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)

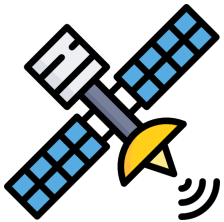
Outline

1) Motivation
2) Background
    a) Coarse-grained reconfigurable arrays (CGRAs)
    b) Prior work
3) ***RipTide***
    a) Overview → Compiler and architecture
    b) Memory ordering
    c) Mapping
    d) Results

Motivation: *Sensors at the extreme edge*

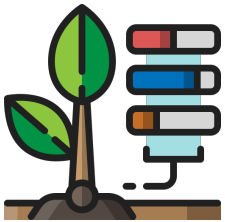
Smart sensor devices at the ***extreme edge*** are quickly emerging.

Devices need to run **complex apps** → inference, DSP, and more.



Nano Satellites

Medical Wearables



Wildlife Monitoring

Trillions of devices[1] coming

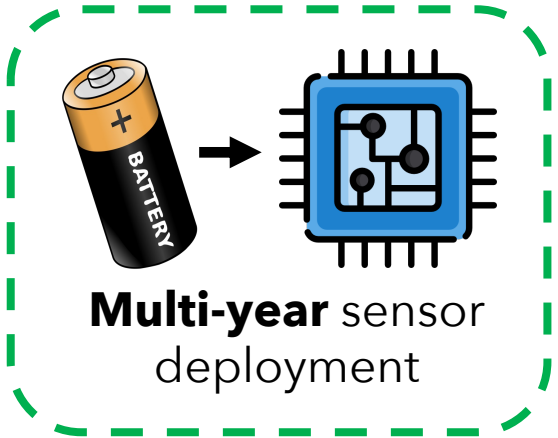
[1]Arm, "How to build a trillion connected things."

Motivation: *Computing at the extreme edge?*

Compute must be ***efficient*** and ***sustainable*** at the extreme edge.

1. Run **variety of apps** on ***ultra low power*** (*ULP*), µWs

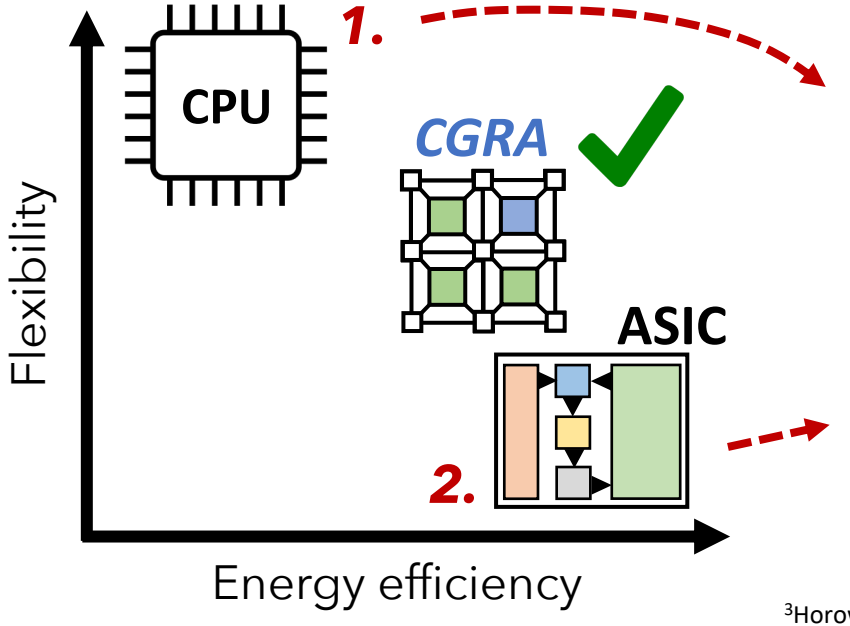
2. More **compute on-device**, less communication[2]



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

Background: *Comparing existing architectures*

Devices must have architectures that are ***flexible*** and ***efficient***.

