Dual-Issue Execution of Mixed Integer and Floating-Point Workloads on Energy-Efficient In-Order RISC-V Cores

Luca Colagrande, Luca Benini

ETH zürich

PULP

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

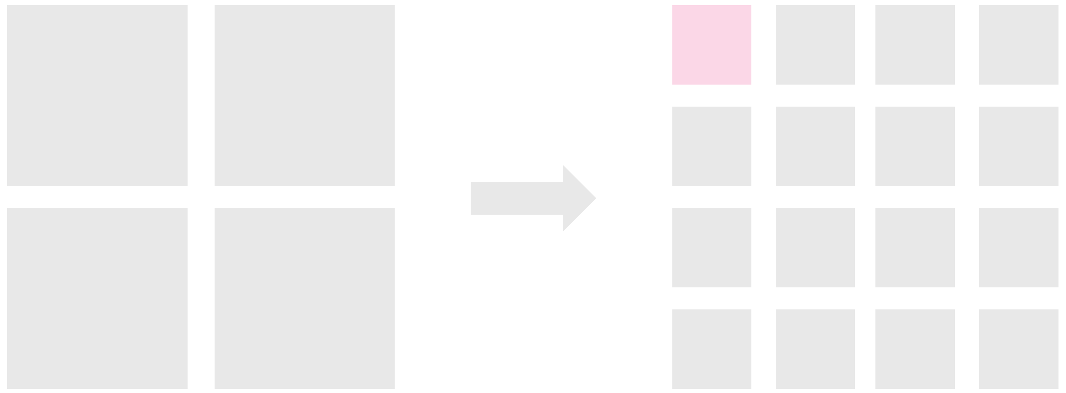
Introduction

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Context

- Many-core general-purpose accelerators
- Built from arrays of *slimmed-down*, *area-* and *energy-efficient* processors
- Typically, *scalar* in-order cores



Motivation

- Few of these propose cores with (limited) *multiple-issue* capabilities
- Nvidia Turing implements *concurrent execution* of FP32 and INT32 operations
  - *+36% throughput*, across several gaming workloads [1]
- *Undisclosed* design, *closed-source* implementation

[1] J. Burgess, “Rtx on—the nvidia turing gpu,” IEEE Micro, vol. 40, no. 2, pp. 36–44, 2020.

Background

- Snitch [2]
  - Tiny, single-issue in-order RV32I core
  - Coupled to a “D” extension capable FPU
- Xfrep extension: zero-overhead loop (ZOL) buffer for FP instructions
- *Pseudo dual-issue* execution
  - Regular instruction fetch in RV32I core
  - Simultaneously, FP instruction issue from ZOL buffer

[2] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, “Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads,” IEEE Transactions on Computers, vol. 70, no. 11, pp. 1845–1860, 2021.

Background

- Concurrent integer and FP threads
- Constraints
    - Fully asynchronous threads, *no synchronization*
    - Fully independent threads, *no memory consistency*
- Threads should access *exclusive resources*
    - Integer thread accesses integer RF, exclusively
    - Floating-point thread accesses FP RF, exclusively

Background

- Instructions accessing both integer and FP RF *disallowed*
  - FP comparisons (`FEQ.*`, `FLT.*`, `FLE.*`, `FCLASS.*`)
  - FP load/stores, conversions and moves (`FLD`, `FSD`, `FCVT.*.*`, `FMV.*.*`)
- Dual-issue execution of mixed-integer-FP kernels *unsupported*
  - e.g. transcendental functions (`exp`, `log`), Monte Carlo, etc.

Goal

- *Enable cooperation* of parallel integer and FP threads
- *Support* dual-issue execution of mixed-integer-FP kernels

COPIFT: Co-**O**perative **P**arallel **I**nteger and **F**loating-point **T**hreads

