

RipTide: A Programmable, Energy-Minimal Dataflow Compiler and Architecture

Graham Gobieski, **Souradip Ghosh**, Marjin Heule, Todd C. Mowry, Tony Nowatzki*, Nathan Beckmann, Brandon Lucia

Carnegie Mellon University, *UCLA | (SRC Task 3019.001)

Smart sensor devices at the extreme edge are emerging with huge industrial impact

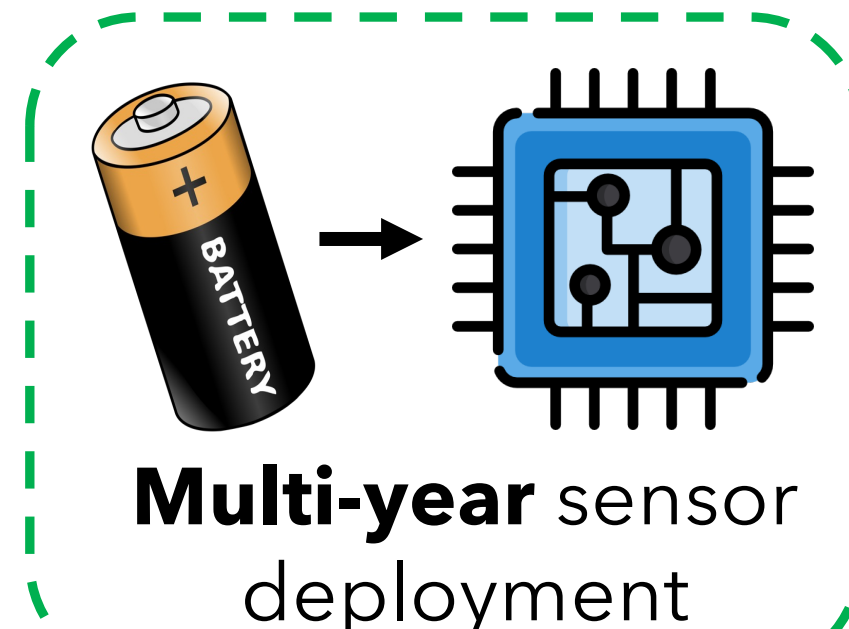


Trillions of devices¹ + **sophisticated** apps

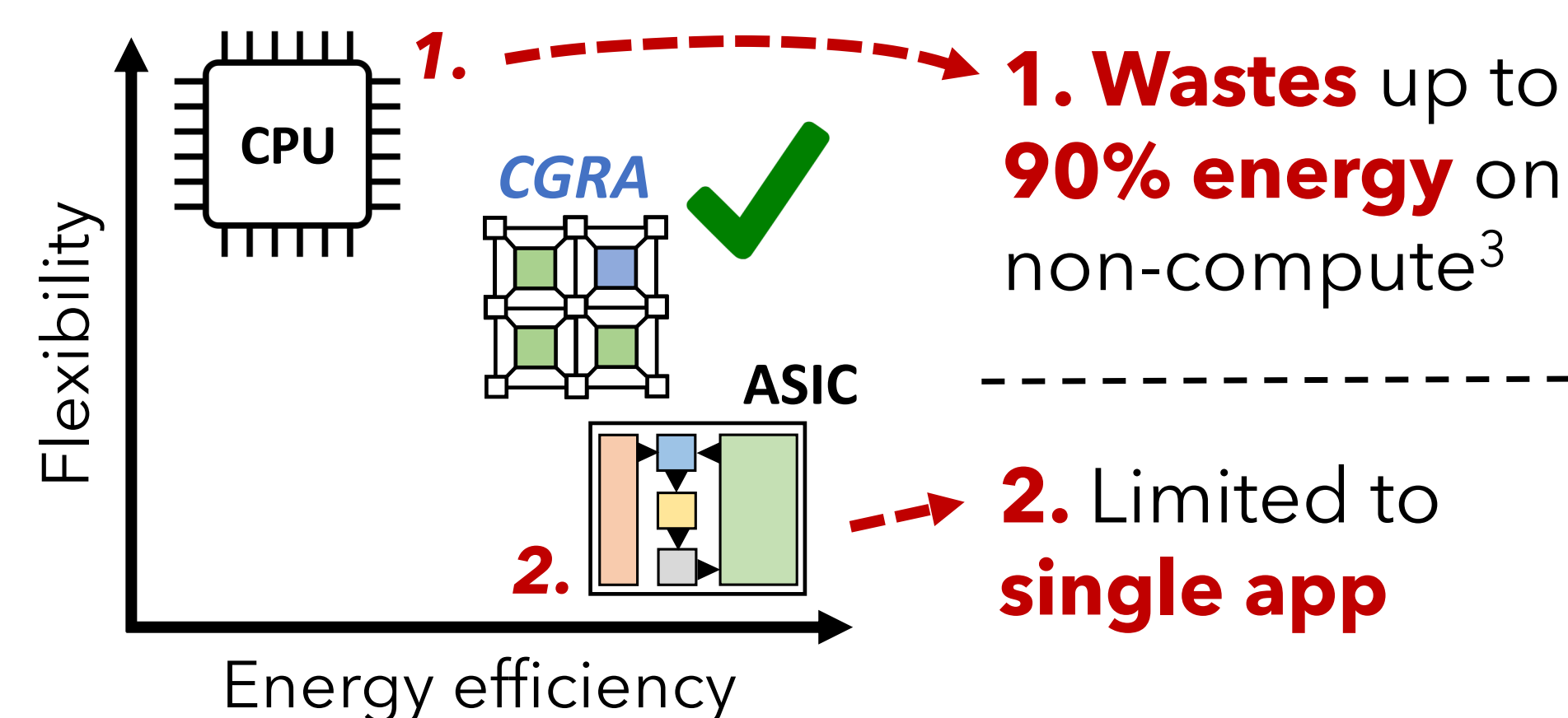
