Better but still WIP: scheduling for multi-threading with constraint solvers

Timothy Bourke

Inria Paris
École normale supérieure, PSL University

32nd Synchron Workshop – Aussois – 2025-11-25

```
static void stabilizerTask(void* param) {
  stabilizerStep_t stabilizerStep = 1;
  ...
  while(1) {
    sensorsWaitDataReady(); // The sensor should unlock at 1kHz
    sensorsAcquire(&sensorData);
    ...
    stateEstimator(&state, stabilizerStep);
    ...
    controller(&control, &setpoint, &sensorData, &state, stabilizerStep);
    controlMotors(&control);

    if (... && RATE_DO_EXECUTE(usddeckFrequency(), stabilizerStep)) {
      // #define RATE_DO_EXECUTE(RATE_HZ, TICK) ((TICK % (RATE_MAIN_LOOP / RATE_HZ)) == 0)
      usddeckTriggerLogging();
    }
    ...
    stabilizerStep++;
    ...
  }
}
```

E.g., Bitcraze Crazyflie 2.1+

- Periodic (FreeRTOS) task that executes step functions ("runnables")
- "Step" functions write and read variables ("direct" comm. on "labels")
- Functions run in order and terminate within a cycle
- Slower functions are conditionally executed in a specific phase

Airbus project “All-in-Lustre”

Current system

- A periodic task that sequences $\approx 5\,000$ individual (Lustre/Scade) functions communicating over $\approx 120\,000$ variables
- Base period = 5ms.
  Functions at 10ms, 20ms, 40ms, and 120ms.
- Choose phases using Integer Linear Programming (ILP):
  - » load balancing
  - » respect upper bounds on bus access
- Manually assign phases for latency chains

Airbus project "All-in-Lustre"

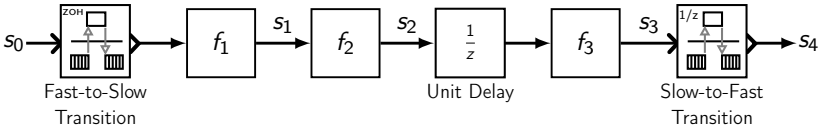
Current system

- A periodic task that sequences $\approx 5\,000$ individual (Lustre/Scade) functions communicating over $\approx 120\,000$ variables
- Base period = 5ms.
  Functions at 10ms, 20ms, 40ms, and 120ms.
- Choose phases using Integer Linear Programming (ILP):
  - » load balancing
  - » respect upper bounds on bus access
- Manually assign phases for latency chains

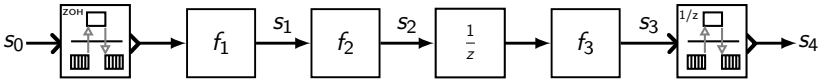
Proposed System

- Specify the whole system as a single Lustre program with new features for specifying periods, resource constraints, and latency constraints
- Like Prelude [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  but with no WCET, no deadlines,
  no real-time tasks
- Rates expressed as $1/n$ of the base clock
- Our contribution:
  - » Formalization of the approach
  - » ILP encoding for end-to-end latency
  - » WiP: multi-task scheduling

Source Language: Rate-Synchronous Lustre

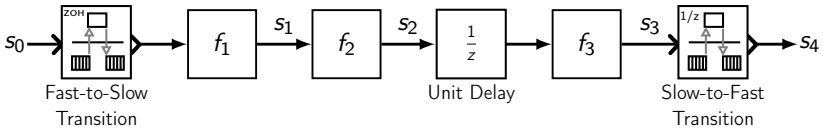


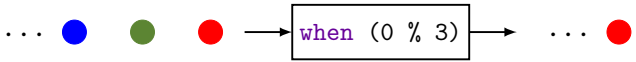
Source Language: Rate-Synchronous Lustre



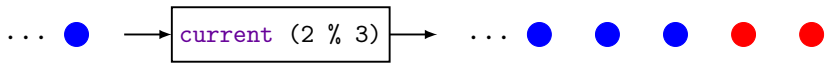

```
s1 = f1(s0 when (0 % 3));
s2 = f2(s1);
s3 = f3(last s2);
s4 = current(s3, (2 % 3));
```

Source Language: Rate-Synchronous Lustre

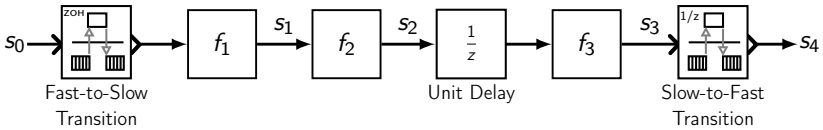





```
s1 = f1(s0 when (0 % 3));
s2 = f2(s1);
s3 = f3(last s2);
s4 = current(s3, (2 % 3));
```

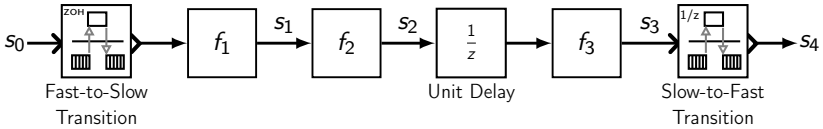


Source Language: Rate-Synchronous Lustre




```
node main(s0 : int) returns (s4 : int)
var s1 : int :: 1/3;
    s2, s3 : int :: 1/3 last = 0;
let
  s1 = f1(s0 when (0 % 3));
  s2 = f2(s1);
  s3 = f3(last s2);
  s4 = current(s3, (2 % 3));

  latency_chain forward <= 4 (s1 -> s2 -> s3);

tel
```

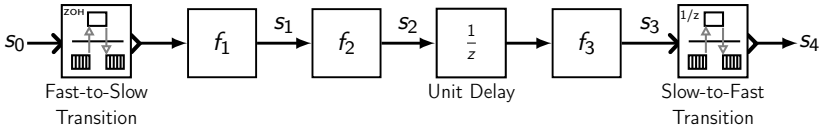
Source Language: Rate-Synchronous Lustre




```
resource cpu : int

node f1(x : int)
returns (y : int)
requires (cpu = 5);

node f2(x : int)
returns (y : int)
requires (cpu = 2);

node f3(x : int)
returns (y : int)
requires (cpu = 2);
```

```
node main(s0 : int) returns (s4 : int)
var s1 : int :: 1/3;
    s2, s3 : int :: 1/3 last = 0;
let
  s1 = f1(s0 when (0 % 3));
  s2 = f2(s1);
  s3 = f3(last s2);
  s4 = current(s3, (2 % 3));

  latency_chain forward <= 4 (s1 -> s2 -> s3);

tel
```

Source Language: Rate-Synchronous Lustre

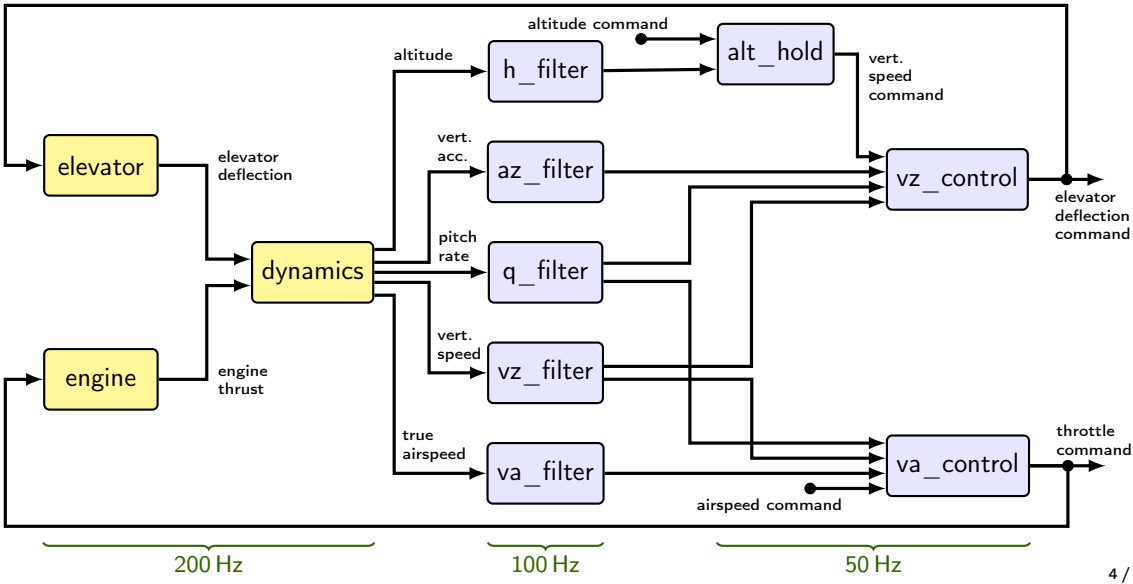



```
resource cpu : int

node f1(x : int)
returns (y : int)
requires (cpu = 5);

node f2(x : int)
returns (y : int)
requires (cpu = 2);

node f3(x : int)
returns (y : int)
requires (cpu = 2);
```

```
node main(s0 : int) returns (s4 : int)
var s1 : int :: 1/3;
    s2, s3 : int :: 1/3 last = 0;
let
  s1 = f1(s0 when (0 % 3));
  s2 = f2(s1);
  s3 = f3(last s2);
  s4 = current(s3, (2 % 3));

  latency_chain forward <= 4 (s1 -> s2 -> s3);
  resource balance cpu;
tel
```