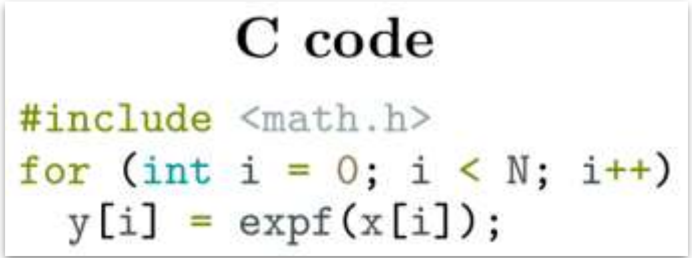
Implementation

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

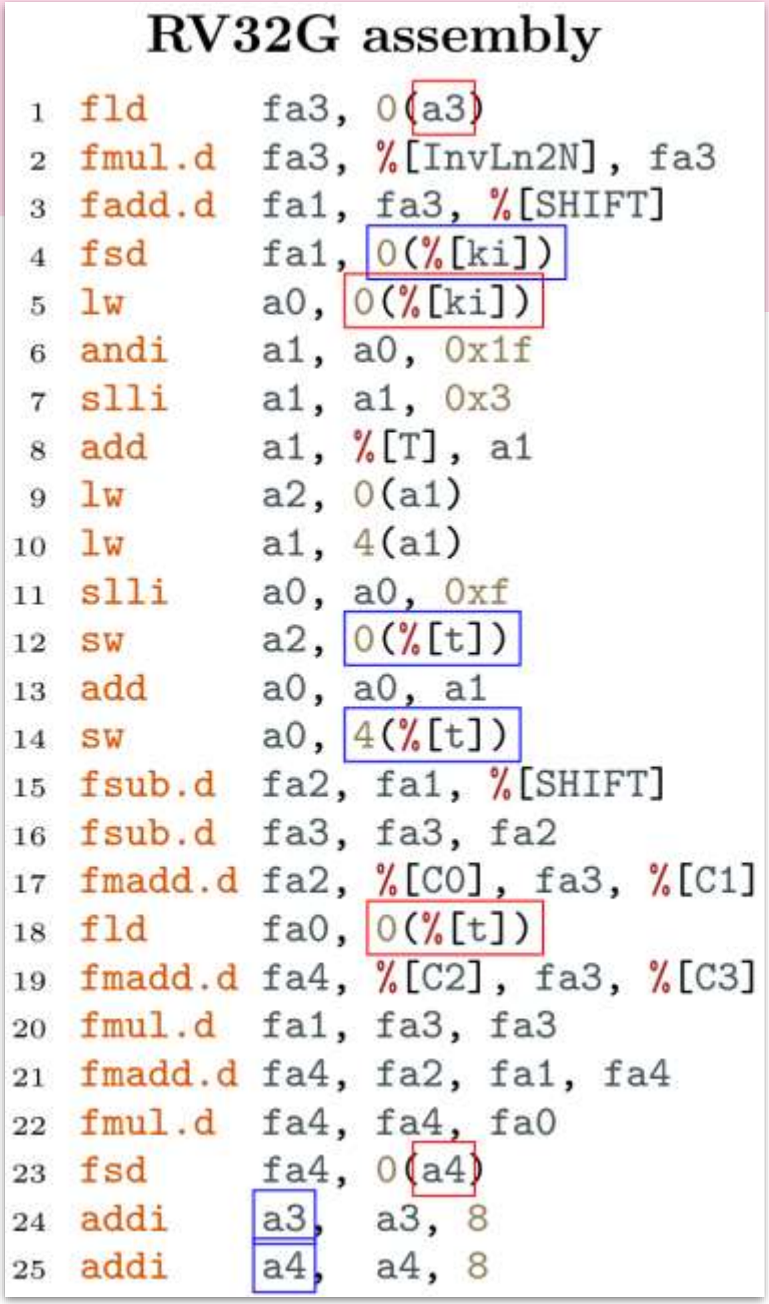
Example

- Exponential of a vector
    - Found e.g. in softmax



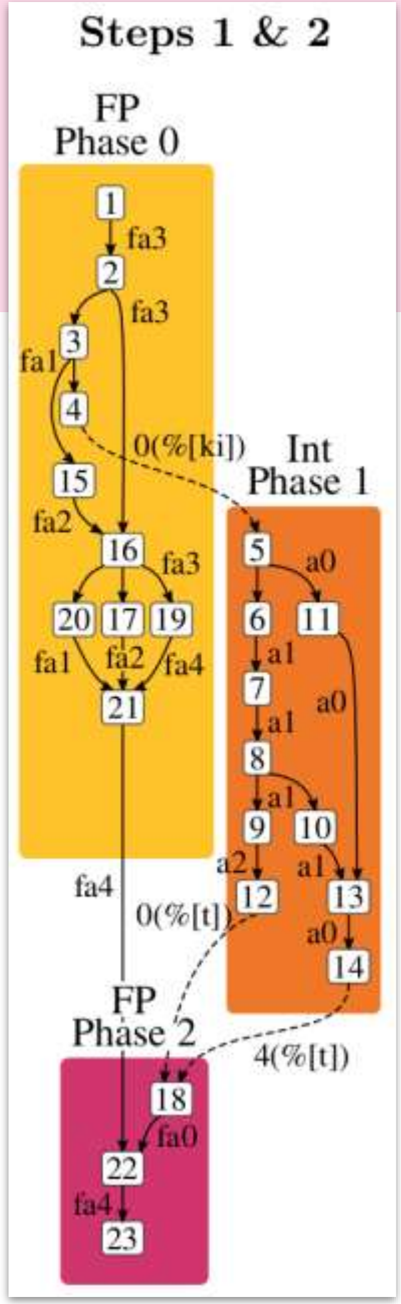
C code

```
#include <math.h>
for (int i = 0; i < N; i++)
  y[i] = expf(x[i]);
```

