Practical Solutions that Are Performant, Portable AND Productive for Programming Extreme Scale Systems

Tim Mattson
timothy.g.mattson@intel.com
Parallel Computing Lab
Intel Corp

Disclaimer

- The views expressed in this talk are those of the speaker and not his employer.
- If I say something "smart" or worthwhile:
  - Credit goes to the many smart people I work with.
- If I say something stupid…
  - It's my own fault

I work in Intel's research labs. I don't build products. Instead, I get to poke into dark corners and think silly thoughts… just to make sure we don't miss any great ideas.

Hence, my views are by design far "off the roadmap".

The three talks I'd give if there was time

- The challenge of Hierarchical, Heterogeneous systems:
  - Hardware complexity is out of control (for good reason … hierarchical heterogeneous systems to optimize Perf/Watt). This is great if you are a hardware architect. This is a nightmare if you develop software.

- "Short Term" Solution: oneAPI with Distributed Computing
  - The oneAPI industry initiative defines an open stack of APIs for programming heterogenous nodes … so one API maps onto multiple hardware targets. We just need to extend it to cover distributed computing.

- Long term Solution: Automate Software Development
  - Machines are now really good at learning. Maybe they can handle programming too? They better because the programmers we are generating these days choose languages such as Python that abstract away ALL the hardware details.

The three talks I'd give if there was time

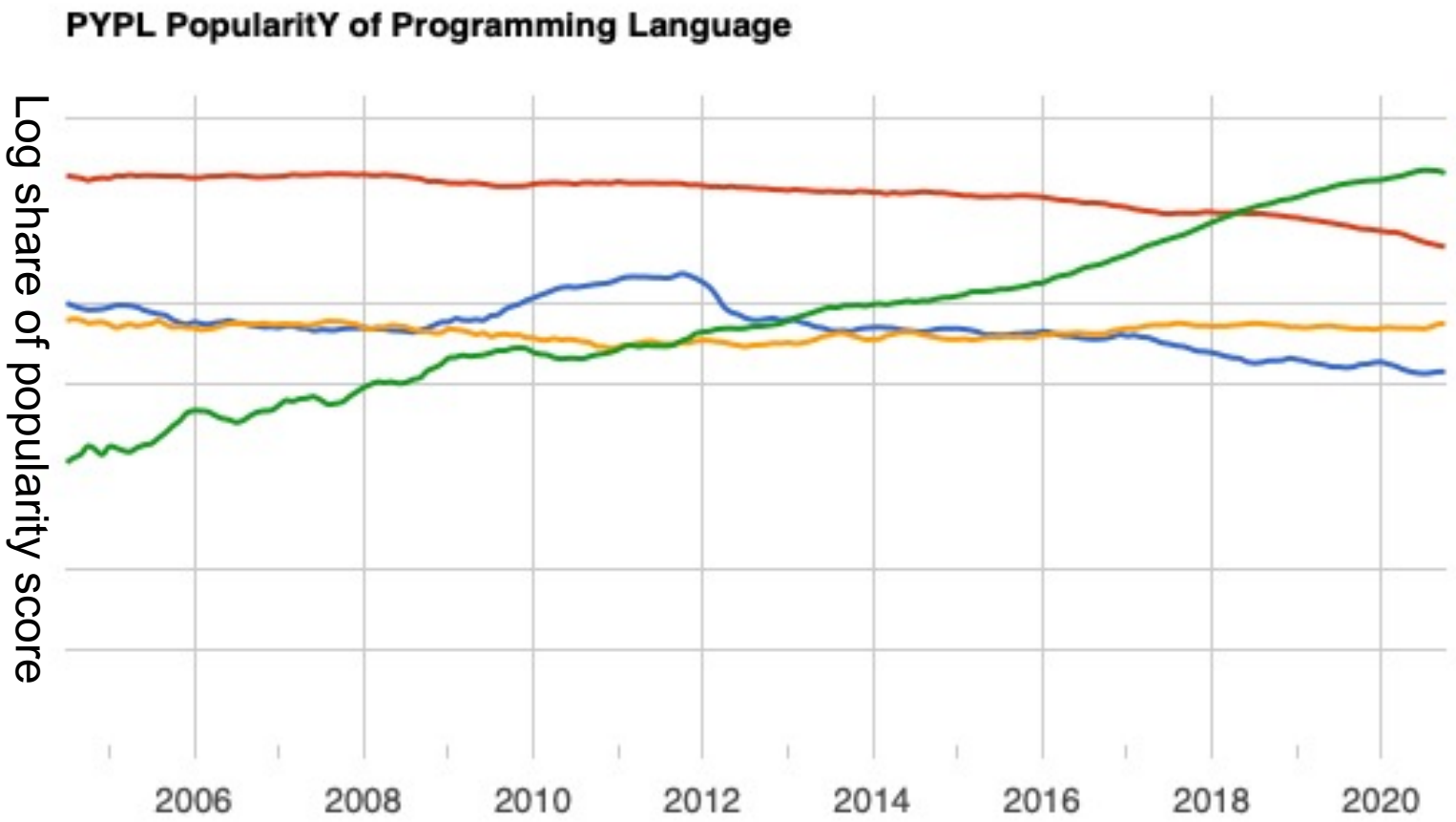
- The challenge of Hierarchical, Heterogeneous systems:
  - Hardware complexity is out of control (for good reason … hierarchical heterogeneous systems to optimize Perf/Watt). This is great if you are a hardware architect. This is a nightmare if you develop software.

- "Short Term" Solution: oneAPI with Distributed Computing
  - The oneAPI industry initiative defines an open stack of APIs for programming heterogenous nodes … so one API maps onto multiple hardware targets. We just need to extend it to cover distributed computing.

- Long term Solution: Automate Software Development
  - Machines are now really good at learning. Maybe they can handle programming too? They better because the programmers we are generating these days choose languages such as Python that abstract away ALL the hardware details.

I think I'll give this talk today

Programming Language trends 2005 to 2020



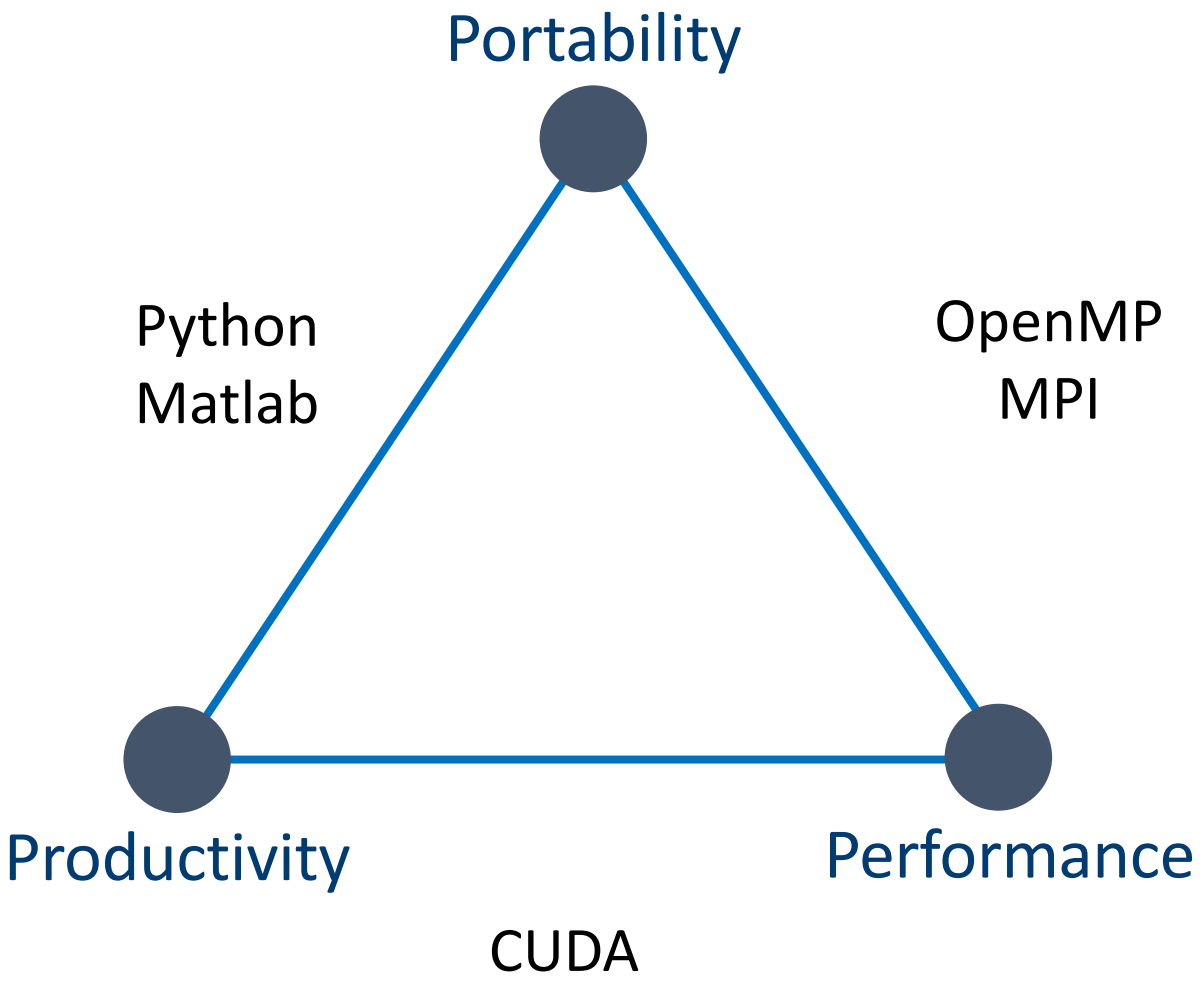
	Share of total scores 2016	2020
C/C++	8%	6%
Java	25%	16%
JavaScript	7%	8%
Python	12%	31%