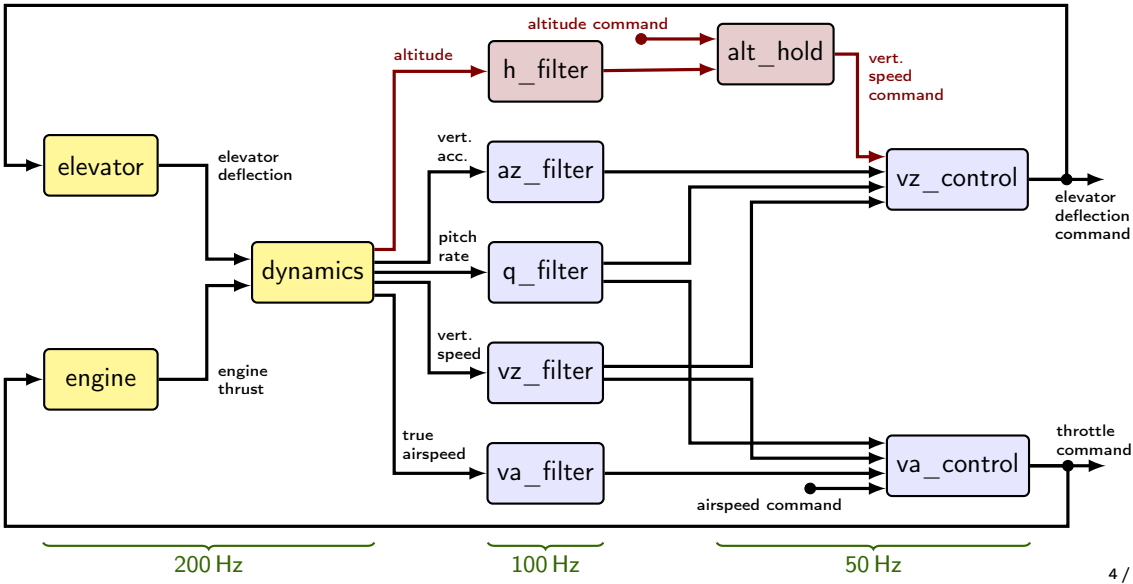
The ROSACE Case Study

[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]

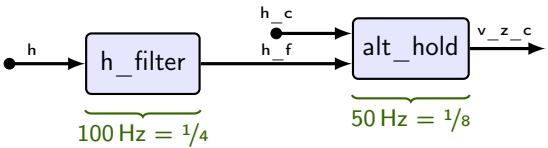


The ROSACE Case Study

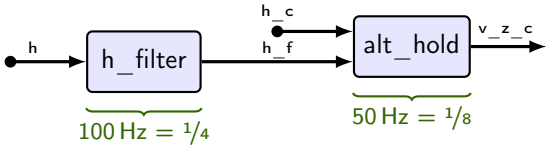
[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]



```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

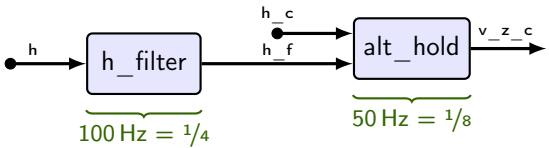


```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

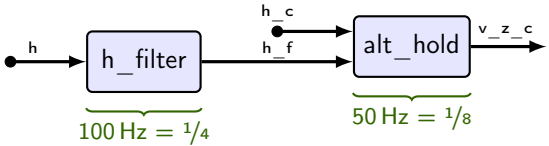
```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```




```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

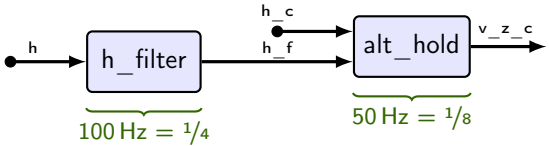


- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



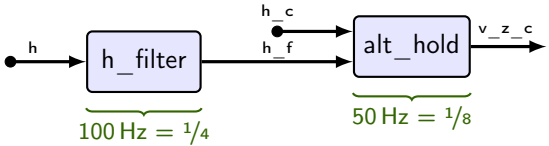
- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

f (concomitance)

Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

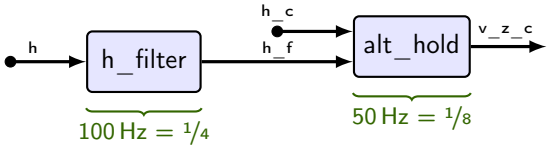


- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

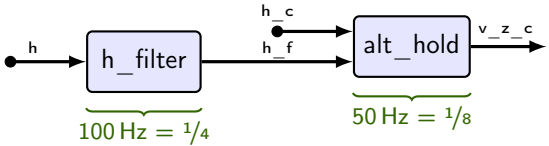


- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Schedule ⇒ Code / Concomitance

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

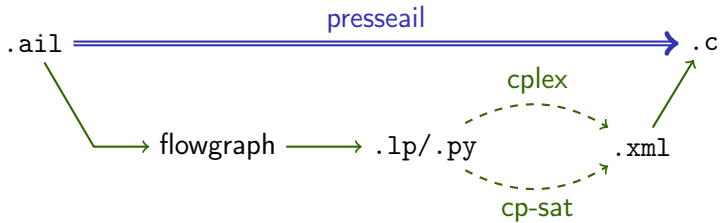


- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold(); // ***
            vz_control();
            break;
    }
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter(); // ***
            ...
        }
    } else {
        ...
    }
    c_30 = (c_30 + 1) % 8;
}
```

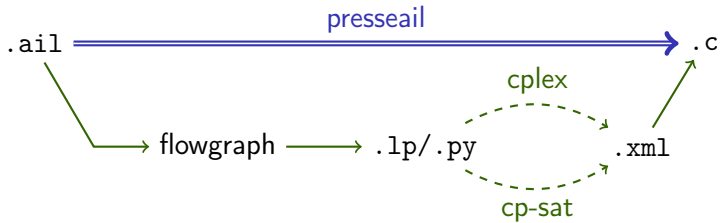
b (concomitance)

Overview: compilation using external solvers



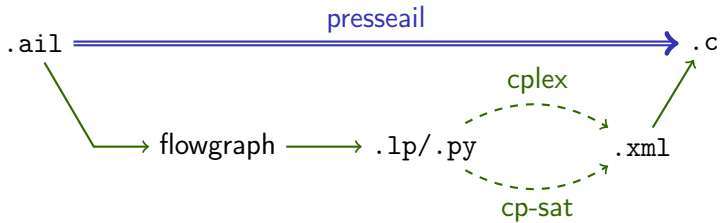
Overview: compilation using external solvers

- Lustre-like source language
- Rates expressed as $1/n$ of the base clock



Overview: compilation using external solvers

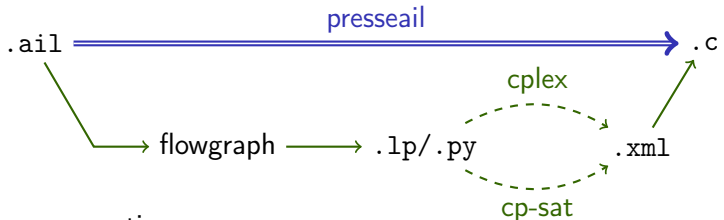
- Lustre-like source language
- Rates expressed as $1/n$ of the base clock
- One or more `step` functions
- Called cyclically at the base rate



Overview: compilation using external solvers

- Lustre-like source language
- Rates expressed as $1/n$ of the base clock

- One or more `step` functions
- Called cyclically at the base rate

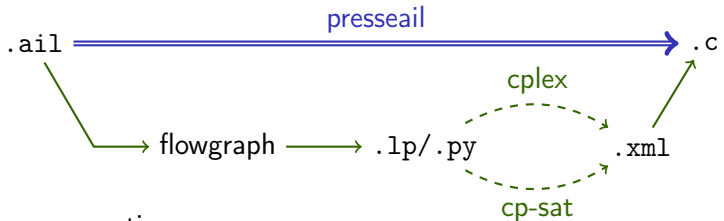


- Vertex = equation
- Arc from producer to consumer
- Independent of source language

Overview: compilation using external solvers

- Lustre-like source language
- Rates expressed as $1/n$ of the base clock

- One or more `step` functions
- Called cyclically at the base rate



- Vertex = equation
- Arc from producer to consumer
- Independent of source language

- Data dependencies
- Resource bounds
- Load balancing
- End-to-end latency