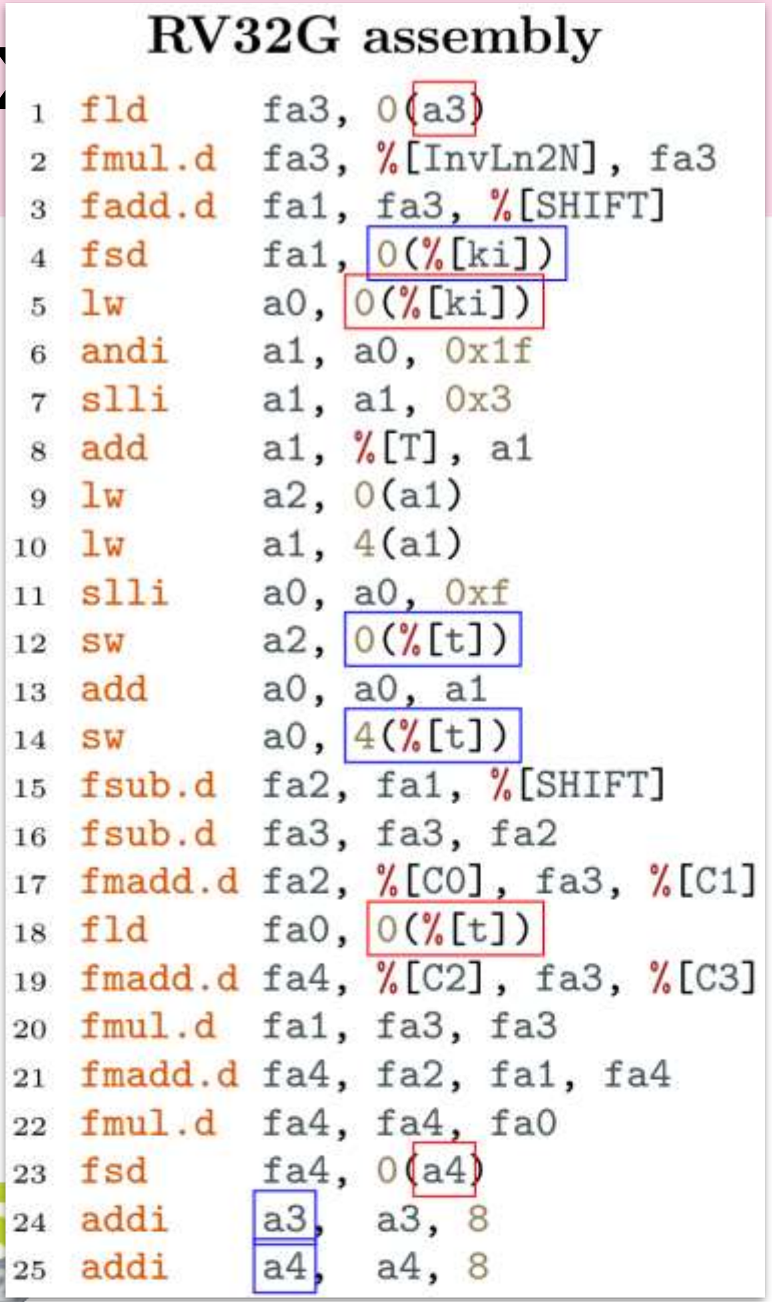


RV32G assembly

```
1  fld      fa3, 0(a3)
2  fmul.d   fa3, %[InvLn2N], fa3
3  fadd.d   fa1, fa3, %[SHIFT]
4  fsd      fa1, 0(%[ki])
5  lw       a0, 0(%[ki])
6  andi     a1, a0, 0x1f
7  slli     a1, a1, 0x3
8  add      a1, %[T], a1
9  lw       a2, 0(a1)
10 lw       a1, 4(a1)
11 slli     a0, a0, 0xf
12 sw       a2, 0(%[t])
13 add      a0, a0, a1
14 sw       a0, 4(%[t])
15 fsub.d   fa2, fa1, %[SHIFT]
16 fsub.d   fa3, fa3, fa2
17 fmadd.d  fa2, %[C0], fa3, %[C1]
18 fld      fa0, 0(%[t])
19 fmadd.d  fa4, %[C2], fa3, %[C3]
20 fmul.d   fa1, fa3, fa3
21 fmadd.d  fa4, fa2, fa1, fa4
22 fmul.d   fa4, fa4, fa0
23 fsd      fa4, 0(a4)
24 addi     a3, a3, 8
25 addi     a4, a4, 8
```

