Programming-Model Centric Debugging for OpenMP

(Philippe Virouleau), Kevin Pouget
Jean-François Méhaut, Miguel Santana

Université Grenoble Alpes / LIG, STMicroelectronics, France
Nano2017-DEMA project

OpenMPCon, Nara, Japan
October 3-5\textsuperscript{th}, 2016
Today’s in parallel computing

- Multicore processors everywhere
- High-level programming environments
- Efficient verification & validation tools
Today’s in parallel computing

- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...

- High-level programming environments

- Efficient verification & validation tools
Today’s in parallel computing

- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...
- High-level programming environments
  - tasks with data-dependencies,
  - fork-join parallelism
  - \[\Rightarrow\] OpenMP
- Efficient verification & validation tools
Today’s in parallel computing

- Multicore processors everywhere
  - HPC systems,
  - laptop and desktop computers,
  - embedded systems ...

- High-level programming environments
  - tasks with data-dependencies,
  - fork-join parallelism
  - $\implies$ OpenMP

- Efficient verification & validation tools
  - our research effort!
1. Research Context

2. Programming Model Centric Debugging

3. Building Blocks of a Model-Centric Debugger

4. OpenMP Case-Study Illustration
1. Research Context

2. Programming Model Centric Debugging

3. Building Blocks of a Model-Centric Debugger

4. OpenMP Case-Study Illustration
Source-Level Interactive Debugging (e.g. GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution
What about programming models?
What about programming models?

Source-level interactive debuggers operate at language-level. They have no knowledge about high-level abstract machines!
What about programming models?

**Source-level interactive debuggers** operate at **language-level**.
What about programming models?

**Source-level interactive debuggers** operate at **language-level**.

They have **no knowledge** about high-level **abstract machines**!
1. Research Context

2. Programming Model Centric Debugging

3. Building Blocks of a Model-Centric Debugger

4. OpenMP Case-Study Illustration
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.
Idea: Integrate programming model concepts in interactive debugging
1. Provide a Structural Representation
   - Draw *application architecture* diagrams
   - Represent the *relationship* between the entities

2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events

3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with *real* machine
1. Provide a Structural Representation
   - Draw application architecture diagrams
   - Represent the relationship between the entities

2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events

3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with real machine
1. Provide a Structural Representation
   - Draw application architecture diagrams
   - Represent the relationship between the entities

2. Monitor Dynamic Behaviors
   - Monitor the collaboration between the tasks
   - Detect communication, synchronization events

3. Interact with the Abstract Machine
   - Control the execution of the entities
   - Support interactions with real machine
1. Provide a Structural Representation
   ▶ Draw application architecture diagrams
   ▶ Represent the relationship between the entities

2. Monitor Dynamic Behaviors
   ▶ Monitor the collaboration between the tasks
   ▶ Detect communication, synchronization events

3. Interact with the Abstract Machine
   ▶ Control the execution of the entities
   ▶ Support interactions with real machine
Agenda

1. Research Context
2. Programming Model Centric Debugging
3. Building Blocks of a Model-Centric Debugger
4. OpenMP Case-Study Illustration
Detect and interpret the execution events of the runtime framework.
Detect and interpret the execution events of the runtime framework.
Detect and interpret the execution events of the runtime framework.
⇒ Detect and interpret the exec. events of the runtime framework
Detect and interpret the execution events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
Detect and interpret the execution events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework

- Task 0 Debugger
  - Breakpoint hit on <task creation>
- Source-level Debugger
  - Breakpoint hit at @ 0x126fd
- Execution Platform
  - Execution context
  - Data dependency
  - Task
⇒ Detect and interpret the exec. events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
Detect and interpret the execution events of the runtime framework.
Detect and interpret the exec. events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
⇒ Detect and interpret the exec. events of the runtime framework
Building Blocks of a Model-Centric Debugger

- Breakpoints and Debug Information:
  - Capturable Info.: High
  - Execution Overhead: Significant
  - Cooperation btw. Debug and Env.: None
  - Portability: Low
## Building Blocks of a Model-Centric Debugger

### Components
- **Application**
- **Preloaded Library**
- **Supportive Environment**

### Key Features
- **Capturable Info.**
  - High
- **Execution Overhead**
  - Significant
- **Cooperation btw. Debug and Env.**
  - None
- **Portability**
  - Low

