STMicroelectronics
LIG
University of Grenoble

life.augmented

Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget

Under the supervision of:

Dr. Miguel Santana, *STMicroelectronics*

Prof. Jean-François Méhaut, *CEA/UJF*

Thesis Defense, Grenoble — February 3rd, 2014

Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Consumer Electronics Devices

- 4K digital televisions
- Smartphones
- Hand-help music players

Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Consumer Electronics Devices

- 4K digital televisions
- Smartphones
- Hand-help music players

- High-resolution multimedia apps
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...

Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Consumer Electronics Devices

- 4K digital televisions
- Smartphones
- Hand-help music players

- High-resolution multimedia apps

  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - . . .

⇒ high performance expectations.

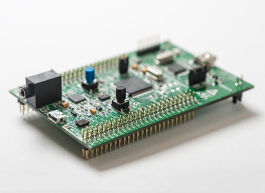
Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Current applications have high performance expectations...



⇒ important demand for:

- Powerful parallel architectures
- High-level development methodologies
- Efficient verification & validation tools

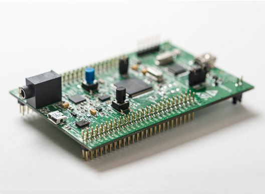
Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Current applications have high performance expectations...



⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies
- Efficient verification & validation tools

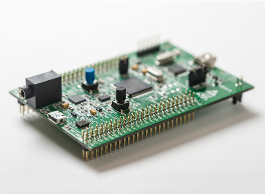
Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Current applications have high performance expectations...



⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies...
  - Programming models & environments
- Efficient verification & validation tools

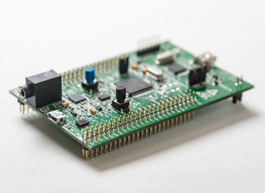
Introduction

Embedded Systems and MPSoC

Embedded Systems (IEEE '92)

A computer system that is part of a larger system and performs some of the requirements of that system.

Current applications have high performance expectations...



⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies. . .
  - Programming models & environments
- Efficient verification & validation tools
  - Our research effort

Introduction

Verification & Validation

Important domain for business, engineering and research

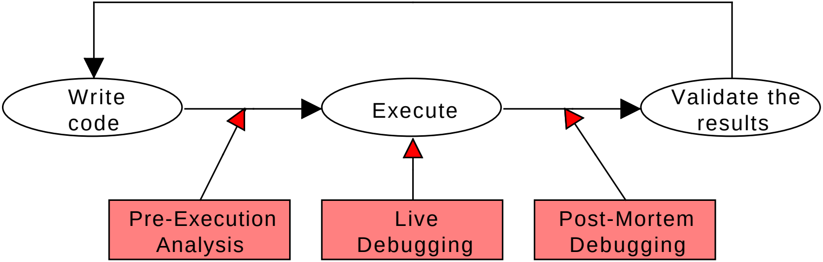
- Time-to-market
- Consumer experience
- Costly maintenance phase

Introduction

Verification & Validation

Important domain for business, engineering and research

- Time-to-market
- Consumer experience
- Costly maintenance phase
- Time and nerves consuming
- Large set of skills/techniques involved
- Still imperfect



Agenda

1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

3. mcGDB Case-Studies

Agenda

1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

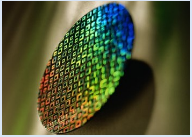
3. mcGDB Case-Studies

Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System-on-Chip

- Many-core processor for embedded systems
- Heterogeneous computing power
- Low energy-consumption

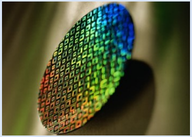


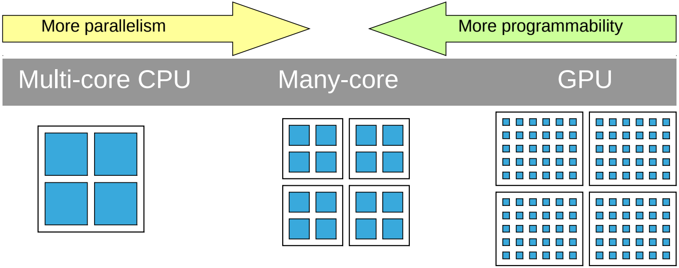
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System-on-Chip

- Many-core processor for embedded systems
- Heterogeneous computing power
- Low energy-consumption



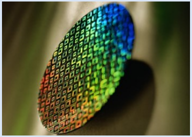


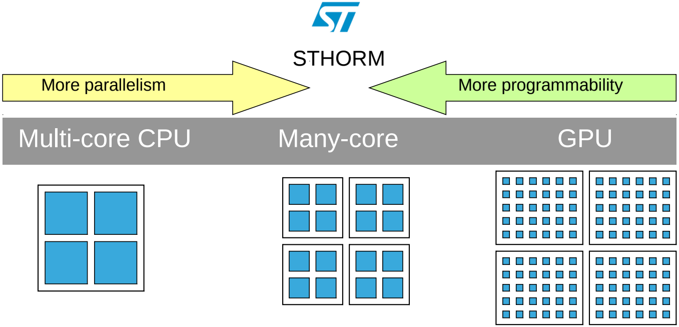
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System-on-Chip

- Many-core processor for embedded systems
- Heterogeneous computing power
- Low energy-consumption



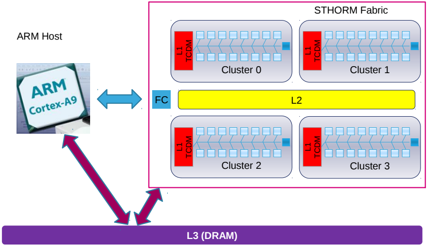


Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

STHORM Platform — our reference MPSoC

- **ST H**eterogeneous **L**o**w**-Powe**r M**any-core (Platform 2012)
- CPU + 4 clusters × 16 lightweight/energy-efficient cores
  - ± dedicated hardware accelerators

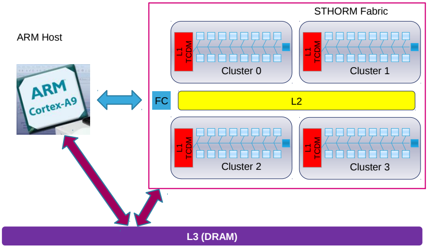


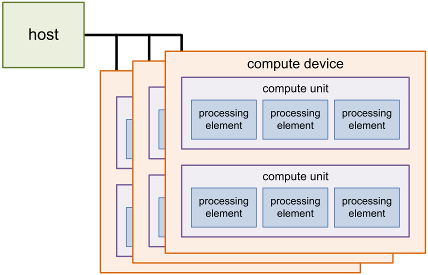
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

STHORM Platform — our reference MPSoC

- **ST Heterogeneous Low-Power Many-core** (Platform 2012)
- CPU + 4 clusters × 16 lightweight/energy-efficient cores
  - ± dedicated hardware accelerators
- GPU-like architecture





Björn König, Wikimedia

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

How to program such complex architectures?

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

How to program such complex architectures?

Programming models and environments!

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

... not so many clear definitions in the literature, so ...

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

... not so many clear definitions in the literature, so ...

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

... not so many clear definitions in the literature, so ...

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

... not so many clear definitions in the literature, so ...

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

broad definition!

→ it's an abstract machine

- that separates application development / lower-level concerns

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

... not so many clear definitions in the literature, so ...

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

broad definition!

→ it's an abstract machine

- that separates application development / lower-level concerns

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine…
- providing certain operations to the programming level above and requiring …

Supportive Environment

- … implementations for each of these operations on all of the architectures below.

- programming frameworks
- runtime libraries
- APIs

Background: MPSoC Programming and Debugging

MPSoCs and Programming Models

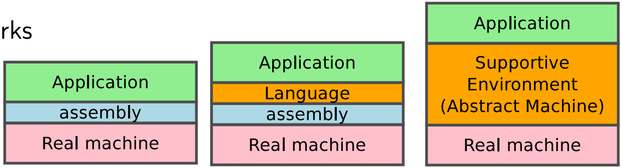
Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and requiring ...

Supportive Environment

- ... implementations for each of these operations on all of the architectures below.