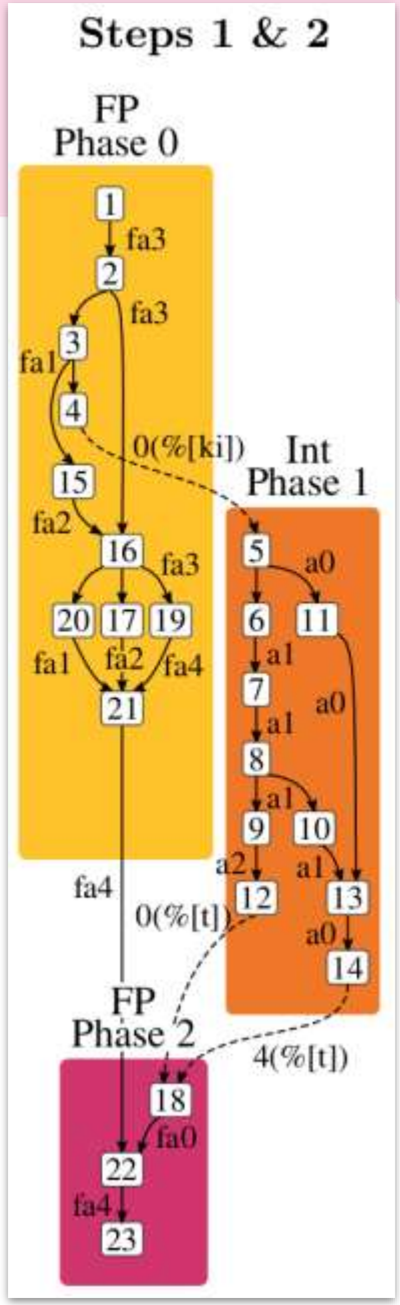


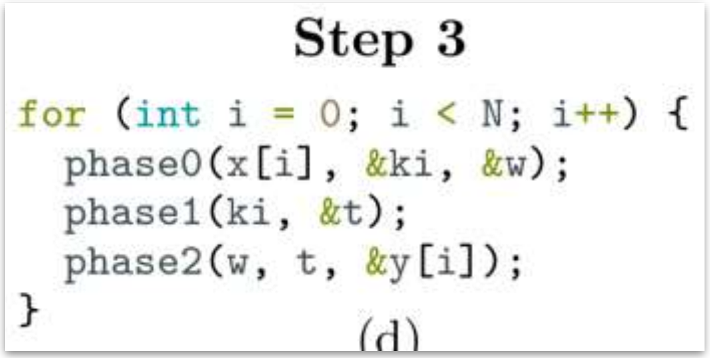
Ex



RV32G assembly

```
1  fld      fa3, 0(a3)
2  fmul.d   fa3, %[InvLn2N], fa3
3  fadd.d   fa1, fa3, %[SHIFT]
4  fsd      fa1, 0(%[ki])
5  lw       a0, 0(%[ki])
6  andi     a1, a0, 0x1f
7  slli     a1, a1, 0x3
8  add      a1, %[T], a1
9  lw       a2, 0(a1)
10 lw       a1, 4(a1)
11 slli     a0, a0, 0xf
12 sw       a2, 0(%[t])
13 add      a0, a0, a1
14 sw       a0, 4(%[t])
15 fsub.d   fa2, fa1, %[SHIFT]
16 fsub.d   fa3, fa3, fa2
17 fmadd.d  fa2, %[C0], fa3, %[C1]
18 fld      fa0, 0(%[t])
19 fmadd.d  fa4, %[C2], fa3, %[C3]
20 fmul.d   fa1, fa3, fa3
21 fmadd.d  fa4, fa2, fa1, fa4
22 fmul.d   fa4, fa4, fa0
23 fsd      fa4, 0(a4)
24 addi     a3, a3, 8
25 addi     a4, a4, 8
```



