Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget, UJF-LIG, STMicroelectronics
Miguel Santana, STMicroelectronics
Jean-François Méhaut, UJF-CEA/LIG
Introduction
Embedded Systems and MPSoC

Consumer Electronics Devices
- 4K digital televisions
- Smartphones
- Hand-held music players
- High-resolution multimedia apps
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...
# Introduction

## Embedded Systems and MPSoC

### Consumer Electronics Devices

- 4K digital televisions
- Smartphones
- Hand-held music players
- **High-resolution multimedia apps**
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...

⇒ high performance expectations.
Introduction
Embedded Systems and MPSoC

*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

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

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

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
    • Workshop and our research effort
Agenda

1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

3. Model-Centric Debugger Case-Study
Agenda

1. Background: MPSoC Programming and Debugging

2. Programming Model Centric Interactive Debugging

3. Model-Centric Debugger Case-Study
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System on-a-Chip

- More parallelism
- More programmability

Multi-core CPU

Many-core

GPU

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

How to program such complex architectures?
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System on-a-Chip

- More parallelism
- More programmability

Multi-core CPU | Many-core | GPU

• Many-core processor for embedded systems
• Low energy-consumption
• Heterogeneous computing power
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

**MultiProcessor System on-a-Chip**

- More parallelism
- More programmability

<table>
<thead>
<tr>
<th>Multi-core CPU</th>
<th>Many-core</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Multi-core CPU" /></td>
<td><img src="image2.png" alt="Many-core" /></td>
<td><img src="image3.png" alt="GPU" /></td>
</tr>
</tbody>
</table>

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

How to program such complex architectures?
Background: MPSoC Programming and Debugging
Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to a hardware-independent interface
- **Separation of concerns** between application / lower levels
Background: MPSoC Programming and Debugging

Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

→ application written on top of an *abstract* machine
Background: MPSoC Programming and Debugging

Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

→ application written on top of an *abstract* machine

... implemented by *supportive environments*: programming frameworks, runtime libraries, APIs
Background: MPSoC Programming and Debugging
Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

→ application written on top of an *abstract* machine

... implemented by *supportive environments*: programming frameworks, runtime libraries, APIs
Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs
Background: MPSoC Programming and Debugging
Programming Models for ST MPSoCs

Components

- code/data encapsulation
- lang.-free interfaces

Dataflow

- Type 1 interface
- Type 2 interface
- «Runnable» interface
- Interface not connected
Background: MPSoC Programming and Debugging
Programming Models for ST MPSoCs

Components

- actor 1
- actor 2
- actor 3
- actor 4

Dataflow

- streams of data
- implicit parallelism
Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

<table>
<thead>
<tr>
<th>Components</th>
<th>Dataflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>• code/data encapsulation</td>
<td>• streams of data</td>
</tr>
<tr>
<td>• lang.-free interfaces</td>
<td>• implicit parallelism</td>
</tr>
</tbody>
</table>

large programming domain coverage...
## Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

<table>
<thead>
<tr>
<th>Components</th>
<th>Dataflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>• code/data encapsulation</td>
<td>• streams of data</td>
</tr>
<tr>
<td>• lang.-free interfaces</td>
<td>• implicit parallelism</td>
</tr>
</tbody>
</table>

large programming domain coverage...

... but what about Verification & Validation of MPSoC applications?
Background: MPSoC Programming and Debugging
Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)
- 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 (eg.: GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution
- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection
Background: MPSoC Programming and Debugging
Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution
- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection

... but nothing related to Supportive Environments ...
Background: MPSoC Programming and Debugging
Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution
- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection

... but nothing related to Supportive Environments ...

⇒ 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. Model-Centric Debugger Case-Study
Idea: Integrate programming model concepts in interactive debugging
Programming Model Centric Interactive Debugging

Provide a Structural Representation

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

Dataflow graph from the case-study
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

Slide 11 — Kevin Pouget — Programming-Model Centric Debugging — MAD Workshop'14, Athens, Greece — October 8th, 2014
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

Slide 11 — Kevin Pouget — Programming-Model Centric Debugging — MAD Workshop'14, Athens, Greece — October 8th, 2014
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

Slide 11 — Kevin Pouget — Programming-Model Centric Debugging — MAD Workshop’14, Athens, Greece — October 8th, 2014
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
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**