```
Minimize rmax.cycle.ops
Subject to
 depw.wr.dynamics.az_filter_2:  p.az_f - p.dyn >= 0
 depw.wr.az_filter.vz_control_1: p.vz_c - p.az_f >= 0

 pw.def1.az_filter: 0 pw.0.az_f + 1 pw.1.az_f + 2 pw.2.az_f + 3 pw.3.az_f -1 p.az_f = 0
 pw.def0.az_filter:  pw.0.az_f +  pw.1.az_f +  pw.2.az_f +  pw.3.az_f = 1

 rsum.pw.0.ops: rsum.pw.0.ops  - 37 pw.0.vz_f - 88 pw.0.vz_c - 38 pw.0.va_f
             - 90 pw.0.va_c  - 37 pw.0.q_f  - 38 pw.0.h_f  - 82 pw.0.eng
             - 98 pw.0.ele  - 1174 pw.0.dyn - 37 pw.0.az_f - 201 pw.0.alt_h = 0 ...

 rsum.pw.7.ops: rsum.pw.7.ops  - 37 pw.3.vz_f - 88 pw.7.vz_c - 38 pw.3.va_f
             - 90 pw.7.va_c  - 37 pw.3.q_f  - 38 pw.3.h_f  - 82 pw.1.eng
             - 98 pw.1.ele  - 1174 pw.1.dyn - 37 pw.3.az_f - 201 pw.7.alt_h = 0 ...

 rbal.rsum.pw.0.ops_24: rmax.cycle.ops - rsum.pw.0.ops >= 0 ...
 rbal.rsum.pw.7.ops_17: rmax.cycle.ops - rsum.pw.7.ops >= 0 ...
Bounds
 0 <= p.az_f <= 3 ...
General
 p.az_f p.dyn p.vz_c ... rsum.pw.7.ops ... rsum.pw.0.ops rmax.cycle.ops ...
Binary
 pw.0.az_f pw.1.az_f pw.2.az_f pw.3.az_f ...
End
```

```
Minimize rmax.cycle.ops
Subject to
  depw.wr.dynamics.az_filter_2:   p.az_f - p.dyn >= 0
  depw.wr.az_filter.vz_control_1: p.vz_c - p.az_f >= 0

  pw.def1.az_filter: 0 pw.0.az_f + 1 pw.1.az_f + 2 pw.2.az_f + 3 pw.3.az_f -1 p.az_f = 0
  pw.def0.az_filter:   pw.0.az_f +   pw.1.az_f +   pw.2.az_f +   pw.3.az_f = 1

  rsum.pw.0.ops: rsum.pw.0.ops   -  37 pw.0.vz_f - 88 pw.0.vz_c -  38 pw.0.va_f
               - 90 pw.0.va_c  -  37 pw.0.q_f  - 38 pw.0.h_f  -  82 pw.0.eng
               - 98 pw.0.ele   - 1174 pw.0.dyn - 37 pw.0.az_f - 201 pw.0.alt_h = 0 ...

  rsum.pw.7.ops: rsum.pw.7.ops   -  37 pw.3.vz_f - 88 pw.7.vz_c -  38 pw.3.va_f
               - 90 pw.7.va_c  -  37 pw.3.q_f  - 38 pw.3.h_f  -  82 pw.1.eng
               - 98 pw.1.ele   - 1174 pw.1.dyn - 37 pw.3.az_f - 201 pw.7.alt_h = 0 ...

  rbal.rsum.pw.0.ops_24: rmax.cycle.ops - rsum.pw.0.ops >= 0 ...
  rbal.rsum.pw.7.ops_17: rmax.cycle.ops - rsum.pw.7.ops >= 0 ...
Bounds
  0 <= p.az_f <= 3 ...
General
  p.az_f p.dyn p.vz_c ... rsum.pw.7.ops ... rsum.pw.0.ops rmax.cycle.ops ...
Binary
  pw.0.az_f pw.1.az_f pw.2.az_f pw.3.az_f ...
End
```

```
(va, az, q, vz, h) = dynamics(th, d_e);
        az_f = az_filter(az when (? % 2));
```

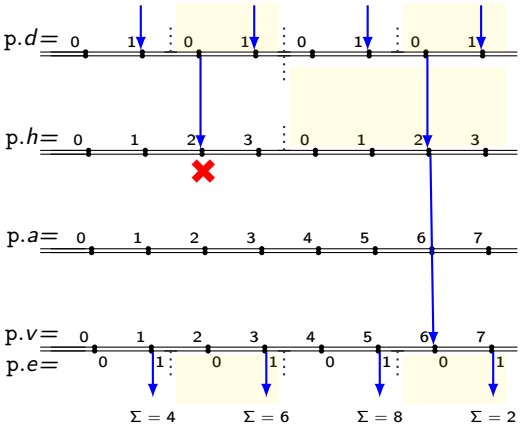
```
Minimize rmax.cycle.ops
Subject to
 depw.wr.dynamics.az_filter_2:   p.az_f - p.dyn >= 0
 depw.wr.az_filter.vz_control_1: p.vz_c - p.az_f >= 0

 pw.def1.az_filter: 0 pw.0.az_f + 1 pw.1.az_f + 2 pw.2.az_f + 3 pw.3.az_f -1 p.az_f = 0
 pw.def0.az_filter:   pw.0.az_f +   pw.1.az_f +   pw.2.az_f +   pw.3.az_f = 1

 rsum.pw.0.ops: rsum.pw.0.ops   -  37 pw.0.vz_f - 88 pw.0.vz_c -  38 pw.0.va_f
             - 90 pw.0.va_c  -  37 pw.0.q_f  - 38 pw.0.h_f  -  82 pw.0.eng
             - 98 pw.0.ele   - 1174 pw.0.dyn - 37 pw.0.az_f - 201 pw.0.alt_h = 0 ...

 rsum.pw.7.ops: rsum.pw.7.ops   -  37 pw.3.vz_f - 88 pw.7.vz_c -  38 pw.3.va_f
             - 90 pw.7.va_c  -  37 pw.3.q_f  - 38 pw.3.h_f  -  82 pw.1.eng
             - 98 pw.1.ele   - 1174 pw.1.dyn - 37 pw.3.az_f - 201 pw.7.alt_h = 0 ...

 rbal.rsum.pw.0.ops_24: rmax.cycle.ops - rsum.pw.0.ops >= 0 ...
 rbal.rsum.pw.7.ops_17: rmax.cycle.ops - rsum.pw.7.ops >= 0 ...
Bounds
 0 <= p.az_f <= 3 ...
General
 p.az_f p.dyn p.vz_c ... rsum.pw.7.ops ... rsum.pw.0.ops rmax.cycle.ops ...
Binary
 pw.0.az_f pw.1.az_f pw.2.az_f pw.3.az_f ...
End
```

```
Minimize rmax.cycle.ops
Subject to
 depw.wr.dynamics.az_filter_2:   p.az_f - p.dyn >= 0
 depw.wr.az_filter.vz_control_1: p.vz_c - p.az_f >= 0

 pw.def1.az_filter: 0 pw.0.az_f + 1 pw.1.az_f + 2 pw.2.az_f + 3 pw.3.az_f -1 p.az_f = 0
 pw.def0.az_filter:   pw.0.az_f +   pw.1.az_f +   pw.2.az_f +   pw.3.az_f = 1

 rsum.pw.0.ops: rsum.pw.0.ops   -  37 pw.0.vz_f - 88 pw.0.vz_c -  38 pw.0.va_f
             - 90 pw.0.va_c  -  37 pw.0.q_f  - 38 pw.0.h_f  -  82 pw.0.eng
             - 98 pw.0.ele   - 1174 pw.0.dyn - 37 pw.0.az_f - 201 pw.0.alt_h = 0 ...

 rsum.pw.7.ops: rsum.pw.7.ops   -  37 pw.3.vz_f - 88 pw.7.vz_c -  38 pw.3.va_f
             - 90 pw.7.va_c  -  37 pw.3.q_f  - 38 pw.3.h_f  -  82 pw.1.eng
             - 98 pw.1.ele   - 1174 pw.1.dyn - 37 pw.3.az_f - 201 pw.7.alt_h = 0 ...

 rbal.rsum.pw.0.ops_24: rmax.cycle.ops - rsum.pw.0.ops >= 0 ...
 rbal.rsum.pw.7.ops_17: rmax.cycle.ops - rsum.pw.7.ops >= 0 ...
Bounds
 0 <= p.az_f <= 3 ...
General
 p.az_f p.dyn p.vz_c ... rsum.pw.7.ops ... rsum.pw.0.ops rmax.cycle.ops ...
Binary
 pw.0.az_f pw.1.az_f pw.2.az_f pw.3.az_f ...
End
```

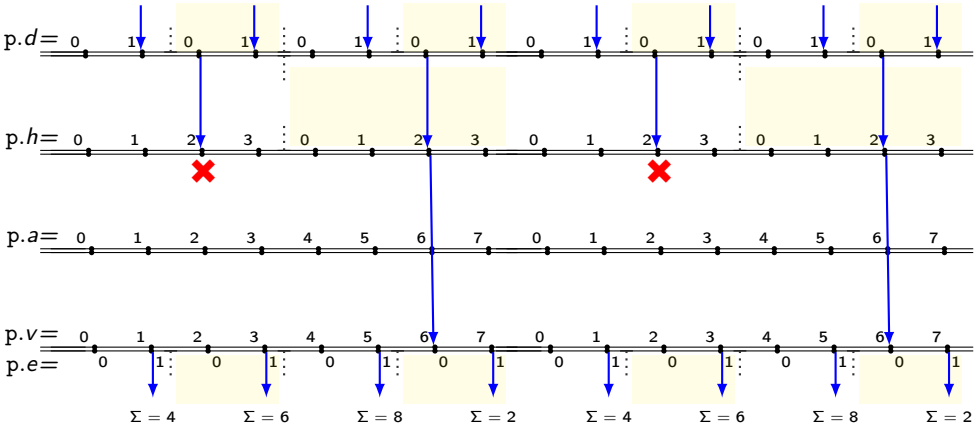
```
node az_filter (az : float) returns (az_f : float)
requires (ops = 37);
```