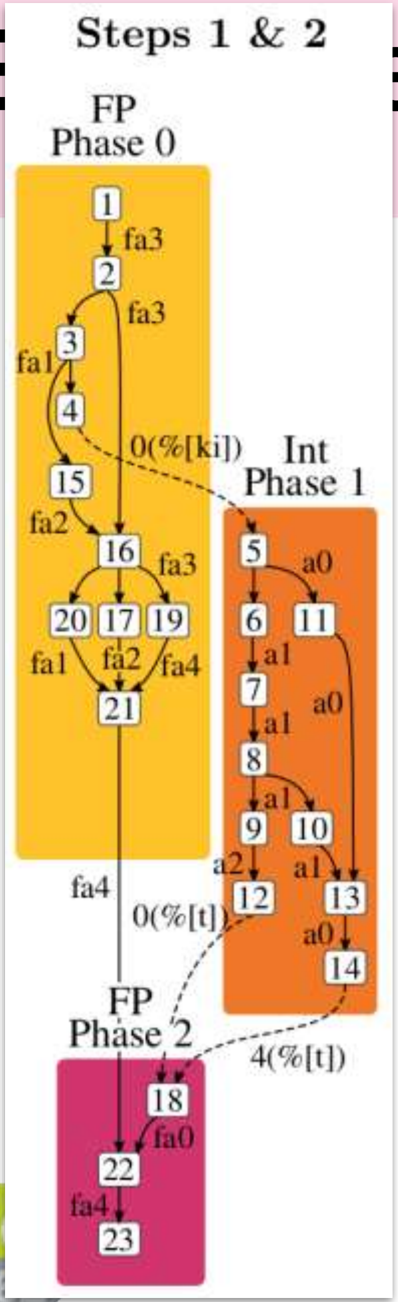


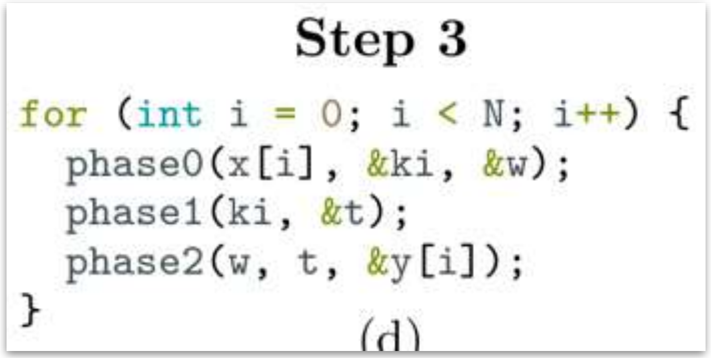
Step 3

```
for (int i = 0; i < N; i++) {
  phase0(x[i], &ki, &w);
  phase1(ki, &t);
  phase2(w, t, &y[i]);
}
```

(d)

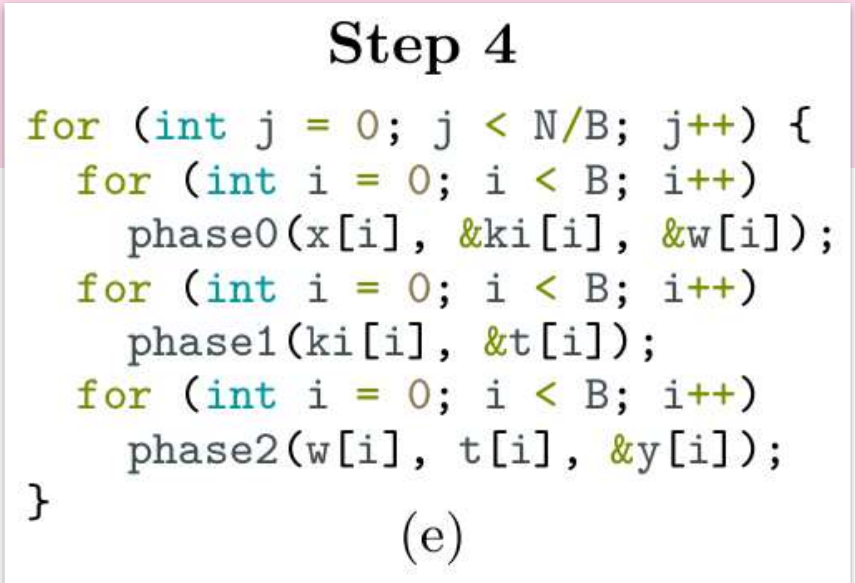
E e


Steps 1 & 2
FP Phase 0
Int Phase 1
FP Phase 2
1 fa3 2 fa3 3 fa1 4 15 fa2 16 fa3 20 17 19 fa1 fa2 fa4 21
0(%[ki])
5 a0 6 11 a1 7 a0 a1 8 a1 9 10 a2 a1 12 13 a0 14
fa4
0(%[t])
4(%[t])
18 fa0 22 fa4 23


Step 3

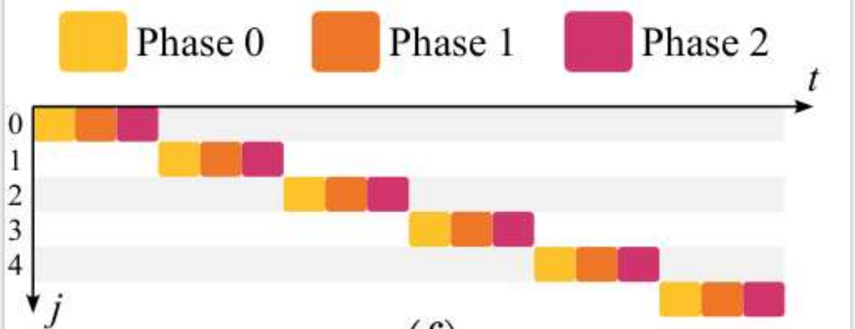
```
for (int i = 0; i < N; i++) {
  phase0(x[i], &ki, &w);
  phase1(ki, &t);
  phase2(w, t, &y[i]);
}
```