Professional programmers use Java, C, and C++.
Professionals who develop software use Python

http://pypl.github.io/PYPL.html

Programmers and their needs

The Triangle of programmer needs



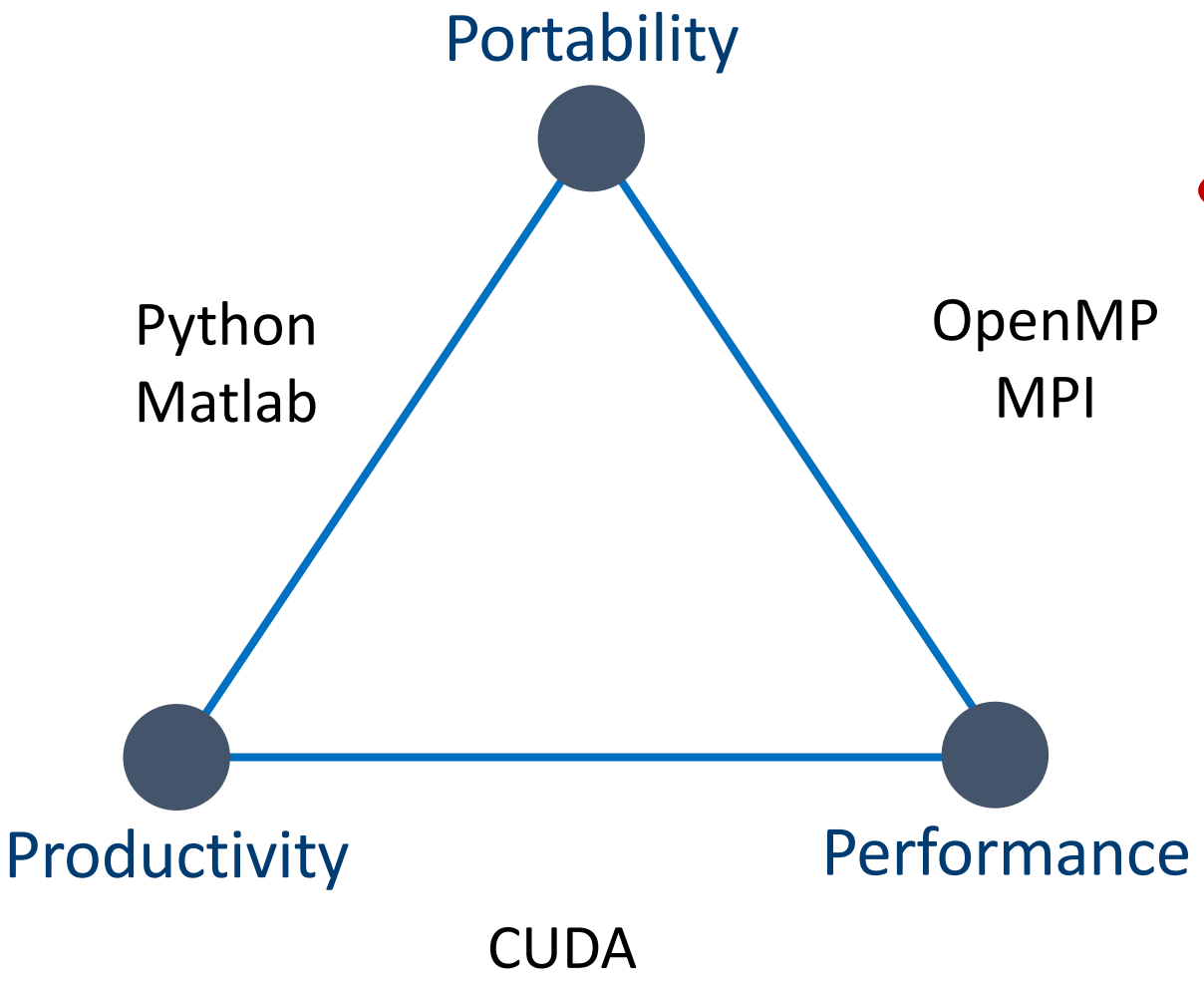
Delivering 2/3 is reasonable,

Getting 3/3 is Prohibitively difficult

3rd party names are the property of their owners

Programmers and their needs

The Triangle of programmer needs



Delivering 2/3 is reasonable,

Getting 3/3 is Prohibitively difficult

We really need 3/3 … it's challenging, but not impossible.

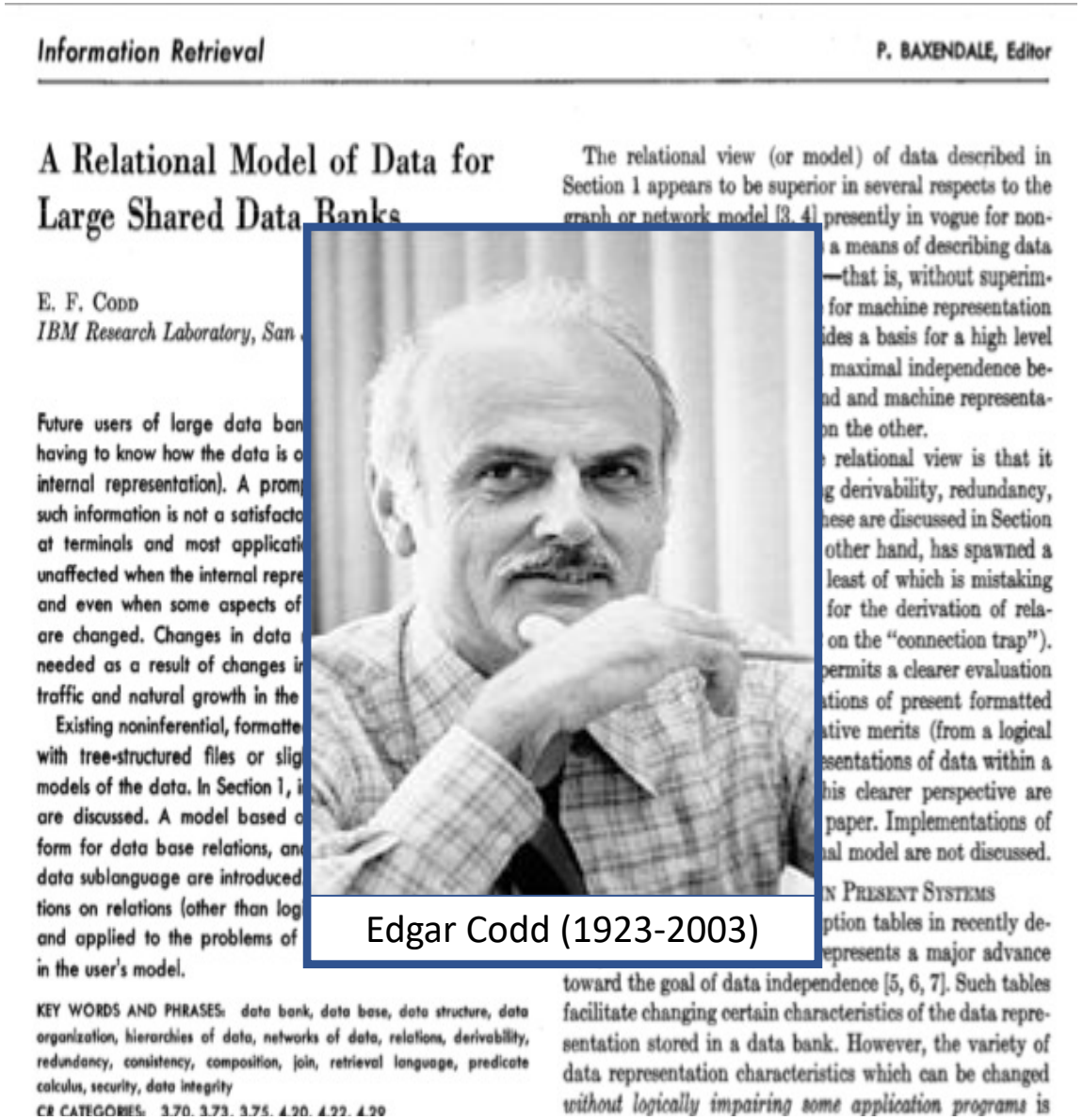
It's been done before

The Relational Model of Databases

- In 1970 Edgar Codd (IBM) published one of the most important papers in the history of computer science.

- It defined a formal algebra* for building databases … the **relational model**.
  - Object: A relation.
    - A set of tuples that share a set of attributes.
    - The set of attributes is defined by a schema
    - A relation is typically represented as a table.
  - A set of operators that act on relations. This set includes:

- Select σ
- Rename ρ
- Join $\bowtie$
- Project π



Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San

Future users of large data ban
having to know how the data is o
internal representation). A prom
such information is not a satisfacto
at terminals and most applicati
unaffected when the internal repre
and even when some aspects of
are changed. Changes in data
needed as a result of changes i
traffic and natural growth in the
Existing noninferential, formatte
with tree-structured files or slig
models of the data. In Section 1, i
are discussed. A model based o
form for data base relations, an
data sublanguage are introduced
tions on relations (other than log
and applied to the problems of
in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-
a means of describing data
—that is, without superim-
for machine representation
des a basis for a high level
maximal independence be-
d and machine representa-
n the other.
relational view is that it
g derivability, redundancy,
ese are discussed in Section
other hand, has spawned a
least of which is mistaking
for the derivation of rela-
on the "connection trap").
permits a clearer evaluation
tions of present formatted
tive merits (from a logical
sentations of data within a
his clearer perspective are
paper. Implementations of
al model are not discussed.