```
Minimize rmax.cycle.ops
Subject to
 depw.wr.dynamics.az_filter_2:   p.az_f - p.dyn >= 0
 depw.wr.az_filter.vz_control_1: p.vz_c - p.az_f >= 0

 pw.def1.az_filter: 0 pw.0.az_f + 1 pw.1.az_f + 2 pw.2.az_f + 3 pw.3.az_f -1 p.az_f = 0
 pw.def0.az_filter:   pw.0.az_f +   pw.1.az_f +   pw.2.az_f +   pw.3.az_f = 1

 rsum.pw.0.ops: rsum.pw.0.ops   -  37 pw.0.vz_f - 88 pw.0.vz_c -  38 pw.0.va_f
             - 90 pw.0.va_c  -  37 pw.0.q_f  - 38 pw.0.h_f  -  82 pw.0.eng
             - 98 pw.0.ele   - 1174 pw.0.dyn - 37 pw.0.az_f - 201 pw.0.alt_h = 0 ...

 rsum.pw.7.ops: rsum.pw.7.ops   -  37 pw.3.vz_f - 88 pw.7.vz_c -  38 pw.3.va_f
             - 90 pw.7.va_c  -  37 pw.3.q_f  - 38 pw.3.h_f  -  82 pw.1.eng
             - 98 pw.1.ele   - 1174 pw.1.dyn - 37 pw.3.az_f - 201 pw.7.alt_h = 0 ...