Must sustainably & efficiently compute at the edge. How?

1. Run apps on **ultra low power** (ULP), μ Ws

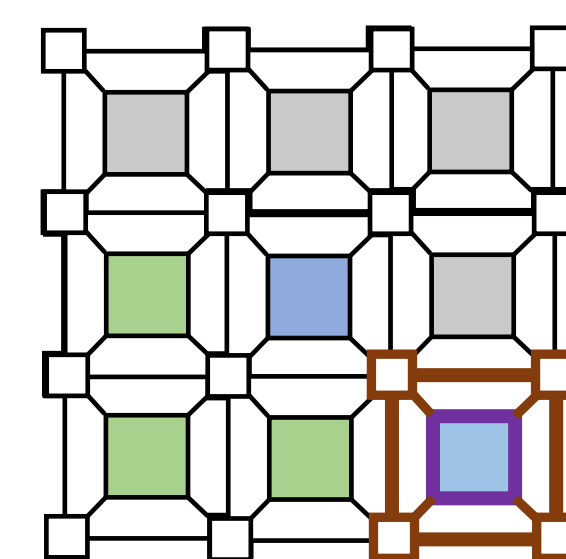
2. More **compute on-device**, less communication²



Goal: build a highly flexible & energy efficient compute



Coarse-grained reconfigurable arrays are flexible & efficient!

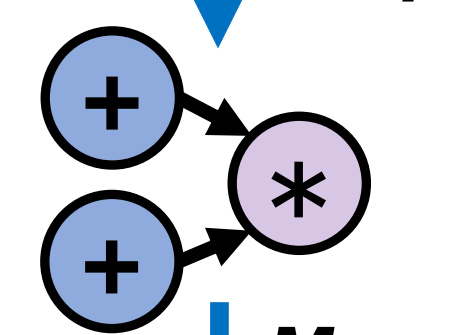


Grid of processing elements (PE) connected by a **NoC**.

