STMicroelectronics
LIG
University of Grenoble

Interactive Debugging of Dynamic Dataflow Embedded Applications.

Kevin Pouget, Patricia Lopez Cueva,
Miguel Santana, Jean-François Méhaut

HIPS'13, Boston, USA — May 20th 2013

Technological Context

Embedded System Development

- High-resolution multimedia app. ⇒ high performance expectations.
  - H.265 HEVC
  - Augmented reality,
  - 4K digital television
  - ...
- Sharp time-to-market constraints

⇒ Important demand for
  - Powerful parallel architectures
    - MultiProcessor on Chip (MPSoC)
  - Convenient programming methodologies
    - Dynamic dataflow programming
  - Efficient verification and validation tools
    - Our research contribution

life.augmented

Technological Context

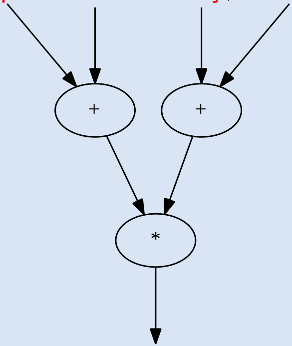
MultiProcessor on Chip (MPSoC)

- Parallel architecture
  - More difficult to program
- Maybe heterogeneous
  - Application-specific processors,
  - Hardware accelerators,
  - GPU-like architecture (OS-less processors)
- Embedded system
  - Constrained environment,
  - On-board debugging complicated
    $\rightarrow$ performance debugging only
  - Limited-scale functional debugging on simulators

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

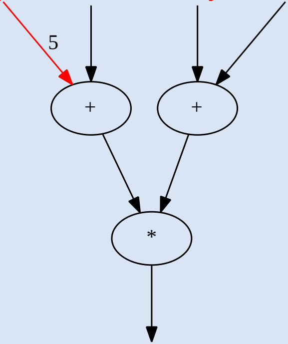
⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

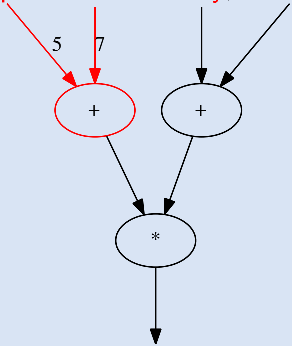
⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

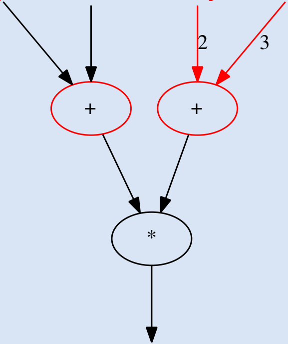
⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model ($\leftrightarrow$ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



$\Rightarrow$ Inherently parallel ("2 PCs" here)

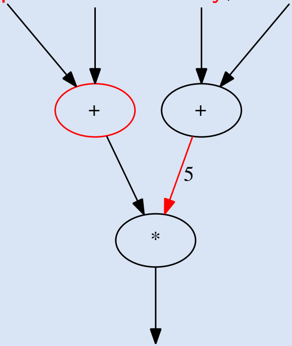
$\Rightarrow$ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

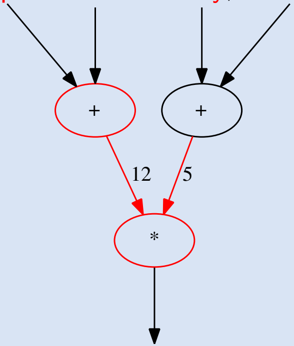
⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

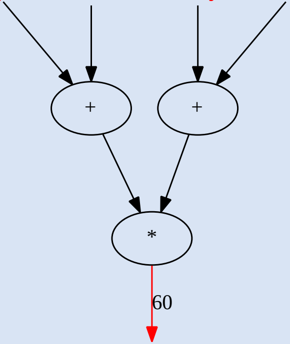
⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model ($\leftrightarrow$ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



$\Rightarrow$ Inherently parallel

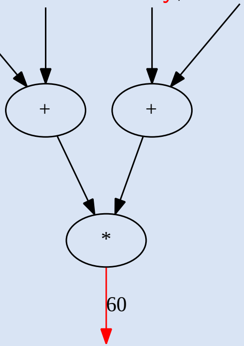
$\Rightarrow$ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Dataflow Programming

- Alternative to *von Neumann* model (↔ C/ASM)
- Instructions executed when their operands[a] are ready, not when the Instruction Pointer (aka. Program Counter, %PC) reaches it.



⇒ Inherently parallel

⇒ Today: coarser granularity, with imperative/object instruction blocks

[a] operand == token == message

Technological Context

Different Dataflow Models

Decidable Dataflow

- Correctness analysis
- Deadlock-free static scheduling
- Powerful optimization

but:

- Strong constraints imposed to dev.
- Reduced expressiveness
  - no dynamic problem

Dynamic Dataflow

Technological Context

Different Dataflow Models

Decidable Dataflow

- Correctness analysis
- Deadlock-free static scheduling
- Powerful optimization

but:

- Strong constraints imposed to dev.
- Reduced expressiveness
  - no dynamic problem

Dynamic Dataflow

- Increased modeling flexibility
- Conditional token emission/rcption
- Variable input/output rates

but:

- Limited static analysis
- Debugging is not straightforward ☺

Technological Context

Different Dataflow Models

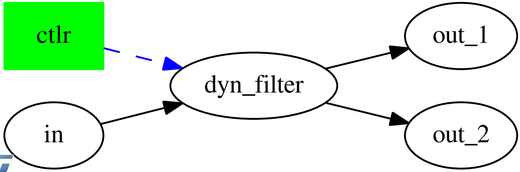
Decidable Dataflow

Dynamic Dataflow

- Increased modeling flexibility
- Conditional token emission/rcption
- Variable input/output rates

but:

- Limited static analysis
- Debugging is not straightforward ☺




```
WORK() { /* dyn_filter.c */
  flg = ctlr.next()
  cnt = ctlr.next()
  if (flg)
   out_1.send(treat(cnt))
  else
   for (i in 0:cnt)
     nxt = in.next()
     out_2.send(treat(nxt))
}
```