 rbal.rsum.pw.0.ops_24: rmax.cycle.ops - rsum.pw.0.ops >= 0 ...
 rbal.rsum.pw.7.ops_17: rmax.cycle.ops - rsum.pw.7.ops >= 0 ...
Bounds
 0 <= p.az_f <= 3 ...                          resource balance ops;
General
 p.az_f p.dyn p.vz_c ... rsum.pw.7.ops ... rsum.pw.0.ops rmax.cycle.ops ...
Binary
 pw.0.az_f pw.1.az_f pw.2.az_f pw.3.az_f ...
End
```

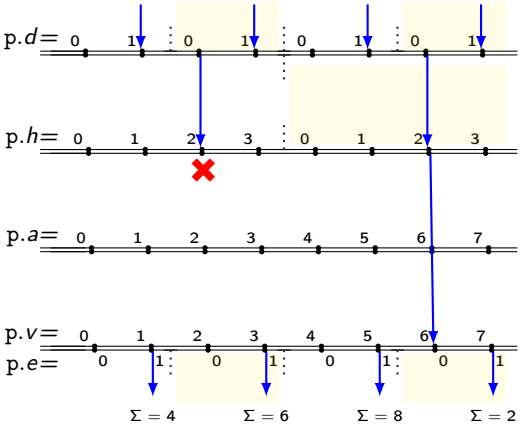
Constraining End-to-end Latency



Constraining End-to-end Latency

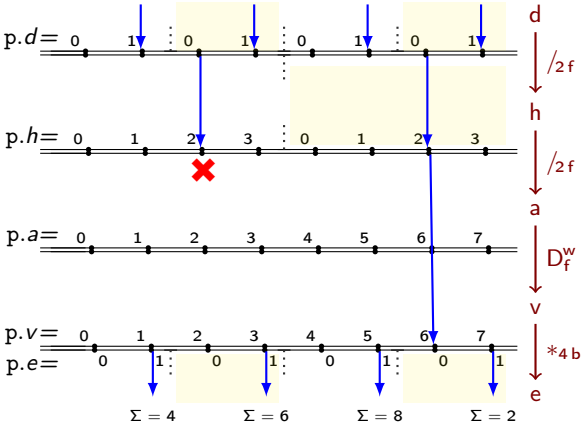


Constraining End-to-end Latency



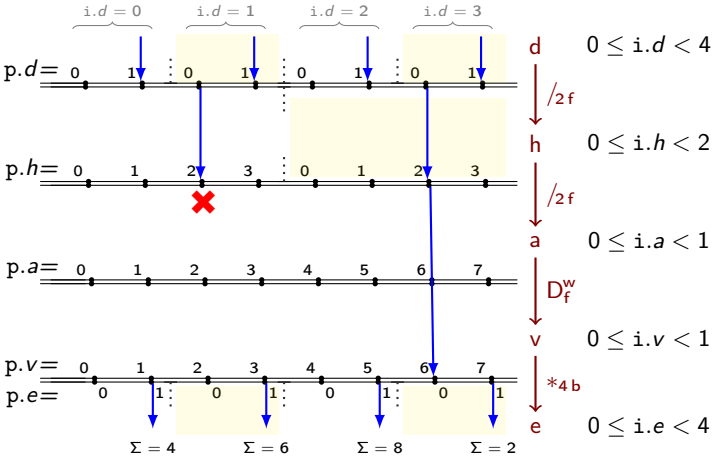
(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency



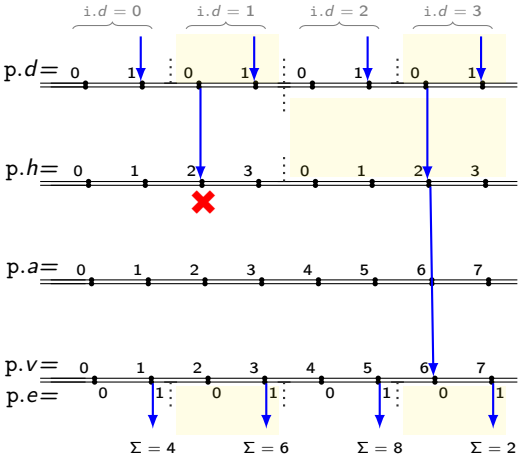
(View online at https://www.tbrk.org/dataflow/showlatency)

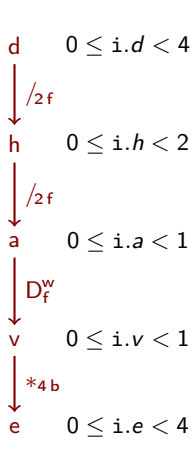
Constraining End-to-end Latency



(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathtt{i}.d < 4$

$0 \le \mathtt{i}.h < 2$

$0 \le \mathtt{i}.a < 1$

$0 \le \mathtt{i}.v < 1$

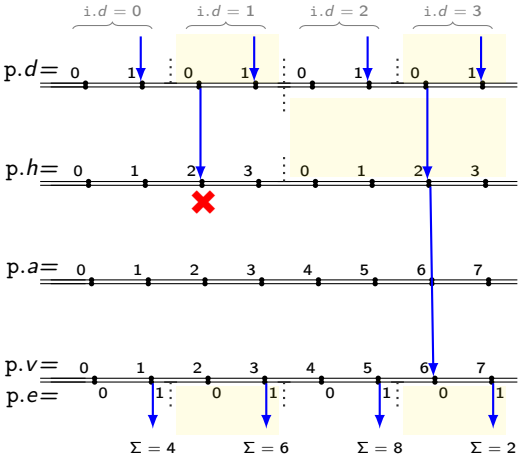
$0 \le \mathtt{i}.e < 4$

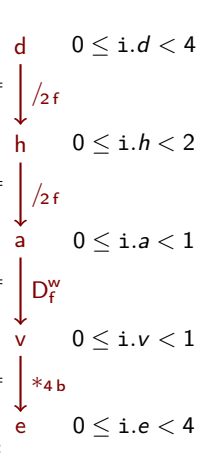
$0 \le \mathtt{lat}_{d,h} < 2$

$2 \cdot \mathtt{i}.d + \mathtt{p}.d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}.h + \mathtt{p}.h$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathtt{i}.d < 4$

$0 \le \mathtt{i}.h < 2$

$0 \le \mathtt{i}.a < 1$

$0 \le \mathtt{i}.v < 1$

$0 \le \mathtt{i}.e < 4$

$0 \le \mathtt{lat}_{d,h} < 2$

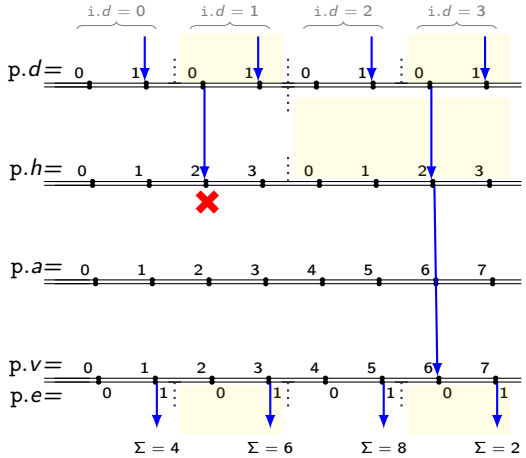
$2 \cdot \mathtt{i}.d + \mathtt{p}.d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}.h + \mathtt{p}.h$

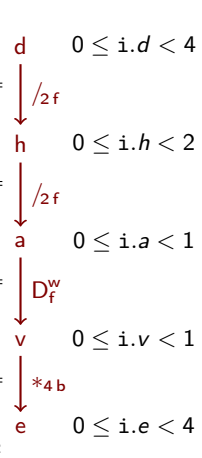
$0 \le \mathtt{lat}_{h,a} < 4$

$4 \cdot \mathtt{i}.h + \mathtt{p}.h + \mathtt{lat}_{h,a} = 8 \cdot \mathtt{i}.a + \mathtt{p}.a$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathtt{i}.d < 4$

$0 \le \mathtt{i}.h < 2$

$0 \le \mathtt{i}.a < 1$

$0 \le \mathtt{i}.v < 1$

$0 \le \mathtt{i}.e < 4$

$0 \le \mathtt{lat}_{d,h} < 2$

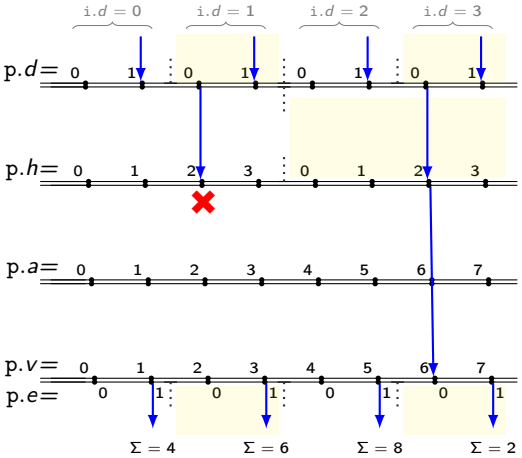
$2 \cdot \mathtt{i}.d + \mathtt{p}.d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}.h + \mathtt{p}.h$

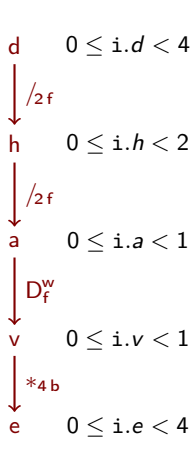
$0 \le \mathtt{lat}_{h,a} < 4$

$4 \cdot \mathtt{i}.h + \mathtt{p}.h + \mathtt{lat}_{h,a} = 8 \cdot \mathtt{i}.a + \mathtt{p}.a$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$$0 \le \mathtt{i}.d < 4$$

$$0 \le \mathtt{lat}_{d,h} < 2$$
$$2 \cdot \mathtt{i}.d + \mathtt{p}.d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}.h + \mathtt{p}.h$$

$$0 \le \mathtt{i}.h < 2$$

$$0 \le \mathtt{lat}_{h,a} < 4$$
$$4 \cdot \mathtt{i}.h + \mathtt{p}.h + \mathtt{lat}_{h,a} = 8 \cdot \mathtt{i}.a + \mathtt{p}.a$$

$$0 \le \mathtt{i}.a < 1$$

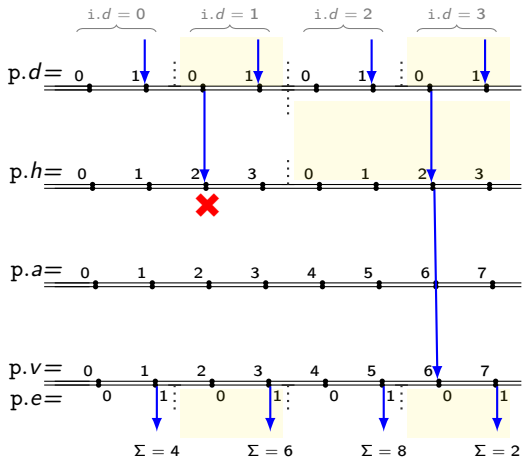
$$0 \le \mathtt{lat}_{a,v} < 8$$
$$8 \cdot \mathtt{i}.a + \mathtt{p}.a + \mathtt{lat}_{a,v} = 8 \cdot \mathtt{i}.v + \mathtt{p}.v$$

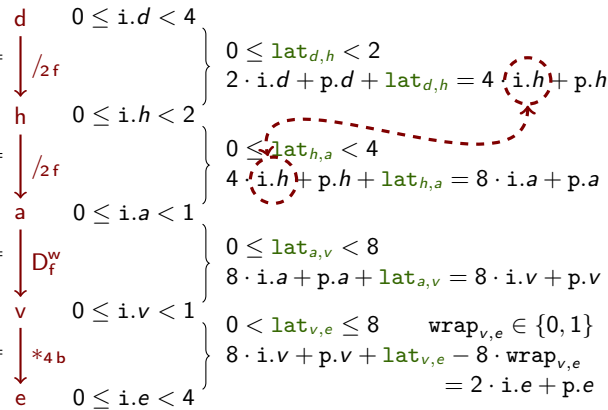
$$0 \le \mathtt{i}.v < 1$$

$$0 \le \mathtt{i}.e < 4$$

(View online at https://www.tbrk.org/dataflow/showlatency)

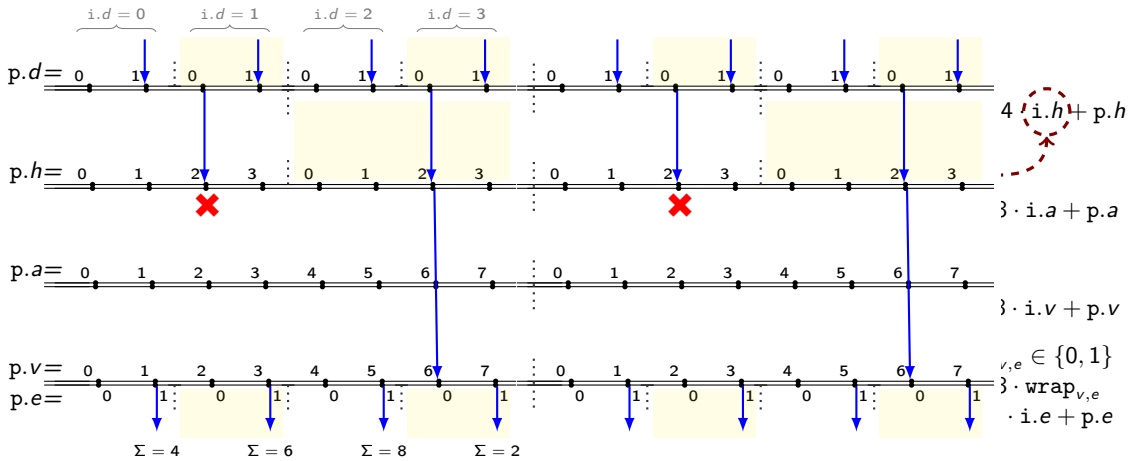
Constraining End-to-end Latency





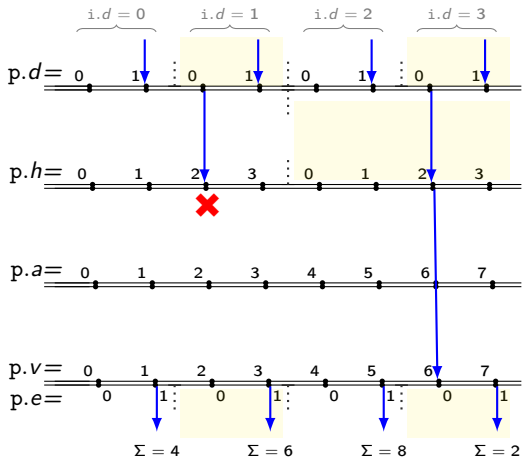
(View online at https://www.tbrk.org/dataflow/showlatency)

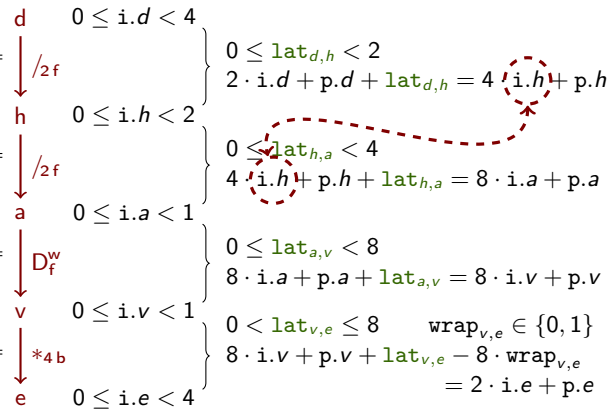
Constraining End-to-end Latency



(View online at https://www.tbrk.org/dataflow/showlatency)

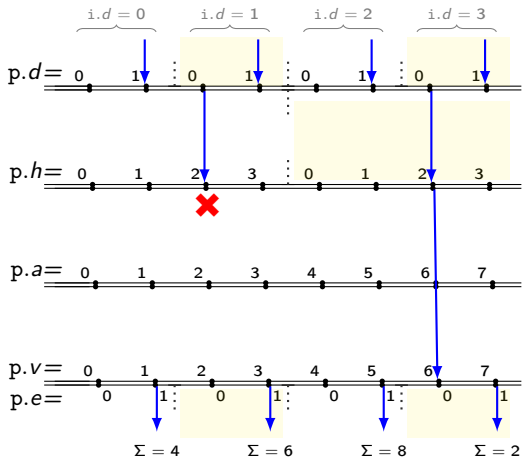
Constraining End-to-end Latency

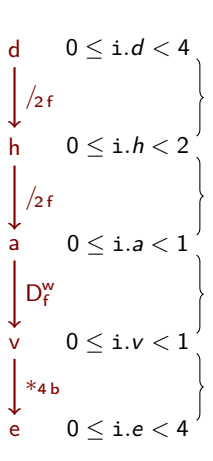




(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$$0 \le \mathtt{i}.d < 4$$
$$0 \le \mathtt{i}.h < 2$$
$$0 \le \mathtt{i}.a < 1$$
$$0 \le \mathtt{i}.v < 1$$
$$0 \le \mathtt{i}.e < 4$$

$$0 \le \mathtt{lat}_{d,h} < 2$$
$$2 \cdot \mathtt{i}.d + \mathtt{p}.d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}.h + \mathtt{p}.h$$

$$0 \le \mathtt{lat}_{h,a} < 4$$
$$4 \cdot \mathtt{i}.h + \mathtt{p}.h + \mathtt{lat}_{h,a} = 8 \cdot \mathtt{i}.a + \mathtt{p}.a$$

$$0 \le \mathtt{lat}_{a,v} < 8$$
$$8 \cdot \mathtt{i}.a + \mathtt{p}.a + \mathtt{lat}_{a,v} = 8 \cdot \mathtt{i}.v + \mathtt{p}.v$$

$$0 < \mathtt{lat}_{v,e} \le 8 \qquad \mathtt{wrap}_{v,e} \in \{0, 1\}$$
$$8 \cdot \mathtt{i}.v + \mathtt{p}.v + \mathtt{lat}_{v,e} - 8 \cdot \mathtt{wrap}_{v,e} = 2 \cdot \mathtt{i}.e + \mathtt{p}.e$$

$$\mathtt{lat}_{d,h} + \mathtt{lat}_{h,a} + \mathtt{lat}_{a,v} + \mathtt{lat}_{v,e} \le 2$$

(View online at https://www.tbrk.org/dataflow/showlatency)

Chains of constraints

`latency forward/backward` $\leq B\ (\ldots, w, r, \ldots)$

Chains of constraints

`latency forward/backward` $\leq B\ (\ldots, w, r, \ldots)$

For each link $w \xrightarrow{s,c} r$,

Chains of constraints

`latency` `forward`/`backward` $\leq B\ (\ldots, w, r, \ldots)$

For each link $w \xrightarrow{s,c} r$,

$$0 \leq \mathtt{i}.r < {}^{hp}/_{\mathrm{period}(r)}$$

$$\begin{array}{l} 0 \leq \mathtt{lat}_{w,r} < L \quad \text{for } c = \mathsf{f} \\ 0 < \mathtt{lat}_{w,r} \leq L \quad \text{for } c = \mathsf{b} \end{array} \quad \left\{ \begin{array}{rl} \text{where} & L = \mathrm{period}(r) \text{ if } \texttt{forward} \\ \text{and} & L = \mathrm{period}(w) \text{ if } \texttt{backward} \end{array} \right.$$

