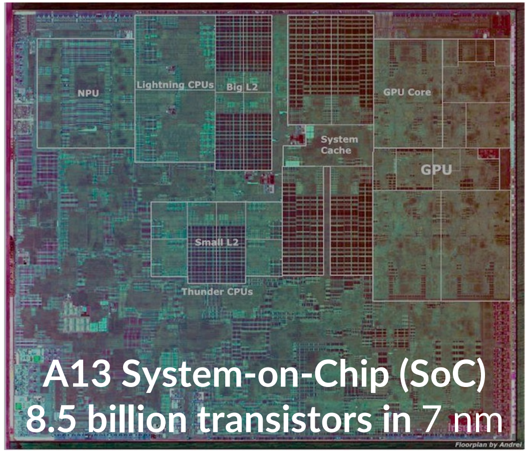
AHA: An Open-Source Framework for Co-design of Programmable Accelerators and Compilers

Kalhan Koul*[1], Jackson Melchert*[1], Keyi Zhang*[1], Taeyoung Kong*[1], Maxwell Strange*[1], Olivia Hsu*[1], Qiaoyi Liu*[1], Jeff Setter*[1], Ross Daly*[1], Caleb Donovick*[1], Alex Carsello*[1], Leonard Truong[1], Po-Han Chen[1], Yuchen Mei[1], Zhouhua Xie[1], Kathleen Feng[1], Gedeon Nyengele[1], Dillon Huff[1], Kavya Sreedhar[1], Huifeng Ke[1], Ankita Nayak[1], Rajsekhar Setaluri[1], Stephen Richardson[1], Christopher Torng[2], Pat Hanrahan[1], Clark Barrett[1], Mark Horowitz[1], Fredrik Kjolstad[1], Priyanka Raina[1]

*Equal Contribution; [1]Stanford University, CA, USA; [2]University of Southern California, CA, USA

Domain-Specific Accelerators

With the slowdown of Moore’s law and end of Dennard scaling, **hardware specialization** is necessary to improve performance and energy efficiency of computing systems



A13 System-on-Chip (SoC)
8.5 billion transistors in 7 nm

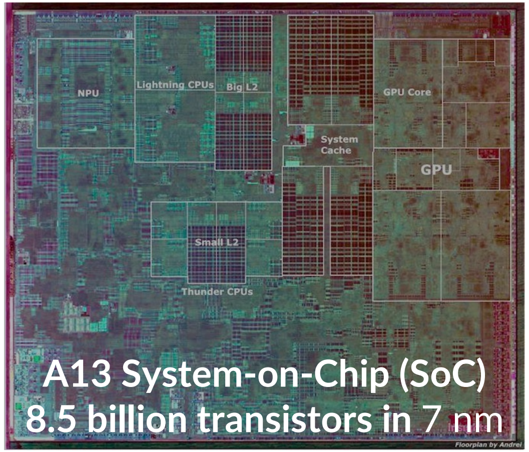
Modern SoCs have dozens of domain-specific accelerators

Graphics
Machine Learning
Image Processing
Video Coding
Cryptography
Wireless …

Image source: https://www.anandtech.com/show/14892/the-apple-iphone-11-pro-and-max-review/2

Domain-Specific Accelerators

With the slowdown of Moore's law and end of Dennard scaling, **hardware specialization** is necessary to improve performance and energy efficiency of computing systems



A13 System-on-Chip (SoC)
8.5 billion transistors in 7 nm

Modern SoCs have dozens of domain-specific accelerators

However, **designing**, **verifying** and **deploying** such systems with accelerators has a **large engineering cost**

Image source: https://www.anandtech.com/show/14892/the-apple-iphone-11-pro-and-max-review/2

Software Cost > Verification Cost > Design Cost

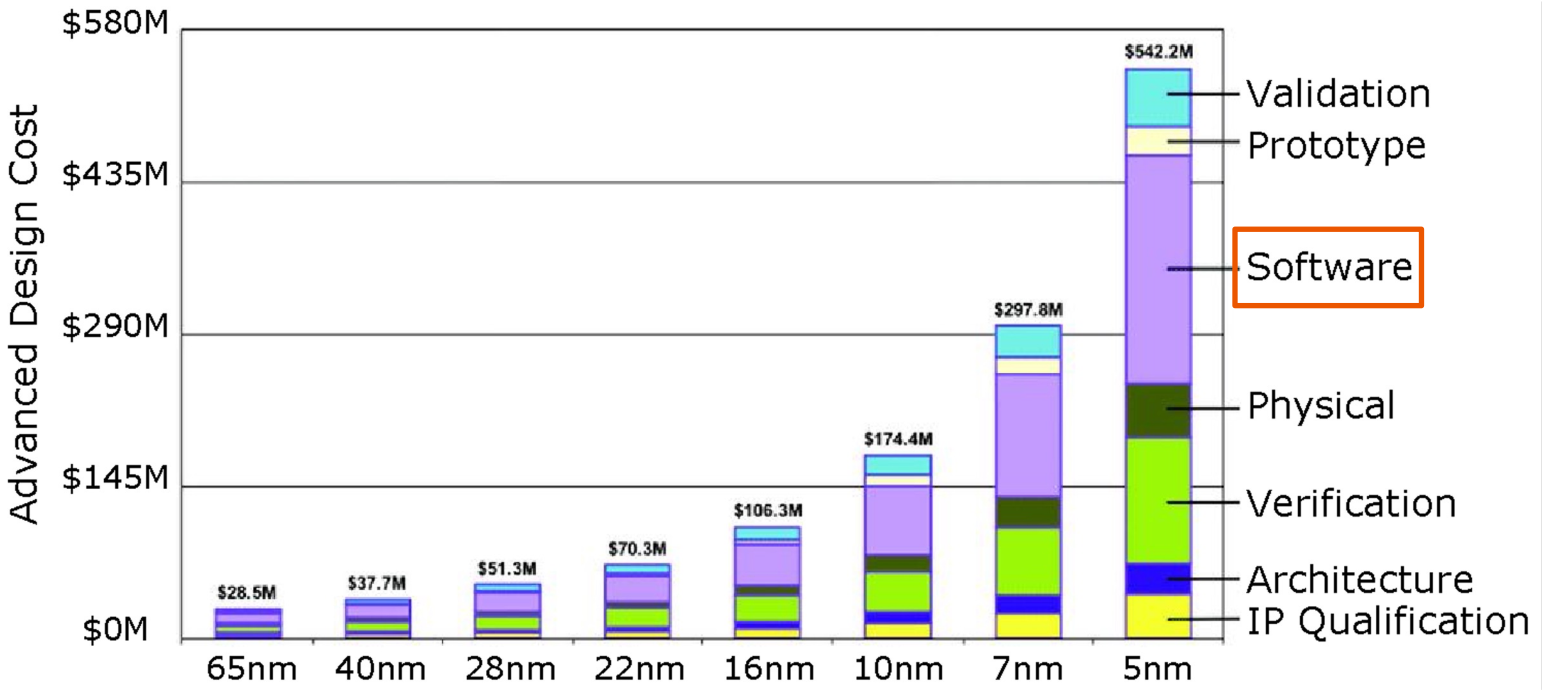
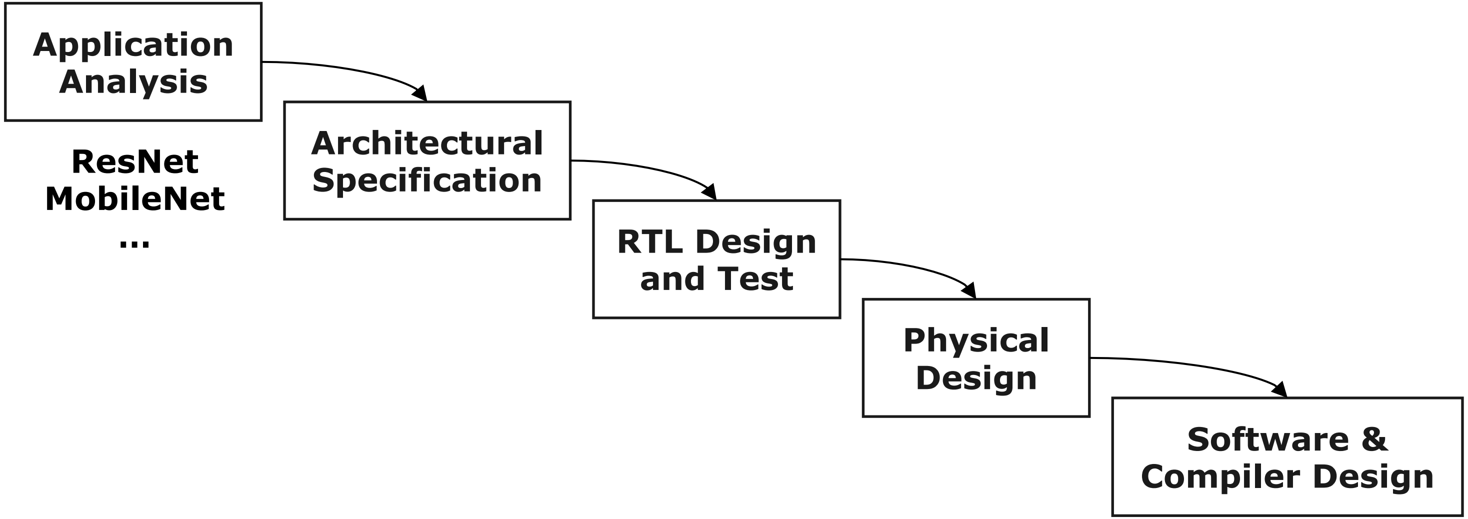


Image source: https://www.researchgate.net/figure/Chip-Design-and-Manufacturing-Cost-under-Different-Process-Nodes-Data-Source-from-IBS_fig1_340843129

Existing Approach to Accelerator Design

- The most common approach to create accelerators is a waterfall approach



Application Analysis

ResNet
MobileNet
...