### Comparison
- **Breakpoints and Debug Information**
  - High
  - Limited to API
- **Preloaded Library**
  - Limited
  - Low
  - Very Good

---

Philippe Virouleau

Programming-Model Centric Debugging

OpenMPCon
# Building Blocks of a Model-Centric Debugger

<table>
<thead>
<tr>
<th>Debugger</th>
<th>Breakpoints and Debug Information</th>
<th>Preloaded Library</th>
<th>Specialized Debug Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supportive Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capturable Info.</td>
<td>High</td>
<td>Limited to API</td>
<td>Full</td>
</tr>
<tr>
<td>Execution Overhead</td>
<td>Significant</td>
<td>Limited</td>
<td>Limited</td>
</tr>
<tr>
<td>Cooperation btw. Debug and Env.</td>
<td>None</td>
<td>Low</td>
<td>Strong</td>
</tr>
<tr>
<td>Portability</td>
<td>Low</td>
<td>Very Good</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>
1. Research Context

2. Programming Model Centric Debugging

3. Building Blocks of a Model-Centric Debugger

4. OpenMP Case-Study Illustration
Nano2017/Dema project

Debugging Embedded and Multicore Applications

ARM Juno

- asymmetric archi.
- ARM big.LITTLE + Mali GPU

OpenMP Parallel Programming

- fork/join multithreading
- tasks with dependencies
- GNU Gomp, Intel OpenMP, ...

mcGDB debugger

- Python extension of GDB
- support for dataflow, components, ...
- developed in partnership with ST
... control the execution of the entities ...
... control the execution of the entities ...
... control the execution of the entities ... 

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

```c
int main() {
    // beginning of main function
    #pragma omp parallel {
        // beginning of parallel region
        #pragma omp single {
            // execute single
        } // implicit barrier
        #pragma omp critical {
            // execute critical
        }
    }
}
```
... control the execution of the entities ...

```c
int main() {
    // beginning of main function
    #pragma omp parallel {
        // beginning of parallel region
        #pragma omp single {
            // execute single
        } // implicit barrier
        #pragma omp critical {
            // execute critical
        }
    }
}
```
... control the execution of the entities ...

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next
... control the execution of the entities ...

1. start
2. omp start
3. omp step
4. omp next barrier
5. omp critical next
6. omp critical next
7. omp critical next
8. omp critical next

int main() {
    // beginning of main function
    #pragma omp parallel {
        // beginning of parallel region
        #pragma omp single {
            // execute single
        } // implicit barrier
        #pragma omp critical { ③④
            ①// execute critical
        } ②
    }
}
... control the execution of the entities ...

1 start
2 omp start
3 omp step
4 omp next barrier
5 omp critical next
6 omp critical next
7 omp critical next
8 omp critical next

int main() {
   // beginning of main function
   #pragma omp parallel {
      // beginning of parallel region
   }
   // execute single
   #pragma omp single {
      // execute single
   } // implicit barrier

   #pragma omp critical {  
      // execute critical
   }  
}
... control the execution of the entities ...

1. start
2. `omp start`
3. `omp step`
4. `omp next barrier`
5. `omp critical next`
6. `omp critical next`
7. `omp critical next`
8. `omp critical next`

```c
int main() {
    // beginning of main function
    #pragma omp parallel {
        // beginning of parallel region
    }
}
```

```c
#pragma omp single {
    // execute single
} // implicit barrier
```

```c
#pragma omp critical {
    // execute critical
}④
```

Philippe Virouleau
Programming-Model Centric Debugging
... provide a structural representation
... provide details about entity state

1 fork-join \implies\ OpenMP sequence diagrams

2 task-based \implies\ mcGDB+Temanejo cooperation
... provide a structural representation
... provide details about entity state

1. fork-join $\implies$ OpenMP sequence diagrams

2. task-based $\implies$ mcGDB+Temanejo cooperation
... provide a structural representation
... provide details about entity state