$$0 \leq \mathtt{wrap}_{w,r} \leq 1 \qquad \begin{array}{l} \text{if } \texttt{forward} \text{ and } s \notin \{\mathsf{D}^{\mathsf{w}}, *_n\} \\ \text{or if } \texttt{backward} \text{ and } s \notin \{\mathsf{D}^{\mathsf{w}}, /_n\} \end{array}$$

$$\mathrm{period}(w) \cdot \mathtt{i}.w + \mathtt{p}.w + \mathtt{lat}_{w,r} - hp \cdot \mathtt{wrap}_{w,r} = \mathrm{period}(r) \cdot \mathtt{i}.r + \mathtt{p}.r.$$

- Each intermediate instance is both a reader and a writer, and thus constrained by two equations.

- `forward`: attach chains from the top
- `backward`: attach chains to the bottom

WiP: multi-task scheduling

Current Approach

1. Generate ILP: equation $\mapsto$ phase
2. Compile to a sequential task (or parallelize each phase [Carle, Potop-Butucaru, Sorel, and Lesens (2015): From Dataflow Specification to Multiprocessor Partitioned Time-Triggered Real-Time Implementation])

WiP: multi-task scheduling

Current Approach

1. Generate ILP: equation $\mapsto$ phase
2. Compile to a sequential task (or parallelize each phase [Carle, Potop-Butucaru, Sorel, and Lesens (2015): From Dataflow Specification to Multiprocessor Partitioned Time-Triggered Real-Time Implementation])

Why not?...

1. Generate ILP: equation $\mapsto$ (thread, phase)
2. Compile to two (or more) sequential tasks

WiP: multi-task scheduling

Current Approach

1. Generate ILP: equation $\mapsto$ phase
2. Compile to a sequential task (or parallelize each phase [Carle, Potop-Butucaru, Sorel, and Lesens (2015): From Dataflow Specification to Multiprocessor Partitioned Time-Triggered Real-Time Implementation])

Why not?...

1. Generate ILP: equation $\mapsto$ (thread, phase)
2. Compile to two (or more) sequential tasks

...because...

- the number of constraints explodes and the ILP solver may not be able to find a solution?
- delayed communications may accumulate and increase end-to-end latency?

WiP: multi-task scheduling

Current Approach

1. Generate ILP: equation $\mapsto$ phase
2. Compile to a sequential task (or parallelize each phase [Carle, Potop-Butucaru, Sorel, and Lesens (2015): From Dataflow Specification to Multiprocessor Partitioned Time-Triggered Real-Time Implementation])

Why not?...

1. Generate ILP: equation $\mapsto$ (thread, phase)
2. Compile to two (or more) sequential tasks

...because...

- the number of constraints explodes and the ILP solver may not be able to find a solution?
- delayed communications may accumulate and increase end-to-end latency?

...Try anyway!

- We've got a big hammer, why not use it?
- Maybe it works for our applications
- The problem is interesting
- Still need to compare to list scheduling

Threads and phases

Source program: `w = e;  r = f(w);`

$(t_w \neq t_r) \wedge (p_w = p_r)$: extra synchronization required

thread 1	thread 2
...	...
if (c % 2 == 0) { w = e; sem_post(wok); }	if (c % 2) == 0) { sem_wait(wok); r = f(w); }
...	...

Threads and phases

Source program: `w = e;  r = f(w);`

$(t_w \neq t_r) \wedge (p_w = p_r)$: extra synchronization required

thread 1	thread 2
...	...
if (c % 2 == 0) { w = e; sem_post(wok); }	if (c % 2) == 0) { sem_wait(wok); r = f(w); }
...	...

$(p_w \neq p_r)$: no race condition due to global synchronization

thread 1	thread 2
...	...
if (c % 2 == 0) { w = e; }	if (c % 2) == 1) { r = f(w); }
...	...

Threads and phases

Source program: `w = e;  r = f(w);`

$(t_w \neq t_r) \wedge (p_w = p_r)$: extra synchronization required

thread 1	thread 2
...	...
if (c % 2 == 0) { w = e; sem_post(wok); }	if (c % 2) == 0) { sem_wait(wok); r = f(w); }
...	...

$(p_w \neq p_r)$: no race condition due to global synchronization

thread 1	thread 2
...	...
if (c % 2 == 0) { w = e; }	if (c % 2) == 1) { r = f(w); }
...	...

require: $t_w = t_r \vee p_w \neq p_r$ "same thread or different phase"

Preliminary Experiments on largest Airbus case-study

- Oct. 2024: It does not work
  - » cplex: runs for days until out-of-memory
  - » `set mip strategy file 3`: runs for longer until dying

Preliminary Experiments on largest Airbus case-study

- Oct. 2024: It does not work
  - » cplex: runs for days until out-of-memory
  - » `set mip strategy file 3`: runs for longer until dying
- Nov. 2024: It works a little bit
  - » Schedule on one thread and generate a *MIP start file*
  - » Scheduling on two cores then starts from a valid solution, which the optimizer can iteratively improve.
  - » Solution after ≈35 hours

Preliminary Experiments on largest Airbus case-study

- Oct. 2024: It does not work
  - » cplex: runs for days until out-of-memory
  - » `set mip strategy file 3`: runs for longer until dying
- Nov. 2024: It works a little bit
  - » Schedule on one thread and generate a *MIP start file*
  - » Scheduling on two cores then starts from a valid solution, which the optimizer can iteratively improve.
  - » Solution after ≈35 hours
- Mar. 2025: "Boolean" encoding for same-thread or different-phase
  - » cplex: 2 hours
  - » cp-sat (with L. Sylvestre): 20 minutes
- Ongoing: Pure SAT encoding (with L. Sylvestre)
  - » Translate pseudo-boolean constraints following Eén and Sörensso (suggested by K. Claessen and M. Sheeran)
  - » Re-encode end-to-end latency constraints