- programming frameworks
- runtime libraries
- APIs



Background: MPSoC Programming and Debugging

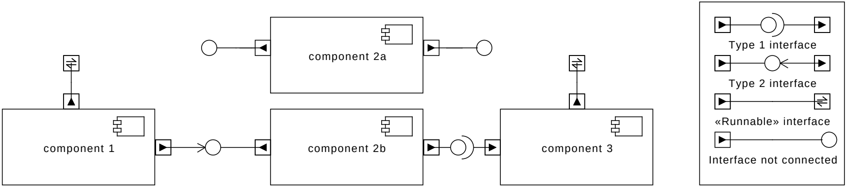
Programming Models for STHORM MPSoC

Components	Dataflow	Kernels

Background: MPSoC Programming and Debugging

Programming Models for Sthorm MPSoC

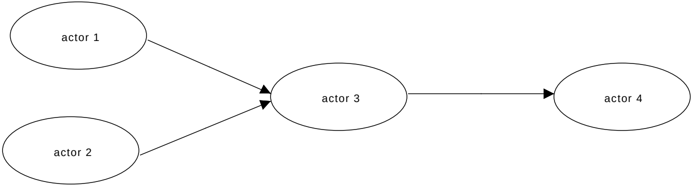
Components	Dataflow	Kernels
- code/data encapsulation - software reuse - service contracts (language-free interfaces)		



Background: MPSoC Programming and Debugging

Programming Models for STHORM MPSoC

Components	Dataflow	Kernels
	• emphasis put on the stream of data • implicit parallelism • roots in graph theory	



Background: MPSoC Programming and Debugging

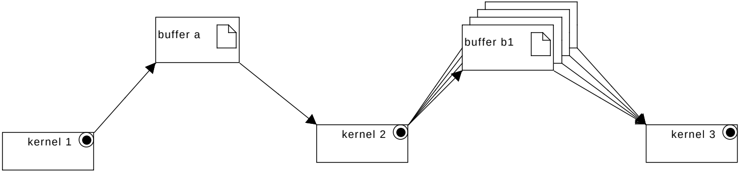
Programming Models for STHORM MPSoC

Components

Dataflow

Kernels

- data parallelism
- massively parallel
- work offloaded to accelerator (GPU/STHORM)



Background: MPSoC Programming and Debugging

Programming Models for STHORM MPSoC

Components

- code/data encapsulation
- software reuse
- service contracts (language-free interfaces)

Dataflow

- emphasis put on the stream of data
- implicit parallelism
- roots in graph theory

Kernels

- data parallelism
- massively parallel
- work offloaded to accelerator (GPU/STHORM)

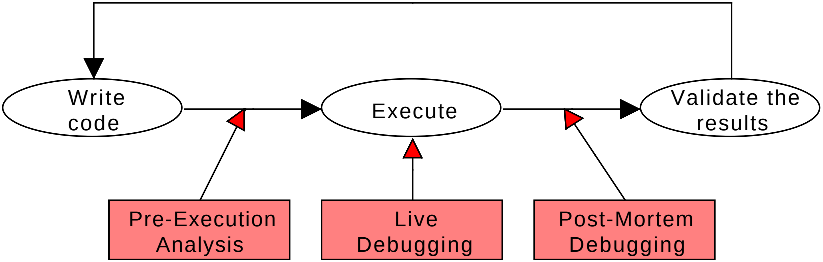
Different models covering large programming domain ...

but what about Verification & Validation?

correctness guarantees vs. development constraints ...

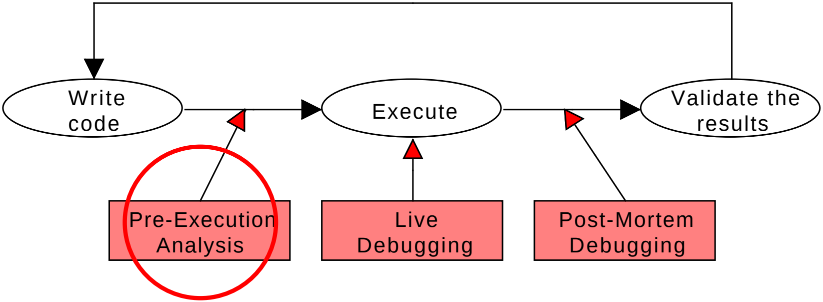
Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging



Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging



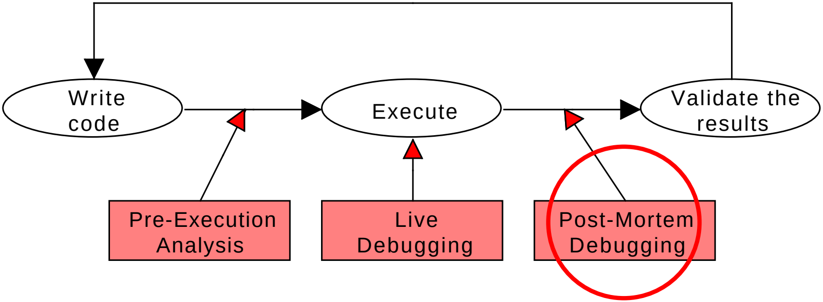
Static/Formal Analysis

+ May be exhaustive
  - synchronous dataflow

– Not always feasible
  - dynamic behaviors

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

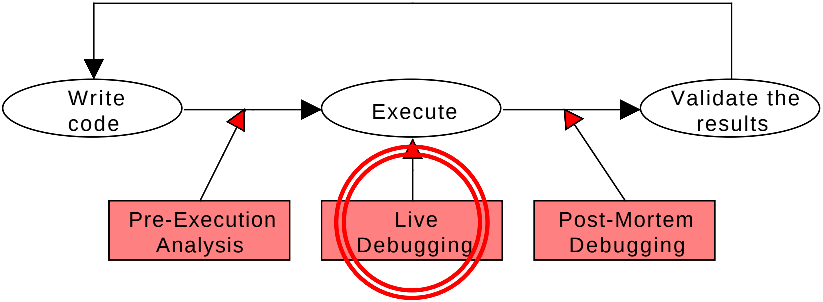


Trace Analysis

- Manual/data-mining
+ long/time critical run
- What to trace?
- fixed # of trace-points

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging



Static/Formal Analysis

+ May be exhaustive
  - synchronous dataflow
– Not always feasible
  - dynamic behaviors

Trace Analysis

- Manual/data-mining
+ long/time critical run
- What to trace?
- fixed # of trace-points

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

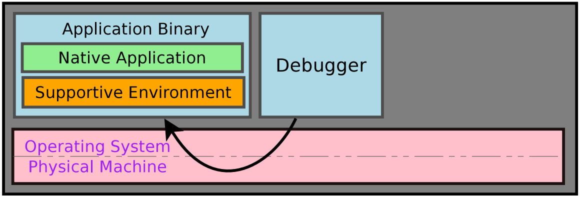
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!



Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

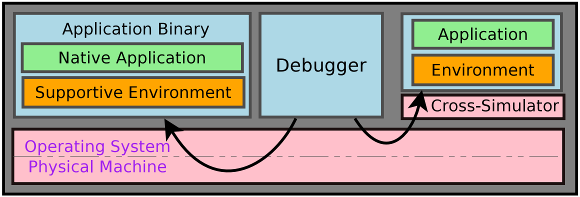
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!



Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

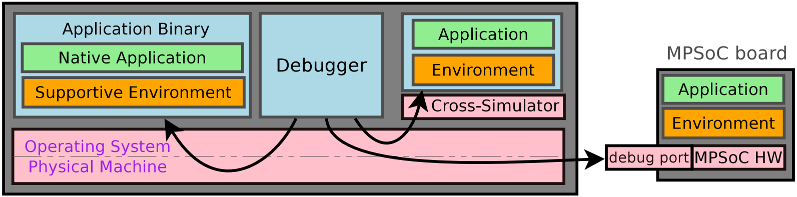
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!



Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging

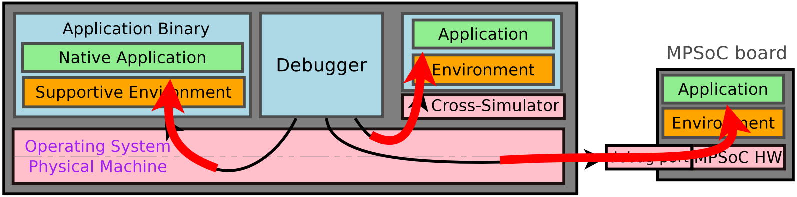
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!



Background: MPSoC Programming and Debugging

Objective

Provide developers with means to
better understand the state of the high-level applications and control more easily their execution,
suitable for various models and environments.

Agenda

1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

3. mcGDB Case-Studies

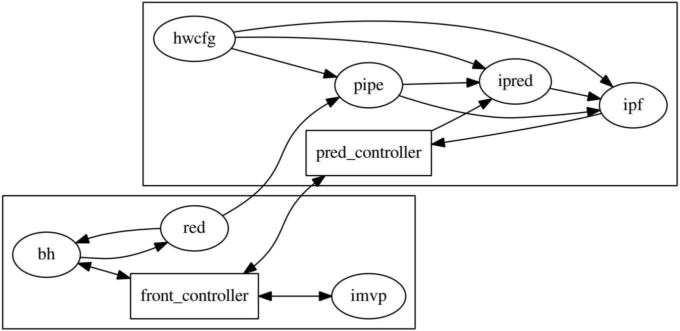
Programming Model Centric Interactive Debugging