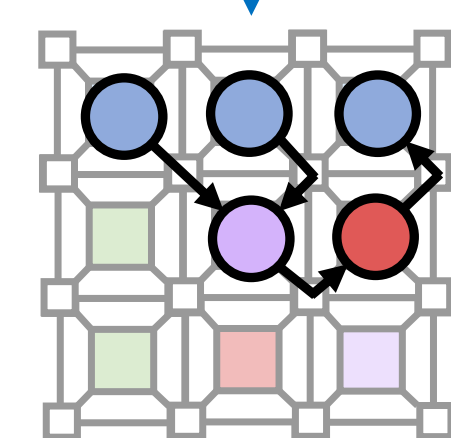
Eliminates fetch/decode & reg. file usage!

```
x: add ...
y: add ...
z: mul x,y
```

Compiler



Mapper



1. Extract a *dataflow graph* from code

2. Map ops to a PE mix and links on the NoC

3. Execute ops w/ "dataflow firing" or a static schedule

Prior ULP CGRAs are limited

```
void foo (...):
for (i = 0..n):
  vlh v1, a + i
  vadd v3, v1, v2
  vsh b + i, v3
```

CGRA code in assembly⁴

Runs **only affine inner loops**. No irregularity or operation ordering.

Insight: To improve efficiency, CGRAs need to run entire apps & support common PL idioms

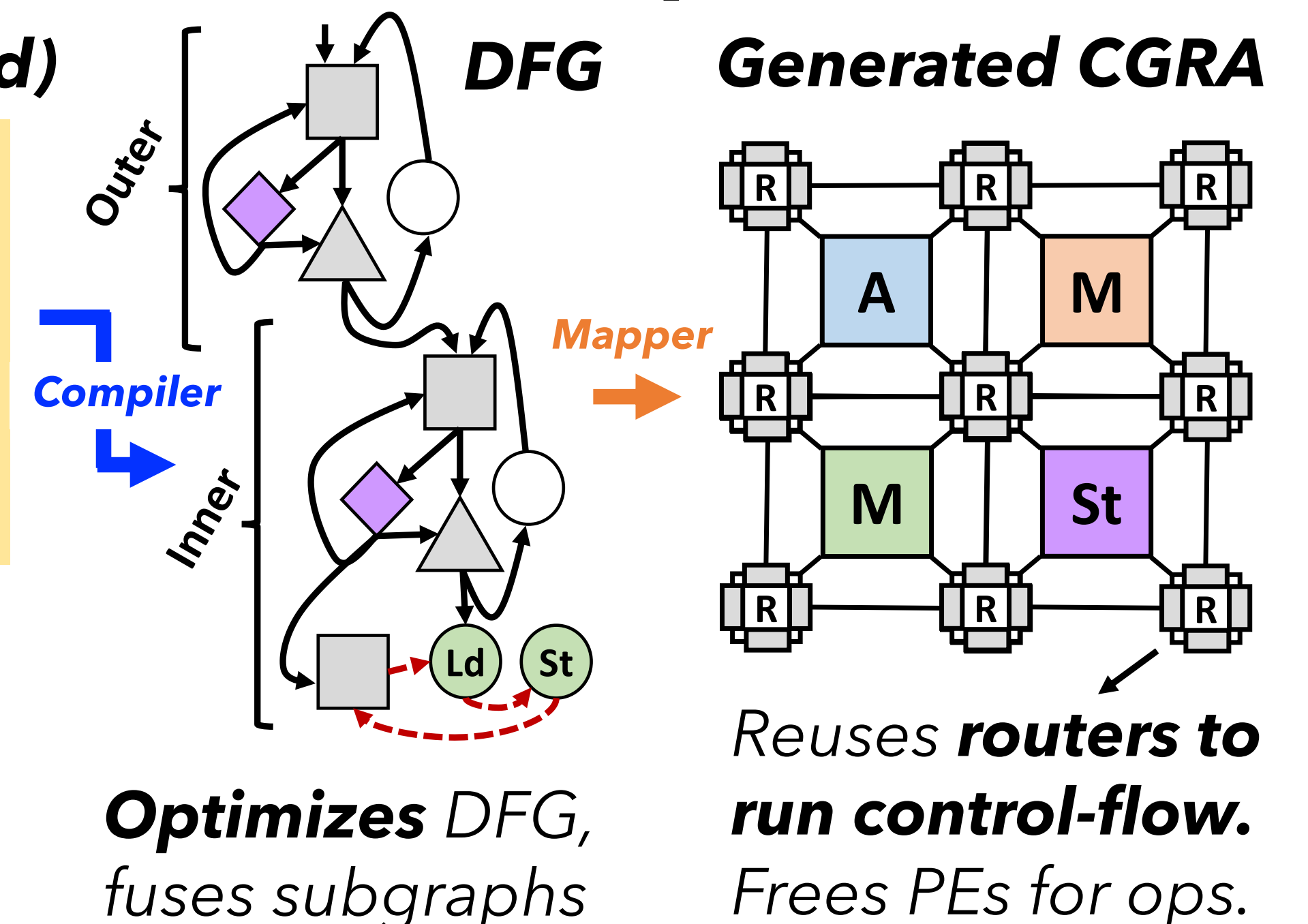
RipTide is a new ULP CGRA compiler & arch.