```
Minimize rmax.threadandcycle.ops
Subject to
  tpw.def1.az_f: -1 p.az_f + 1 tw.0.pw.1.az_f + 2 tw.0.pw.2.az_f + 3 tw.0.pw.3.az_f
                          + 1 tw.1.pw.1.az_f + 2 tw.1.pw.2.az_f + 3 tw.1.pw.3.az_f  = 0

  tpw.def0.az_f:  tw.0.pw.0.az_f + tw.0.pw.1.az_f + tw.0.pw.2.az_f + tw.0.pw.3.az_f
                + tw.1.pw.0.az_f + tw.1.pw.1.az_f + tw.1.pw.2.az_f + tw.1.pw.3.az_f = 1 ...

  threads.tw.0.pw.1.dyn.az_f: tw.0.pw.1.dyn + tw.1.pw.1.az_f + tw.1.pw.3.az_f <= 1
  threads.tw.0.pw.0.dyn.az_f: tw.0.pw.0.dyn + tw.1.pw.0.az_f + tw.1.pw.2.az_f <= 1

  threads.tw.1.pw.1.dyn.az_f: tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f <= 1
  threads.tw.1.pw.0.dyn.az_f: tw.1.pw.0.dyn + tw.0.pw.0.az_f + tw.0.pw.2.az_f <= 1 ...

  rsum.tw.1.pw.7.ops: rsum.tw.1.pw.7.ops  - 37 tw.1.pw.3.vz_f - 88 tw.1.pw.7.vz_c
                    - 38 tw.1.pw.3.va_f - 90 tw.1.pw.7.va_c - 37 tw.1.pw.3.q_f
                    - 38 tw.1.pw.3.h_f   - 82 tw.1.pw.1.eng  - 98 tw.1.pw.1.ele
                    - 1174 tw.1.pw.1.dyn - 37 tw.1.pw.3.az_f - 201 tw.1.pw.7.alt_h = 0 ...
Bounds
  0 <= p.az_f <= 3 ...
General
  p.az_f ... rsum.tw.0.pw.0.ops ... rsum.tw.1.pw.7.ops rmax.threadandcycle.ops ...
Binary
  tw.0.pw.0.az_f tw.0.pw.1.az_f tw.0.pw.2.az_f tw.0.pw.3.az_f
  tw.1.pw.0.az_f tw.1.pw.1.az_f tw.1.pw.2.az_f tw.1.pw.3.az_f ...
End
```

```
Minimize rmax.threadandcycle.ops
Subject to
 tpw.def1.az_f: -1 p.az_f + 1 tw.0.pw.1.az_f + 2 tw.0.pw.2.az_f + 3 tw.0.pw.3.az_f
                          + 1 tw.1.pw.1.az_f + 2 tw.1.pw.2.az_f + 3 tw.1.pw.3.az_f  = 0

 tpw.def0.az_f:  tw.0.pw.0.az_f + tw.0.pw.1.az_f + tw.0.pw.2.az_f + tw.0.pw.3.az_f
               + tw.1.pw.0.az_f + tw.1.pw.1.az_f + tw.1.pw.2.az_f + tw.1.pw.3.az_f = 1 ...

 threads.tw.0.pw.1.dyn.az_f: tw.0.pw.1.dyn + tw.1.pw.1.az_f + tw.1.pw.3.az_f <= 1
 threads.tw.0.pw.0.dyn.az_f: tw.0.pw.0.dyn + tw.1.pw.0.az_f + tw.1.pw.2.az_f <= 1

 threads.tw.1.pw.1.dyn.az_f: tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f <= 1
 threads.tw.1.pw.0.dyn.az_f: tw.1.pw.0.dyn + tw.0.pw.0.az_f + tw.0.pw.2.az_f <= 1 ...

 rsum.tw.1.pw.7.ops: rsum.tw.1.pw.7.ops  - 37 tw.1.pw.3.vz_f - 88 tw.1.pw.7.vz_c
                   - 38 tw.1.pw.3.va_f - 90 tw.1.pw.7.va_c - 37 tw.1.pw.3.q_f
                   - 38 tw.1.pw.3.h_f   - 82 tw.1.pw.1.eng  - 98 tw.1.pw.1.ele
                   - 1174 tw.1.pw.1.dyn - 37 tw.1.pw.3.az_f - 201 tw.1.pw.7.alt_h = 0 ...
Bounds
 0 <= p.az_f <= 3 ...
General
 p.az_f ... rsum.tw.0.pw.0.ops ... rsum.tw.1.pw.7.ops rmax.threadandcycle.ops ...
Binary
 tw.0.pw.0.az_f tw.0.pw.1.az_f tw.0.pw.2.az_f tw.0.pw.3.az_f
 tw.1.pw.0.az_f tw.1.pw.1.az_f tw.1.pw.2.az_f tw.1.pw.3.az_f ...
End
```

```
Minimize rmax.threadandcycle.ops
Subject to
  tpw.def1.az_f: -1 p.az_f + 1 tw.0.pw.1.az_f + 2 tw.0.pw.2.az_f + 3 tw.0.pw.3.az_f
                           + 1 tw.1.pw.1.az_f + 2 tw.1.pw.2.az_f + 3 tw.1.pw.3.az_f  = 0

  tpw.def0.az_f:   tw.0.pw.0.az_f + tw.0.pw.1.az_f + tw.0.pw.2.az_f + tw.0.pw.3.az_f
                 + tw.1.pw.0.az_f + tw.1.pw.1.az_f + tw.1.pw.2.az_f + tw.1.pw.3.az_f = 1 ...

  threads.tw.0.pw.1.dyn.az_f: tw.0.pw.1.dyn + tw.1.pw.1.az_f + tw.1.pw.3.az_f <= 1
  threads.tw.0.pw.0.dyn.az_f: tw.0.pw.0.dyn + tw.1.pw.0.az_f + tw.1.pw.2.az_f <= 1

  threads.tw.1.pw.1.dyn.az_f: tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f <= 1
  threads.tw.1.pw.0.dyn.az_f: tw.1.pw.0.dyn + tw.0.pw.0.az_f + tw.0.pw.2.az_f <= 1 ...

  rsum.tw.1.pw.7.ops: rsum.tw.1.pw.7.ops  - 37 tw.1.pw.3.vz_f - 88 tw.1.pw.7.vz_c
                    - 38 tw.1.pw.3.va_f - 90 tw.1.pw.7.va_c - 37 tw.1.pw.3.q_f
                    - 38 tw.1.pw.3.h_f   - 82 tw.1.pw.1.eng  - 98 tw.1.pw.1.ele
                    - 1174 tw.1.pw.1.dyn - 37 tw.1.pw.3.az_f - 201 tw.1.pw.7.alt_h = 0 ...
Bounds
  0 <= p.az_f <= 3 ...
General
  p.az_f ... rsum.tw.0.pw.0.ops ... rsum.tw.1.pw.7.ops rmax.threadandcycle.ops ...
Binary
  tw.0.pw.0.az_f tw.0.pw.1.az_f tw.0.pw.2.az_f tw.0.pw.3.az_f
  tw.1.pw.0.az_f tw.1.pw.1.az_f tw.1.pw.2.az_f tw.1.pw.3.az_f ...
End
```

```
(va, az, q, vz, h) = dynamics(th, d_e);
az_f = az_filter(az when (? % 2));
```

```
Minimize rmax.threadandcycle.ops
Subject to
  tpw.def1.az_f: -1 p.az_f + 1 tw.0.pw.1.az_f + 2 tw.0.pw.2.az_f + 3 tw.0.pw.3.az_f
                          + 1 tw.1.pw.1.az_f + 2 tw.1.pw.2.az_f + 3 tw.1.pw.3.az_f  = 0

  tpw.def0.az_f:   tw.0.pw.0.az_f + tw.0.pw.1.az_f + tw.0.pw.2.az_f + tw.0.pw.3.az_f
                 + tw.1.pw.0.az_f + tw.1.pw.1.az_f + tw.1.pw.2.az_f + tw.1.pw.3.az_f = 1 ...

  threads.tw.0.pw.1.dyn.az_f: tw.0.pw.1.dyn + tw.1.pw.1.az_f + tw.1.pw.3.az_f <= 1
  threads.tw.0.pw.0.dyn.az_f: tw.0.pw.0.dyn + tw.1.pw.0.az_f + tw.1.pw.2.az_f <= 1

  threads.tw.1.pw.1.dyn.az_f: tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f <= 1
  threads.tw.1.pw.0.dyn.az_f: tw.1.pw.0.dyn + tw.0.pw.0.az_f + tw.0.pw.2.az_f <= 1 ...