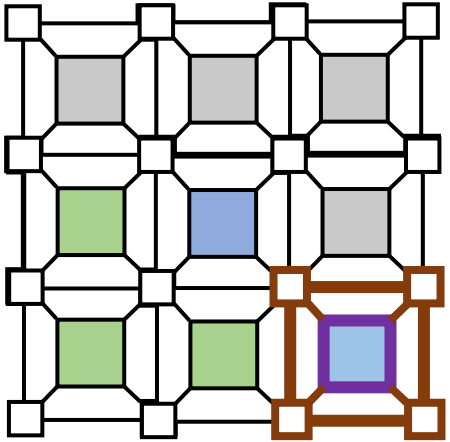


1. Wastes up to **90% energy** on non-compute[3]

2. Inflexible by design, limited to **single app**

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

CGRAs: *What is a CGRA?*

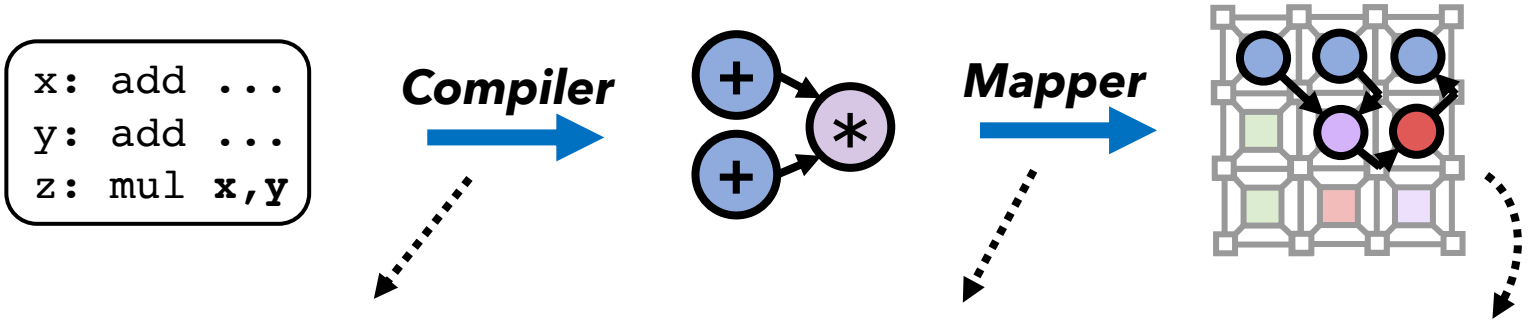


Coarse-grained configurable arrays:

Grid of processing elements (**PE**, to execute ops) connected by a **NoC** (to send values).

Eliminates **fetch/decode** & **reg. file usage!**

CGRAs: *An end-to-end pipeline*



1. Extract a *dataflow graph* (DFG) from code, optimize

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

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

Prior Work: *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 assembly[4]

Runs ***only*** **affine inner loops.**

No complex control-flow, irregular memory accesses, or operation ordering.

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

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

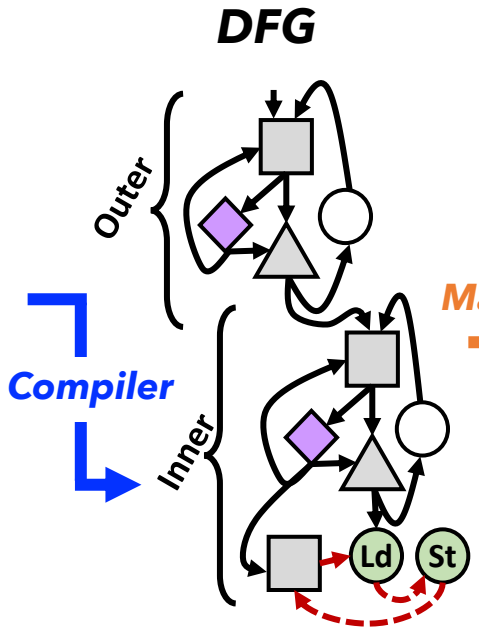
RipTide: *Overview*

C code (lightly annotated)

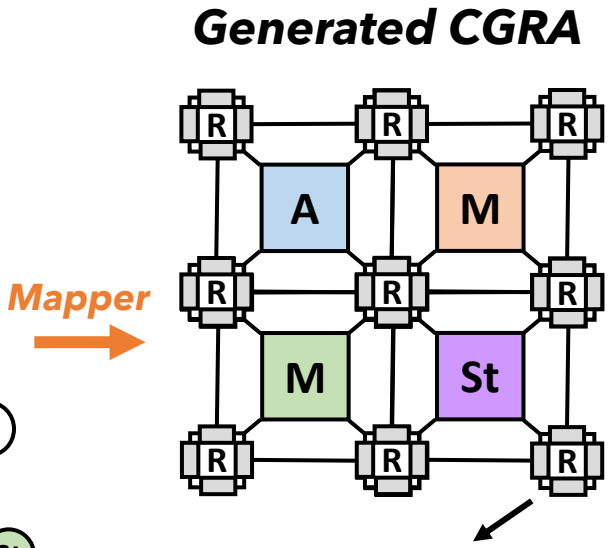
```
#riptide void foo
(int * restrict a, b) {
  while (!q.empty()) {
    for (i in 0..q.pop())
      if (b[a[i]]) ...
  }
}
```