Architectural Specification

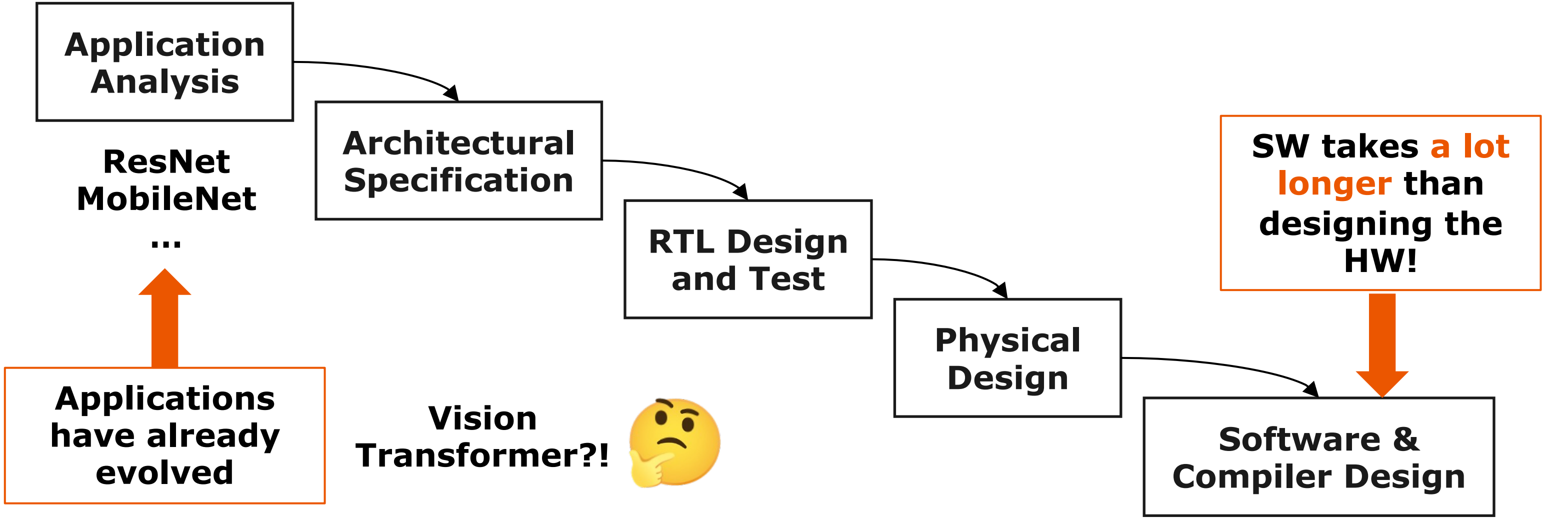
RTL Design and Test

Physical Design

Software & Compiler Design

Existing Approach to Accelerator Design

- The most common approach to create accelerators is a waterfall approach



Application Analysis

Architectural Specification

RTL Design and Test

Physical Design

Software & Compiler Design

ResNet
MobileNet
...

Applications have already evolved

Vision Transformer?! 🤔

SW takes **a lot longer** than designing the HW!

Challenges to Adopting Hardware Specialization at Scale

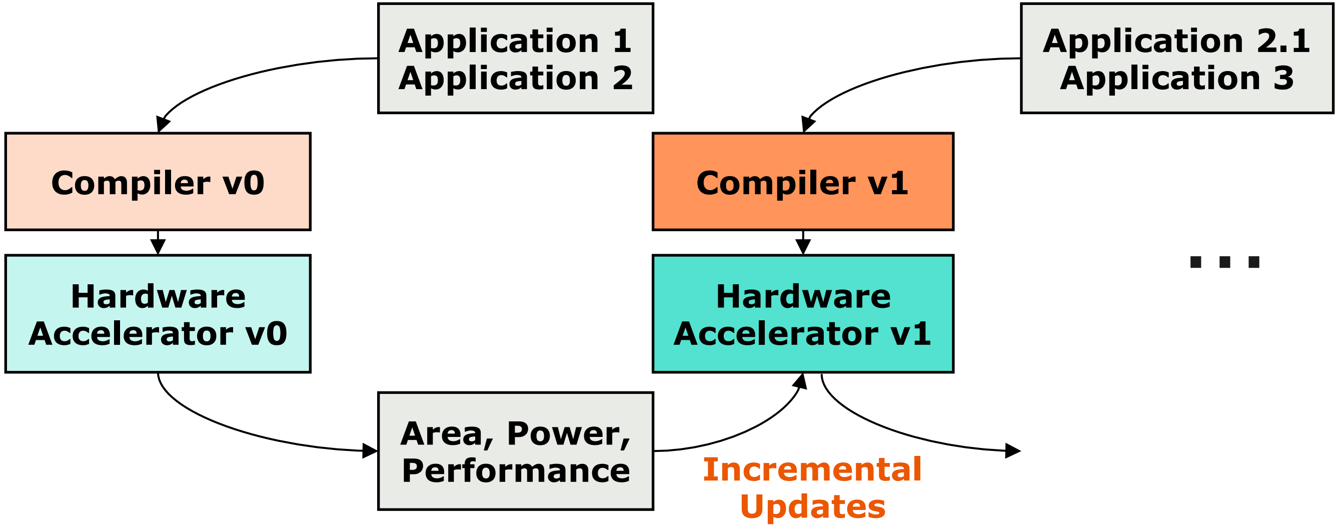
- Prohibitive **cost of design and verification** of accelerator-based systems

- Lack of a **structured approach for evolving the software stack** as the underlying hardware becomes more specialized

- No general **methodology for specializing hardware to *domains***, rather than a few benchmarks

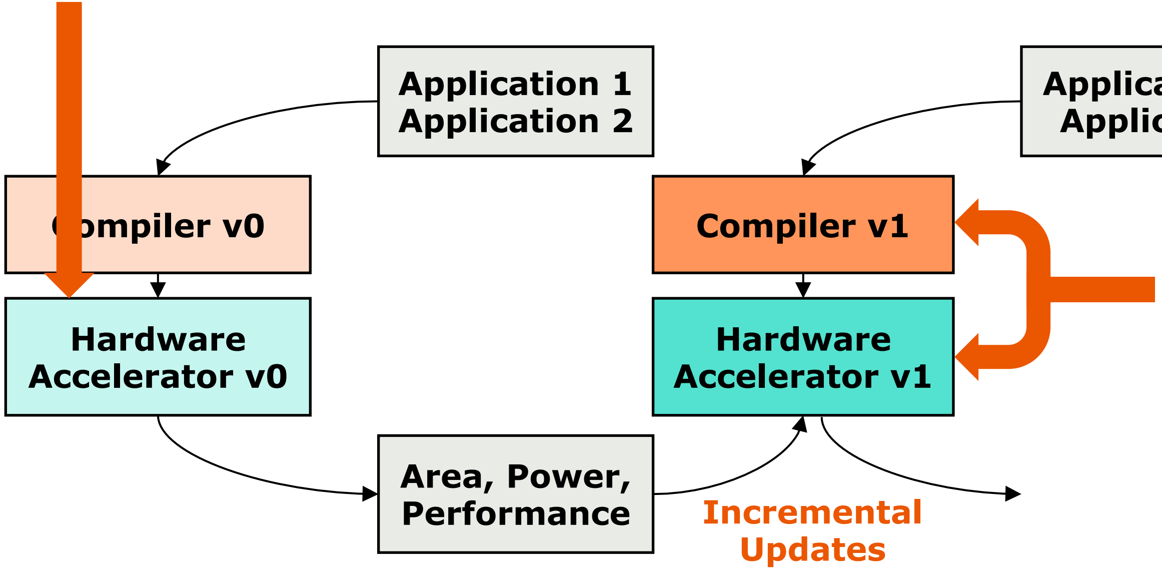
Waterfall Approach -> Agile Approach

- Agile approaches are very common in SW development --- we create tools to adapt them to hardware/compiler co-design
- Incrementally update the hardware accelerator and software to map to it



Requirements for the Agile Approach

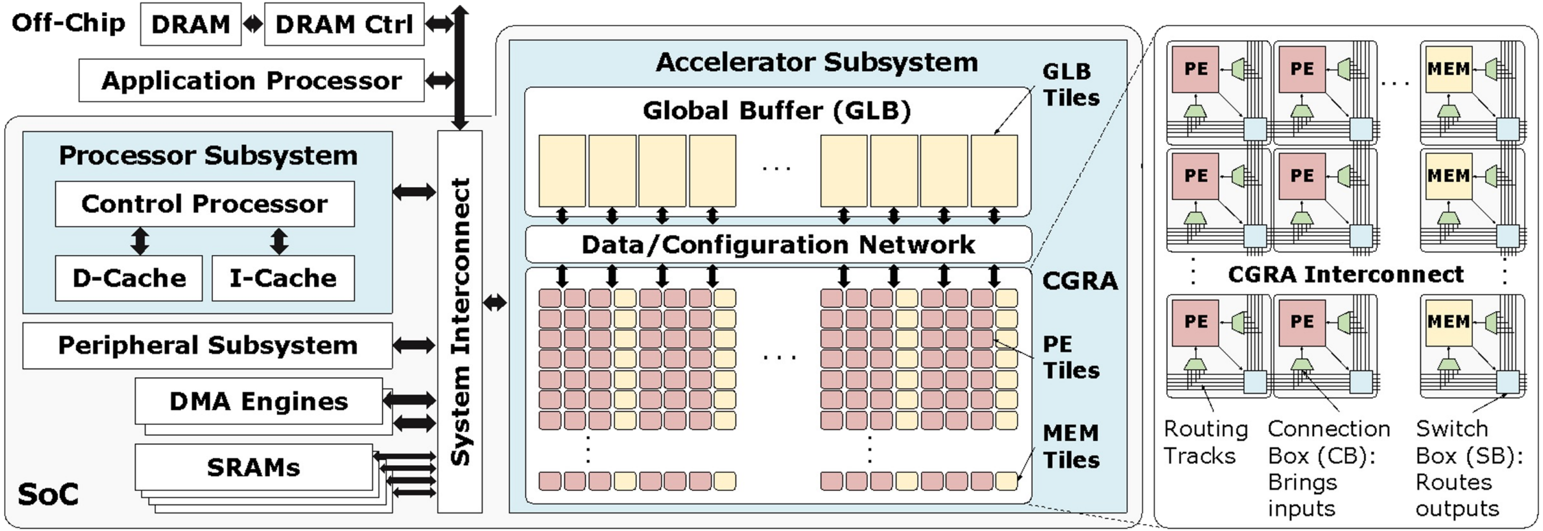
1. Accelerator must be **configurable**
   - So we can map new or modified applications to it (although with lower efficiency)