N Present Systems
ption tables in recently de-
represents a major advance
toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is

Edgar Codd (1923-2003)

Communications of the ACM, vol 13, no. 6 p. 337, 1970

* Note: An "algebra" is a set of objects, operators that act on those objects, and rules for how those operators interact with each other

Database Queries and the Relational Model

- Users interact with the relational database by issuing queries.
- Codd proposed elegant and mathematically rich procedural queries:

$$\pi_{e.name}\left(\sigma_{e.salary > m.salary}\left(\rho_e(employee) \bowtie_{e.manager = m.name} \rho_m(employee) \right) \right)$$

- When applied to this relation:

Name	Salary	Manager
Smith	45,000	Harker
Jones	40,000	Smith
Baker	50,000	Smith
Nelson	55,000	Baker

- The output is **Baker** and **Nelson** …. employees who earn more than their managers

*This example comes from ”Early History of SQL” by D. Chamberlin, IEEE Annals of the history of computing, 2012

Queries and the Structured Query Language

- Codd's notation was too obtuse for the general user

$$\pi_{e.name}\ (\ \sigma_{e.salary > m.salary}\ (\ \rho_e(employee) \bowtie_{e.manager = m.name} \rho_m(employee)\ )\ )$$

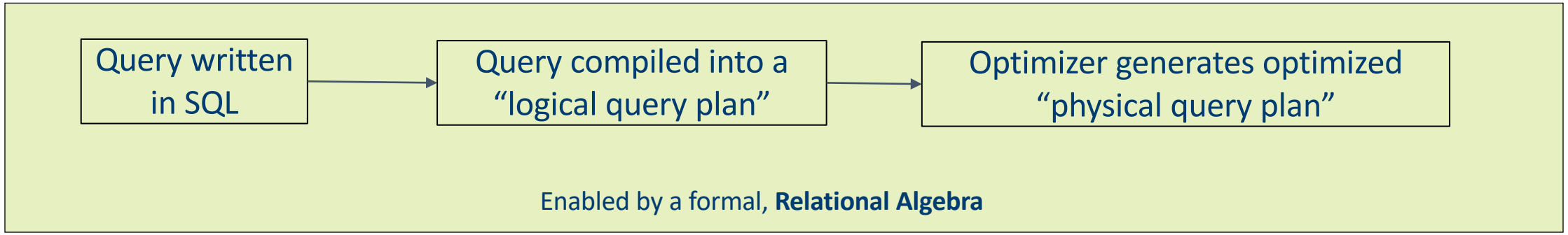
- In 1974, Codd's colleagues at IBM (Ray Boyce and Don Chamberlin) created a Query Language for Codd's relational Algebra called the Structured Query Language (SQL … Pronounced as Sequel)
- The SQL Query equivalent to Codd's notation above reads:

```
select e.name
from employee e, employee m
where e.manager = m.name and e.salary > m.salary
```

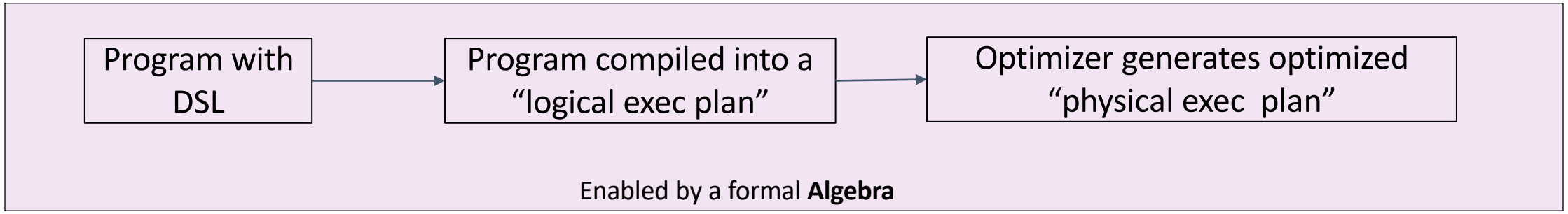
- Codd's notation is procedural. SQL is Declarative … That difference is an important reason for why SQL is the most successful DSL of all time.

Productivity, Performance and Portability in one programming framework

- By the 1980s, database researchers at IBM and UC Berkeley exploited the declarative nature of SQL to build systems that delivered on the “3 Ps” …



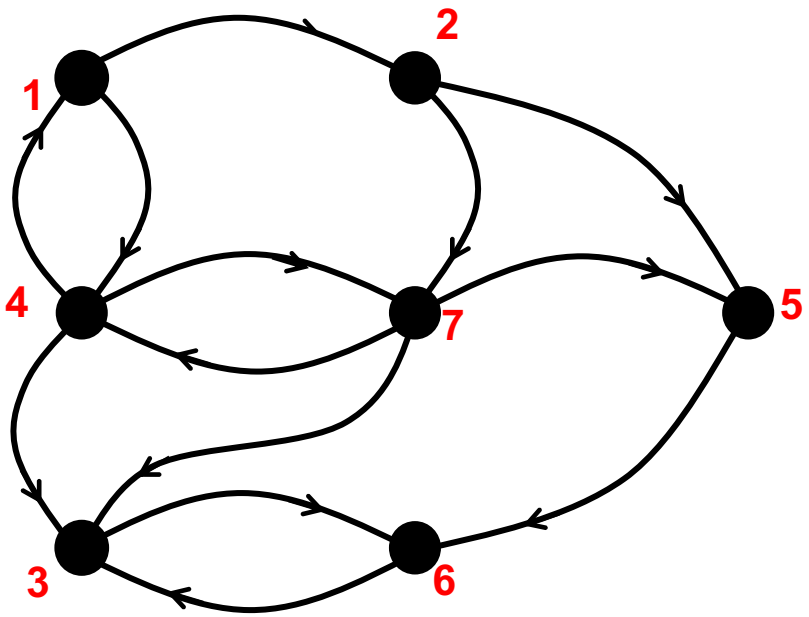
- Can we replicate this strategy for programming heterogeneous systems?

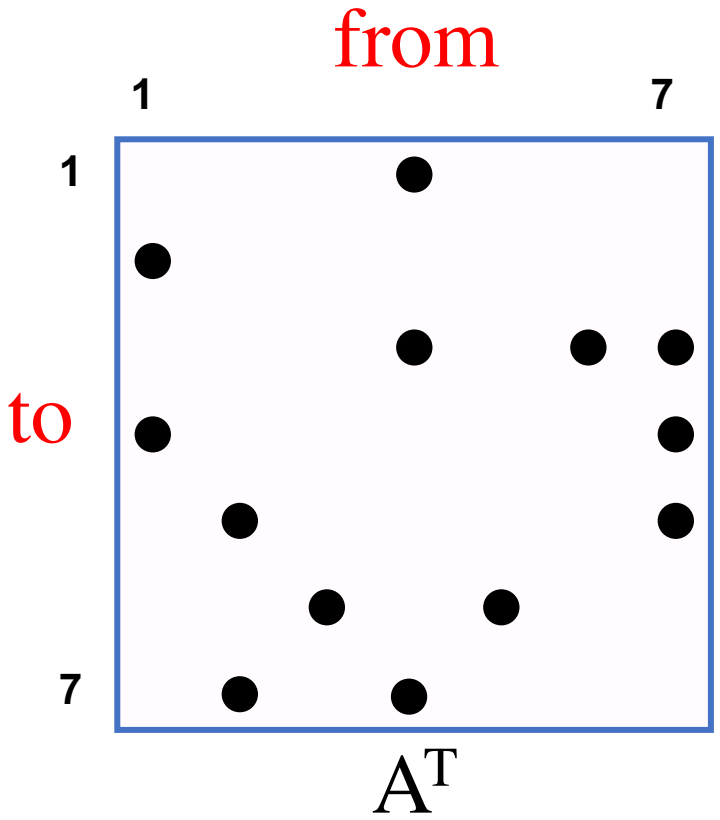


SQL: Structured Query Language

Graphs in the Language of Linear Algebra

These two diagrams are equivalent representations of a graph.



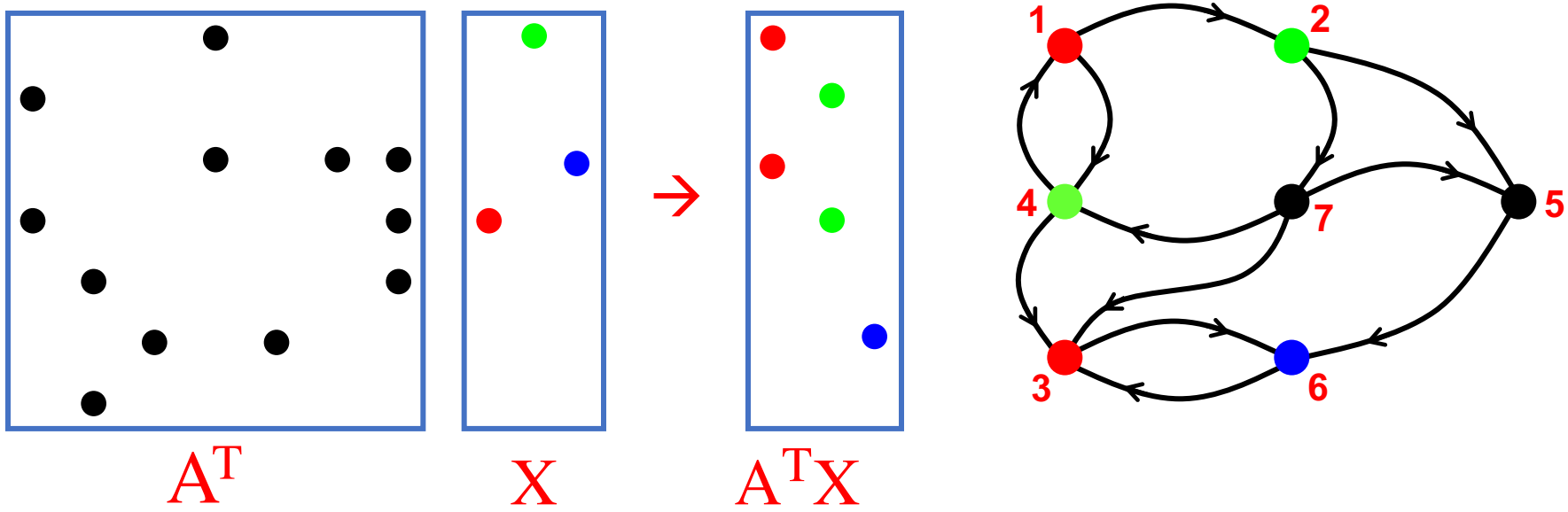


A = the adjacency matrix
Elements nonzero edges connect vertices

Source: Pictures from Aydin Buluc and my colleagues in the GraphBLAS Forum

Multiple-source breadth-first search

Multiplication of sparse matrices captures Breadth first search and serves as the foundation of all algorithms based on BFS



- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

Halide: Focusing on programmer intent

Halide separates the

Algorithm

from the

Schedule

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y)  + input(x, y)  + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32).vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

- Algorithm:
  - What the program does,
  - Written by a domain specialist
- Schedule:
  - How the program runs
  - Written by SW/HW expert

Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines, J Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, PLDI, 2013, https://doi.org/10.1145/2491956.2462176

The common theme between SQL and Halide? Separation of concerns

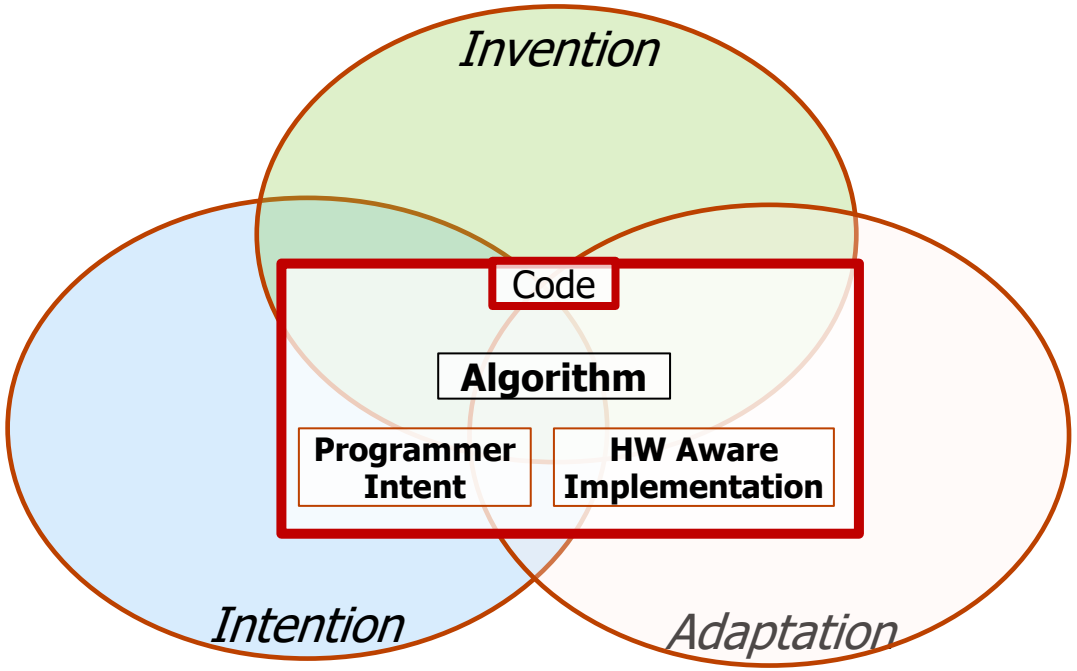
- SQL separates logical plan from the physical plan
  - The query optimizer invents new algorithms using the relational algebra to drive transformations into the emerging plan.
  - The query optimizer than adapts the emerging plan into the physical plan to match the hardware.

- Halide separates programmer intention (the DSL) from how it adapts onto the final hardware

- Isolates the three fundamental concerns of programming
  - Intension
  - Invention
  - adaptation

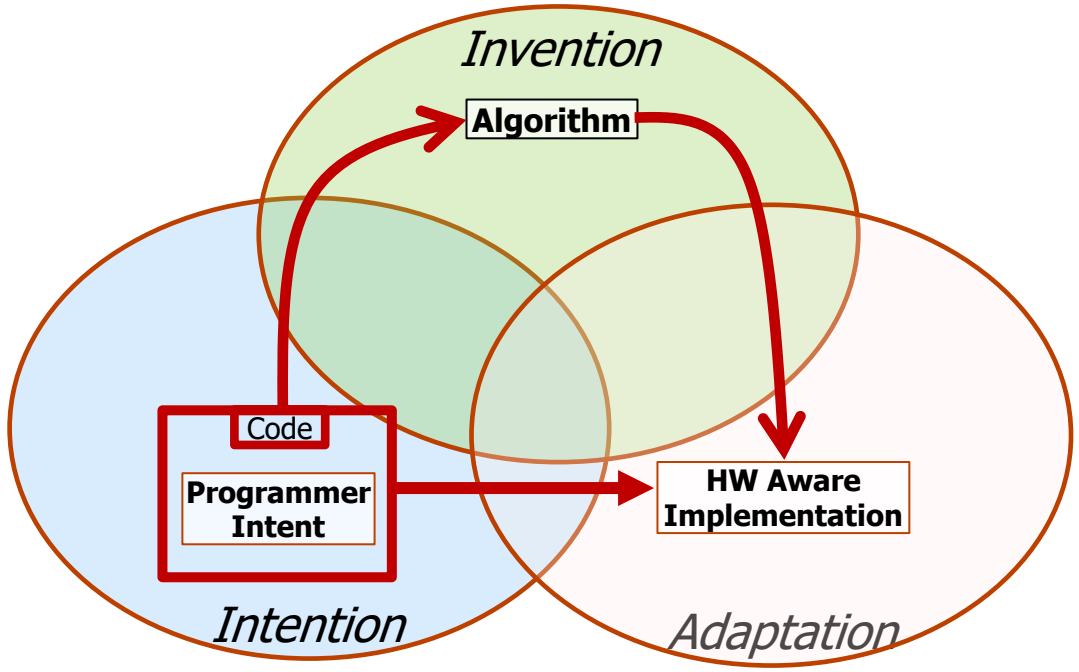
Separation of concerns

- Let's break up the software development process and consider each aspect Separately



Separation of concerns

- Let's break up the software development process and consider each aspect Separately

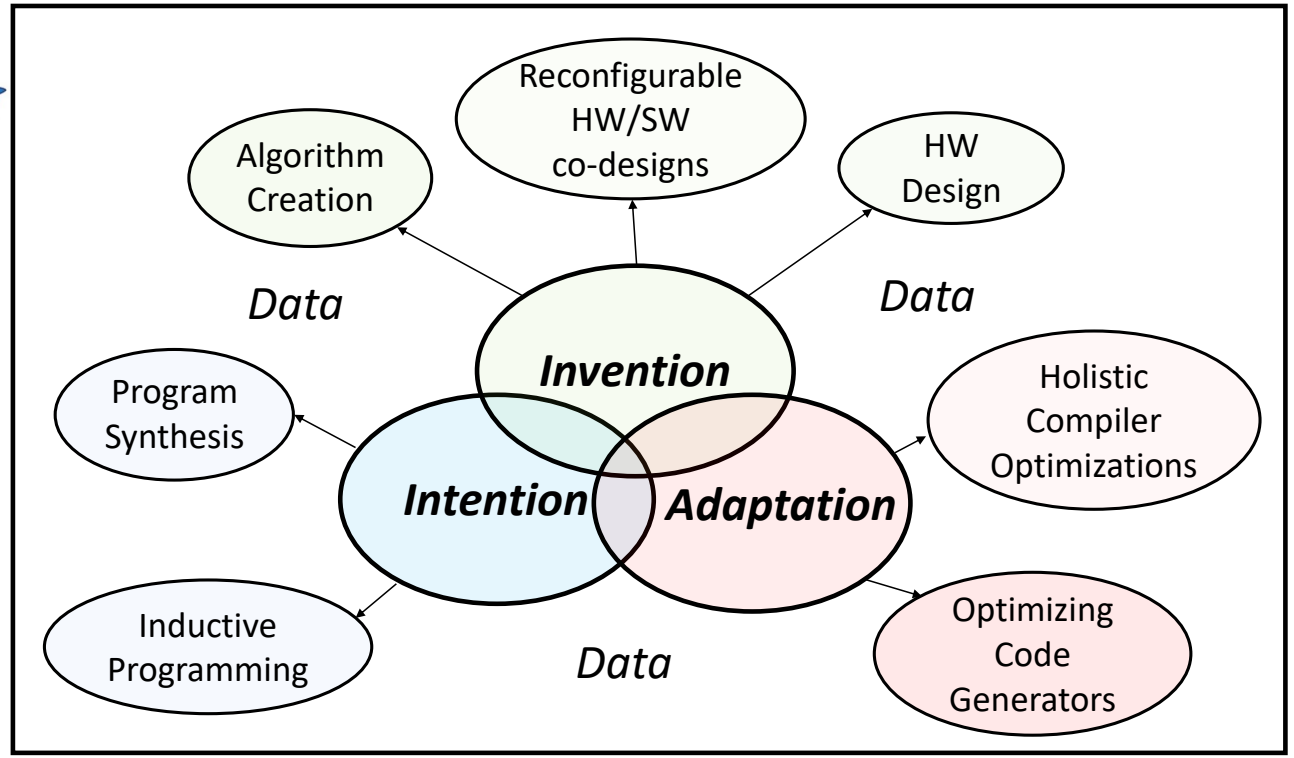


Programmers should just worry about expressing their intent. We will automate the Invention and Adaptation work

The Three Pillars of Machine Programming

merly.ai

Justin Gottschlich, ~~Intel~~
Armando Solar-Lezama, MIT
Nesime Tatbul, Intel
Michael Carbin, MIT
Martin, Rinard, MIT
Regina Barzilay, MIT
Saman Amarasinghe, MIT
Joshua B Tenebaum, MIT
Tim Mattson, Intel