Agenda

1. Debugging Challenges of Dataflow Applications

2. Dataflow-Aware Interactive Debugging

3. Proof of Concept Implementation

4. Case Study: a H.264 Video Decoder

5. Conclusion

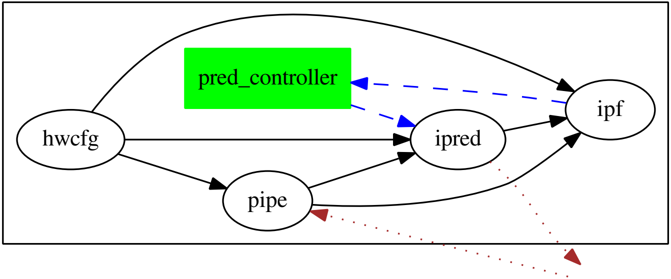
Agenda

1. Debugging Challenges of Dataflow Applications
2. Dataflow-Aware Interactive Debugging
3. Proof of Concept Implementation
4. Case Study: a H.264 Video Decoder
5. Conclusion

Debugging Challenges of Dataflow Applications

Dataflow applications

Graph-Based Architect. | Flow-Fork Instructions | Token-Based Execution



Single-threaded applications

- only one execution context

Multi-threaded applications

- multi-sequential execution
- flat organization:
  - no inter-thread relationship

Debugging Challenges of Dataflow Applications

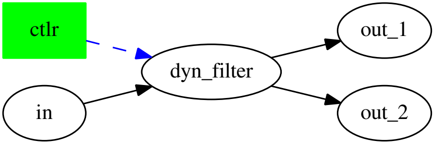
Dataflow applications

Graph-Based Architect. | Flow-Fork Instructions | Token-Based Execution

```
WORK() { /* dyn_filter.c */
  flg = ctlr.next()
  cnt = ctlr.next()
  if (flg)
   out_1.send(treat(cnt))   ⟸
  else
   for (i in 0:cnt)
     nxt = in.next()
     out_2.send(treat(nxt))
}
```

after this instruction:

- dyn_filr continues
- out_1 can run



Single-threaded applications | Multi-threaded applications

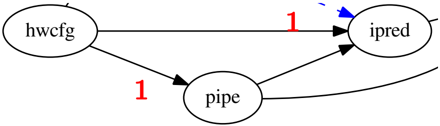
- %PC sequential execution
- simple flow-ctrl mechanisms:
  - functions, if-else, loops

Debugging Challenges of Dataflow Applications

Dataflow applications

Graph-Based Architect. | Flow-Fork Instructions | Token-Based Execution

- no function calls
  - only async. filter activation
- tokens exchanged among filters
- filter execution conditioned by input tokens generation

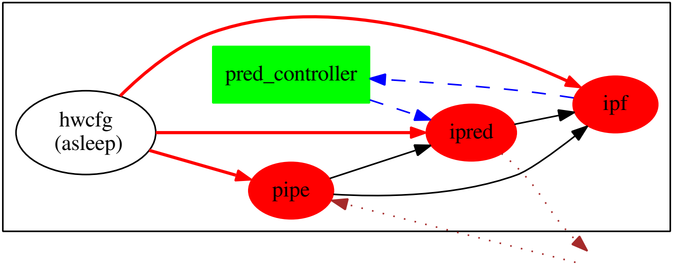


Single-threaded applications — **Multi-threaded applications**

- %PC sequential execution
- simple flow-ctrl mechanisms:
  - functions, if-else, loops

Debugging Challenges of Dataflow Applications

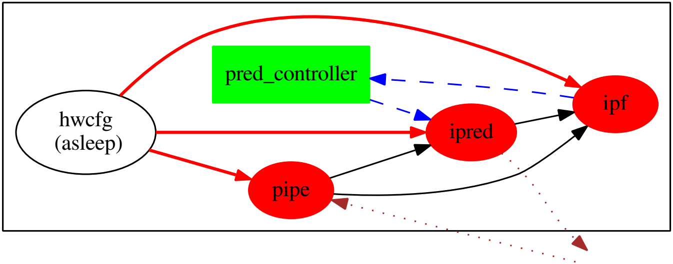
Example



- The application is frozen, how can GDB help us?
  - red filters are starving, waiting for data from the red link
  - filter `hwcfg` was not scheduled for execution by `pred_controller`
- *hint: not much!*

Debugging Challenges of Dataflow Applications

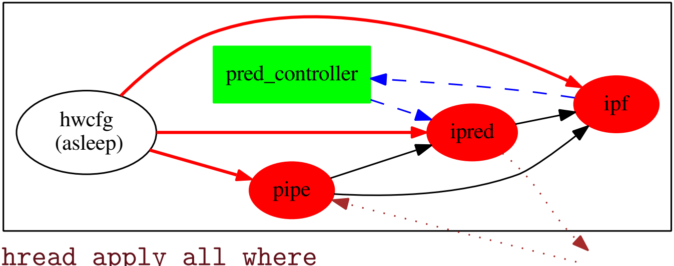
Example




```
(gdb) info threads
 Id    Target Id          Frame
  1    Thread 0xf7e77b   0xf7ffd430 in __kernel_vsyscall ()
* 2    Thread 0xf7e797   operator= (val=..., this=0xa0a1330)
```

Debugging Challenges of Dataflow Applications

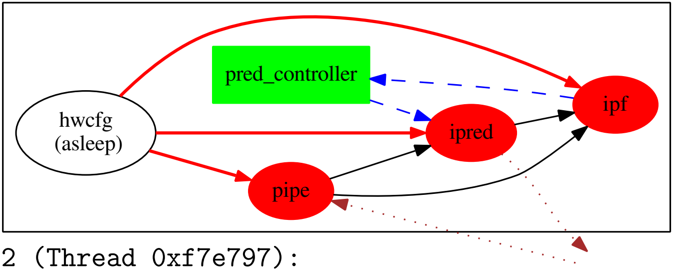
Example




```
(gdb) thread apply all where
Thread 1 (Thread 0xf7e77b):
#0  0xf7ffd430 in __kernel_vsyscall ()
#1  0xf7fcd18c in pthread_cond_wait@ ()
#2  0x0809748f in wait_for_command_completion(struct ... *)
#3  0x0809596e in pred_controller_work_function()
#4  0x08095cbc in entry(int, char**) ()
#5  0x0809740a in host_launcher_entry_point ()
#6  0xf7fc9aff in start_thread ()
```