(d)


Step 4

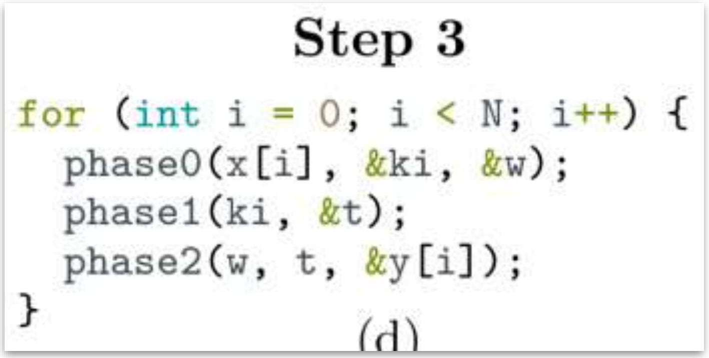
```
for (int j = 0; j < N/B; j++) {
  for (int i = 0; i < B; i++)
    phase0(x[i], &ki[i], &w[i]);
  for (int i = 0; i < B; i++)
    phase1(ki[i], &t[i]);
  for (int i = 0; i < B; i++)
    phase2(w[i], t[i], &y[i]);
}
```

(e)



(f)

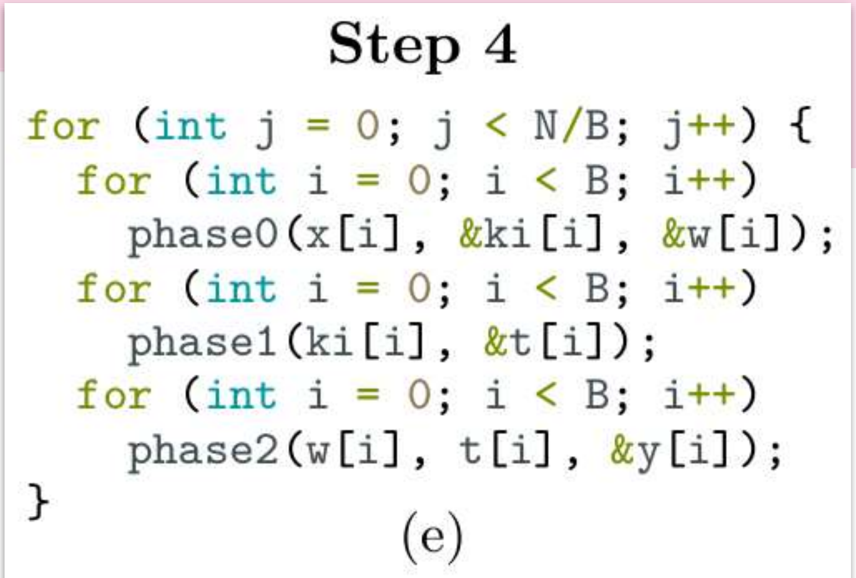
Example



Step 3

```
for (int i = 0; i < N; i++) {
  phase0(x[i], &ki, &w);
  phase1(ki, &t);
  phase2(w, t, &y[i]);
}
```

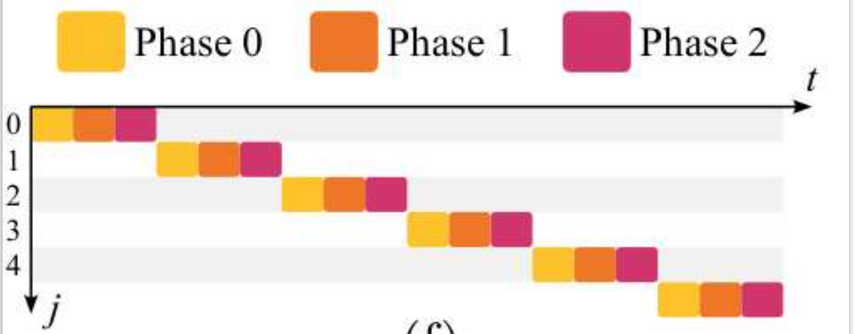
(d)



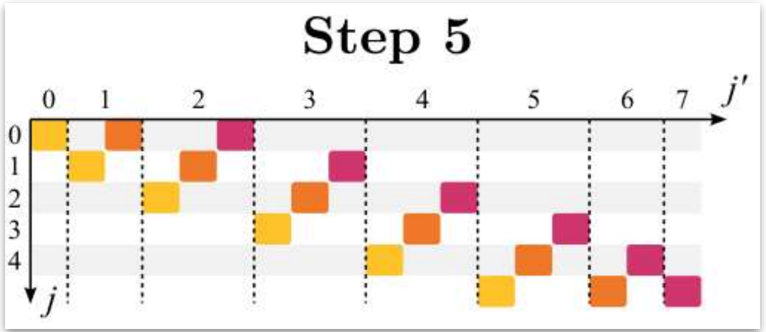
Step 4

```
for (int j = 0; j < N/B; j++) {
  for (int i = 0; i < B; i++)
    phase0(x[i], &ki[i], &w[i]);
  for (int i = 0; i < B; i++)
    phase1(ki[i], &t[i]);
  for (int i = 0; i < B; i++)
    phase2(w[i], t[i], &y[i]);
}
```

