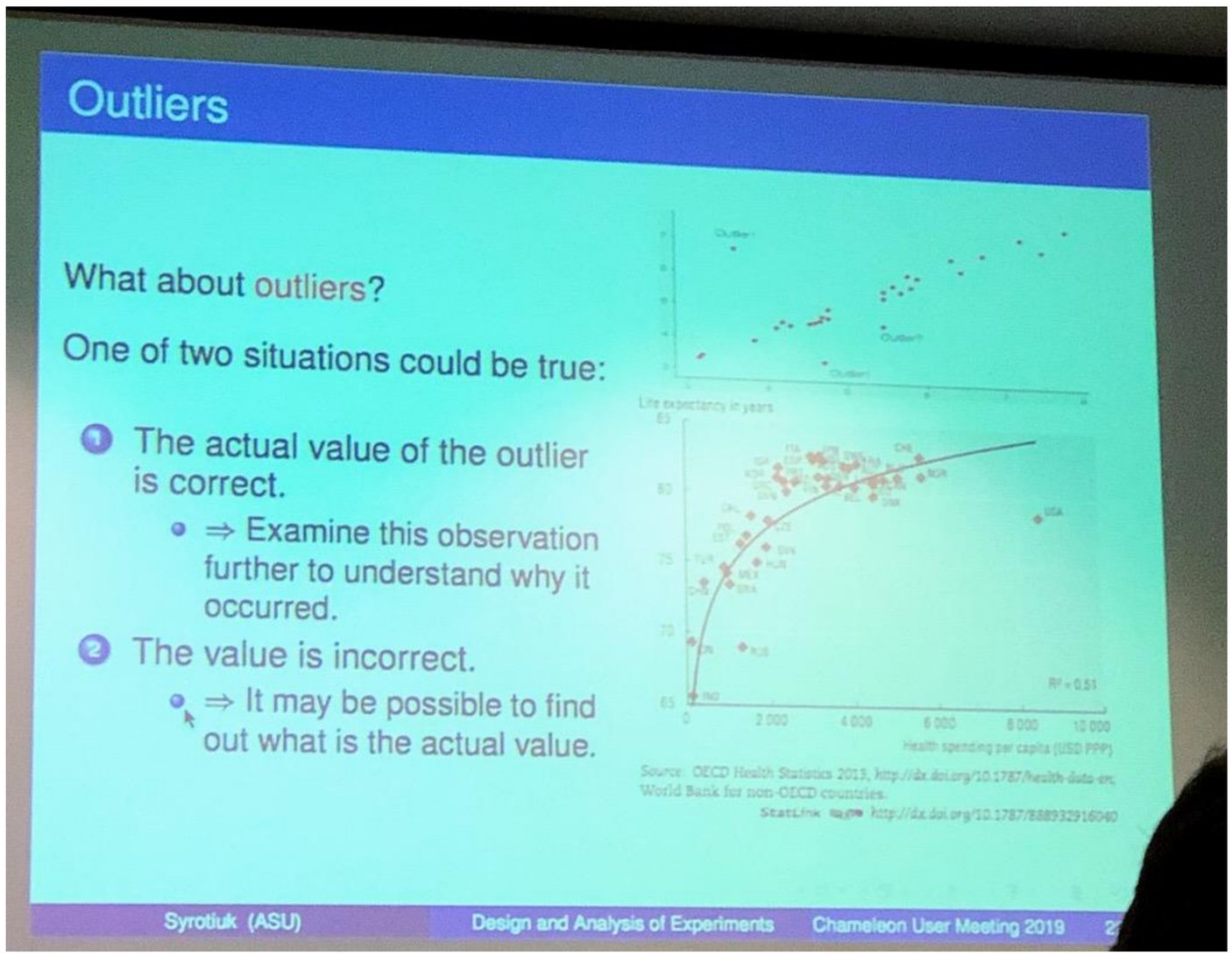
[4] L. Bjertnes, Jacob O. Tørring, and A. C. Elster. “LS-CAT: A Large-Scale CUDA AutoTuning Dataset”. In: 2021 IEEE International Conference on Applied Artifical Intelligence (ICAPAI). Apr. 2021. To appear.

HPC LAB

SFI Centre for Geophysical Forecasting

What about Outliers?

Slide from Dr. Violet R. Syrotiuk, Arizona State
on Design Engineering at TACC Chameleon meeting Feb 2018:



Note:

Variations in datasets
may make ML unpredictable!

Have Fun!

Thank you! / Grazie mille/ Tusen takk!

anne.elster@gmail.com

HPC-Lab members & friends with Tucker Taft, Spring 2014