2. Hardware and compiler must **evolve together**
   - Any change in hardware must propagate to compiler automatically

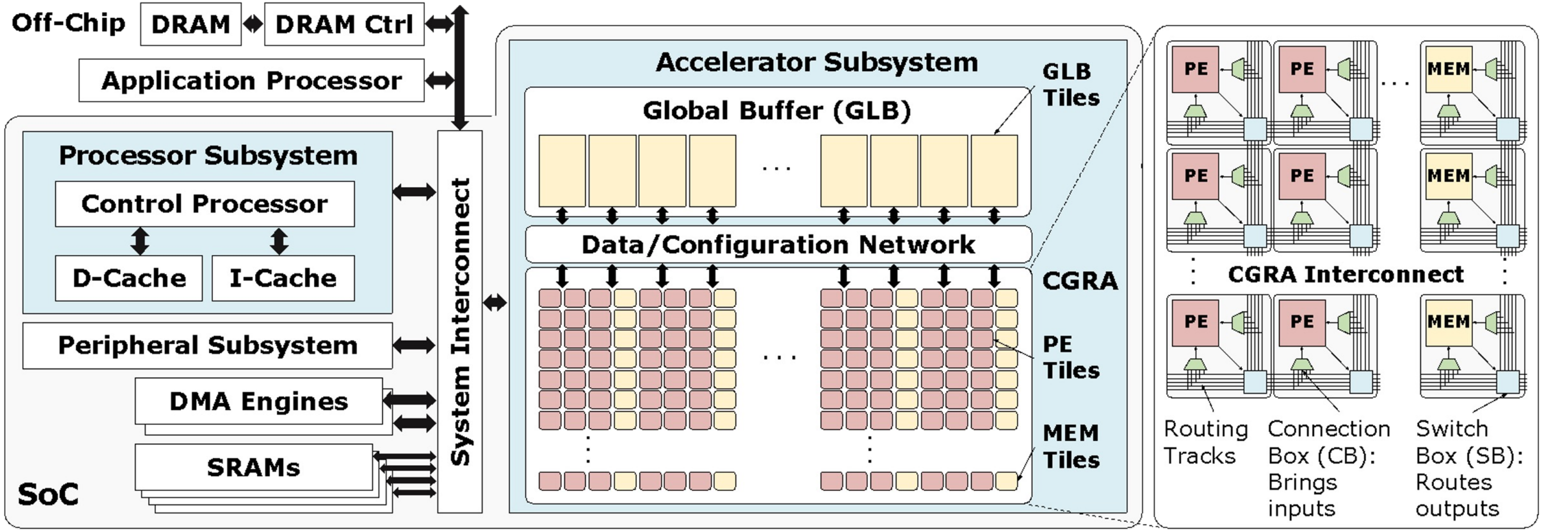
CGRAs as Accelerator Templates

- Think about **accelerators as specialized coarse-grained reconfigurable arrays** (CGRAs) --- similar to an FPGA but with larger compute and memory units, and word-level interconnect



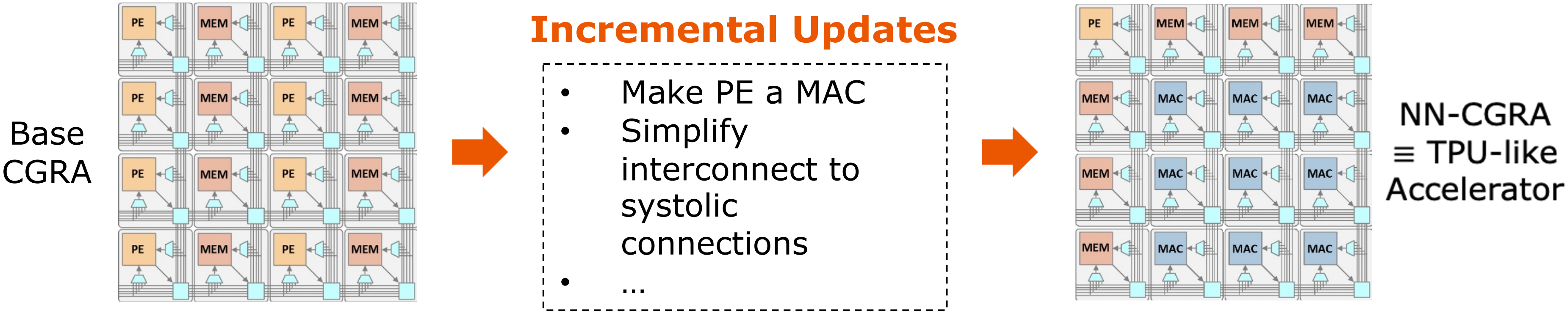
CGRAs as Accelerator Templates

- Is **programmable** enough to accommodate **application evolution**
- Allows **specialization** and exploiting **parallelism and locality** --- characteristics that make an accelerator efficient



CGRAs as Accelerator Templates

- By **tuning the amount of configurability** in CGRA PEs, MEMs and the interconnect, we can create more specialized (closer to ASICs) or more general-purpose accelerators (closer to FPGAs)



Base CGRA

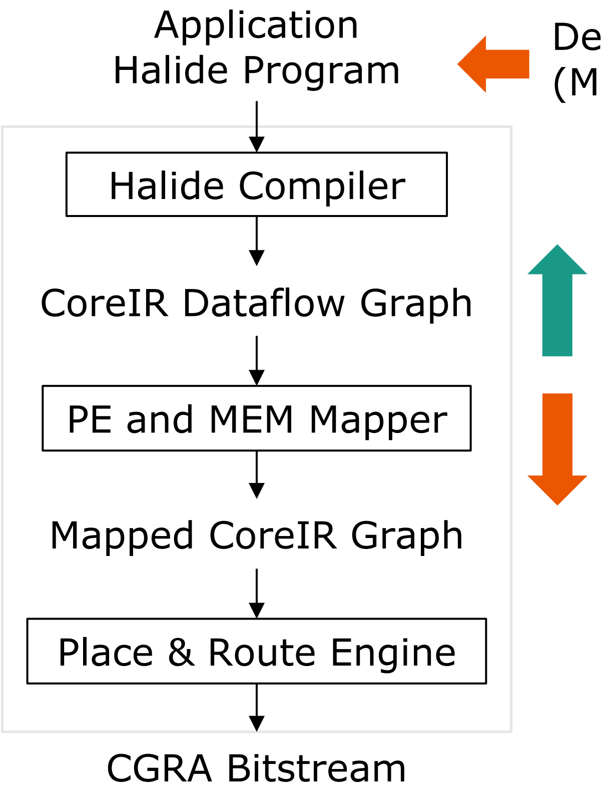
Incremental Updates

- Make PE a MAC
- Simplify interconnect to systolic connections
- …

NN-CGRA ≡ TPU-like Accelerator

- More importantly, thinking of accelerators as specialized CGRAs provides a **standard accelerator template for a compiler to target**

Compiler from Halide Applications to CGRAs



Application
Halide Program

Halide Compiler

CoreIR Dataflow Graph

PE and MEM Mapper

Mapped CoreIR Graph

Place & Route Engine

CGRA Bitstream

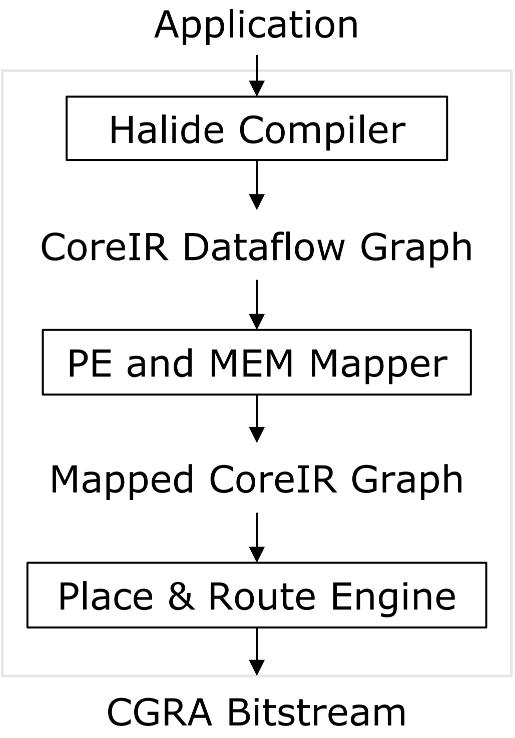
Dense Linear Algebra
(Machine Learning, Image Processing, etc.)