Debugging Challenges of Dataflow Applications

Example




```
Thread 2 (Thread 0xf7e797):
#0 operator= (val=..., this=0xa0a1330)
#1 pipeRead (data=0) at pipeFilter.c:154
154  Smb = pedf.io.hwcfgSmb[count];
#2 0x0804da63 in PipeFilter_work_function () at pipe.c:361
#3 0x080a4132 in PedfBaseFilter::controller (this=0xa0a0d18)
#4 0x080bec81 in sc_core::sc_process_b::semantics (this=0xa0a35
#5 0x080c12f0 in sc_core::sc_thread_cor_fn (arg=0xa0a3598)
#6 0x08111831 in sc_core::sc_cor_qt_wrapper (...)
```

Debugging Challenges of Dataflow Applications

Objective

Provide debugger users with means to
better understand the state of the dataflow execution
and **easily reach key transition events**.

Agenda

1. Debugging Challenges of Dataflow Applications
2. **Dataflow-Aware Interactive Debugging**
3. Proof of Concept Implementation
4. Case Study: a H.264 Video Decoder
5. Conclusion

Dataflow-Aware Interactive Debugging

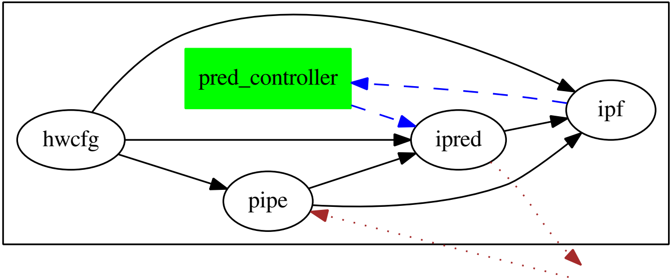
Idea: **Integrate dataflow programming model concepts in interactive debugging**

Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Application State

- model the application as a graph
- view token distribution
- sent/received counters on filter interfaces
- filter state: blocked waiting for more data? deadlocked ?

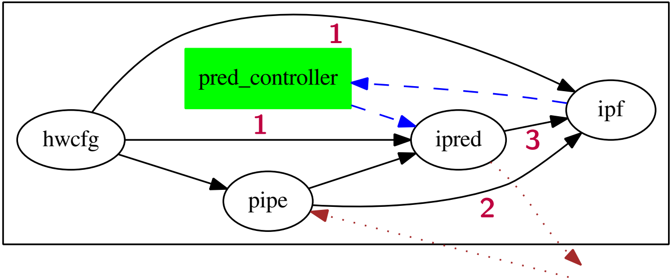


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Application State

- model the application as a graph
- view token distribution
- sent/received counters on filter interfaces
- filter state: blocked waiting for more data? deadlocked ?

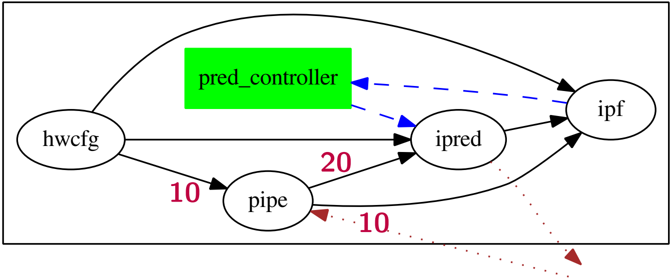


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Application State

- model the application as a graph
- view token distribution
- sent/received counters on filter interfaces
- filter state: blocked waiting for more data? deadlocked ?

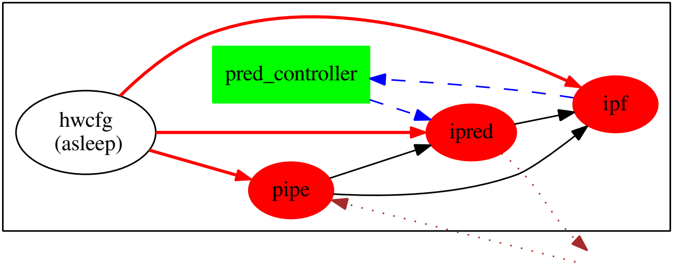


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Application State

- model the application as a graph
- view token distribution
- sent/received counters on filter interfaces
- filter state: blocked waiting for more data? deadlocked ?

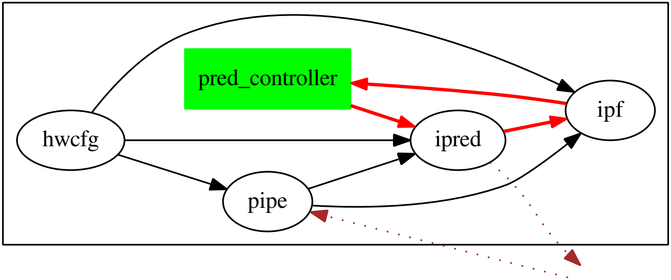


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Application State

- model the application as a graph
- view token distribution
- sent/received counters on filter interfaces
- filter state: blocked waiting for more data? deadlocked ?



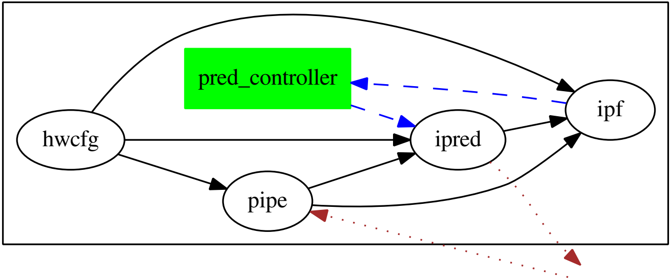
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



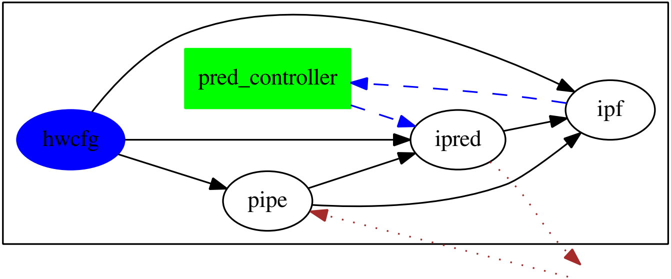
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