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

Programming Model Centric Interactive Debugging

1 Provide a Structural Representation

- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations

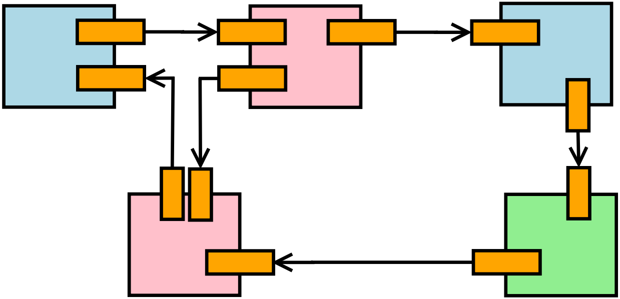


Graph of a dataflow from the case-study

Programming Model Centric Interactive Debugging

1 Provide a Structural Representation

- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations

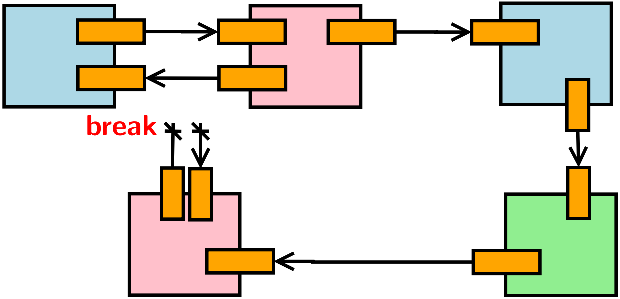


Reconfiguration of an application based on components

Programming Model Centric Interactive Debugging

1 Provide a Structural Representation

- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations

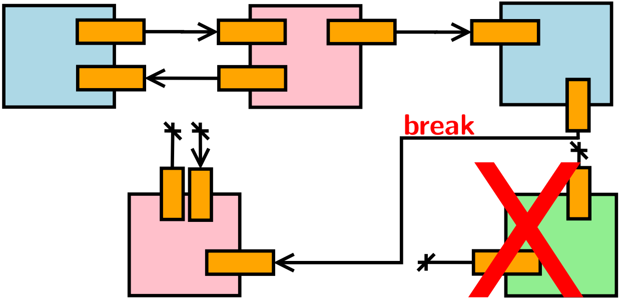


Reconfiguration of an application based on components

Programming Model Centric Interactive Debugging

1 Provide a Structural Representation

- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations

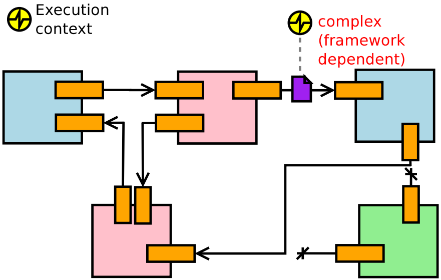


Reconfiguration of an application based on components

Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms

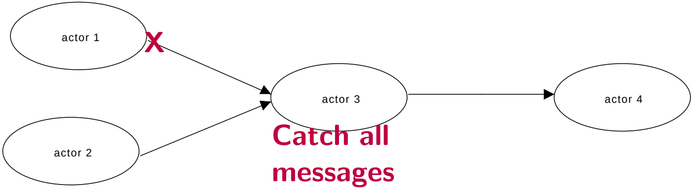


```
(pink) ask_service()
(env.) transmit request...
(blue) exec_service() {...}
```

Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics
    (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms

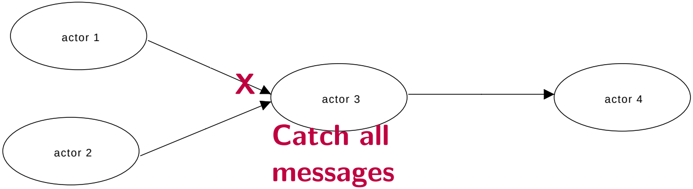


Catch all messages

Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

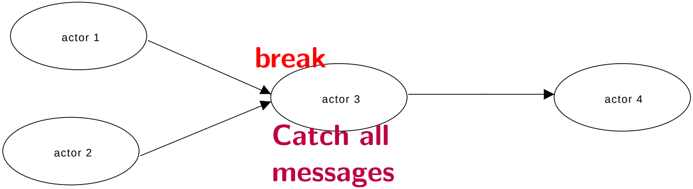
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

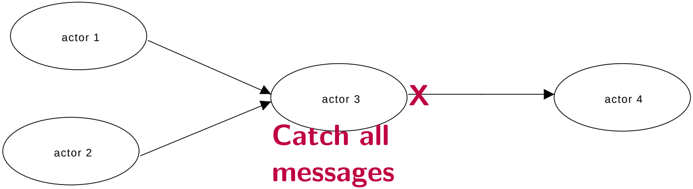
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics
    (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics
    (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms

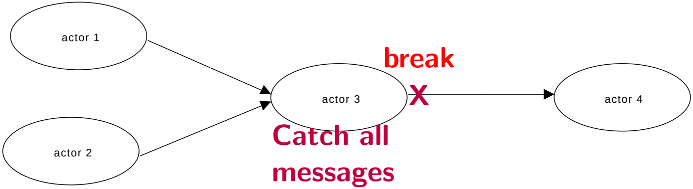


Catch all messages

Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

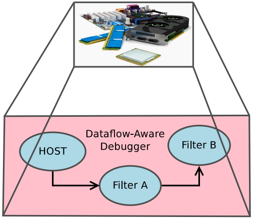
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

3 Interact with the Abstract Machine

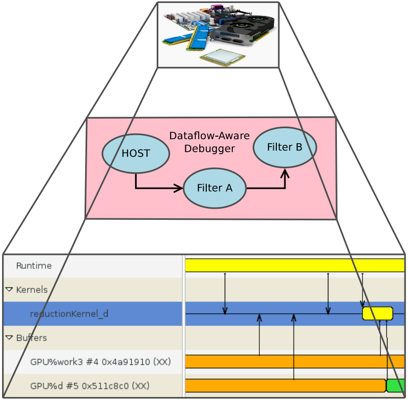
- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



Programming Model Centric Interactive Debugging

3 Interact with the Abstract Machine

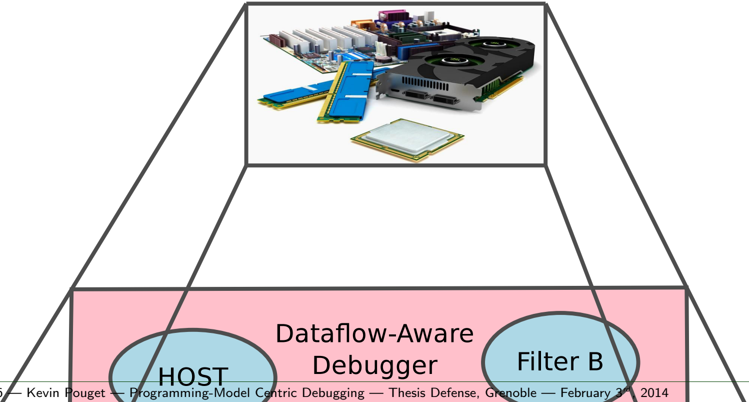
- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



Programming Model Centric Interactive Debugging

3 Interact with the (abstract) Machine

- Support interactions with *real* machine
  - memory and processor inspection
  - breakpoints and watchpoints (maybe per entity)
  - step-by-step execution ...



Programming Model Centric Interactive Debugging

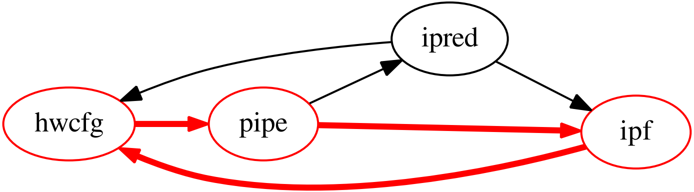
4 Open Up to Model and Environment Specific Features

- Follow messages over *multiple* entities
- User-defined constraints on the graph topology
- Deadlock detection in task-based models

Programming Model Centric Interactive Debugging

4 Open Up to Model and Environment Specific Features

- Follow messages over *multiple* entities
- User-defined constraints on the graph topology
- Deadlock detection in task-based models



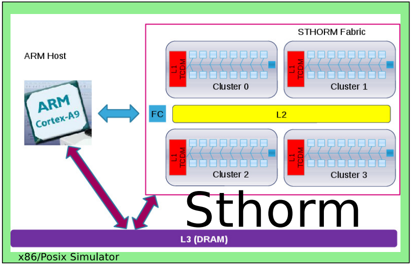
cycles in the graph of blocking communications ⟹ deadlock

Programming Model Centric Interactive Debugging

1. Provide a Structural Representation
2. Monitor Dynamic Behaviors
3. Interact with the Abstract Machine
4. Open Up to Model and Environment Specific Features

Programming Model Centric Interactive Debugging

Proof-of-concept Environment



STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators

Programming Model Centric Interactive Debugging

Proof-of-concept Environment

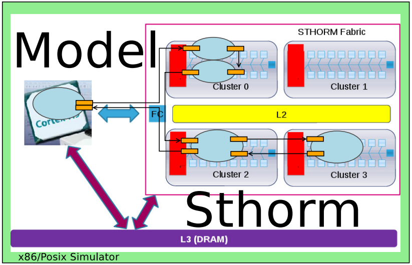
STHORM Progr. Environments

- Components (NPM)
- Dataflow (PEDF)
- Kernel (OpenCL)

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Programming Model Centric Interactive Debugging