Hardware **independent**

Hardware **dependent**

Q. Liu et al, “Unified Buffer: Compiling image processing and machine learning applications to push-memory accelerators,” TACO 2022

Automatically Generate HW and Compiler Collateral from a Single Source of Truth



Application

Halide Compiler

CoreIR Dataflow Graph

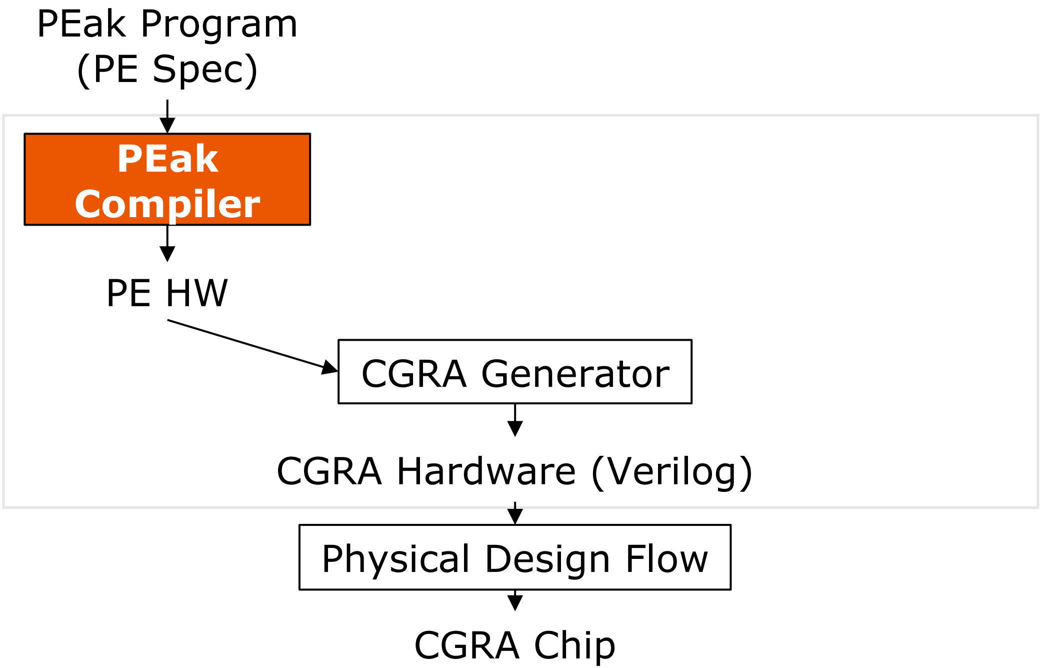
PE and MEM Mapper

Mapped CoreIR Graph

Place & Route Engine

CGRA Bitstream

Formal Specifications for Accelerator Components



PEak Program
(PE Spec)

PEak Compiler

PE HW

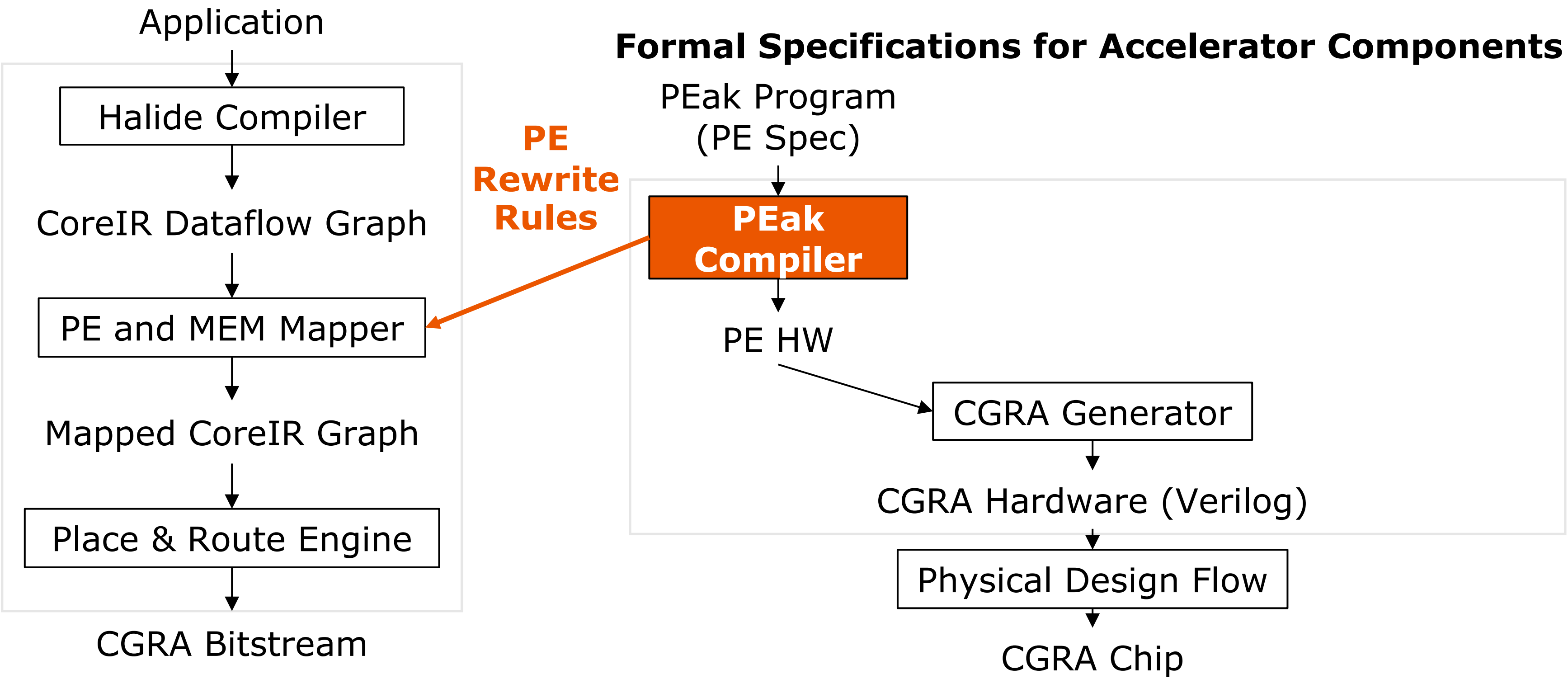
CGRA Generator

CGRA Hardware (Verilog)

Physical Design Flow

CGRA Chip

Automatically Generate HW and Compiler Collateral from a Single Source of Truth



Application

Halide Compiler

CoreIR Dataflow Graph

PE and MEM Mapper

Mapped CoreIR Graph

Place & Route Engine

CGRA Bitstream

PE Rewrite Rules

Formal Specifications for Accelerator Components

PEak Program (PE Spec)

PEak Compiler

PE HW

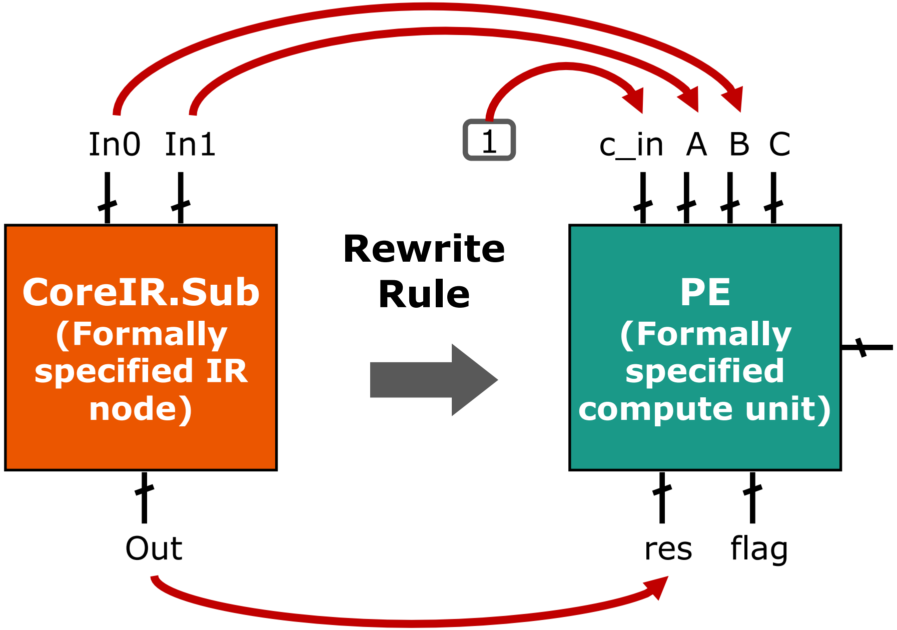
CGRA Generator

CGRA Hardware (Verilog)

Physical Design Flow

CGRA Chip

Automatic Rewrite Rule Generation with SMT


In0
In1
1
c_in
A
B
C
CoreIR.Sub
(Formally specified IR node)
Rewrite Rule
PE
(Formally specified compute unit)
Out
res
flag