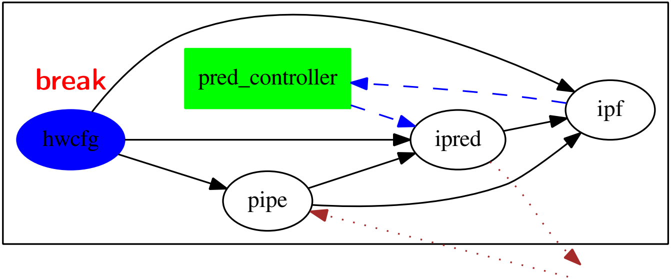
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
    - and allow conditional stops with token inspection



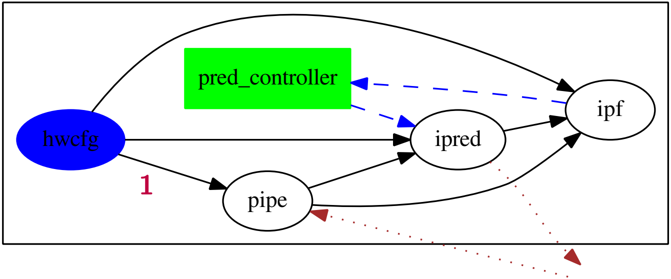
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



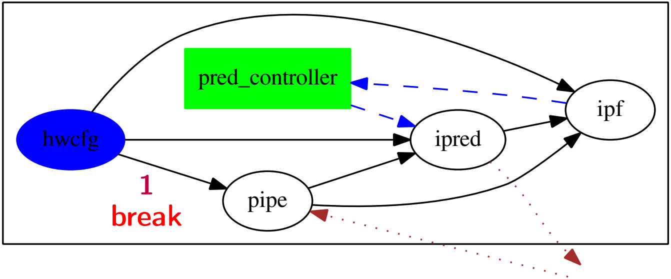
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



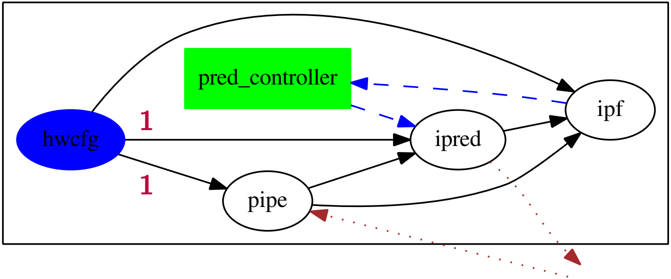
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



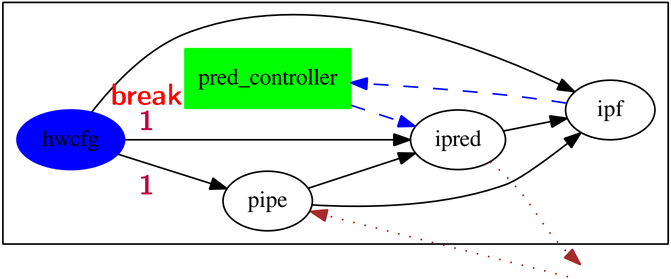
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



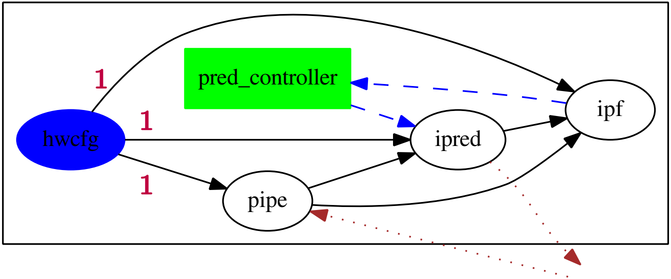
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
  - and allow conditional stops with token inspection



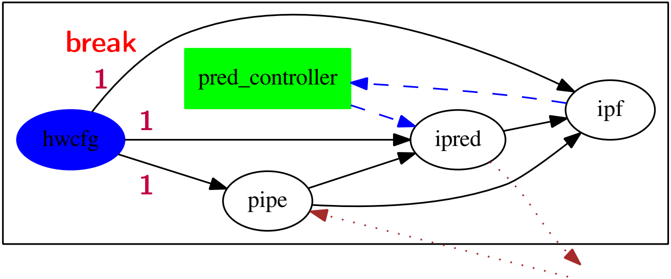
Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control

Catchpoints on dataflow events:

- filter is activated or terminates
- token generation/consumption
    - and allow conditional stops with token inspection