**break**
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 inspection
  - breakpoints
  - step-by-step
Slide 13 — Kevin Pouget — Programming-Model Centric Debugging — MAD Workshop'14, Athens, Greece — October 8th, 2014

Agenda

1. Background: MPSoC Programming and Debugging
2. Programming Model Centric Interactive Debugging
3. Model-Centric Debugger Case-Study
Model-Centric Debugger Case-Study
Proof-of-concept Environment

STHORM / Platform 2012
ST/CEA MPSoC research platform
• x86 platform simulators
Model-Centric Debugger Case-Study
Proof-of-concept Environment

STHORM Progr. Environments
- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

STHORM / Platform 2012
ST/CEA MPSoC research platform
- x86 platform simulators
Model-Centric Debugger Case-Study

Proof-of-concept Environment

The GNU Debugger
- Adapted to low level/C debugging
- Large user community

STHORM Progr. Environments
- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

STHORM / Platform 2012
ST/CEA MPSoC research platform
- x86 platform simulators

GDB

Model

Sthorm
Model-Centric Debugger Case-Study
Proof-of-concept Environment

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

STHORM Progr. Environments
- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

STHORM / Platform 2012
ST/CEA MPSoC research platform
- x86 platform simulators
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

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

Slide 15 — Kevin Pouget — Programming-Model Centric Debugging — MAD Workshop '14, Athens, Greece — October 8th, 2014
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study

Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework
Model-Centric Debugger Case-Study
Interpreting Execution Events

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

![Diagram showing the relationship between the host, dataflow-centric debugger, source-level debugger, and execution platform. The diagram illustrates the execution flow and breakpoints.]
Model-Centric Debugger Case-Study
Dataflow Video Decoder
Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?

*hint: not much!*

(static graph provided by the compiler)
Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?

(gdb) info threads

<table>
<thead>
<tr>
<th>Id</th>
<th>Target Id</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thread 0xf7e77b</td>
<td>0xf7fffd430 in __kernel_vsycall ()</td>
</tr>
<tr>
<td>* 2</td>
<td>Thread 0xf7e797</td>
<td>operator= (val=..., this=0xa0a1330)</td>
</tr>
</tbody>
</table>
Model-Centric Debugger Case-Study: Dataflow 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 ()
Model-Centric Debugger Case-Study: Dataflow 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)
```
Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can mcGDB help us?

(mcgdb) info graph
Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can mcGDB help us?

```
(mcgdb) info graph
```

```
pred_controller
  ↓
  iwcfg
  ↓
  pipe
  ↓
  ipred
  ↓
  ipf
```

```
(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
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, dataflow and kernel (GPU) 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, dataflow and kernel (GPU) programming

Proof-of-concept: mcGDB, a prototype for STHORM platform
- Extends GDB and its Python interface:
  - Interface patches contributed to the community
- Usage studied through embedded and scientific applications
Conclusions and Future Work

Perspectives for programming-model centric debugging:

- Industrial side
  - Strengthen the implementation
  - Conduct extensive impact studies

- Research side
  - Apply to different programming models
  - Visualization-assisted interactive debugging
  - Enrich debugging information generated by compilers
    - work funded by Nano 2017 R&D project
Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget, UJF-LIG, STMicroelectronics
Miguel Santana, STMicroelectronics
Jean-François Méhaut, UJF-CEA/LIG

MAD Workshop’14, Athens, Greece — October 8th, 2014
Publications

Kevin Pouget.

Kevin Pouget, Marc Pérache, Patrick Carribault, and Hervé Jourdren.

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.
Conclusions and Future Work

Interact with the Abstract Machine

**ST STHORM Platform — our reference MPSoC**

- CPU + 4 clusters × 16 lightweight/energy-efficient cores
  - ± dedicated hardware accelerators
- GPU-like architecture

How to program such complex architectures?
Conclusions and Future Work

Dataflow Video Decoder

**Dataflow Environment (PEDF)**
- Dynamic dataflow programming
- Good for multimedia application
- No verification/validation help
- Heterogeneous computing:
  - actors $\Rightarrow$ HW accelerators

**Flexible video decoding standard**
- for HD television, blu-ray disks, broadcast, telephony, . . .

**Good dataflow decomposition**
- Developed to validate PEDF design