1. **fork-join**  $\Rightarrow$ OpenMP sequence diagrams

2. **task-based**  $\Rightarrow$ mcGDB+Temanejo cooperation
1 start
2 omp start
3 omp step
4 omp next barrier
5 thread 2
6 omp critical next
7 omp critical next
8 omp critical next
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
1. start
2. `omp start`
3. `omp step`
4. `omp next barrier`
5. thread 2
6. `omp critical next`
7. `omp critical next`
8. `omp critical next`
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
OpenMP: mcGDB sequence diagram

1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
1. start
2. omp start
3. omp step
4. omp next barrier
5. thread 2
6. omp critical next
7. omp critical next
8. omp critical next
... provide a structural representation
... provide details about entity state

1 fork-join $\Rightarrow$ OpenMP sequence diagrams

2 task-based $\Rightarrow$ mcGDB + Temanefio cooperation
(HLRS Stuttgart) Temanejo ...

✔ is a great visualization tool for task debugging,
✘ and does not support source-level debugging.
(HLRS Stuttgart) Temanejo ...

✓ is a great visualization tool for task debugging,
✗ and does not support source-level debugging.

GDB/mcGDB ...

✗ has no visualization engine,
✓ but provides source debugging at language (gdb) and model level.
(HLRS Stuttgart) Temanejo ...

✓ is a great visualization tool for task debugging,
✗ and does not support source-level debugging.

GDB/mcGDB ...

✗ has no visualization engine,
✓ but provides source debugging at language (gdb) and model level.

So let’s combine them!
mcGDB – Temanejo cooperation:

**Temanejo**
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

**mcGDB**
- task graph and exec. events capture,
- advanced model-level exec. control.

**GDB**
- language and assembly level execution control, memory inspection.
mcGDB – Temanejo cooperation:

Temanejo
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

mcGDB
- task graph and exec. events capture,
- advanced model-level exec. control.

GDB
- language and assembly level execution control, memory inspection.
mcGDB – Temanejo cooperation:

**Temanejo**
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

**mcGDB**
- task graph and exec. events capture,
- advanced model-level exec. control.

**GDB**
- language and assembly level execution control, memory inspection.
mcGDB – Temanejo cooperation:

**Temanejo**
- task graph visualization
- simple model-level execution control.
- sequence diagram visualization.

**mcGDB**
- task graph and exec. events capture,
- advanced model-level exec. control.

**GDB**
- language and assembly level execution control, memory inspection.
Node color
▶ sources files
Node color
- sources files

Links color
- dependencies
Node color
- sources files
- debug state

Links color
- dependencies

Task 3 blocked
- blue finished
- purple blocked
Node color
- sources/files
- debug state
- executed by

Links color
- dependencies

Task 3 blocked
- blue finished
- purple blocked

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

Our contribution: model-centric interactive debugging

- mcGDB extends GDB through its Python interface:
  - Framework for model-centric debugging
  - Py interface patches contributed to the community
  - Source code soon-to-be open source (Apache licence)

- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control of fork-join and task-based execution
  - Better current-state understanding
    - OpenMP sequence diagrams
    - Temanejo graph visualization
Debugging high-level applications is challenging
Lack of information about programming models and frameworks

Our contribution: model-centric interactive debugging

mcGDB extends GDB through its Python interface:
- Framework for model-centric debugging
- Py interface patches contributed to the community
- Source code soon-to-be open source (Apache licence)

mcGDB OpenMP support:
- Developed for GNU GOMP and Intel OpenMP
- Better control of fork-join and task-based execution
- Better current-state understanding
  - OpenMP sequence diagrams
  - Temanejo graph visualization
Debugging high-level applications is challenging

Lack of information about programming models and frameworks

**Our contribution:** model-centric interactive debugging

- mcGDB extends GDB through its Python interface:
  - Framework for model-centric debugging
  - Py interface patches contributed to the community
  - Source code soon-to-be *open source* (Apache licence)

- mcGDB OpenMP support:
  - Developed for GNU GOMP and Intel OpenMP
  - Better control of fork-join and task-based execution
  - Better current-state understanding
    - OpenMP sequence diagrams
    - Temanejo graph visualization
Programming-Model Centric Debugging for OpenMP

(Philippe Virouleau), Kevin Pouget
Jean-François Méhaut, Miguel Santana

Université Grenoble Alpes / LIG, STMicroelectronics, France
Nano2017-DEMA project

OpenMPCon, Nara, Japan
October 3-5th, 2016