(e)



(f)



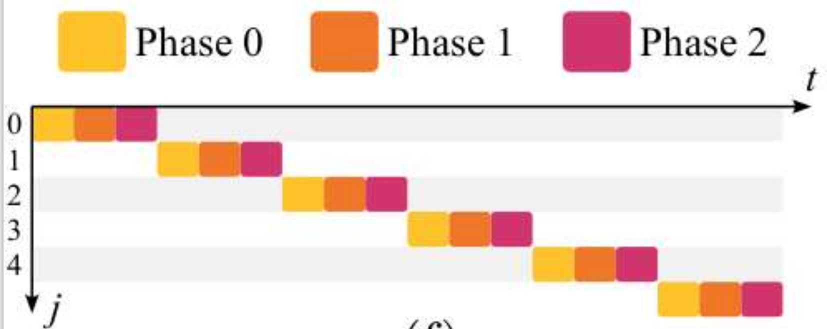
Step 5

Example

Step 4

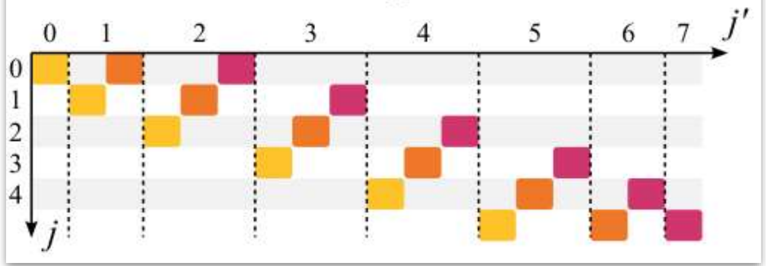
```
for (int j = 0; j < N/B; j++) {
  for (int i = 0; i < B; i++)
    phase0(x[i], &ki[i], &w[i]);
  for (int i = 0; i < B; i++)
    phase1(ki[i], &t[i]);
  for (int i = 0; i < B; i++)
    phase2(w[i], t[i], &y[i]);
}
```

(e)

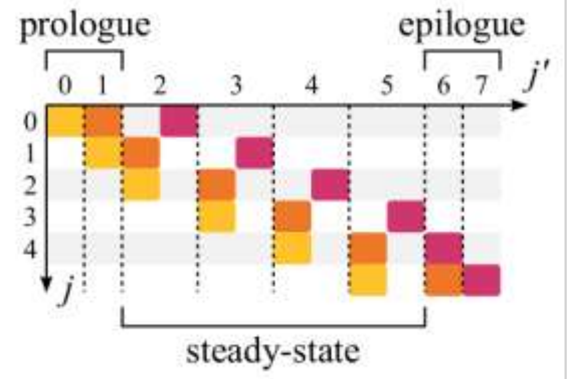


(f)

Step 5

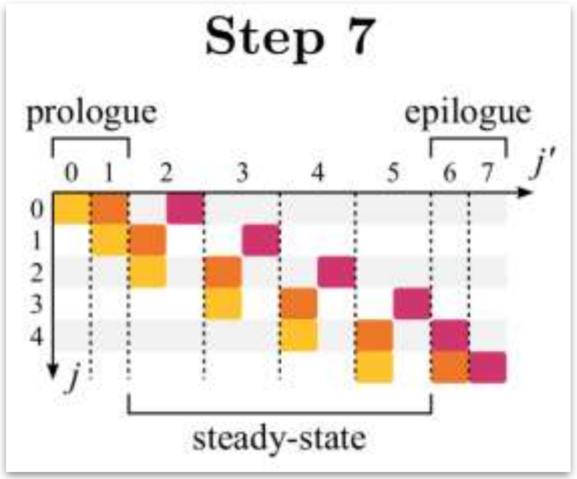


Step 7



Example

- FP phases 0 and 2 are mapped to FREP loops
- Execute in parallel with integer phase 1
- No communication and synchronization within blocks
- Explicit communication and synchronization across blocks
- Details omitted from presentation for brevity
    - Step 6
    - Minor ISA extensions to broaden COPIFT scope