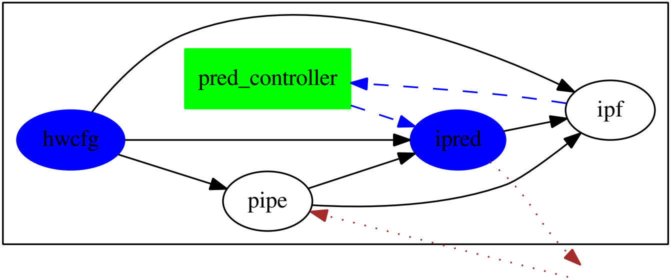


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control / Step-by-step

- token exchange $\Longleftrightarrow$ function calls

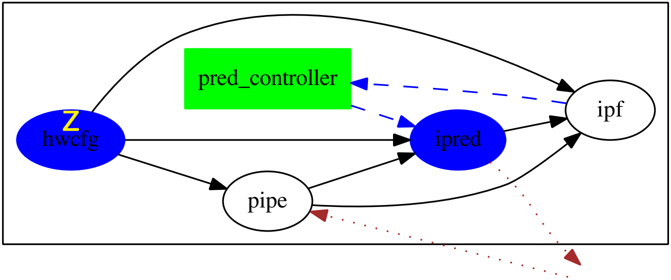


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control / Step-by-step

- token exchange $\Longleftrightarrow$ function calls

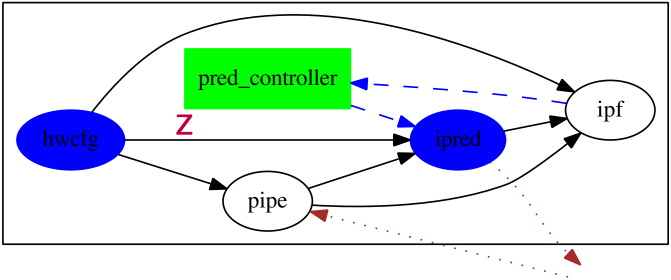


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control / Step-by-step

- token exchange $\Longleftrightarrow$ function calls

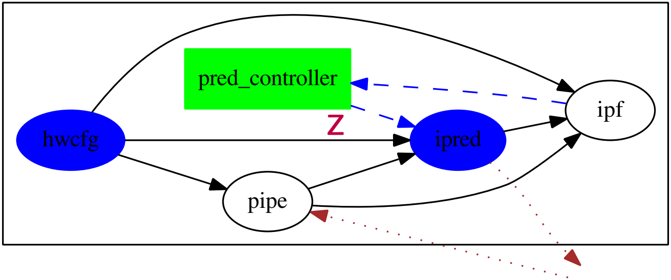


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control / Step-by-step

- token exchange $\Longleftrightarrow$ function calls

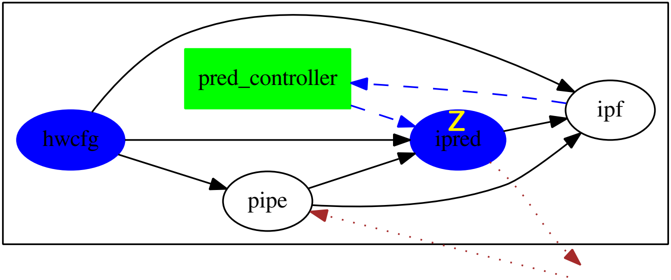


Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Flow Control / Step-by-step

- token exchange $\Longleftrightarrow$ function calls



Dataflow-Aware Interactive Debugging

Integrate dataflow programming model concepts in interactive debugging

Two-level Debugging

- source-code and symbol breakpoints
- line-by-line
- watchpoints, processor inspection, etc.

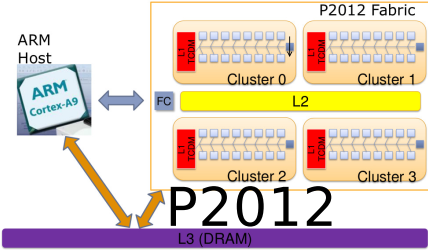
```
(gdb) next
(gdb) step
(gdb) break $pc + 0x45F
(gdb) break hwcfgFilter.c:27 if *mbType != 0xFFFFFFFF
(gdb) watch *pedf.attr.cHwcfgQuant
```

Agenda

life.augmented

Proof of Concept Implementation

Proof-of-concept environment



Platform 2012

ST/CEA MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric

Proof of Concept Implementation

Proof-of-concept environment

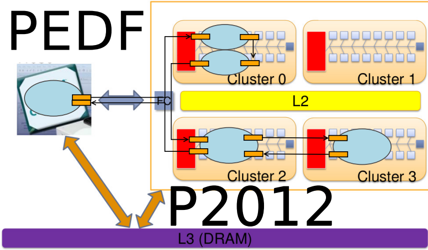
Dataflow Programming Model

- Predicated Execution DataFlow
  - Dataflow framework for H.265

Platform 2012

ST/CEA MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Proof of Concept Implementation

Proof-of-concept environment

The Gnu Debugger

- Adapted to low level/C debugging
- Large user community

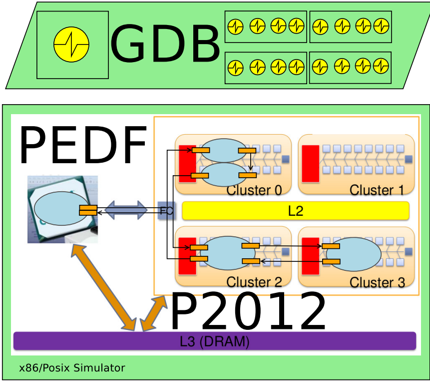
Dataflow Programming Model

- Predicated Execution DataFlow
  - Dataflow framework for H.265

Platform 2012

ST/CEA MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Proof of Concept Implementation

Proof-of-concept environment

The Gnu Debugger

- Adapted to low level/C debugging
- Large user community
- Extendable with Python API