Proof-of-concept Environment

The GNU Debugger

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

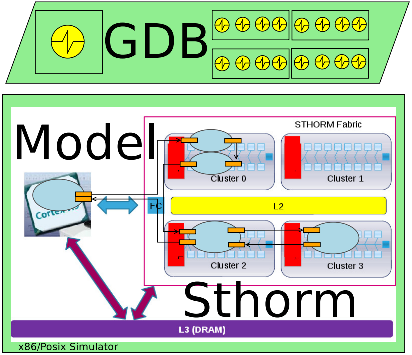
STHORM Progr. Environments

- Components (NPM)
- Dataflow (PEDF)
- Kernel (OpenCL)

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Programming Model Centric Interactive Debugging

Proof-of-concept Environment

The GNU Debugger

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

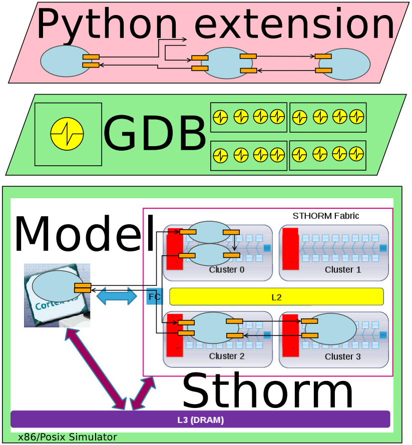
STHORM Progr. Environments

- Components (NPM)
- Dataflow (PEDF)
- Kernel (OpenCL)

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Programming Model Centric Interactive Debugging

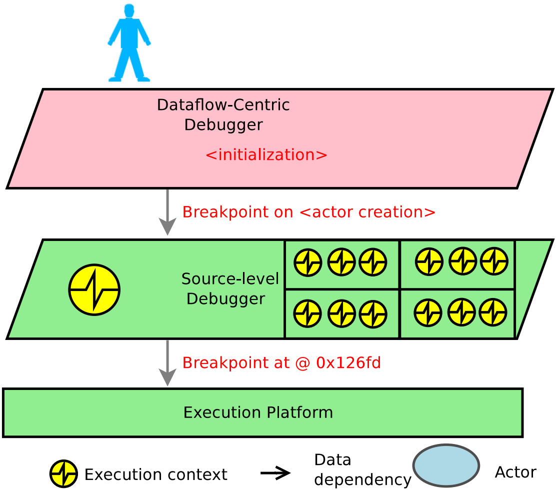
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework

Programming Model Centric Interactive Debugging

Interpreting Execution Events

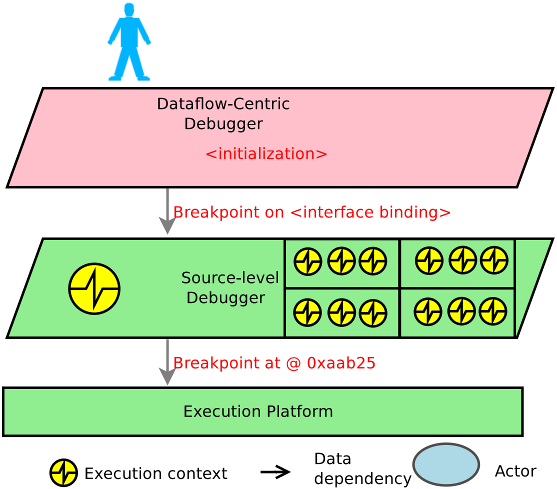
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

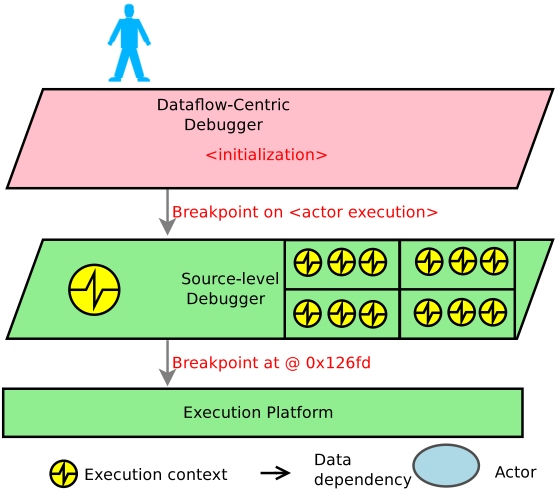
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

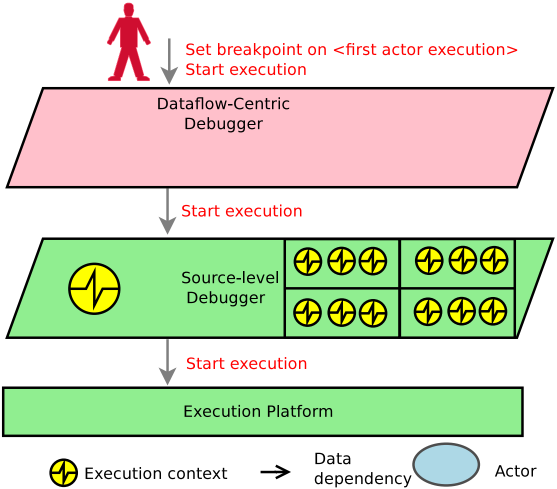
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

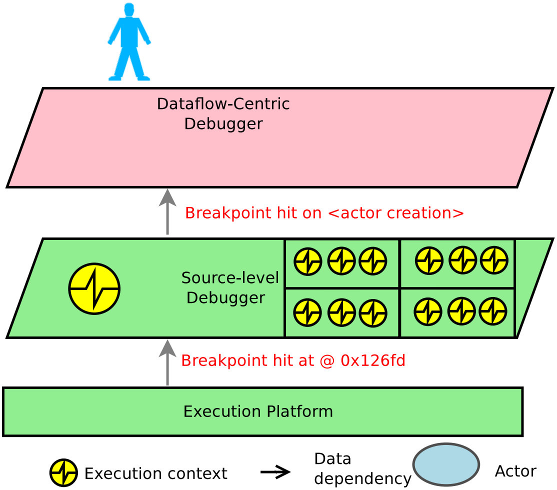
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

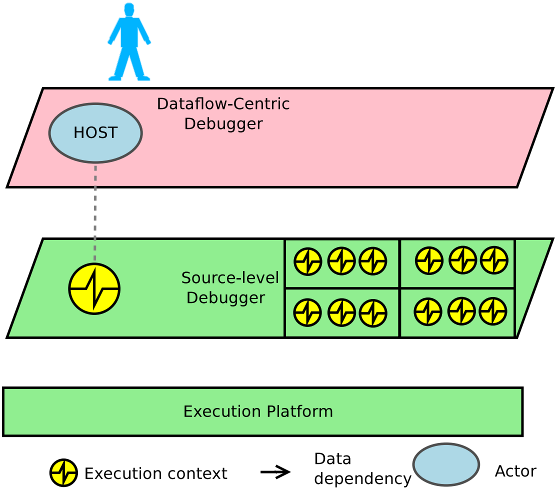
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

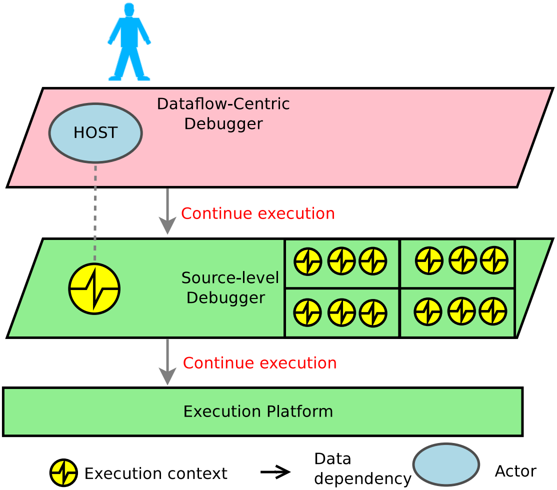
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

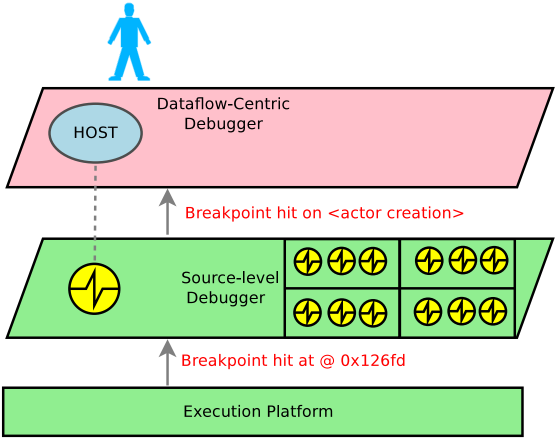
⇒ Detect and interpret the exec. events of the runtime framework