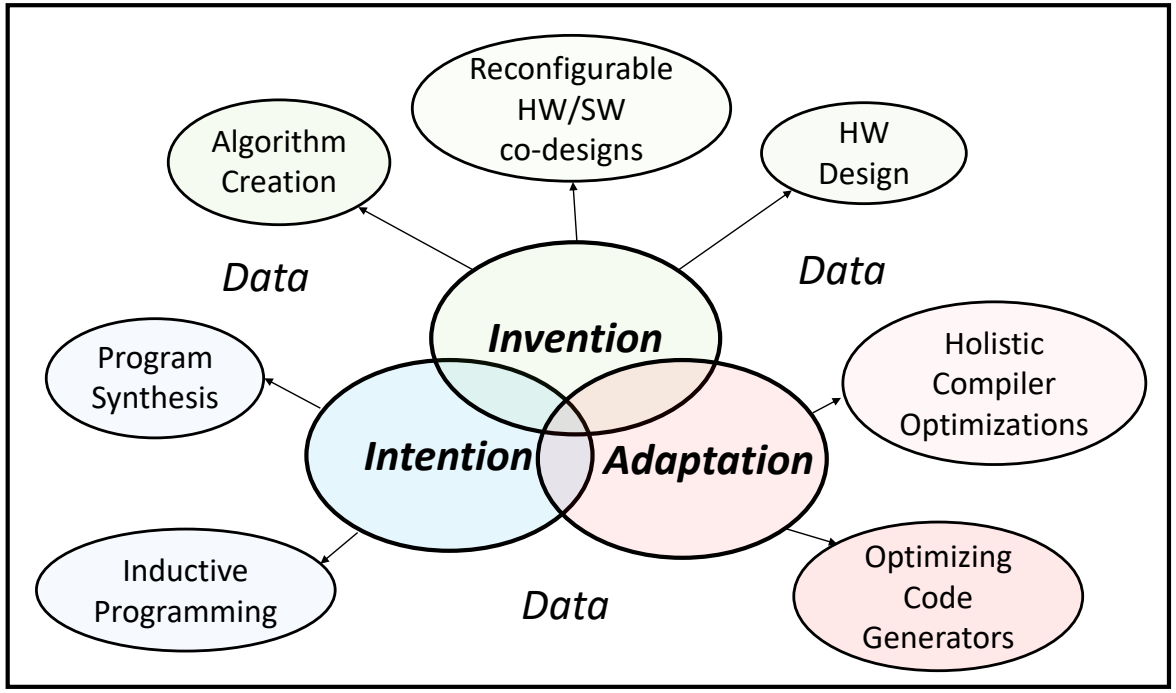
A position paper laying out our vision for how to solve the machine programming problem. The three Pillars:

- **Intention**: Discover the intent of a programmer
- **Invention**: Create new algorithms and data structures
- **Adaption**: Evolve in a changing hardware/software world

Three Pillar Examples*

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

- **Intention**
  - ***"Verified Lifting of Stencil Computations",*** S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)
- **Invention**
  - ***"Speedy Point: Optimized Cpu code from Cuda",*** Adam Herr and the Intel Inteon group
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

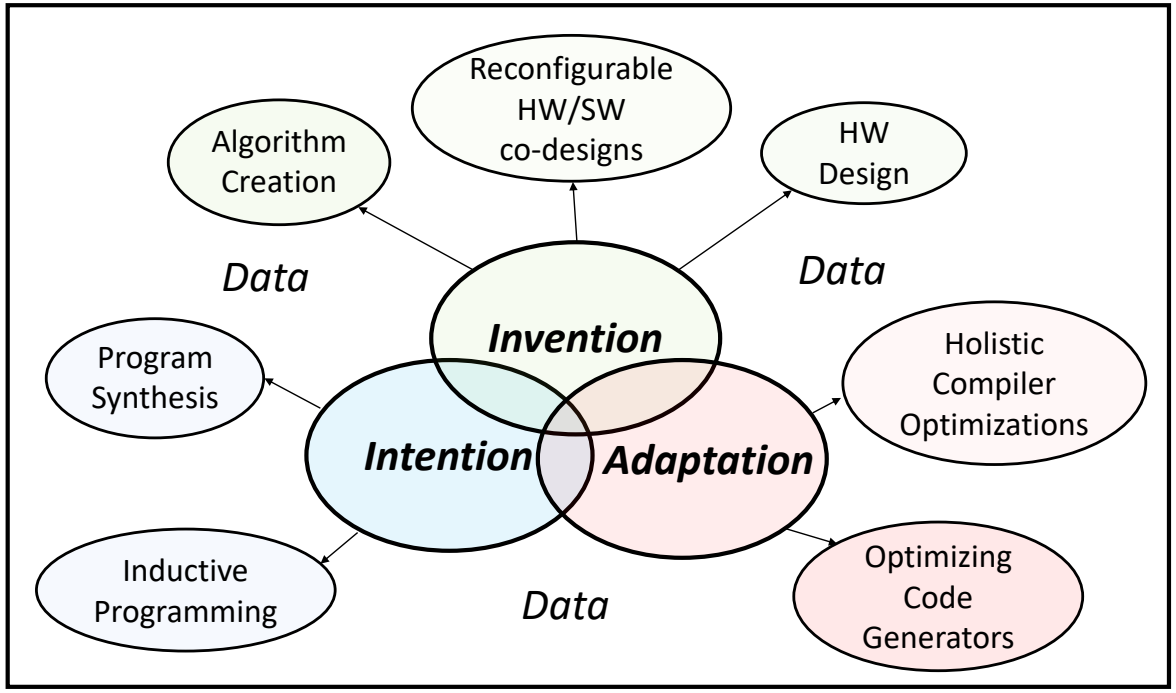


- **Put all three together … and something awesome happens**
  - ***"ScaMP: Scalable Machine Programming"*** a research program at MIT funded by Intel and MIT … launched fall'22

Three Pillar Examples*

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

- **Intention**
  - ***"Verified Lifting of Stencil Computations",*** S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)
- **Invention**
  - ***"Speedy Point: Optimized Cpu code from Cuda",*** Adam Herr and the Intel Inteon group
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



- **Put all three together … and something awesome happens**
  - ***"ScaMP: Scalable Machine Programming"*** a research program at MIT funded by Intel and MIT … launched fall'22

MetaLift and Verified Lifting

("Verified Lifting of Stencil Computations", S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)

Syntax guided generator to transform input code into a high-level DSL for automated analysis.
(STNG … Fortran → halide)

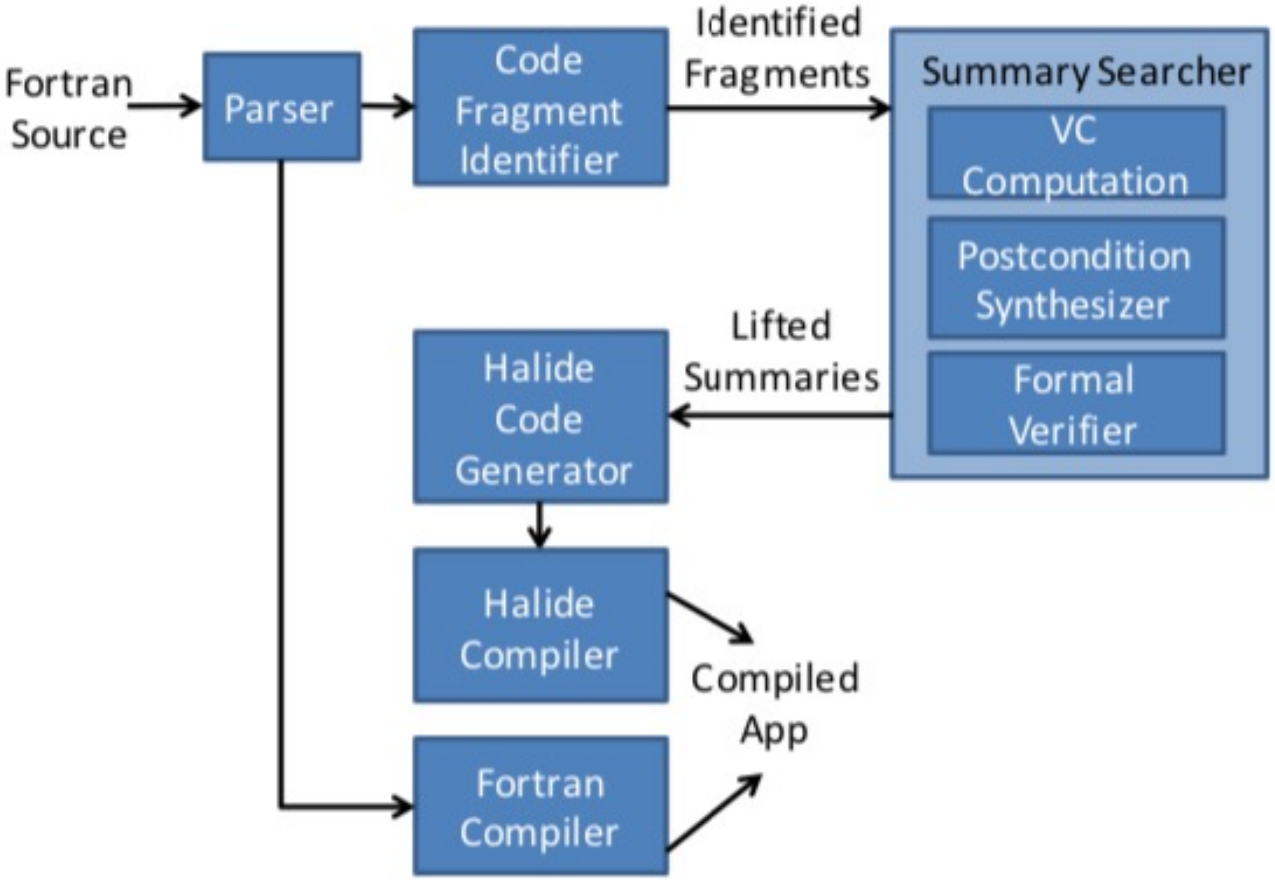


Figure 3. The STNG compiler toolchain

VC Computation: Verified condition generator

http://metalift.uwplse.org

https://homes.cs.washington.edu/~akcheung/papers/pldi16.pdf

MetaLift and Verified Lifting

("Verified Lifting of Stencil Computations", S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)

Syntax guided generator to transform input code into a high-level DSL for automated analysis.
(STNG … Fortran → halide)

- Input: serial Fortran (reference for speedup), a stencil code
- Output: Halide

- Results: NAS Parallel Benchmarks MG15, speedups
  - 17.51 CPU,
  - 4.3 GPU

- CPU: 24-core dual-socket Intel Xeon E5-2695v2 CPUs at 2.4 Ghz, 128 GB mem.
- GPU: NVIDIA K80 … NOTE: their autotuning framework did not optimize the GPU so don't read too much into the GPU numbers