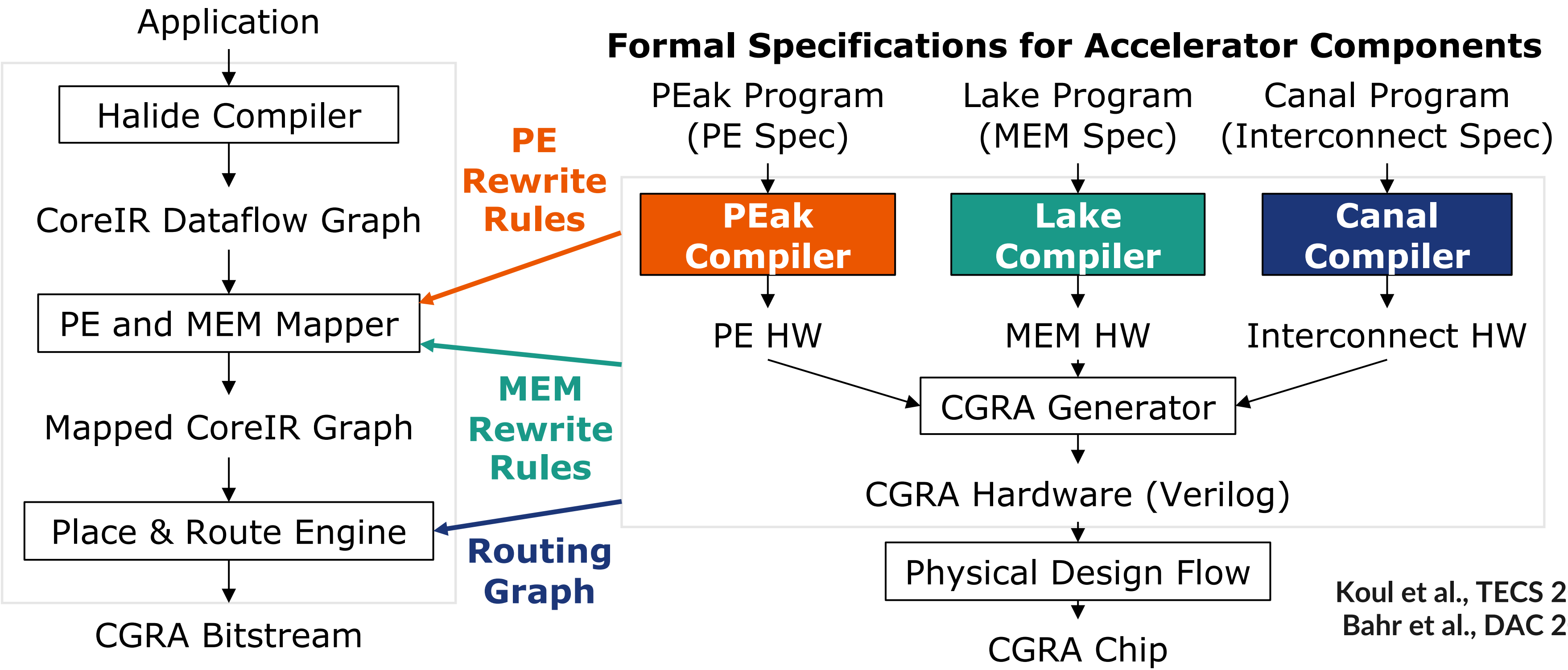
```
Instruction (
  op = Add,
  invert_B = 1,
  …
)
```

∃inst ∀inputs :
CoreIR.Op(inputs)
== PE(inst,inputs)

Solve using an SMT solver

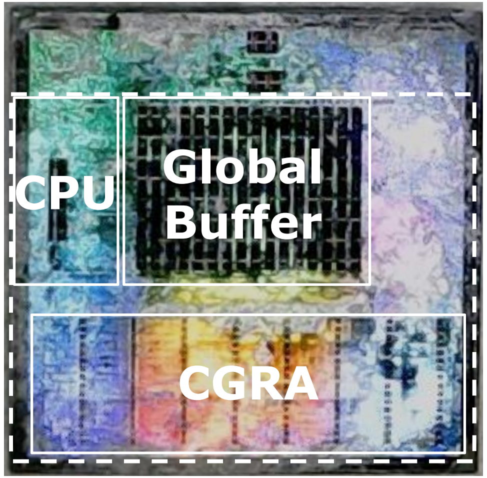
Daly et al., Synthesizing Instruction Selection Rewrite Rules from RTL using SMT, FMCAD 2022

Automatically Generate HW and Compiler Collateral from a Single Source of Truth



Koul et al., TECS 2022
Bahr et al., DAC 2020

Accelerators Designed Using AHA Flow

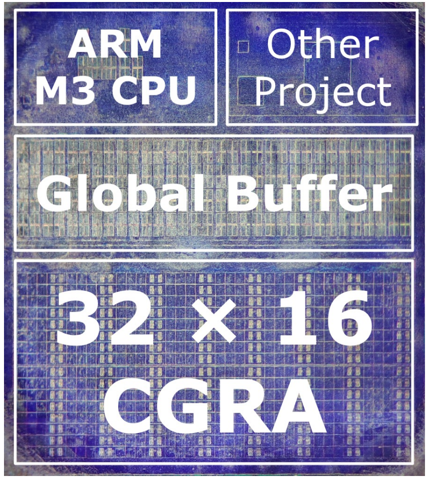


Amber SoC

TSMC 16

Statically scheduled dense data processing e.g. image processing and ML

VLSI 2022, Hot Chips 2022, JSSC 2023

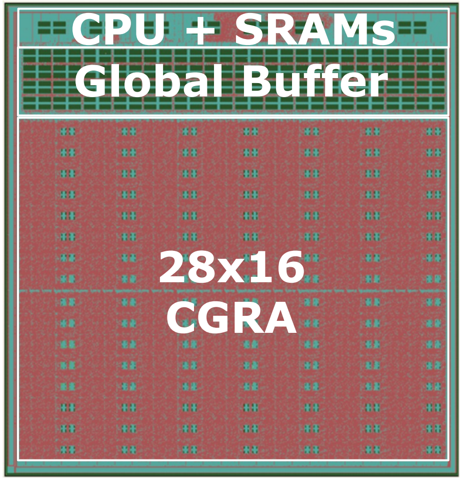


Onyx SoC

GF 12

+ Higher compute density
+ Energy-efficient memories
+ Sparse tensor algebra

VLSI 2024, Hot Chips 2024



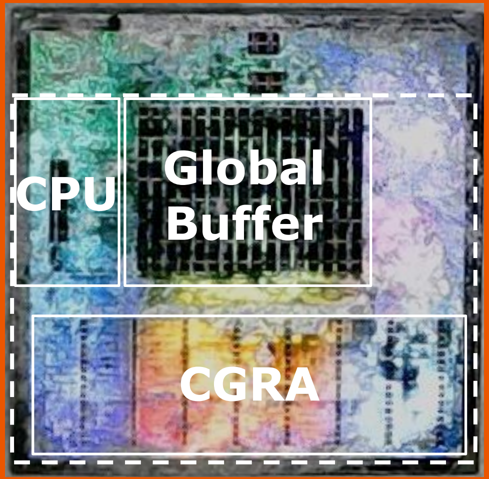
Opal SoC

Intel 16

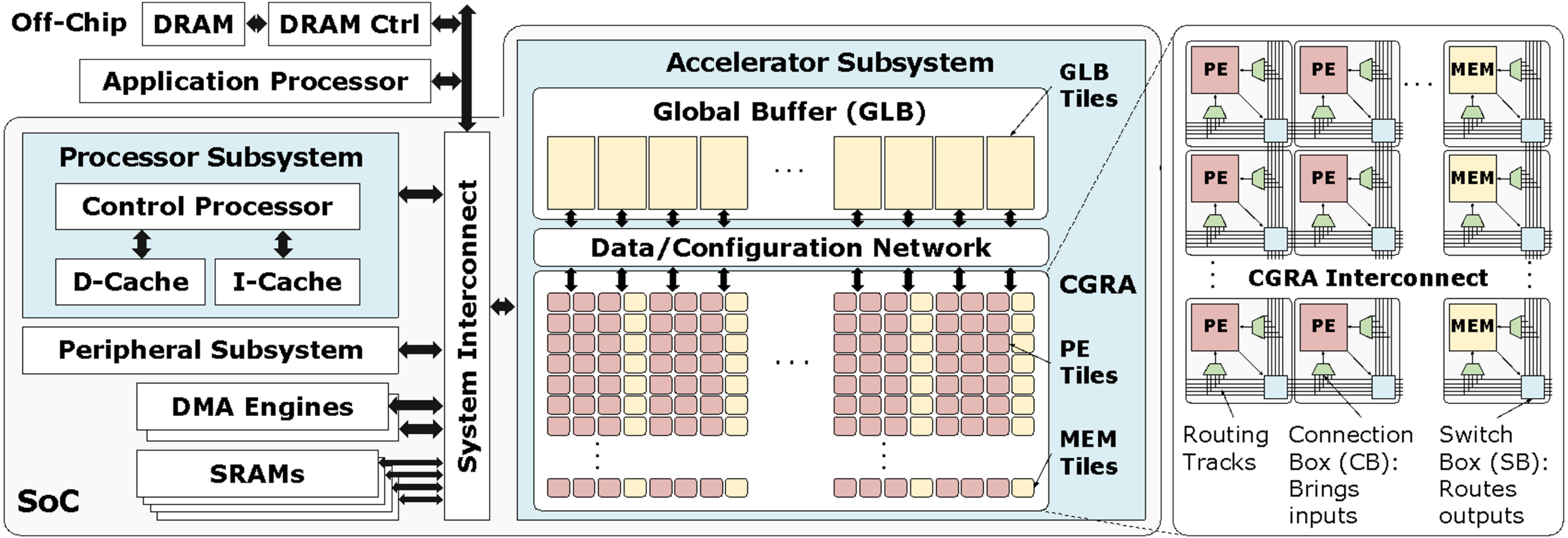
(Taped out: Dec 2023)

+ Higher performance on sparse tensor algebra
+ Sparse machine learning

Amber CGRA – Dense Image Processing and Machine Learning Accelerator



Amber: CGRA SoC Designed with Agile Approach



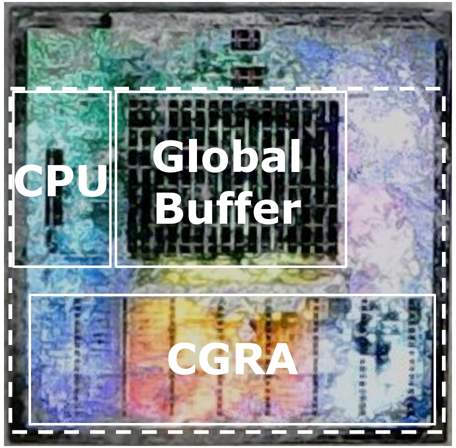
Carsello et al., VLSI 2022, Feng et al., JSSC 2023