Handles arbitrary code via

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

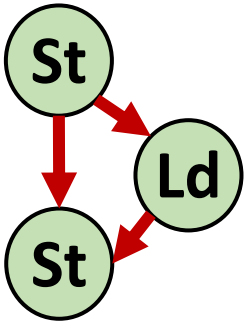


Optimizes *DFG, fuses subgraphs*

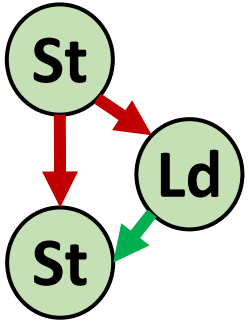


Reuses ***routers to run control-flow.*** *Frees PEs for ops.*

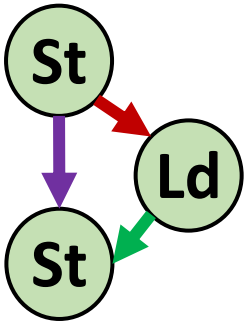
RipTide: *Memory ordering*



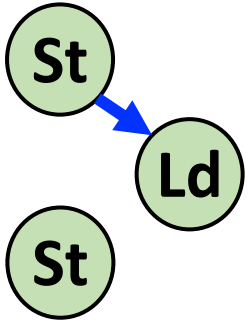
a) Build an ***ordering graph*** *for mem. deps*



b) ***Prune*** *already-enforced arcs via existing data and control deps*



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

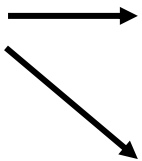


d) Remaining ordering arcs are ***enforced****.*

RipTide: *Mapping*

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

RipTide also only maps *spatially* (no static execution schedule).

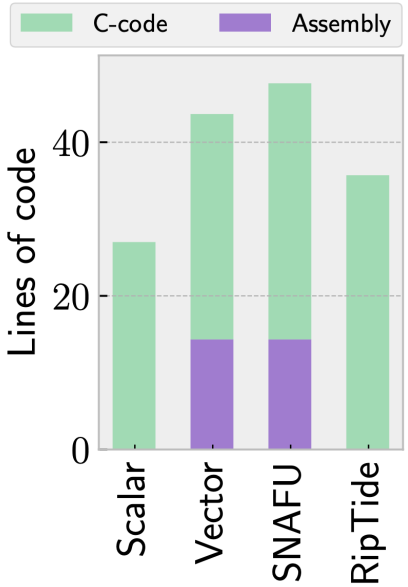


SAT: *Adequate soln. Quickly solves.*

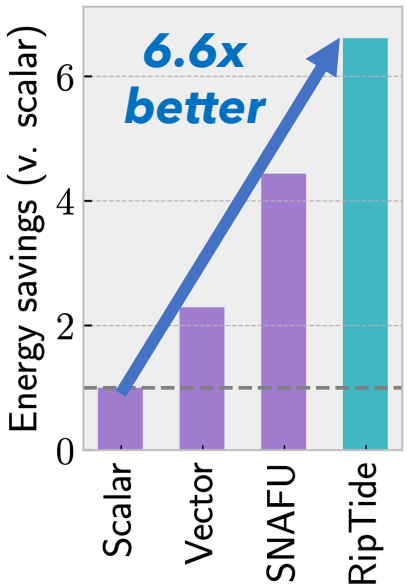
ILP: *Better soln. Slowly solves.*

RipTide: *Results*

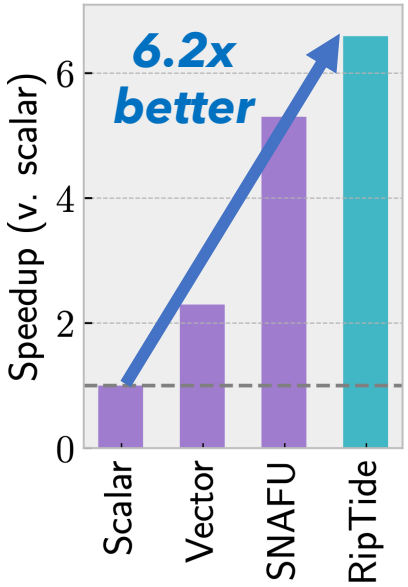
Programmability



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****!*

Thank you!

Correspondence:

- Souradip Ghosh (souradip@cmu.edu)
- Graham Gobieski (gobieski@cmu.edu)

For more details, visit:

- Abstract Group (abstract.ece.cmu.edu)
- CORGi Group (cmu-corgi.github.io)