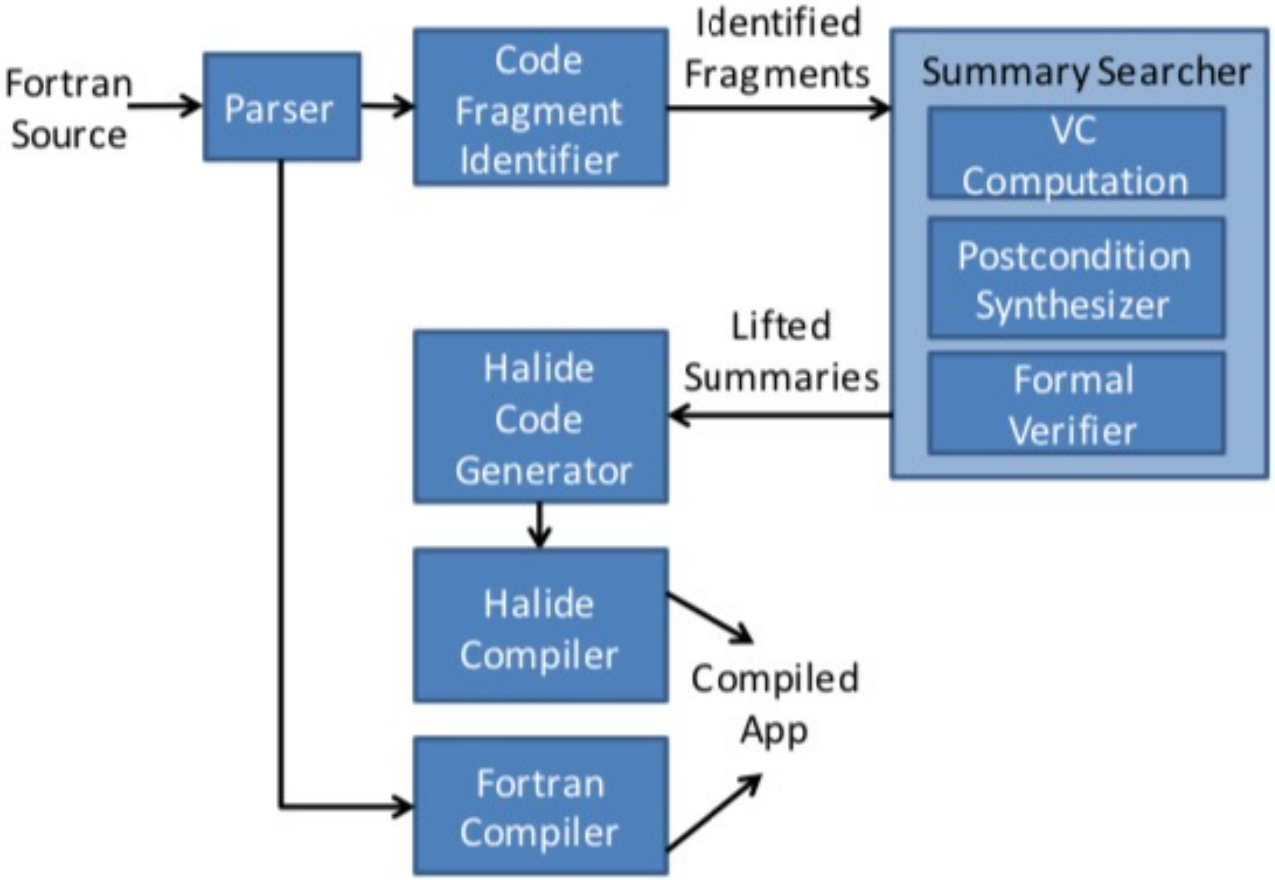


Figure 3. The STNG compiler toolchain

VC Computation: Verified condition generator

3rd party names are the property of their owners

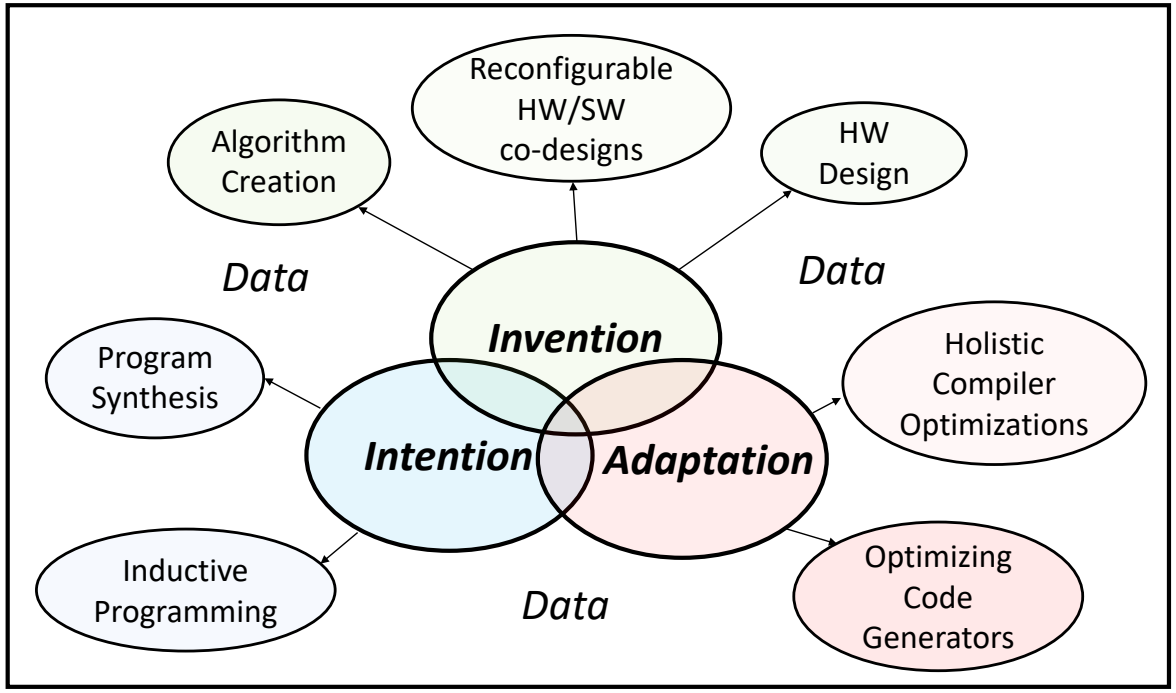
http://metalift.uwplse.org

https://homes.cs.washington.edu/~akcheung/papers/pldi16.pdf

Three Pillar Examples*

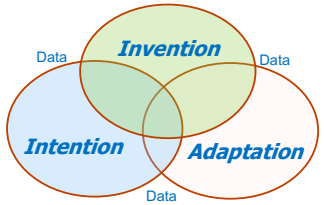
*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

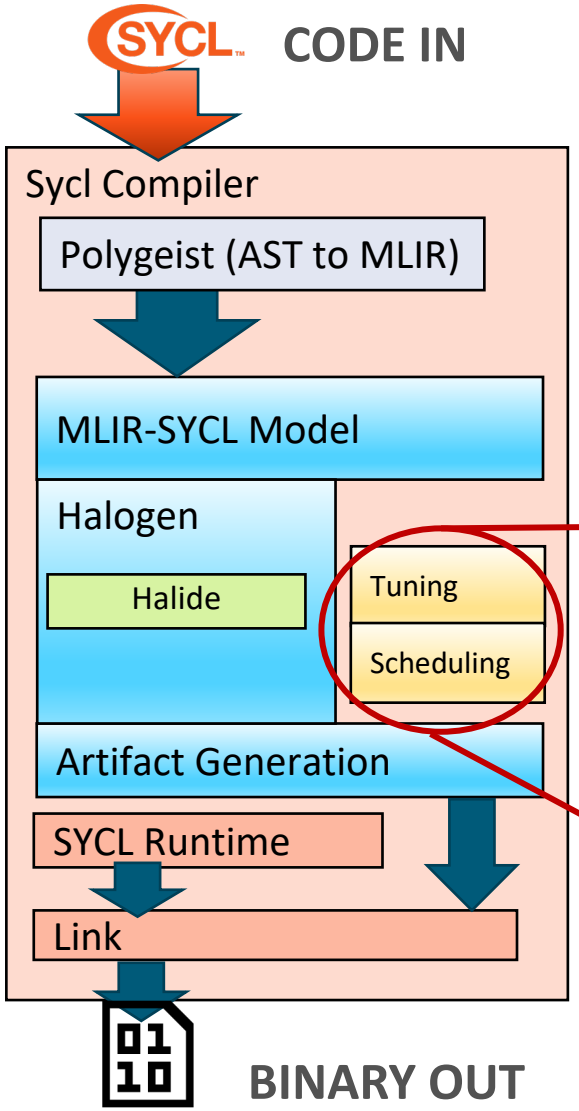
- **Intention**
  - ***"Verified Lifting of Stencil Computations",*** S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)
- **Invention**
  - ***"Speedy Point: Optimized Cpu code from Cuda",*** Adam Herr and the Intel Inteon group
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