  rsum.tw.1.pw.7.ops: rsum.tw.1.pw.7.ops  - 37 tw.1.pw.3.vz_f - 88 tw.1.pw.7.vz_c
                    - 38 tw.1.pw.3.va_f - 90 tw.1.pw.7.va_c - 37 tw.1.pw.3.q_f
                    - 38 tw.1.pw.3.h_f   - 82 tw.1.pw.1.eng  - 98 tw.1.pw.1.ele
                    - 1174 tw.1.pw.1.dyn - 37 tw.1.pw.3.az_f - 201 tw.1.pw.7.alt_h = 0 ...
Bounds
  0 <= p.az_f <= 3 ...
General
  p.az_f ... rsum.tw.0.pw.0.ops ... rsum.tw.1.pw.7.ops rmax.threadandcycle.ops ...
Binary
  tw.0.pw.0.az_f tw.0.pw.1.az_f tw.0.pw.2.az_f tw.0.pw.3.az_f
  tw.1.pw.0.az_f tw.1.pw.1.az_f tw.1.pw.2.az_f tw.1.pw.3.az_f ...
End
```

```
node az_filter (az : float) returns (az_f : float)
requires (ops = 37);
```

```
Minimize rmax.threadandcycle.ops
Subject to
 tpw.def1.az_f: -1 p.az_f + 1 tw.0.pw.1.az_f + 2 tw.0.pw.2.az_f + 3 tw.0.pw.3.az_f
                          + 1 tw.1.pw.1.az_f + 2 tw.1.pw.2.az_f + 3 tw.1.pw.3.az_f  = 0

 tpw.def0.az_f:   tw.0.pw.0.az_f + tw.0.pw.1.az_f + tw.0.pw.2.az_f + tw.0.pw.3.az_f
                + tw.1.pw.0.az_f + tw.1.pw.1.az_f + tw.1.pw.2.az_f + tw.1.pw.3.az_f = 1 ...

 threads.tw.0.pw.1.dyn.az_f: tw.0.pw.1.dyn + tw.1.pw.1.az_f + tw.1.pw.3.az_f <= 1
 threads.tw.0.pw.0.dyn.az_f: tw.0.pw.0.dyn + tw.1.pw.0.az_f + tw.1.pw.2.az_f <= 1

 threads.tw.1.pw.1.dyn.az_f: tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f <= 1
 threads.tw.1.pw.0.dyn.az_f: tw.1.pw.0.dyn + tw.0.pw.0.az_f + tw.0.pw.2.az_f <= 1 ...

 rsum.tw.1.pw.7.ops: rsum.tw.1.pw.7.ops  - 37 tw.1.pw.3.vz_f - 88 tw.1.pw.7.vz_c
                   - 38 tw.1.pw.3.va_f - 90 tw.1.pw.7.va_c - 37 tw.1.pw.3.q_f
                   - 38 tw.1.pw.3.h_f   - 82 tw.1.pw.1.eng  - 98 tw.1.pw.1.ele
                   - 1174 tw.1.pw.1.dyn - 37 tw.1.pw.3.az_f - 201 tw.1.pw.7.alt_h = 0 ...
Bounds
 0 <= p.az_f <= 3 ...
General
 p.az_f ... rsum.tw.0.pw.0.ops ... rsum.tw.1.pw.7.ops rmax.threadandcycle.ops ...
Binary
 tw.0.pw.0.az_f tw.0.pw.1.az_f tw.0.pw.2.az_f tw.0.pw.3.az_f
 tw.1.pw.0.az_f tw.1.pw.1.az_f tw.1.pw.2.az_f tw.1.pw.3.az_f ...
End
```

Preliminary Results on the ≈5000 function example

- `cplex` runs for days until an out-of-memory error

Preliminary Results on the ≈5000 function example

- `cplex` runs for days until an out-of-memory error
  - » Schedule with `-nthreads 1` and generate a **MIP start file (.mst)**
  - » Rescheduling with `-nthreads 2` then starts from a valid solution, which the optimizer iteratively improves over ≈1 hour.

Preliminary Results on the ≈5000 function example

- `cplex` runs for days until an out-of-memory error
  - » Schedule with `-nthreads 1` and generate a **MIP start file (.mst)**
  - » Rescheduling with `-nthreads 2` then starts from a valid solution, which the optimizer iteratively improves over ≈1 hour.

- With L. Sylvestre, we added new back-ends
  - » `cp-sat`, translate .lp: ≈15 minutes (3 threads: ≈8 hours)
  - » `cp-sat`, all booleans: ≈25 minutes
  - » Z3 (single core): fails (so far)
  - » `minisat+`to expand the pseudo-boolean constraints: fails (so far)

Preliminary Results on the ≈5000 function example

- `cplex` runs for days until an out-of-memory error
  - » Schedule with `-nthreads 1` and generate a **MIP start file (.mst)**
  - » Rescheduling with `-nthreads 2` then starts from a valid solution, which the optimizer iteratively improves over ≈1 hour.

- With L. Sylvestre, we added new back-ends
  - » `cp-sat`, translate .lp: ≈15 minutes (3 threads: ≈8 hours)
  - » `cp-sat`, all booleans: ≈25 minutes
  - » Z3 (single core): fails (so far)
  - » `minisat+`to expand the pseudo-boolean constraints: fails (so far)

- What about `--nthreads 4`? Or a bigger application?
- More tricks needed (schedule for 2, then reschedule for 4, etc.) or completely doomed?

Same thread or different phase: what about fast functions?

- For functions at rate $1/n$ when $1 < n$, the $t_w = t_r \vee p_w \neq p_r$ rule works well enough
- But what about for $n = 1$?

Same thread or different phase: what about fast functions?

- For functions at rate $1/n$ when $1 < n$, the $t_w = t_r \vee p_w \neq p_r$ rule works well enough
- But what about for $n = 1$?
- Simple solution: multiply all rates by 2, giving $1/2n$
  - » Each cycle, every thread executes: `step(); barrier(); step()`
  - » Breaks the `--fast-first` rule (faster functions before slower ones)

Same thread or different phase: what about fast functions?

- For functions at rate $1/n$ when $1 < n$, the $t_w = t_r \vee p_w \neq p_r$ rule works well enough
- But what about for $n = 1$?
- Simple solution: multiply all rates by 2, giving $1/2n$
  - » Each cycle, every thread executes: `step(); barrier(); step()`
  - » Breaks the `--fast-first` rule (faster functions before slower ones)
- Another solution: add the barrier within `step()`
  - » Schedule $n = 1$ functions ante-barrier or post-barrier
  - » New rule: $t_w = t_r \vee$ (if $n > 1$ then $p_w \neq p_r$ else $b_w \neq b_r$)
  - » Even more variables and constraints
  - » Need to balance resources used in ante-barrier and in the whole cycle

Ongoing work

- Still refining the SAT backend
  - » boolean encoding of end-to-end latency
  - » treatment of pseudo-boolean constraints for resource balancing
- Difference between WCET used for balancing and WCET of generated code
- Need robust measures for case-studies and more benchmarks
- Use a real-time scheduling algorithm to find a solution or an initial solution?

References I

- Carle, T., D. Potop-Butucaru, Y. Sorel, and D. Lesens (Nov. 2015). "From Dataflow Specification to Multiprocessor Partitioned Time-Triggered Real-Time Implementation". In: *Leibniz Trans. Embedded Systems (LITES)* 2.2, 01:1–01:30.
- Forget, J., F. Boniol, D. Lesens, and C. Pagetti (Mar. 2010). "A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems". In: *Proc. 25th ACM Symp. Applied Computing (SAC'10)*. Ed. by S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C.-C. Hung. Sierre, Switzerland: ACM, pp. 527–534.
- Pagetti, C., D. Saussié, R. Gratia, E. Noulard, and P. Siron (Apr. 2014). "The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution". In: *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2014)*. IEEE. Berlin, Germany, pp. 309–318.

Tricks that do not seem to help: 1/2

Minimize Races

- Rather than prohibit different-thread/same-phase communications, try to minimize them.
- For each writer-reader: add a binary variable `sync.dyn.az_f`
- Include it in the previous constraints:

  ```
  tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f - sync.dyn.az_f <= 1
  ```

- Add an additional objective to minimize the sum of such variables

Tricks that do not seem to help: 1/2

Minimize Races

- Rather than prohibit different-thread/same-phase communications, try to minimize them.
- For each writer-reader: add a binary variable `sync.dyn.az_f`
- Include it in the previous constraints:

  `tw.1.pw.1.dyn + tw.0.pw.1.az_f + tw.0.pw.3.az_f - sync.dyn.az_f <= 1`
- Add an additional objective to minimize the sum of such variables
- Results: load balances quite quickly (objective 1),
  then very, very slowly converges on eliminating races (objective 2).
- Requires adding synchronizations within a phase.

Tricks that do not seem to help: 2/2

Clustering prior to ILP

- Many partitions: does not seem very effective
- Few partitions: precludes finding a valid schedule
  - » E.g., with four partitions, we can rapidly test all combinations for feasability: 1100, 0110, 1010

Tricks that do not seem to help: 2/2

Clustering prior to ILP

- Many partitions: does not seem very effective
- Few partitions: precludes finding a valid schedule
  - » E.g., with four partitions, we can rapidly test all combinations for feasability: 1100, 0110, 1010
- Try tweaking the heuristic?
- Cluster in a different way?
- Try cplex conflict analysis features?