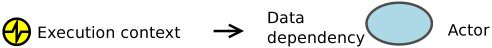


Programming Model Centric Interactive Debugging

Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework

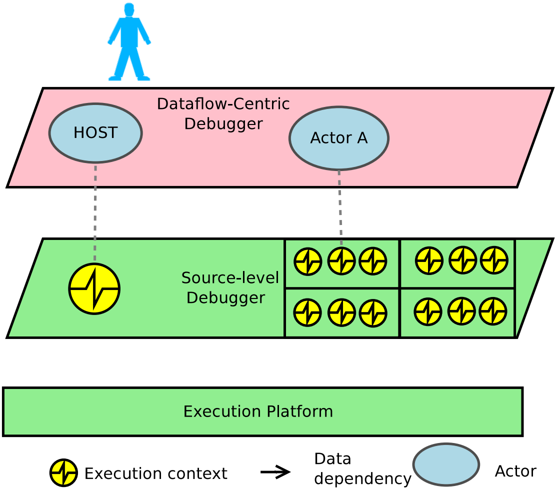

Dataflow-Centric Debugger
HOST
Breakpoint hit on <actor creation>
Source-level Debugger
Breakpoint hit at @ 0x126fd
Execution Platform



Programming Model Centric Interactive Debugging

Interpreting Execution Events

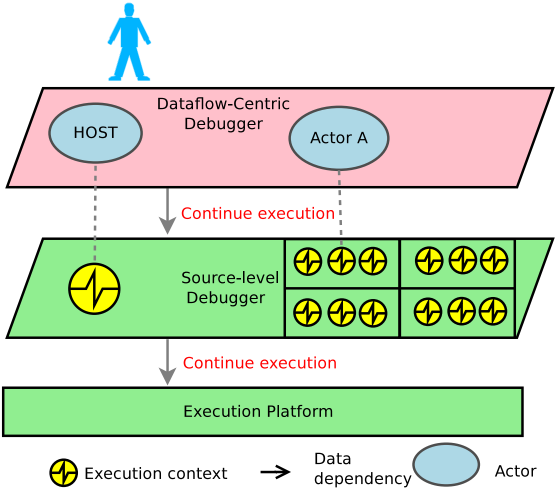
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

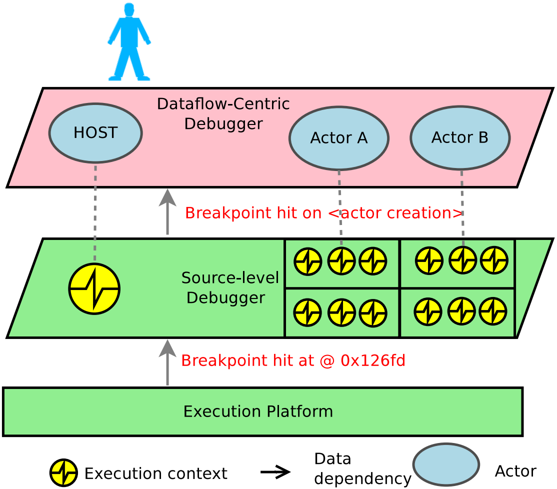
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

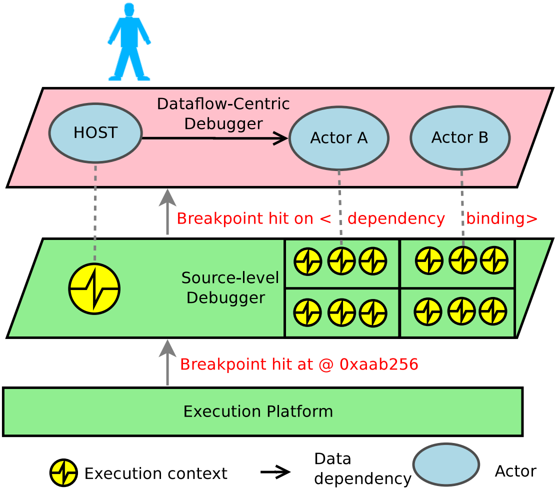
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

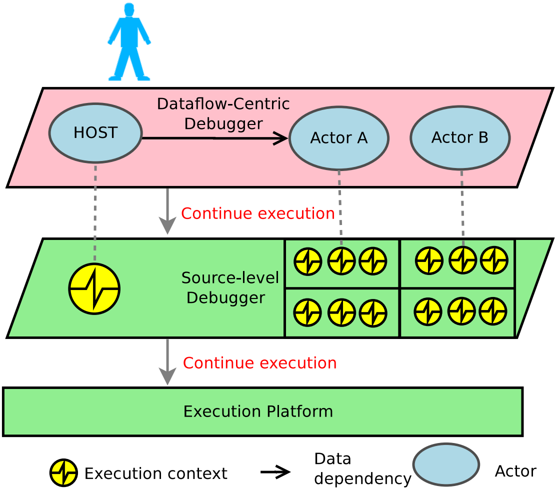
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

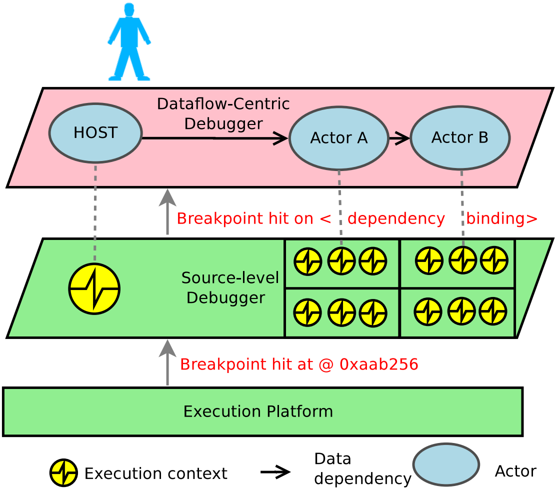
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

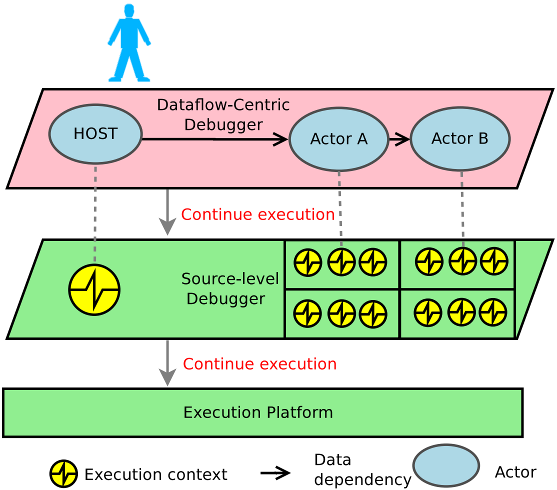
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

Interpreting Execution Events

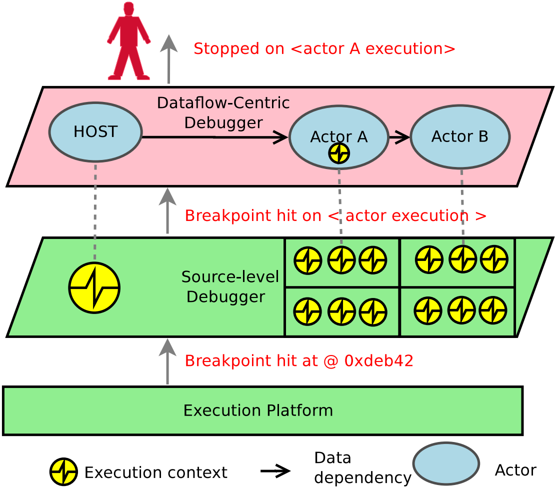
⇒ Detect and interpret the exec. events of the runtime framework



Programming Model Centric Interactive Debugging

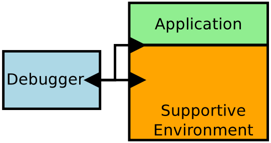
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework


Stopped on <actor A execution>
Dataflow-Centric Debugger
HOST
Actor A
Actor B
Breakpoint hit on < actor execution >
Source-level Debugger
Breakpoint hit at @ 0xdeb42
Execution Platform
Execution context
Data dependency
Actor

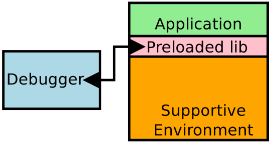
Programming Model Centric Interactive Debugging

Capture Mechanism Evaluation and Alternatives

	**Breakpoints and Debug Information**		
Capturable Info.	**High**		
Execution Overhead	Significant		
Cooperation btw. Debug and Env.	**None**		
Portability	Low		

Programming Model Centric Interactive Debugging

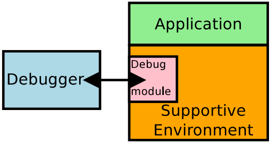
Capture Mechanism Evaluation and Alternatives



	Breakpoints and Debug Information	**Preloaded Library**
Capturable Info.	**High**	Limited to API
Execution Overhead	Significant	Limited
Cooperation btw. Debug and Env.	**None**	Low
Portability	Low	**Very Good**

Programming Model Centric Interactive Debugging

Capture Mechanism Evaluation and Alternatives



	Breakpoints and Debug Information	**Preloaded Library**	**Specialized Debug Module**
Capturable Info.	**High**	Limited to API	**Full**
Execution Overhead	Significant	Limited	Limited
Cooperation btw. Debug and Env.	**None**	Low	Strong
Portability	Low	**Very Good**	Vendor Specific