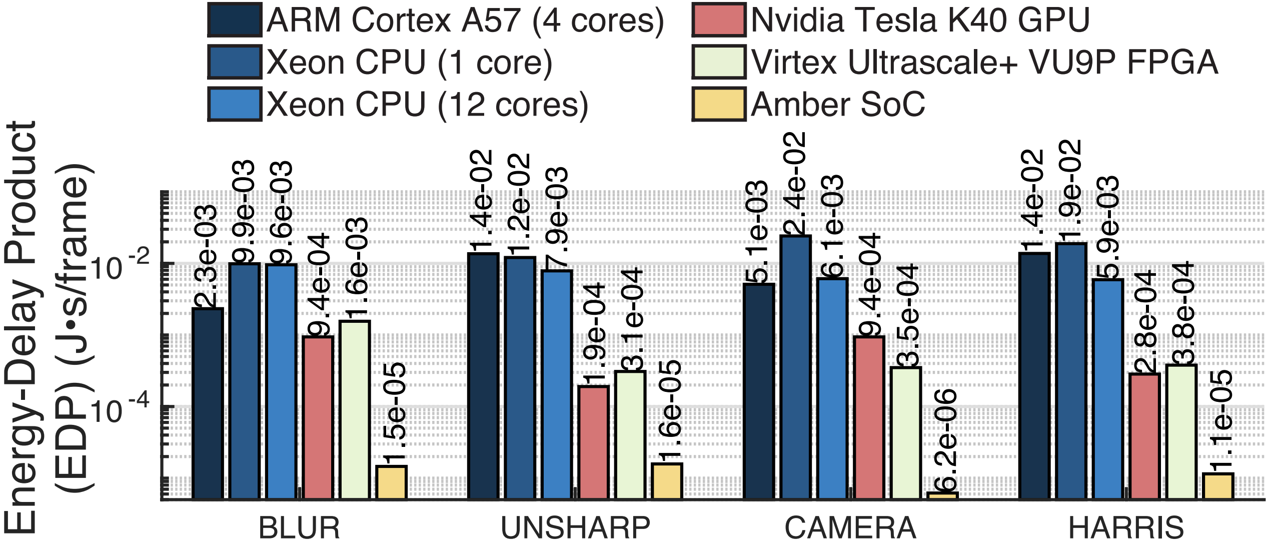
ISSCC 2022 Student Research Preview Poster Award
VLSI 2022 Best Demo Paper Award

Amber: CGRA SoC Designed with Agile Approach

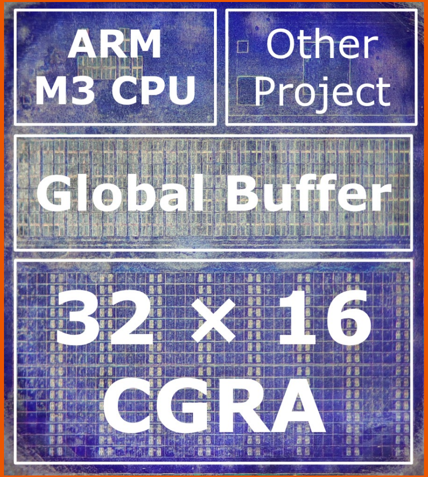
- Amber achieves 160-1200x, 678-3902x, 498-988x, 12-152x, and 20-107x better EDP vs. Cortex A57, 1-core and 12-core Xeon CPUs, GPU, and FPGA respectively

Amber in TSMC 16 nm

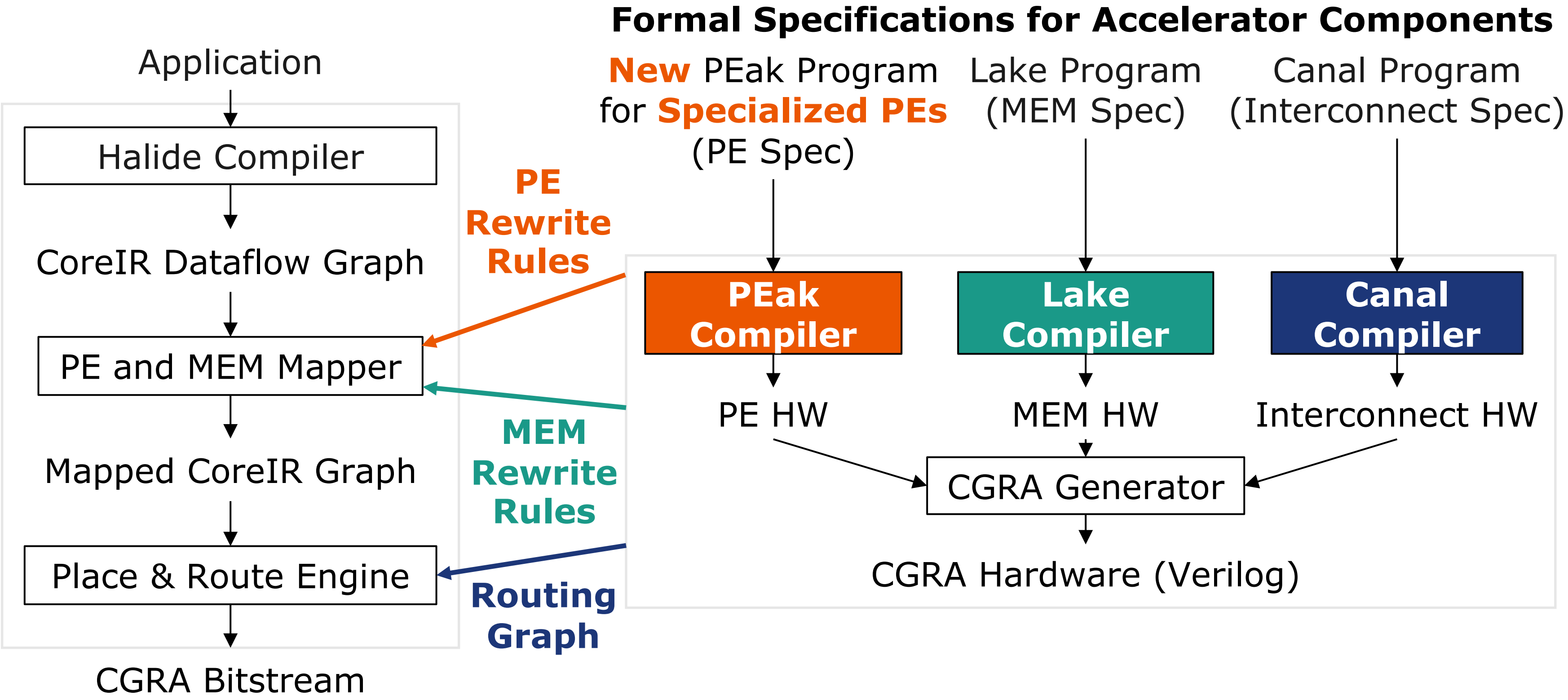




Onyx CGRA – Improving Dense Acceleration and Adding Sparse Tensor Algebra Acceleration