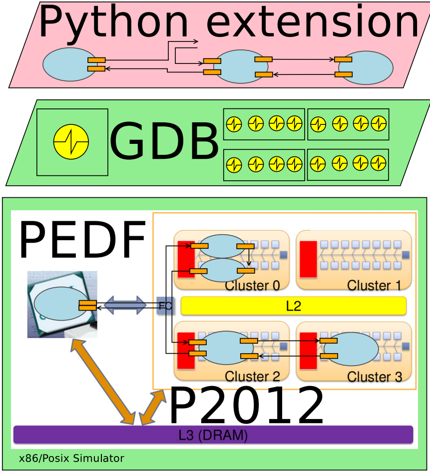
Dataflow Programming Model

- Predicated Execution DataFlow
  - Dataflow framework for H.265

Platform 2012

ST/CEA MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



life.augmented

Proof of Concept Implementation

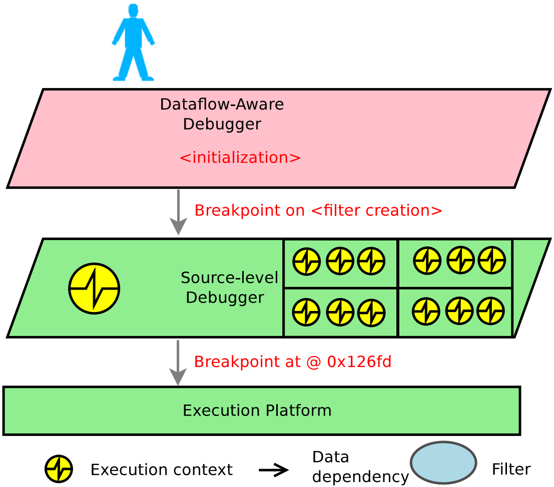
Work with framework events

⇒ Detect and interpret key events in the programming framework

Proof of Concept Implementation

Work with framework events

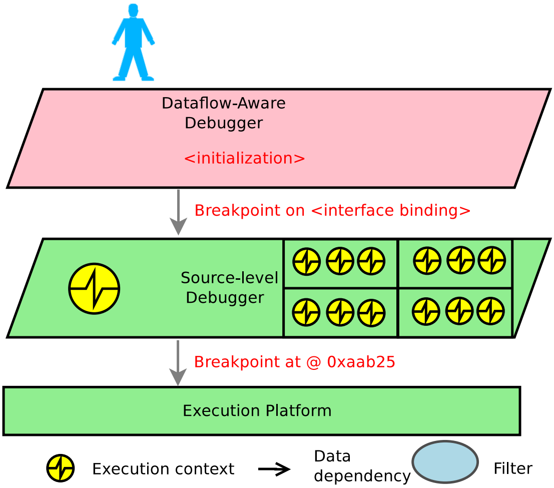
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

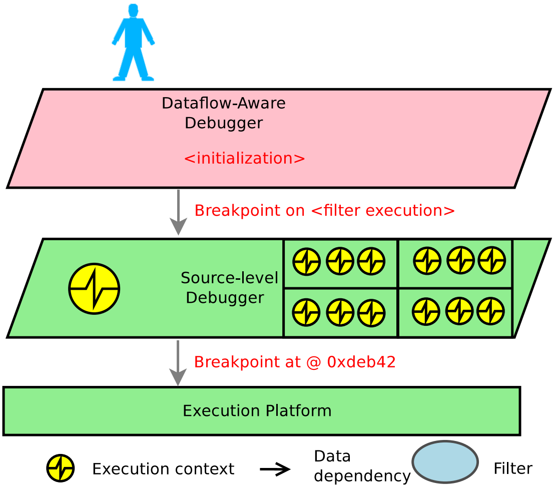
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

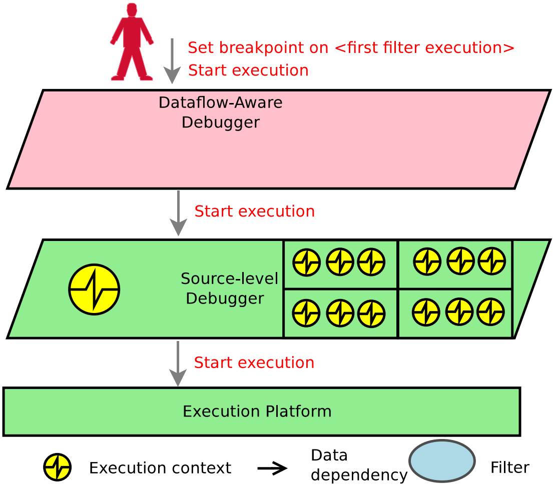
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

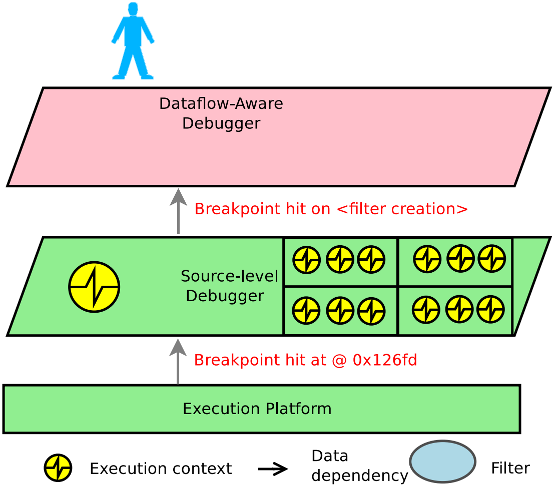
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

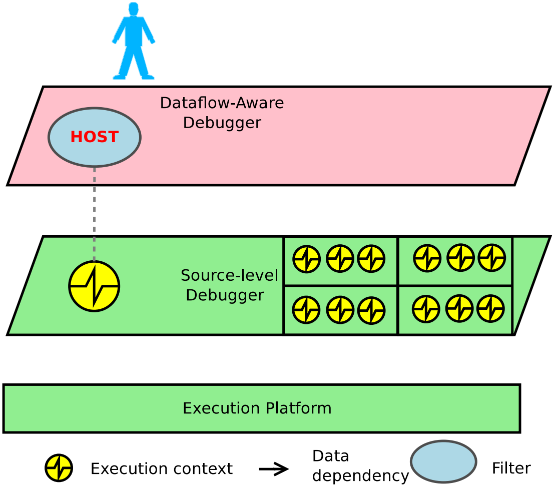
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

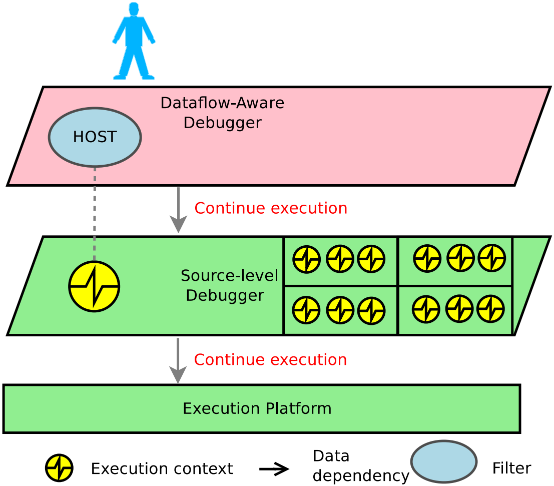
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

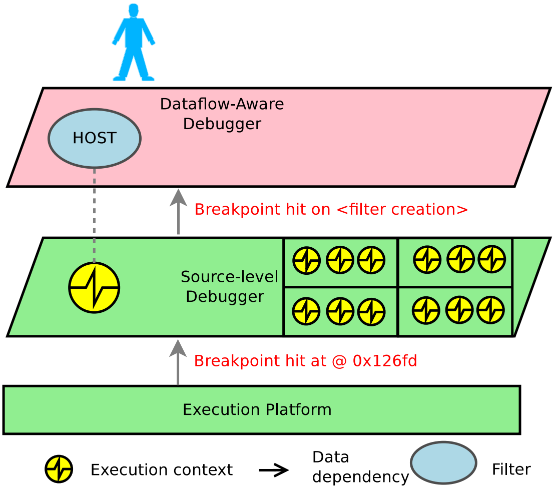
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