Agenda

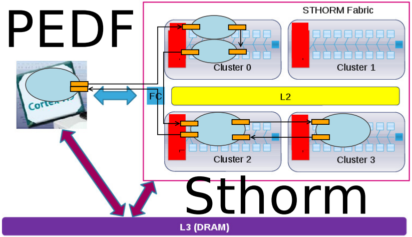
1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

3. mcGDB Case-Studies

mcGDB Case-Studies

A PEDF Dataflow H.264 Video Decoder

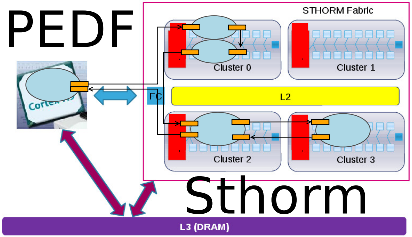


Dataflow Environment (PEDF)

- Dynamic dataflow programming
- Good for multimedia application
- No verification/validation help
- Heterogeneous computing:
  - actors ⇒ HW accelerators

mcGDB Case-Studies

A PEDF Dataflow H.264 Video Decoder



logo by bullboykennels

Dataflow Environment (PEDF)

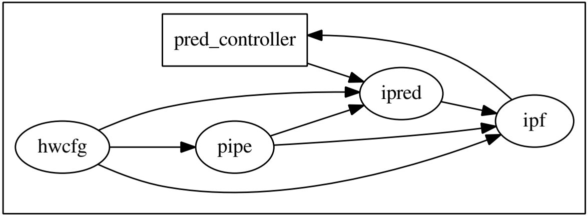
- Dynamic dataflow programming
- Good for multimedia application
- No verification/validation help
- Heterogeneous computing:
  - actors ⇒ HW accelerators

- Flexible video decoding standard
  - for HD television, blu-ray disks, broadcast, telephony, . . .
- Good dataflow decomposition
- Developed to validate PEDF design

mcGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can GDB help us?

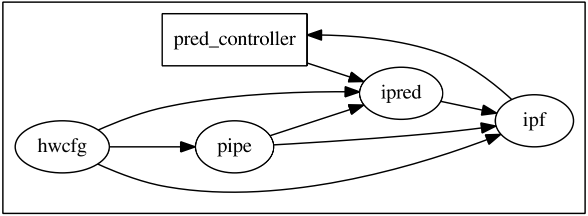
hint: not much!



(static graph provided by the compiler)

mcGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can GDB help us?

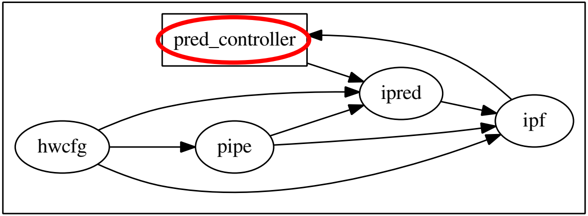


(gdb) info threads

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

mcGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can GDB help us?

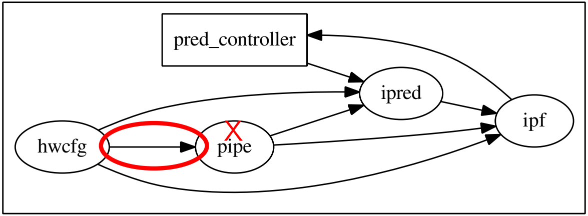


(gdb) thread apply all where

```
Thread 1 (Thread 0xf7e77b):
#0  0xf7ffd430 in __kernel_vsyscall ()
#1  0xf7fcd18c in pthread_cond_wait@ ()
#2  0x0809748f in wait_for_step_completion(struct... *)
#3  0x0809596e in pred_controller_work_function()
#4  0x08095cbc in entry(int, char**) ()
#5  0x0809740a in host_launcher_entry_point ()
```

MCGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can GDB help us?



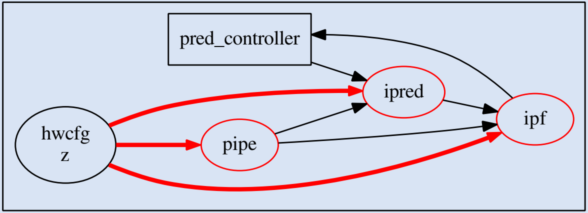
(gdb) thread apply all where

```
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=0xa0d18)
#4 0x080c12f0 in sc_core::sc_thread_cor_fn (arg=0xa0a3598)
```

MCGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can mcGDB help us?

(mcgdb) info graph

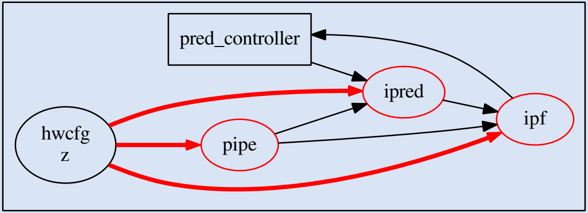


- `pipe`, `ipred` and `ipf` are blocked
- `hwcfg` is asleep

mcGDB Case-Studies: A PEDF Dataflow H.264 Video Decoder

The application is frozen, how can mcGDB help us?

(mcgdb) info graph



(mcgdb) info actors +state

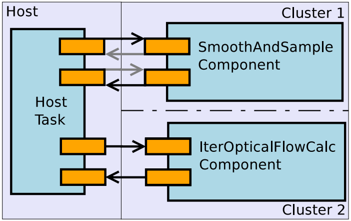
```
#0 Controller ‘pred_controller’:
    Blocked, waiting for step completion
#1/2/3 Actor ‘pipe/ipref/ipf’:
    Blocked, reading from #4 ‘hwcfg’
#4 Actor ‘hwcfg’:
    Asleep, Step completed
```

mcGDB Case-Studies

A Feature Tracker Based on NPM Components

Component Framework (NPM)

- Low-level access to STHORM architecture
- Optimized communication components
- 1 component per cluster and fork-join //



mcGDB Case-Studies

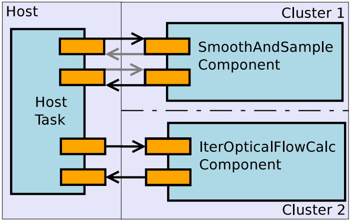
A Feature Tracker Based on NPM Components

Component Framework (NPM)

- Low-level access to STHORM architecture
- Optimized communication components
- 1 component per cluster and fork-join //

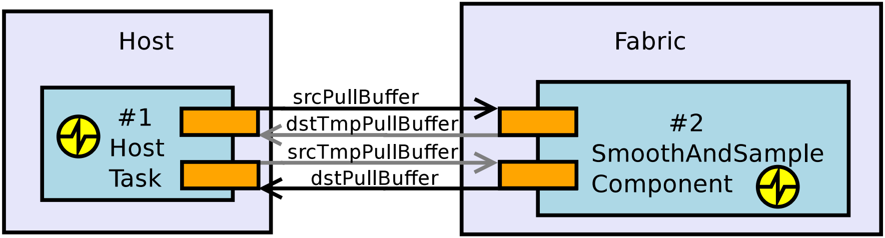
PKLT Feature Tracker

- Track interesting features btw. frames
- Part of an augmented reality application



mcGDB Case-Studies

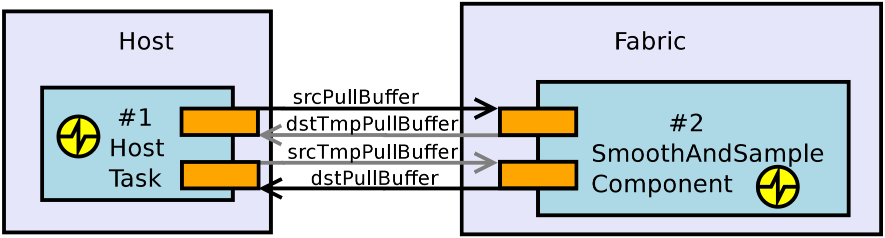
A Feature Tracker Based on NPM Components



```
(mcgdb) info component 2 +interfaces +counts
#2 Component[SmoothAndSampleComponent.so]
    srcPullBuffer #35 msgs
    dstTmpPushBuffer #36 msgs
    srcTmpPullBuffer #35 msgs
    dstPushBuffer #34 msgs
```

mcGDB Case-Studies

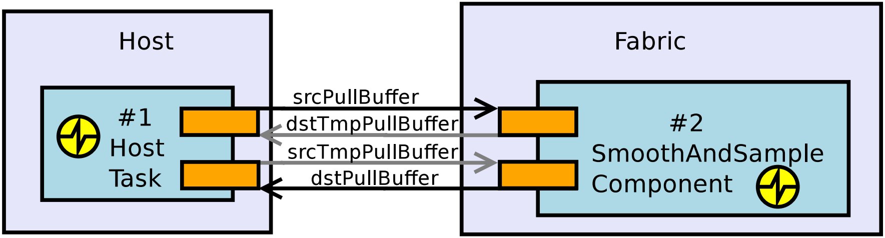
A Feature Tracker Based on NPM Components



```
(mcgdb) info component 2 +interfaces +counts
#2 Component[SmoothAndSampleComponent.so]
    srcPullBuffer #35 msgs
    dstTmpPushBuffer #36 msgs
    srcTmpPullBuffer #35 msgs
    dstPushBuffer #34 msgs
```