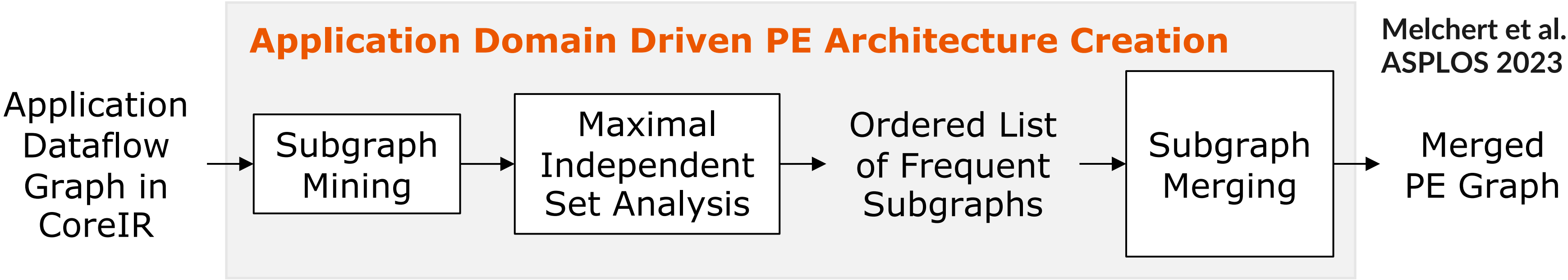


Improving Performance for Dense Applications



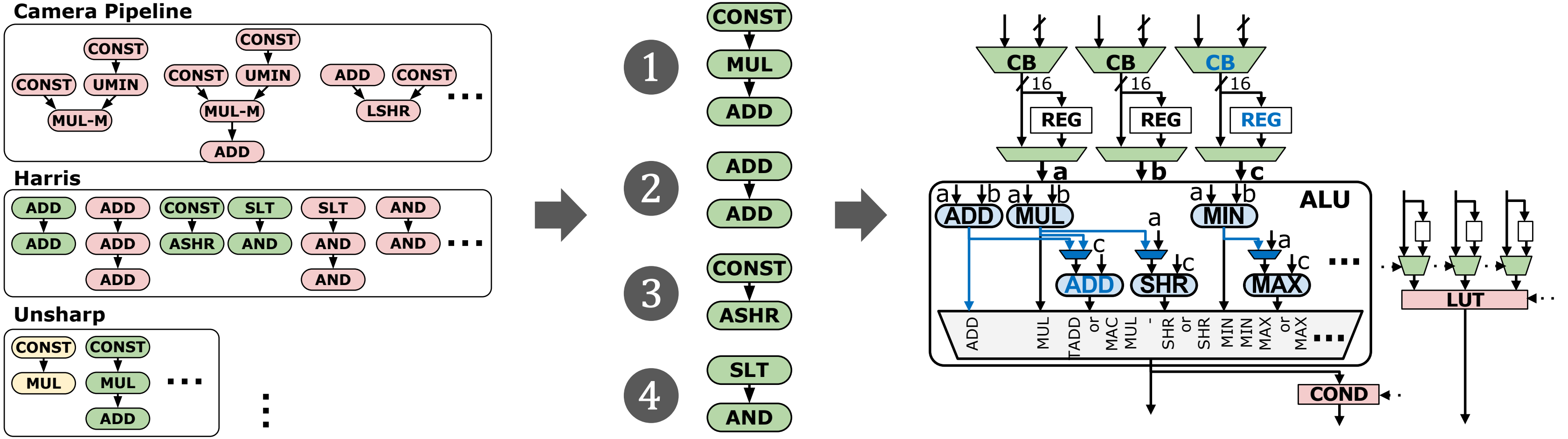
APEX: Application-Driven PE Exploration

Melchert et al., ASPLOS 2023

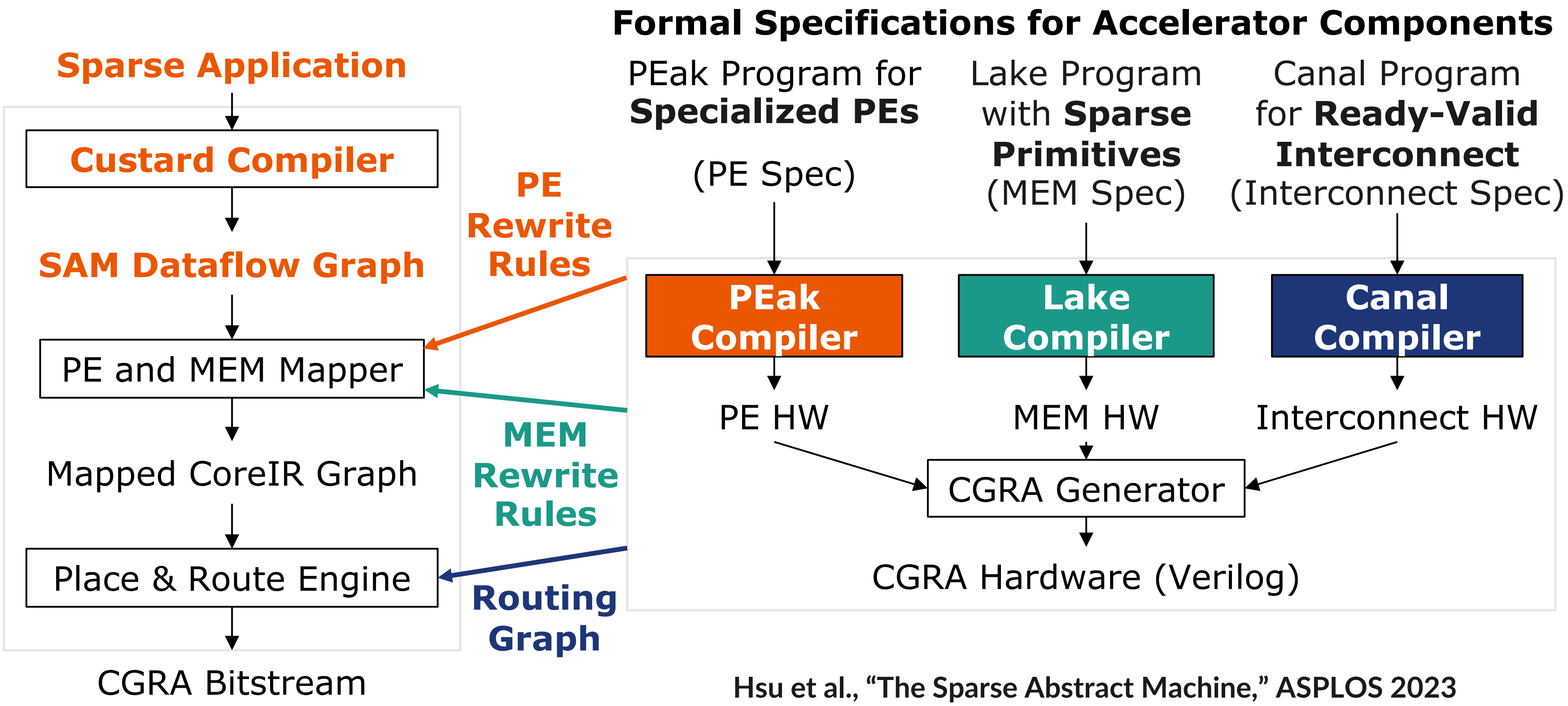


Application Domain Driven PE Architecture Creation

Application Dataflow Graph in CoreIR → Subgraph Mining → Maximal Independent Set Analysis → Ordered List of Frequent Subgraphs → Subgraph Merging → Merged PE Graph



Extending the CGRA to Sparse Applications



Hsu et al., “The Sparse Abstract Machine,” ASPLOS 2023

Extending the CGRA to Sparse Applications

Example Sparse Application

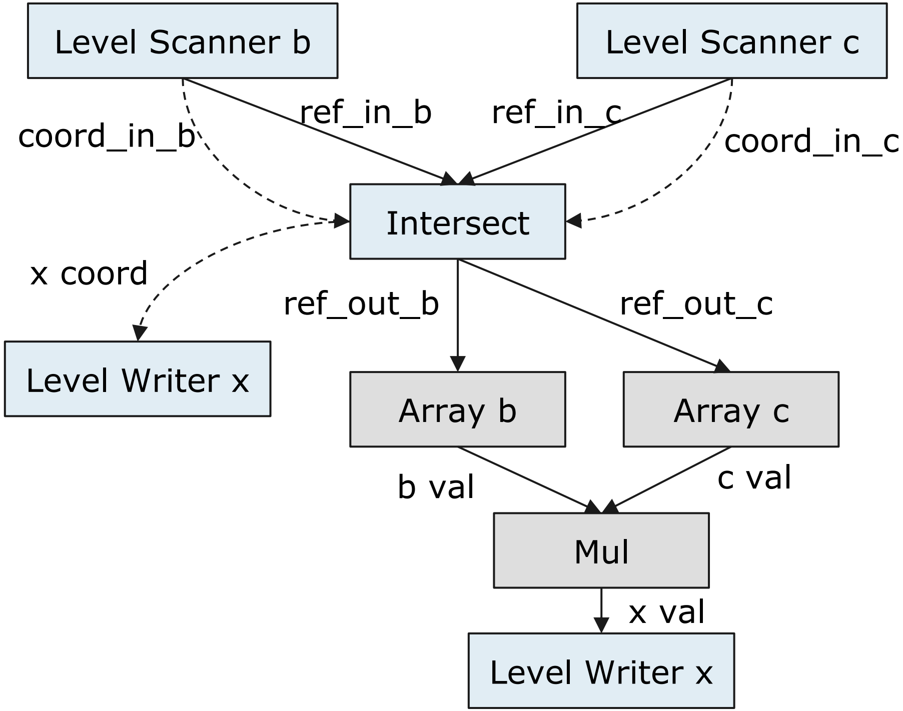
Compute (stmt):

```
X(i,j) = B(i) * C(i);
```

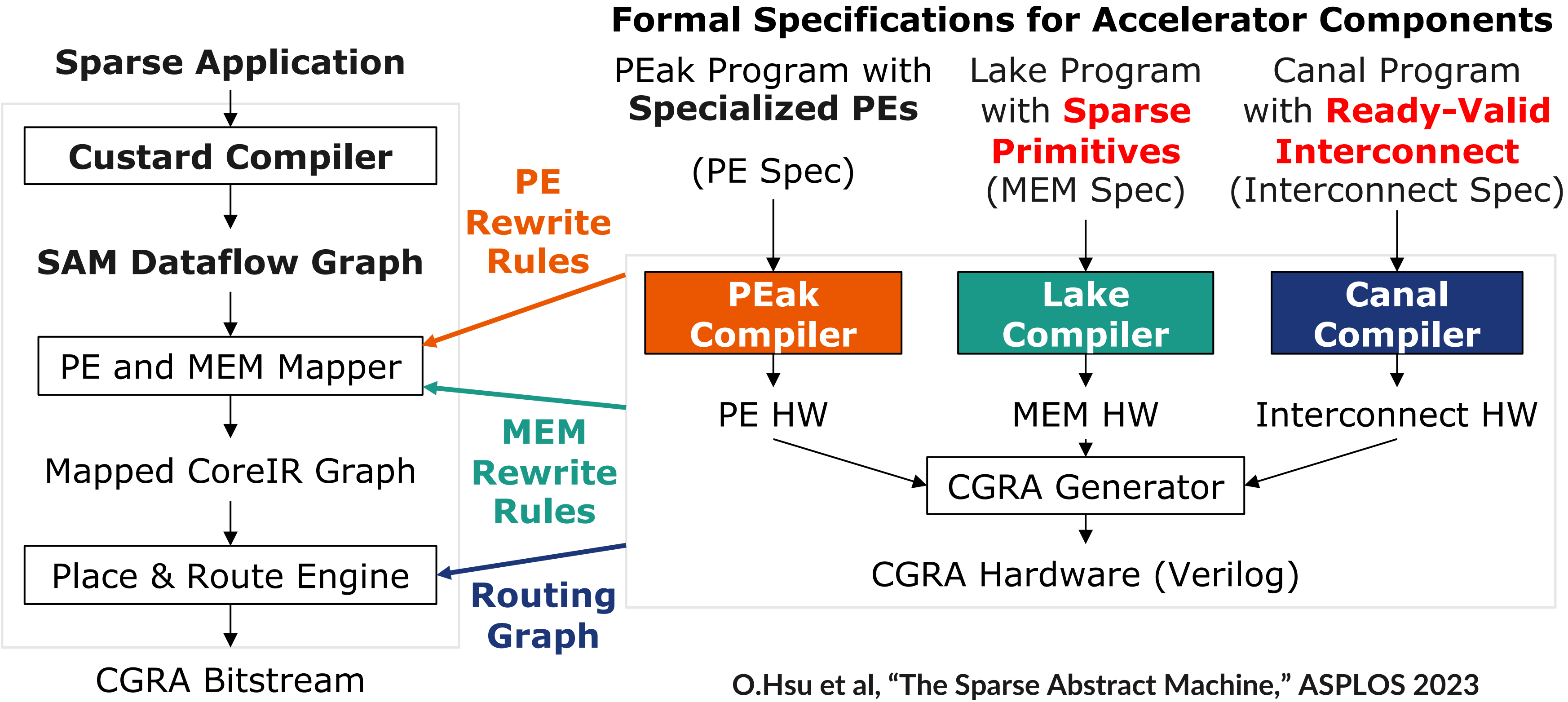
Scheduling and Format:

```
Format sp = Sparse;
Tensor X({sp});
Tensor B({sp});
Tensor C({sp});
```