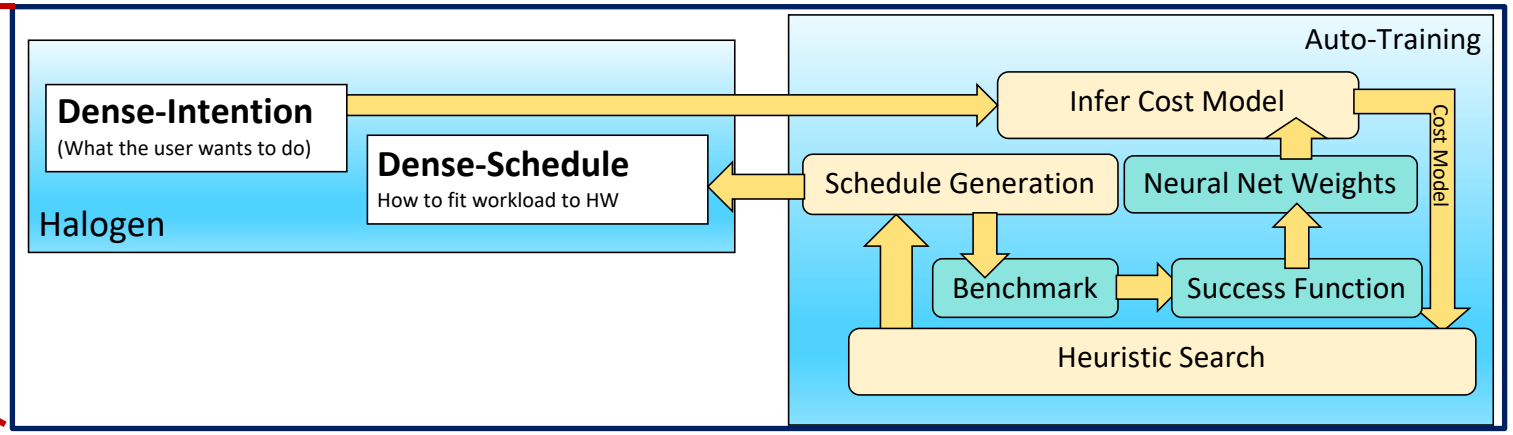
- **Put all three together … and something awesome happens**
  - ***"ScaMP: Scalable Machine Programming"*** a research program at MIT funded by Intel and MIT … launched fall'22

Speedy Point#: Automatically optimizing Sycl code





- Extend Sycl Compiler with Machine Programming (MP) flags
- Use Polygeist* to lower device code to MLIR
- Translate MLIR-SYCL to Halide
- MP Codegen to Halogen
- Link result with SYCL runtime and auto-scheduling
- Produce SYCL complete binary



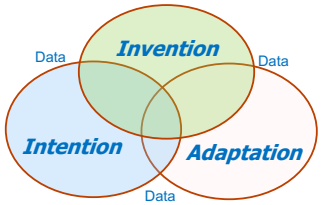
Performance Optimization as a search problem:

- AI-based cost model to guide heuristic search (alpha-go), trained with millions of wall-time measurements of real workloads

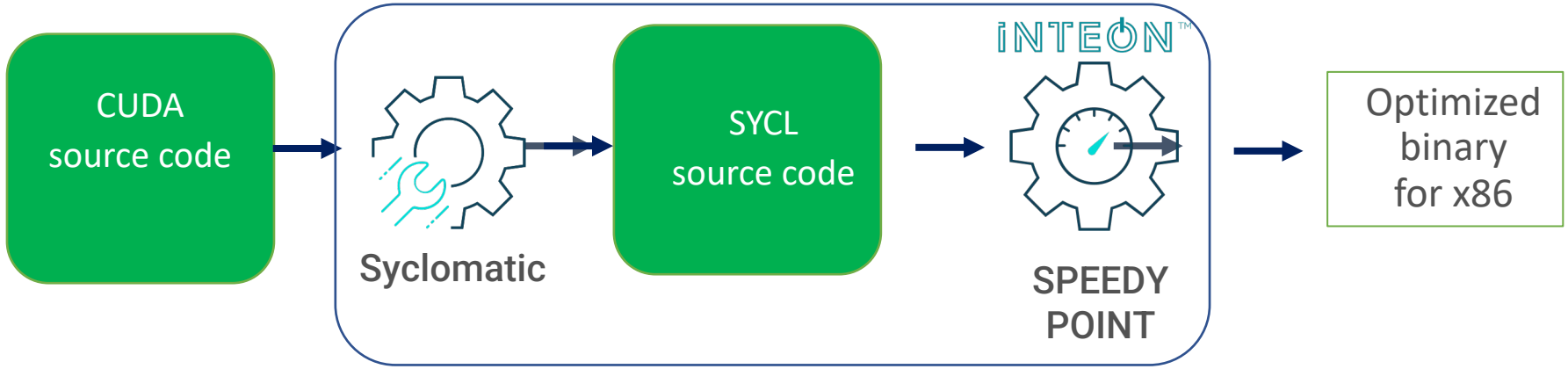
#From Intel's Inteon group.

*https://polygeist.mit.edu/

From CUDA to optimized CPU code



- Syclomatic: A oneAPI tool from Intel to convert CUDA into Sycl
- SPEEDY POINT: Convert code to Halide and then auto-schedule to optimize performance

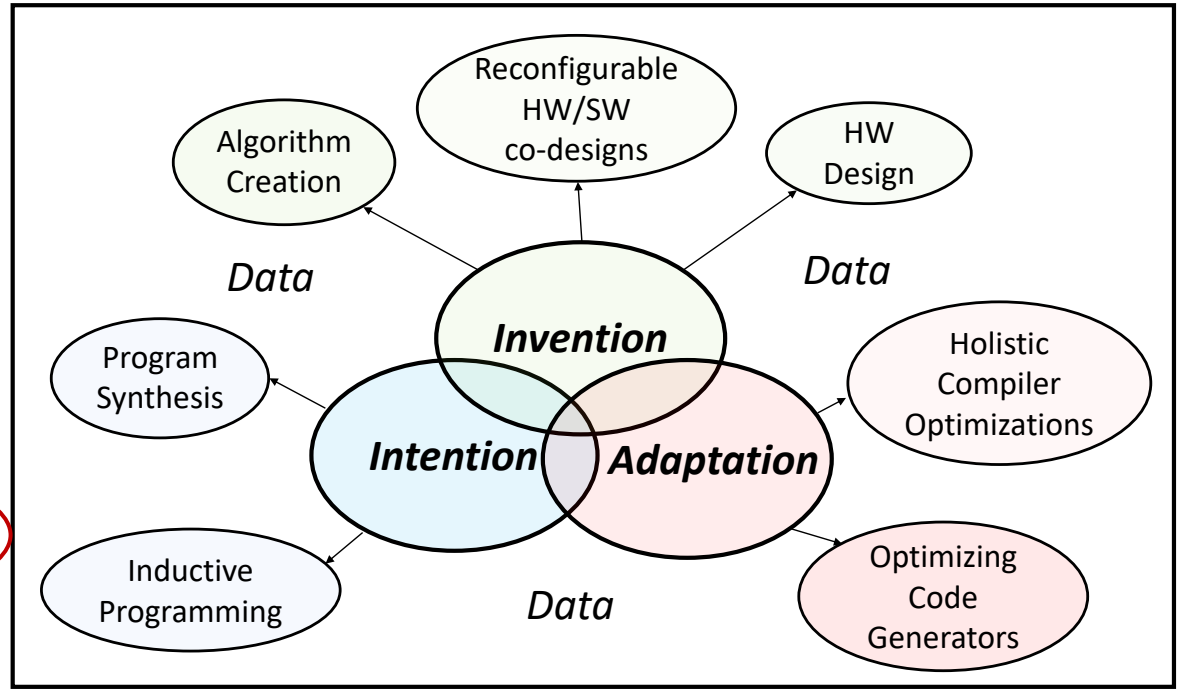


Preliminary results show that **gains of 40x or more** are possible vs. the translated but unoptimized SYCL code.

Three Pillar Examples*

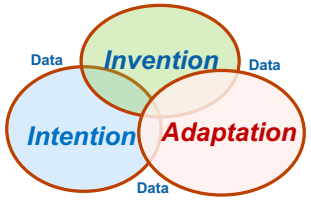
*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

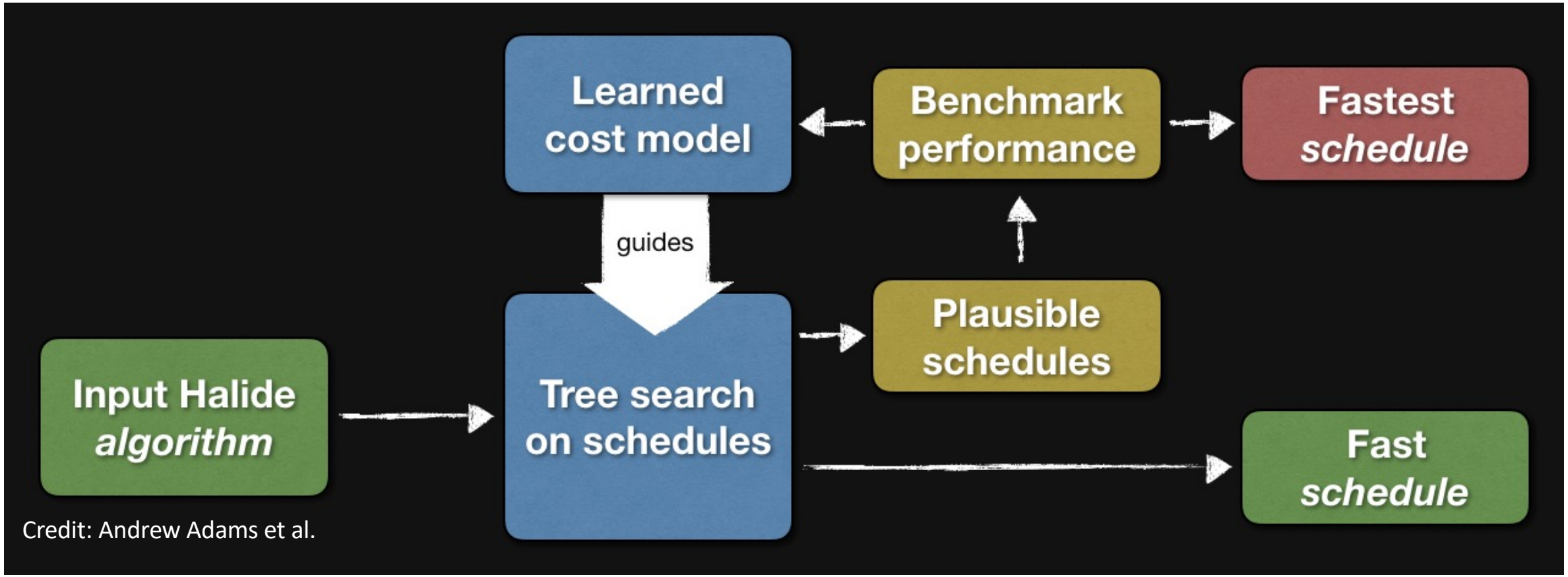
- **Intention**
  - ***"Verified Lifting of Stencil Computations",*** S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)
- **Invention**
  - ***"Speedy Point: Optimized Cpu code from Cuda",*** Adam Herr and the Intel Inteon group
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