MCGDB Case-Studies

A Feature Tracker Based on NPM Components



(mcgdb) info component 2 +interfaces +counts

```
#2 Component[SmoothAndSampleComponent.so]
    srcPullBuffer #35 msgs
    dstTmpPushBuffer #36 msgs
    srcTmpPullBuffer #35 msgs
    dstPushBuffer #34 msgs
```

- that was a bug of the application! (msg sent to the wrong interface)

mcGDB Case-Studies

A Feature Tracker Based on NPM Components

Excerpt from a 300-line-of-code file

```
/* Compute last lines if necessary */
if (tmp_size > 0) {
    ...
    /* Transmit the last lines computed */
    CALL(srcTmpPullBuffer, release)(...);
    CALL(dstTmpPushBuffer, push)(...);
}
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming

OpenCL

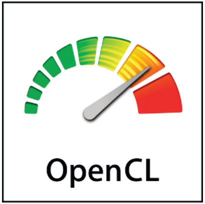
Kernel Programming Standard

(no environment distinction for us)

- Running on STHORM, but primarily used with GPU

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming



Kernel Programming Standard

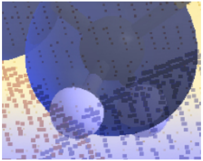
(no environment distinction for us)

- Running on STHORM, but primarily used with GPU
- MC debugging only on host-side

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming

OpenCL



BigDFT

Kernel Programming Standard

(no environment distinction for us)

- Running on STHORM, but primarily used with GPU
- MC debugging only on host-side

Density functional theory solver based on Daubechies wavelets. (electronic structure calculations)

- High performance requirements
- Hybrid CPU/GPU computing
- MPI - OpenCL (Fortran / C)

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming

Generic OpenCL skeleton

```
cl_kernel ker = clCreateKernel(...);

cl_buffer buf = clCreateBuffer(...);

clSetKernelArg(ker, 0, buf, ...);
clSetKernelArg(ker, 1, N, ...);

clEnqueueNDRangeKernel(ker, ...);

clEnqueueReadBuffer(buf, ...);
```

That's common operations found in any OpenCL code ...

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

So let's instrument these events …

```
cl_program clCreateProgramWithSource (context, code, ...);
cl_kernel  clCreateKernel (program, kernel_name, ..);

cl_int clSetKernelArg (kernel, arg_index,..., arg_val);
cl_int clEnqueueNDRangeKernel (cmd_q, kernel, work_dim);

cl_mem clCreateBuffer (context, size, host_ptr, ..);
cl_int clEnqueueReadBuffer (cmd_q, buf, ..., ptr, ...);
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

So let's instrument these events ...

```
cl_program clCreateProgramWithSource (context, code, ...);
cl_kernel  clCreateKernel (program, kernel_name, ..);

cl_int clSetKernelArg (kernel, arg_index,..., arg_val);
cl_int clEnqueueNDRangeKernel (cmd_q, kernel, work_dim);

cl_mem clCreateBuffer (context, size, host_ptr, ..);
cl_int clEnqueueReadBuffer (cmd_q, buf, ..., ptr, ...);
```

... to implement model-centric debugging commands:

```
(mcgdb) info programs
(mcgdb) info kernels [name|ID] {+where|+params}
(mcgdb) info buffers [ID] {+where|+params}
```

MCGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

So let's instrument these events ...

```
cl_program clCreateProgramWithSource (context, code, ...);
cl_kernel  clCreateKernel (program, kernel_name, ..);

cl_int clSetKernelArg (kernel, arg_index,..., arg_val);
cl_int clEnqueueNDRangeKernel (cmd_q, kernel, work_dim);

cl_mem clCreateBuffer (context, size, host_ptr, ..);
cl_int clEnqueueReadBuffer (cmd_q, buf, ..., ptr, ...);
```

... to implement model-centric debugging commands:

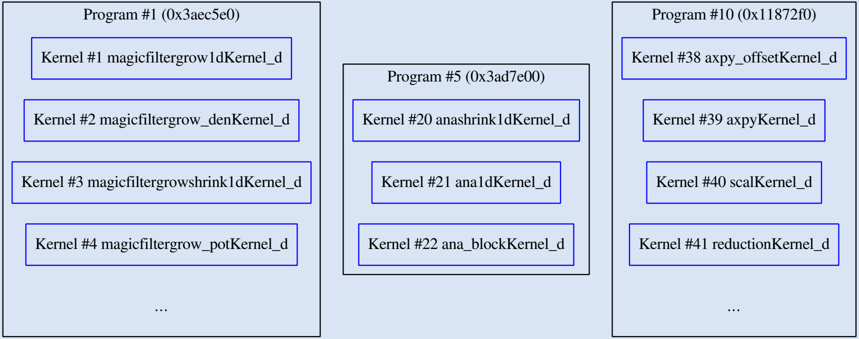
```
(mcgdb) info programs
(mcgdb) info kernels [name|ID] {+where|+params}
(mcgdb) info buffers [ID] {+where|+params}

(mcgdb) kernel [name|ID] catch {all|enqueue|set_arg|...}
(mcgdb) buffer [ID] catch {all|transfer|read|write|...}
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

(mcgdb) info programs +kernels



Program #1 (0x3aec5e0)

Kernel #1 magicfiltergrow1dKernel_d

Kernel #2 magicfiltergrow_denKernel_d

Kernel #3 magicfiltergrowshrink1dKernel_d

Kernel #4 magicfiltergrow_potKernel_d

...

Program #5 (0x3ad7e00)

Kernel #20 anashrink1dKernel_d

Kernel #21 ana1dKernel_d

Kernel #22 ana_blockKernel_d

Program #10 (0x11872f0)

Kernel #38 axpy_offsetKernel_d

Kernel #39 axpyKernel_d

Kernel #40 scalKernel_d

Kernel #41 reductionKernel_d

...

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

```
(mcgdb) info programs +kernels
```

```
(mcgdb) info kernels 41 +where +use_count +handle
```

```
Kernel #41  reductionKernel_d
  Creation stack:
    #0 0x0912a13 in create_reduction_kernels (...)
    #1 0x0900a10 in create_kernels (...)
    #2 0x0902744 in ocl_create_command_queue_id_ (...)
  Use count: 5
  Handle: (cl_kernel) 0x3aea590
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

```
(mcgdb) info programs +kernels
```

```
(mcgdb) info kernels 41 +where +use_count +handle
```

```
Kernel #41  reductionKernel_d
  Creation stack:
    #0 0x0912a13 in create_reduction_kernels (...)
    #1 0x0900a10 in create_kernels (...)
    #2 0x0902744 in ocl_create_command_queue_id_ (...)
  Use count: 5
  Handle: (cl_kernel) 0x3aea590
```

```
cl_kernel kernel = 0x3aea590 = ...  ; //somewhere in the code
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

```
(mcgdb) info kernels 41 +where +use_count +handle
```

```
Kernel #41  reductionKernel_d
  Creation stack:
    #0 0x0912a13 in create_reduction_kernels (...)
    #1 0x0900a10 in create_kernels (...)
    #2 0x0902744 in ocl_create_command_queue_id_ (...)
  Use count: 5
  Handle: (cl_kernel) 0x3aea590
```

```
cl_kernel kernel = 0x3aea590 = ...  ; //somewhere in the code
```

```
(gdb) print kernel, print *kernel **native support**
```

```
$1 = (cl_kernel) 0x3ae9d50
$2 = <incomplete type>
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Architecture Representation

```
(mcgdb) info kernels 41 +where +use_count +handle

Kernel #41  reductionKernel_d
  Creation stack:
    #0 0x0912a13 in create_reduction_kernels (...)
    #1 0x0900a10 in create_kernels (...)
    #2 0x0902744 in ocl_create_command_queue_id_ (...)
  Use count: 5
  Handle: (cl_kernel) 0x3aea590
```

```
cl_kernel kernel = 0x3aea590 = ...  ; //somewhere in the code
```

```
(mcgdb) info kernels +handle=0x3aea590
Kernel #41  reductionKernel_d
  Creation stack:                          (++ with GDB 7.6
  ...                                       pretty printers)
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Why Execution Visualization ?

Why Execution Visualization ?

let's consider an example ...

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Why Execution Visualization ?

C code

```
reductionKernel (int n, double *in, double *out){...}
checkStatus(int *ptr, char *msg) { if(ptr == 0) exit(-1);}

void main()
{
 double *in = malloc(...) ; checkStatus(in,"in failed");
 double *out = malloc(...); checkStatus(out,"out failed");

 initialize(in);
 reductionKernel(N, in, out);
 // free ...
}
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Why Execution Visualization ?