C code (lightly annotated)

```
#riptide void foo
(int * restrict a, b) {
  while (!q.empty()) {
    for (i in 0..q.pop())
      if (b[a[i]]) ...
  }
}
```

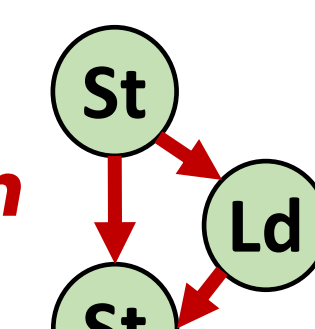
Handles arbitrary code via

- 1) Complex **control-flow**
- 2) **Irregular mem. access**
- 3) Operation **ordering**

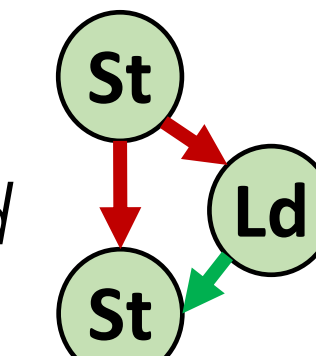


Memory ordering:

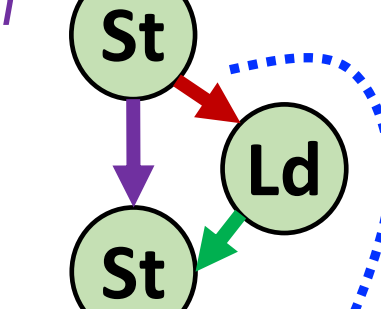
a) Build an **ordering graph** for mem. deps



b) **Prune arcs** via existing data and control deps



c) Perform a **path-sensitive transitive reduction**



d) Remaining **ordering arcs are enforced**.