- **Put all three together … and something awesome happens**
  - ***"ScaMP: Scalable Machine Programming"*** a research program at MIT funded by Intel and MIT … launched fall'22

Halide Learned Schedules

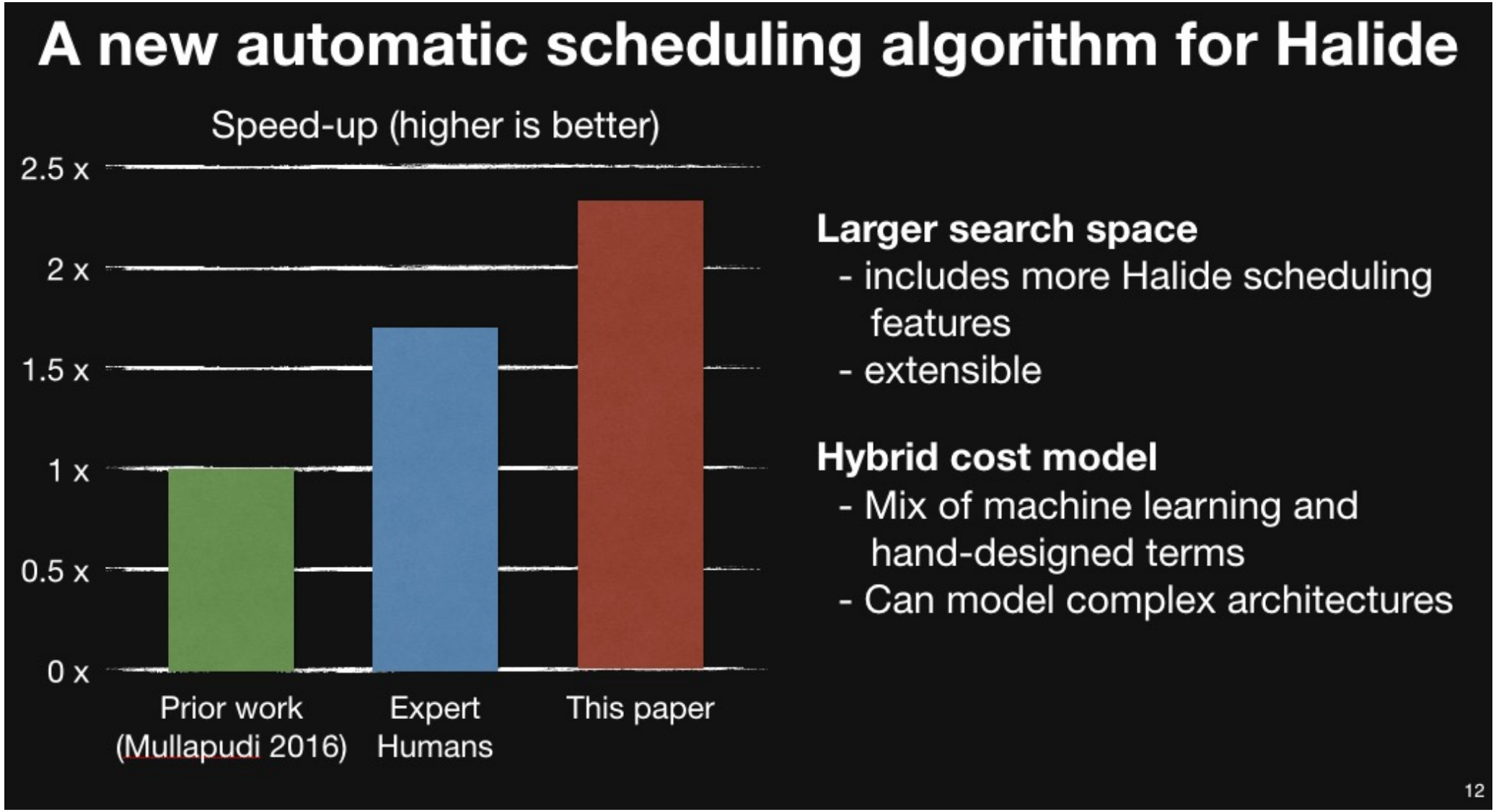




Credit: Andrew Adams et al.

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, Jonathan Ragan-Kelley. Learning to Optimize Halide with Tree Search and Random Programs ACM Transactions on Graphics 38(4) (Proceedings of ACM SIGGRAPH 2019)

Superhuman Performance

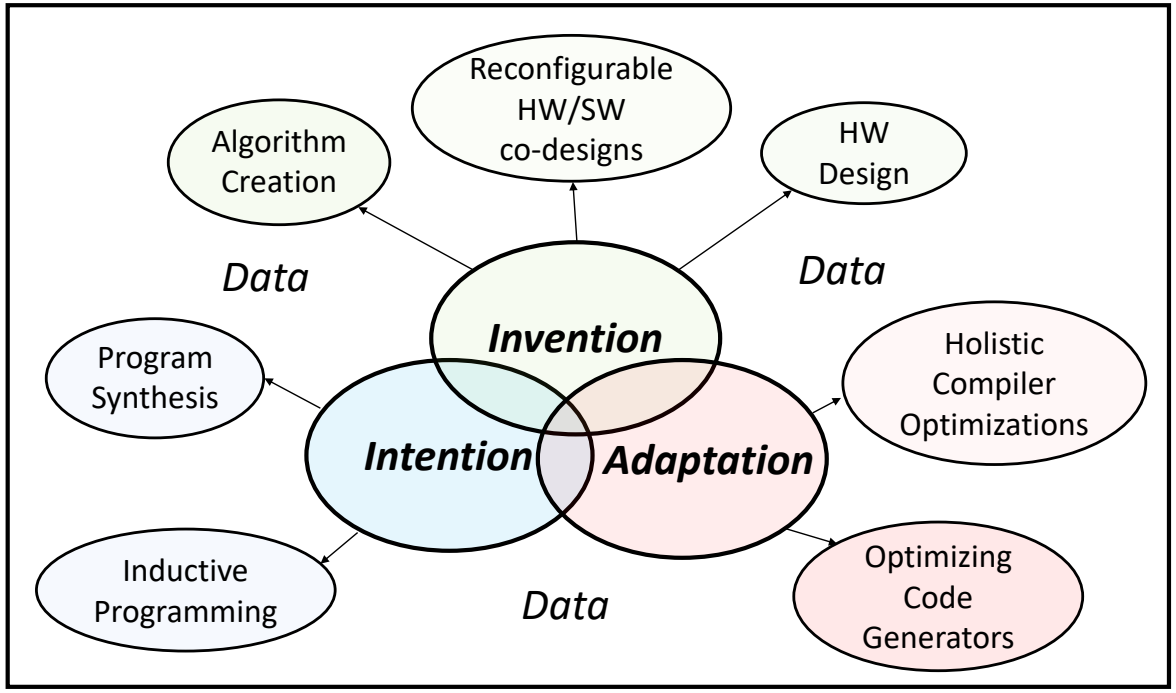


Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, Jonathan Ragan-Kelley. Learning to Optimize Halide with Tree Search and Random Programs ACM Transactions on Graphics 38(4) (Proceedings of ACM SIGGRAPH 2019)

Three Pillar Examples*

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

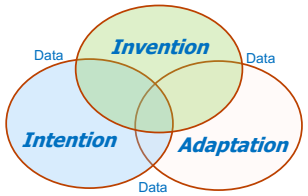
- **Intention**
  - ***"Verified Lifting of Stencil Computations",*** S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI'16)
- **Invention**
  - ***"Speedy Point: Optimized Cpu code from Cuda",*** Adam Herr and the Intel Inteon group
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



- **Put all three together … and something awesome happens**
  - ***"ScaMP: Scalable Machine Programming"*** a research program at MIT funded by Intel and MIT … launched fall'22

ScaMP: Scalable Machine Programming

A five-year research program at MIT funded by Intel and NSF (Launched Oct 2022)



Initial "spec"

I want to build a web based workflow management system for fluid dynamics simulations

Conversation (iterative refinement)

Incremental Multimodal Specification (IMS)

- Progressive Reification — MP, SP
- DSL Synthesis — MP, PL
- Coordination Synthesis — MP, PL, HPC

Source code expressed in terms of composable DSLs

Learned autoschedulers

Verified DSL compilers

Correct by construction code generators

executable

Clusters of heterogeneous nodes

Performance History database

Learned cost models

MIT Pis

- Saman Amarasinghe
- Michael Carbin
- Adam Chlipala
- Jonathan Ragan-Kelley
- Armondo Solar-Lezama

Source: https://www.intel.com/content/www/us/en/research/news/new-machine-programming-research-at-mit.html?cid=iosm&source=linkedin&campid=intel_ai_-_@intelai_social_media_content_calendar&content=100003476010351&icid=always-on&linkId=100000158650577

Programming in the 23rd Century: Conversational Computing

- With the three pillars we are on our way to 23rd century programming
  - **Intention**: Natural language processing plus visual information
  - **Invention**: Lifting into a DSL, ML to invent algorithms, Theorem prover to verify.
  - **Automation**: Autotuning + ML to optimize for “any” HW

source: Star Trek IV: the journey home.

But there is much more we need to do on each of the pillars (especially the intention pillar)

… and we have partnered with NSF and MIT to make it happen!

Conclusion

- Hardware is heterogeneous … we need to support heterogeneous programming … i.e. a single programs that runs on all processors (GPUs, CPUs, and more)
- We need Productivity, portability and Performance (the 3 Ps).
- The Database community pointed the way back in the 1970s … a declarative programming interface build on a formal algebra
- Our research … Machine Programming to map intention onto HPC hardware

My Greenland, skin-on-frame kayak in the middle of Budd Inlet WA during a negative tide