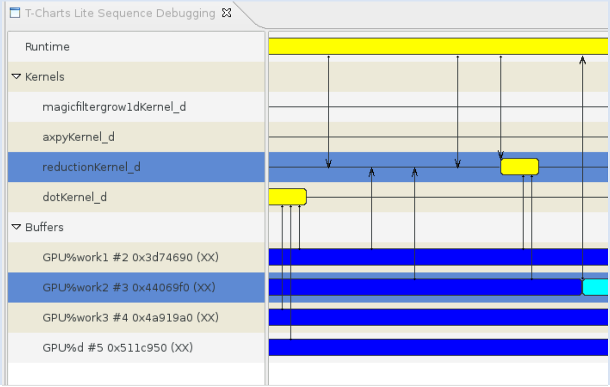
or in one page:

```
/* Instantiate the runtime.  */
command_queue = clCreateCommandQueue((*context)->context, aDevices[0], 0, &ciErrNum);
kerns->reduction_kernel_d=clCreateKernel(reductionProgram, "reductionKernel_d",&ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create kernel!");
/* Allocate the buffers on the GPU.  */
*buff_ptr = clCreateBuffer((*context)->context,  CL_MEM_READ_ONLY, *size, NULL, &ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create read buffer!");
/* Push the initial values to the GPU memory.  */
cl_int ciErrNum = clEnqueueWriteBuffer((*command_queue)->command_queue,  *buffer, CL_TRUE, 0, *size, p...
oclErrorCheck(ciErrNum,"Failed to enqueue write buffer!");
/* Set the kernel parameters.  */
clSetKernelArg(kernel, i++,sizeof(*ndat), (void*)ndat); clSetKernelArg(kernel, i++,sizeof(*in), (void*...
clSetKernelArg(kernel, i++,sizeof(*out), (void*)out);   clSetKernelArg(kernel, i++,sizeof(cl_dbl)*blk_...
/* Trigger the kernel execution.  */
size_t localWorkSize[] = { block_size_i };
size_t globalWorkSize[] ={ roundUp(blk_size_i*2,*ndat)/2 };
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_queue, kernel, 1, NULL, globalWorkSz, localWo...
oclErrorCheck(errNum,"Failed to enqueue reduction kernel!");
/* Get the result back.  */
cl_int ciErrNum = clEnqueueReadBuffer((*command_queue)->command_queue, *input, CL_TRUE, 0, sizeof(cl_d...
oclErrorCheck(ciErrNum,"Failed to enqueue read buffer!");
/* Then release the memory ... */
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Execution Visualization

(mcgdb) print_flow (an Eclipse visualization engine)



T-Charts Lite Sequence Debugging

Runtime

Kernels

magicfiltergrow1dKernel_d

axpyKernel_d

reductionKernel_d

dotKernel_d

Buffers

GPU%work1 #2 0x3d74690 (XX)

GPU%work2 #3 0x44069f0 (XX)

GPU%work3 #4 0x4a919a0 (XX)

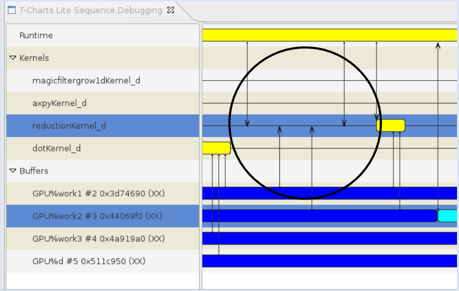
GPU%d #5 0x511c950 (XX)

- updated on user request, or
- automatically on execution stops, step-by-step, ...

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Execution Visualization

(mcgdb) print_flow (an Eclipse visualization engine)


T-Charts Lite Sequence Debugging
Runtime
Kernels
magicfiltergrow1dKernel_d
axpyKernel_d
reductionKernel_d
dotKernel_d
Buffers
GPU%work1 #2 0x3d74690 (XX)
GPU%work2 #3 0x44069f0 (XX)
GPU%work3 #4 0x4a919a0 (XX)
GPU%d #5 0x511c950 (XX)

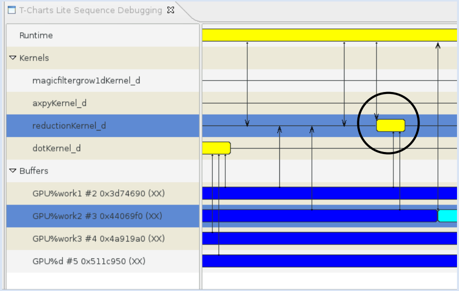
- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers

```
clSetKernelArg(kernel, i++, sizeof(*ndat),(void*)ndat);
clSetKernelArg(kernel, i++, sizeof(*in), (void*)in);
clSetKernelArg(kernel, i++, sizeof(*out), (void*)out);
clSetKernelArg(kernel, i++, sizeof(*sz), (void*)sz);
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Execution Visualization

(mcgdb) print_flow (an Eclipse visualization engine)


T-Charts Lite Sequence Debugging
Runtime
Kernels
magicfiltergrow1dKernel_d
axpyKernel_d
reductionKernel_d
dotKernel_d
Buffers
GPU%work1 #2 0x3d74690 (XX)
GPU%work2 #3 0x44069f0 (XX)
GPU%work3 #4 0x4a919a0 (XX)
GPU%d #5 0x511c950 (XX)

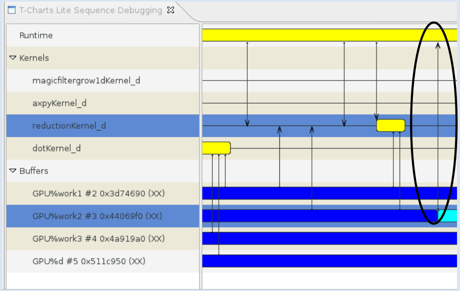
- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved

```
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_q,
                                  kernel, 1, NULL, globalWorkSz,
                                  localWorkSize, 0, NULL, NULL);
```

mcGDB Case-Studies

OpenCL and BigDFT Kernel Programming: Execution Visualization

(mcgdb) print_flow (an Eclipse visualization engine)


T-Charts Lite Sequence Debugging
Runtime
Kernels
magicfiltergrow1dKernel_d
axpyKernel_d
reductionKernel_d
dotKernel_d
Buffers
GPU%work1 #2 0x3d74690 (XX)
GPU%work2 #3 0x44069f0 (XX)
GPU%work3 #4 0x4a919a0 (XX)
GPU%d #5 0x511c950 (XX)

- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved
- Retrieve the result
  - buffer content is saved

```
cl_int ciErrNum = clEnqueueReadBuffer(
                      (*command_queue)->command_queue,
                      *input, CL_TRUE, 0, sizeof(cl_double),
                      out, 0, NULL, NULL);
```

Agenda

Conclusions and Future Work

Conclusions and Future Work

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

Our contribution: model-centric interactive debugging, applied to

- Component-software engineering (SCOPES '12)
- Dataflow programming (SAC and HIPS '13)
- Kernels for accelerator programming

Conclusions and Future Work

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

Our contribution: model-centric interactive debugging, applied to

- Component-software engineering (SCOPES '12)
- Dataflow programming (SAC and HIPS '13)
- Kernels for accelerator programming

Proof-of-concept: MCGDB, a prototype for STHORM platform

- Extends GDB and its Python interface:
  - Framework for model-centric debugging
  - Interface patches contributed to the community
- Usage studied through embedded and scientific applications

Conclusions and Future Work

Perspectives with programming-model centric debugging:

- Industrial side
  - Strengthen the implementation for production
  - Conduct extensive impact studies
  - Integrate within graphical debugging and visualization environments
- Research side
  - Apply to different programming models
    - multi-level of abstraction for embedded systems,
    - hardware (ARM big.LITTLE architecture)
  - Continue the study on visualization-assisted interactive debugging
  - Enrich debugging information generated by compilers
    - work funded by NANO 2017 R&D project

Publications

- Kevin Pouget, Marc Pérache, Patrick Carribault, and Hervé Jourdren.
  User level DB: a debugging API for user-level thread libraries. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–7, 2010.

- Kevin Pouget, Miguel Santana, Vania Marangozova-Martin, and Jean-François Mehaut.
  Debugging Component-Based Embedded Applications. In *Joint Workshop Map2MPSoC (Mapping of Applications to MPSoCs) and SCOPES (Software and Compilers for Embedded Systems)*, St Goar, Germany, may 2012. Published in the ACM library.

- Kevin Pouget, Patricia López Cueva, Miguel Santana, and Jean-François Méhaut.
  Interactive Debugging of Dynamic Dataflow Embedded Applications. In *Proceedings of the 18th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, Boston, Massachusetts, USA, may 2013. Held in conjunction of IPDPS.

- Kevin Pouget, Patricia López Cueva, Miguel Santana, and Jean-François Méhaut.
  A novel approach for interactive debugging of dynamic dataflow embedded applications. In *Proceedings of the 28th Symposium On Applied Computing (SAC)*, pages 1547–1549, Coimbra, Portugal, apr 2013.

Thanks for your attention

STMicroelectronics
LIG
University of Grenoble

Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget

Under the supervision of:

Dr. Miguel Santana, *STMicroelectronics*

Prof. Jean-François Méhaut, *CEA/UJF*

Thesis Defense, Grenoble — February 3rd, 2014