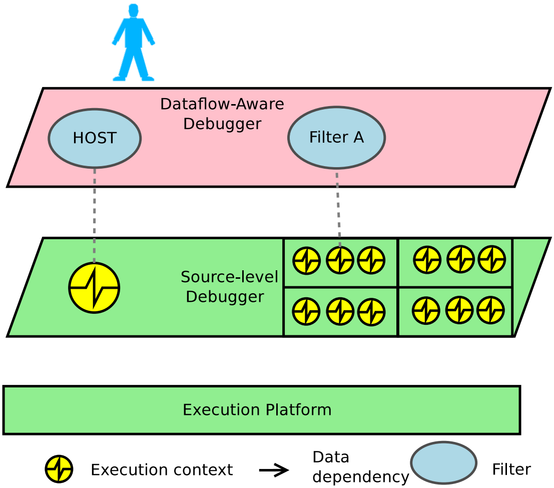
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

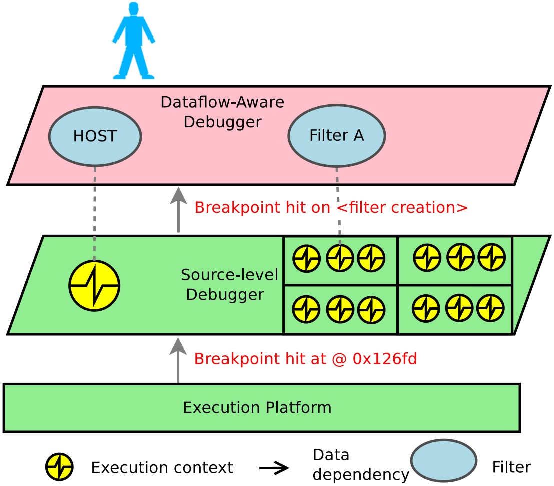
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

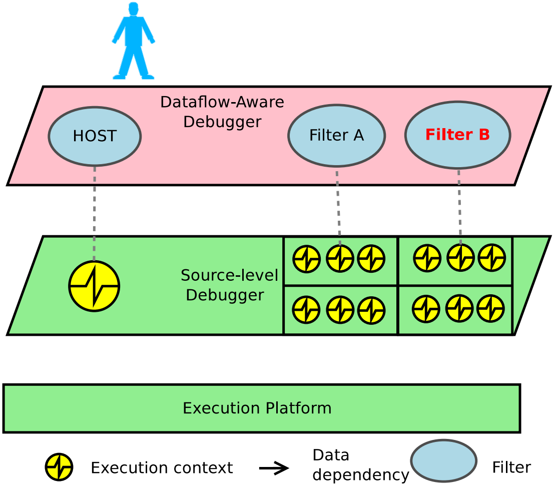
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

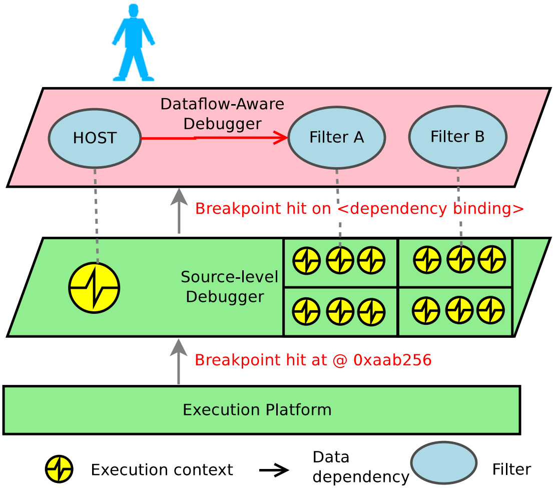
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

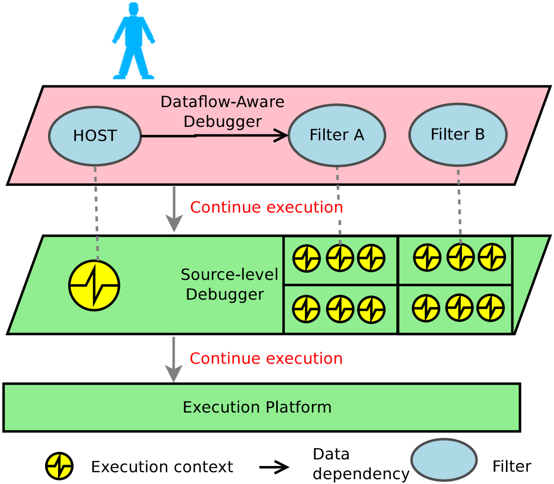
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

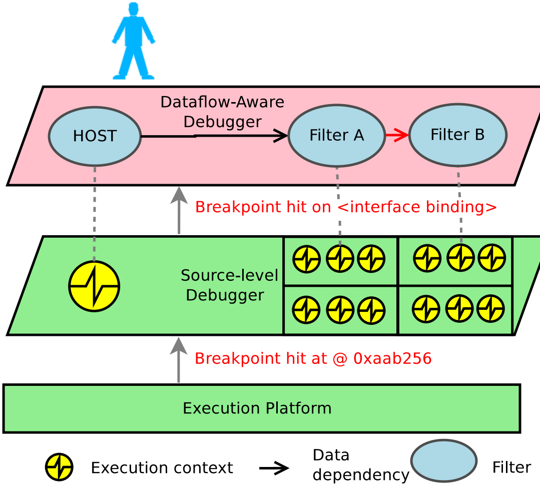
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

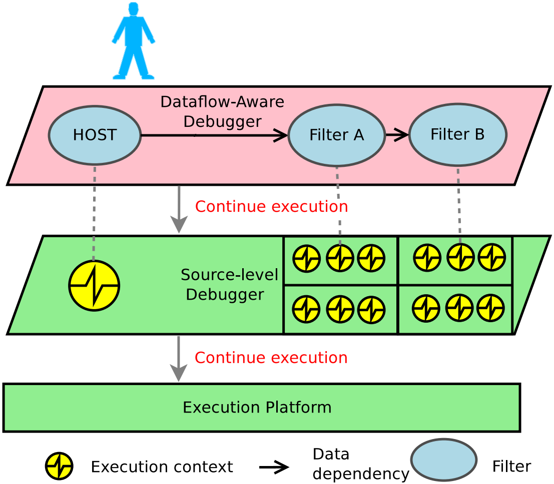
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