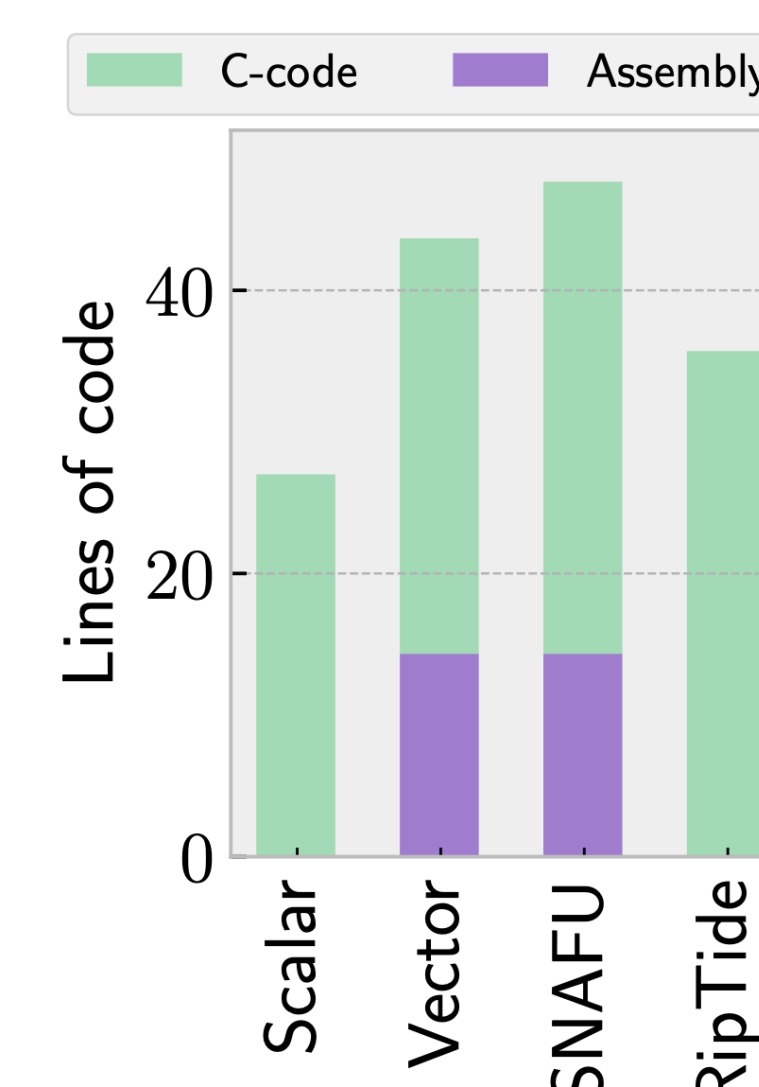
Mapping DFGs to the CGRA:

RipTide formulates PE/link place & route as **SAT** or **ILP** problems.

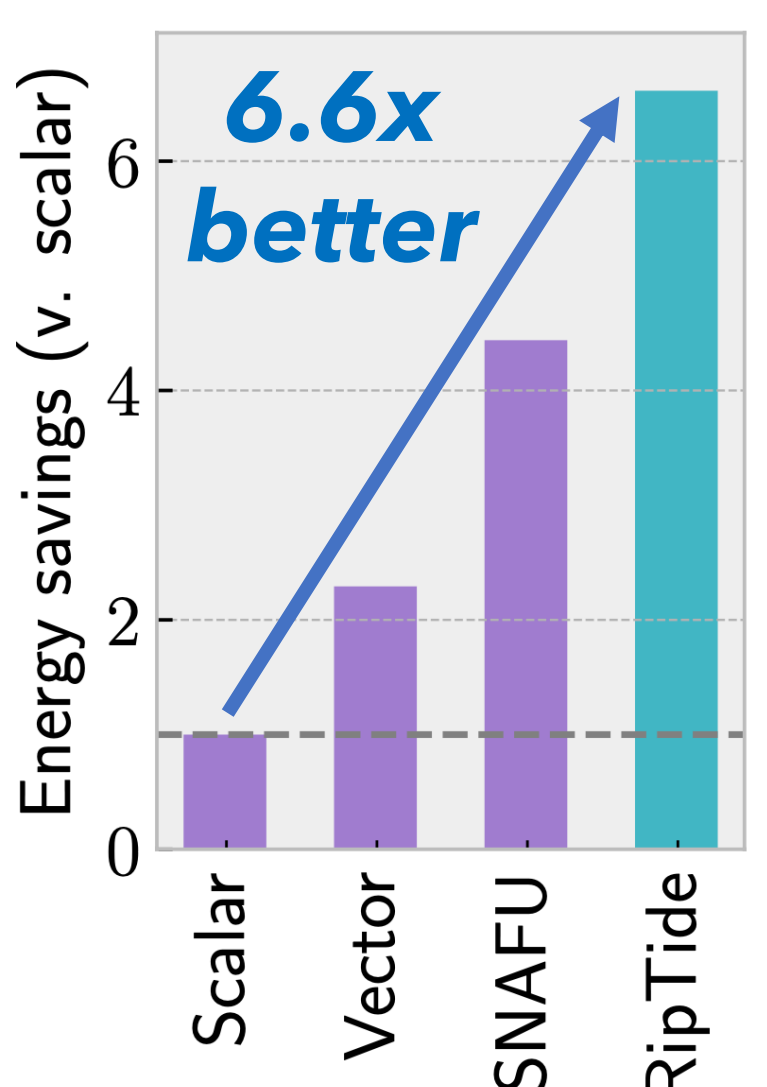
SAT: Adequate soln. Quickly solves.