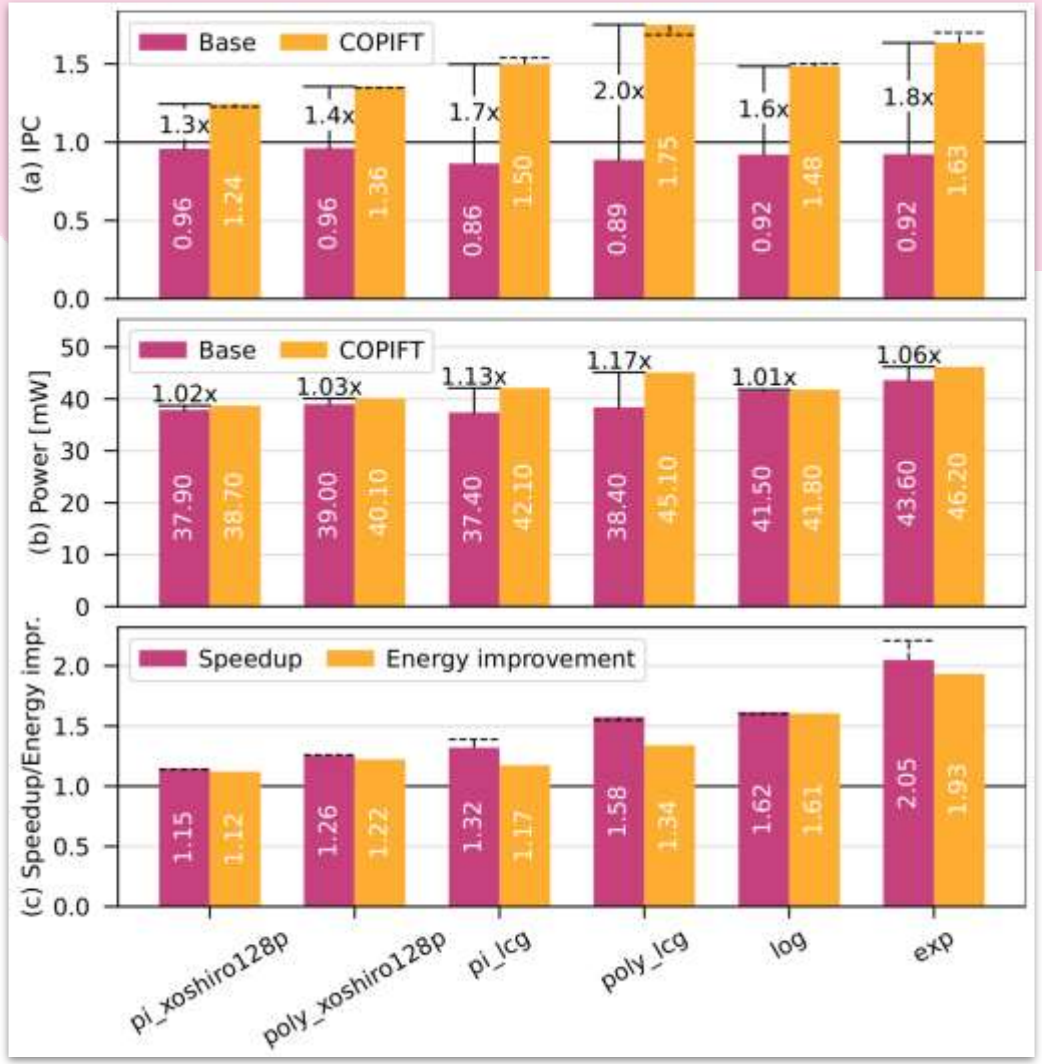
Evaluation

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Results

- Negligible area and timing overheads due to our ISA extensions [1]
- Benchmarks: `expf` and `logf` functions and a few sample Monte Carlo integration kernels
- Geomean **1.47x** speedup, peak **1.75** IPC [2]
- Geomean **1.37x** increase in energy efficiency, peak **1.93x** on `expf` [3]



[1] Implementation using Fusion Compiler in GlobalFoundries 12LP+ FinFET technology, with a 1GHz target clock frequency.
[2] From cycle-accurate RTL simulation in QuestaSim 2023.4.
[3] Switching activities extracted from post-layout simulation and used for power estimation in PrimeTime, assuming typical operating conditions of 25°C and 0.8V.

Conclusion

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Contributions

- We develop **COPIFT**, a **generic** methodology to enable **dual-issue execution** of **mixed-integer-FP** workloads on **energy-efficient cores**
- We implement **COPIFT-accelerated** transcendental function and MC **codes**
- Measuring a geomean **1.47x speedup**, **1.37x energy efficiency** improvement and a peak **IPC** of **1.75**
- Demonstrating that effective **dual-issue execution** of mixed-integer-FP workloads **is possible on** area- and **energy-efficient** in-order **cores**
- All code is **open source** and performance experiments are **reproducible using free software** [3]

[3] https://github.com/colluca/snitch_cluster/tree/copift