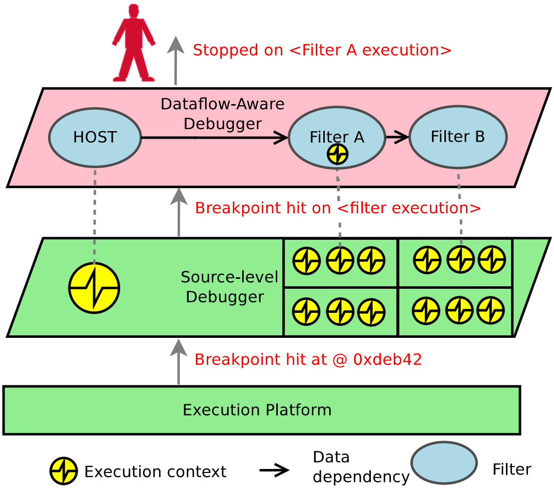
⇒ Detect and interpret key events in the programming framework



Proof of Concept Implementation

Work with framework events

⇒ Detect and interpret key events in the programming framework



Agenda

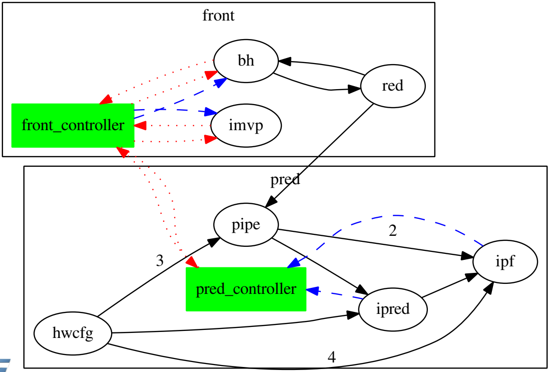
1. Debugging Challenges of Dataflow Applications
2. Dataflow-Aware Interactive Debugging
3. Proof of Concept Implementation
4. Case Study: a H.264 Video Decoder
5. Conclusion

life.augmented

Case Study: a H.264 Video Decoder

Overview

- dynamic dataflow application
- exploit P2012 heterogeneous capabilities
- eventually, filters ⇒ HW accelerators

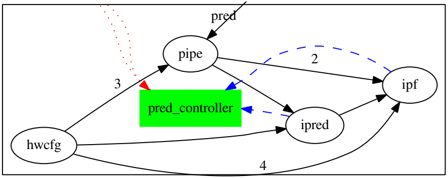


H.264 logo by http://bullboykennels.deviantart.com

Case Study: a H.264 Video Decoder

Application State

- Model the application as a graph
- View token distribution




```
(gdb) info connections pipe
#tk #interface            #remote itf           #remote filter
    Red2PipeLumaDC_in   < Red2PipeLumaDC_out   front.red
(3) HwcfgQuant_in       < pipe_HwcfgQuant_out  hwcfg
    Pipe2AddCrMB_out    > Pipe2AddCrMB_in      ipred
(2) LumaCBF_out         > LumaCBF_in           ipf
```

Case Study: a H.264 Video Decoder

Step-by-step

- Next token send / received

```
(gdb) filter pipe next token out
...
[Stopped on token enqueued in ‘Pipe2IpredCrMB_out -> ipred’]
120 pedf.io.Pipe2IpredCrMB_out[count++] = *Pipe2IpredCrMB;
```

```
(gdb) filter ipred next token in
...
[Stopped on token received from ‘Pipe2IpredCrMB_in <- pipe’]
204 pedf.data.Pipe2IpredCrMB = pedf.io.Pipe2IpredCrMB_in[count+
```

life.augmented

Case Study: a H.264 Video Decoder

Step-by-step

- Next token send / received

```
(gdb) filter pipe next token out
...
[Stopped on token enqueued in ‘Pipe2IpredCrMB_out -> ipred’]
120 pedf.io.Pipe2IpredCrMB_out[count++] = *Pipe2IpredCrMB;
```

```
(gdb) filter ipred next token in
...
[Stopped on token received from ‘Pipe2IpredCrMB_in <- pipe’]
204 pedf.data.Pipe2IpredCrMB = pedf.io.Pipe2IpredCrMB_in[count+
```

```
(gdb) filter pipe itf Pipe2IpredCrMB_out follow last
```

Agenda

1. Debugging Challenges of Dataflow Applications
2. Dataflow-Aware Interactive Debugging
3. Proof of Concept Implementation
4. Case Study: a H.264 Video Decoder
5. Conclusion

Conclusion

- Debugging complex applications is challenging
- Lack of high level information about programming frameworks

- **Our work:** bring debuggers closer to dataflow programming models
  - Better understanding application behavior
  - Keep programmers focused on bug tracking

Conclusion

- Debugging complex applications is challenging
- Lack of high level information about programming frameworks

- **Our work:** bring debuggers closer to dataflow programming models
  - Better understanding application behavior
  - Keep programmers focused on bug tracking

- **Proof-of-concept**
  - P2012 dataflow programming environment
    - component debugging published earlier
    - different models, same approach
      $\Rightarrow$ first step towards programming-model centric debugging
  - GDB and its Python interface
    - missing hooks contributed to the project

Conclusion

- Debugging complex applications is challenging
- Lack of high level information about programming frameworks

- **Our work:** bring debuggers closer to dataflow programming models
  - Better understanding application behavior
  - Keep programmers focused on bug tracking

- **Proof-of-concept**
  - P2012 dataflow programming environment
    - component debugging published earlier
    - different models, same approach

      $\Rightarrow$ first step towards programming-model centric debugging
  - GDB and its Python interface
    - missing hooks contributed to the project

- Going further with programming-model aware debugging
  - GPU computing, OpenCL API
  - Visualization to aid in understanding app. behavior

Thanks for your attention