ILP: Better soln. Slowly solves.

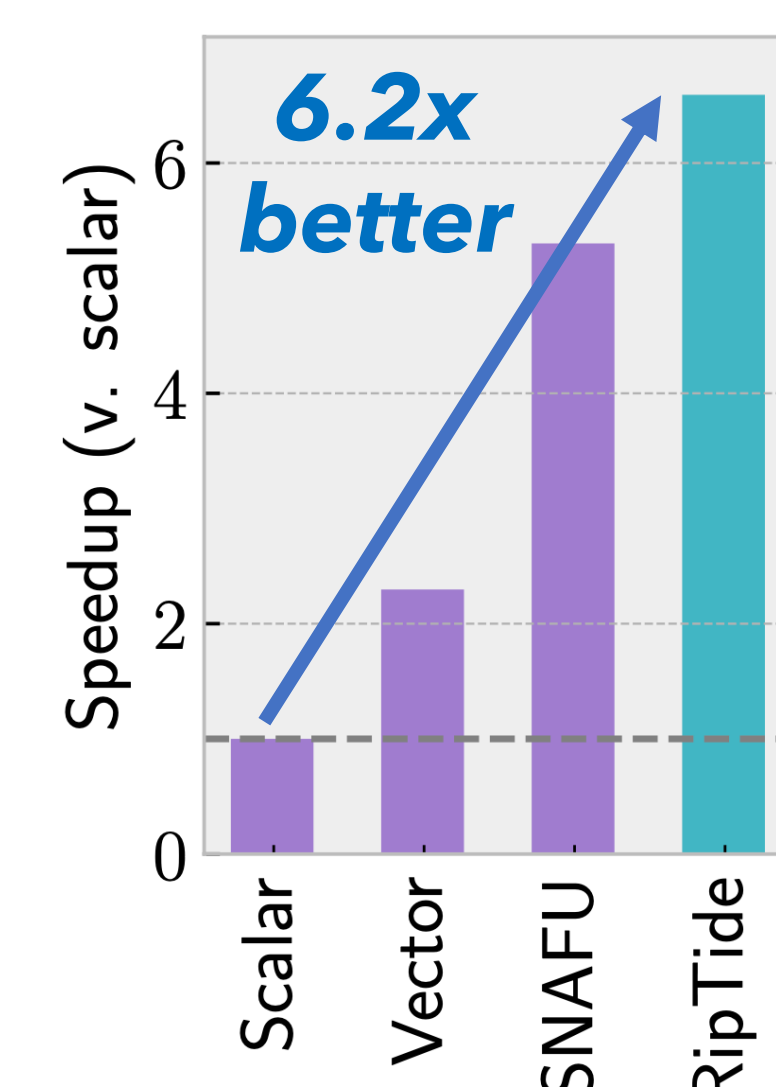
Programmability



Energy-Efficiency



Performance



10 apps from linalg, graph processing, and signal processing

+ **Full compiler** built with **LLVM**

+ **Full RTL design** and **synthesized**

+ Ran an **entire DNN** on **RipTide!**

¹Arm, "How to build a trillion connected things."

²Gobieski et al., "Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems." (ASPLOS '19).

³Horowitz, "Computing's energy problem (and what we can do about it)." (ISSCC '14).

⁴Gobieski et al., "SNAFU: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture." (ISCA '21).