Example SAM Graph



Extending the CGRA to Sparse Applications

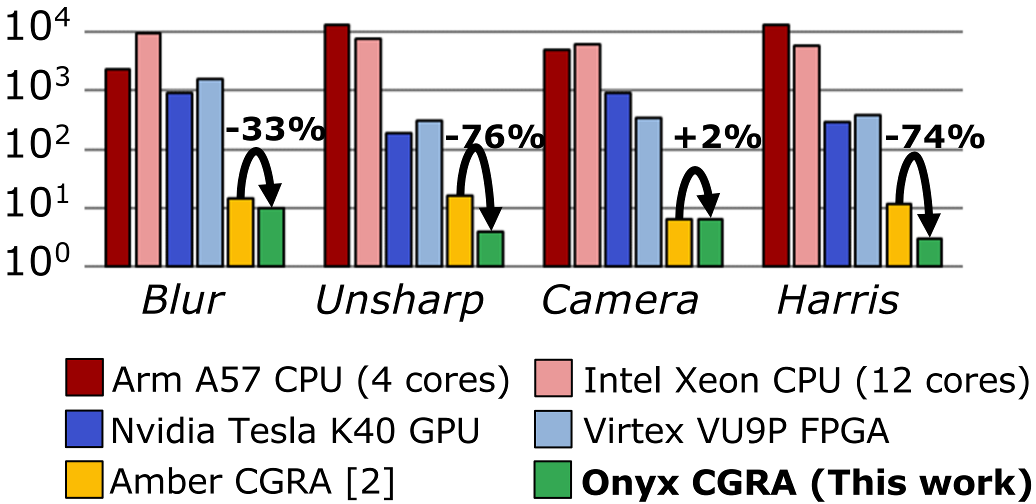


O.Hsu et al, “The Sparse Abstract Machine,” ASPLOS 2023

Onyx: CGRA with Dense and Sparse Acceleration

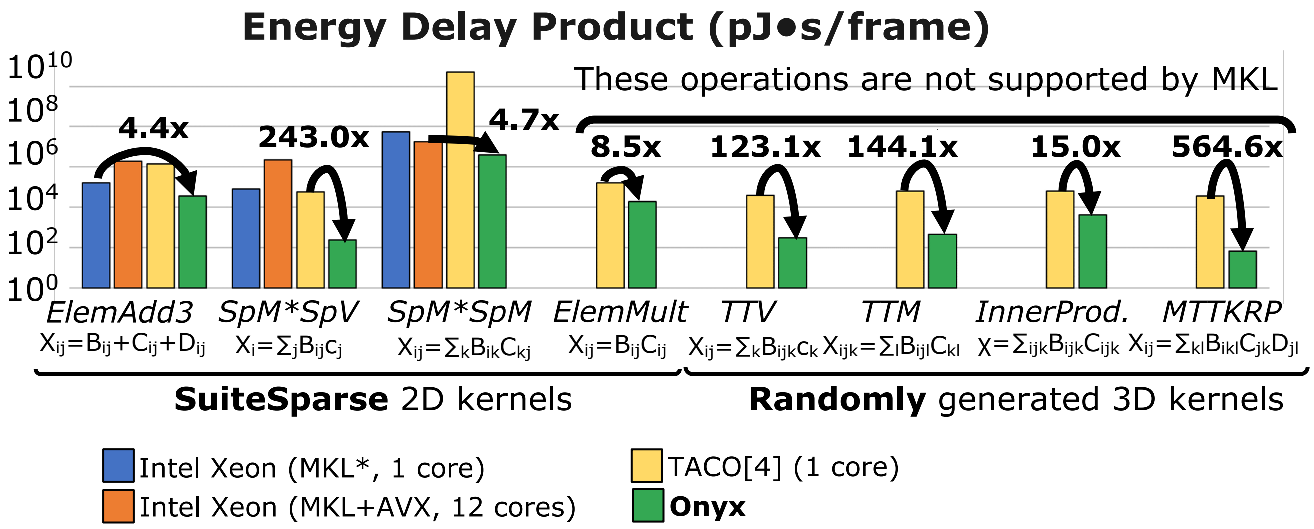
- Onyx achieves up to 85% lower EDP versus Amber on dense applications

Energy Delay Product (µJ•s/frame)



Onyx: CGRA with Dense and Sparse Acceleration

- Onyx achieves up to 565x EDP improvement over CPU sparse libraries



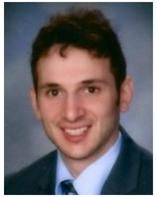
AHA Summary

- Demonstrated an **agile hardware-software co-design methodology**, using three key insights
    - Thinking about **accelerators as specialized CGRAs** provides a standard accelerator template for a compiler to target
    - Using a combination of new specification languages and formal methods, we can **automatically generate the accelerator hardware and its compiler** from a single source of truth
    - This enables creating higher-level **design-space exploration frameworks** for application domain-driven optimization of accelerator
- Used to create multiple end-to-end chip demonstrations, with significant reuse in both the hardware and the software toolchain

The AHA Team

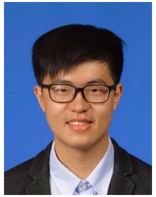
Rajsekhar
Setaluri

Lenny
Truong


Caleb
Donovick

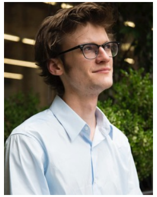
Alex
Carsello

Keyi
Zhang


Qiaoyi
Liu

Maxwell
Strange

Yuchen
Mei


Dillon
Huff

Olivia
Hsu

Ross
Daly

Ankita
Nayak

Kavya
Sreedhar

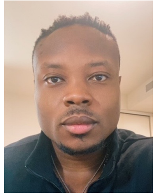
Jackson
Melchert

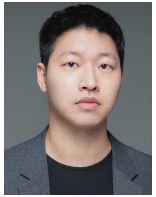
Jeff
Setter

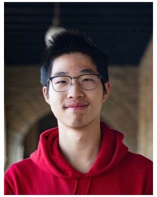
Steve
Richardson

Kalhan
Koul

Po-Han
Chen

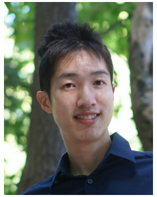

Gedeon
Nyengele


Taeyoung
Kong


Zhouhua
Xie

Huifeng
Ke

Kathleen
Feng


Christopher
Torng

Clark
Barrett

Priyanka
Raina

Mark
Horowitz

Pat
Hanrahan

Fredrik
Kjolstad

Acknowledgements for Funding

- DARPA DSSoC
- AHA Center
- Stanford SystemX Alliance
- SRC JUMP 2.0 PRISM Center
- NSF Award 2238006
- Intel HIP
- Apple
- Samsung

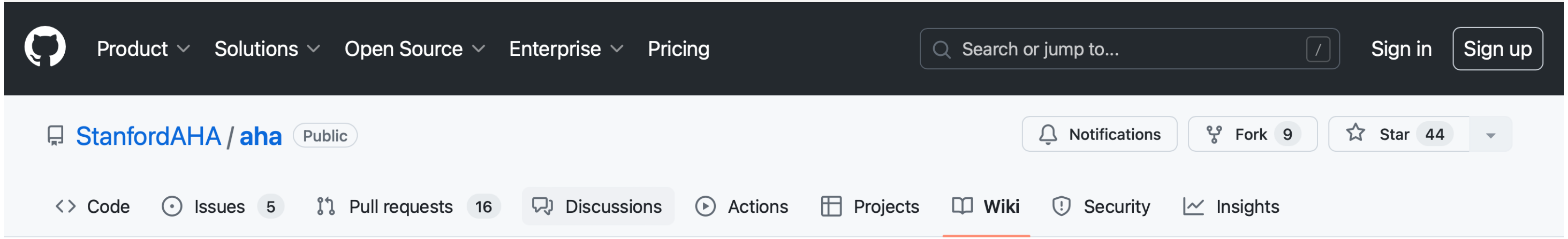
Publications

Hardware Generators:
AHA Overview – DAC 2020 https://ieeexplore.ieee.org/document/9218553
AHA Details – TECS 2023 https://ieeexplore.ieee.org/document/10258121
PEak – arXiv 2023 https://arxiv.org/abs/2308.13106
Canal – CAL 2023 https://ieeexplore.ieee.org/document/10105430
APEX – ASPLOS 2023 https://dl.acm.org/doi/10.1145/3582016.3582070

Compilers:
Dense Compiler – TACO 2023 https://dl.acm.org/doi/10.1145/3572908
Sparse Compiler & SAM – ASPLOS 2023 https://dl.acm.org/doi/10.1145/3582016.3582051
Rewrite Rule Generation – FMCAD 2022 https://ieeexplore.ieee.org/document/10026577
CGRA Pipelining – TCAD 2024 https://ieeexplore.ieee.org/document/10504565

Chips:
Onyx – VLSI 2024
Amber – VLSI 2022, JSSC 2023 https://ieeexplore.ieee.org/document/10258121

Code and Upcoming Tutorial!



- Try it out! : https://github.com/StanfordAHA/aha
- Wiki: https://github.com/StanfordAHA/aha/wiki
- Tutorial at MICRO 2024!
  Nov 2 – 6, 2024
  Austin, Texas, USA