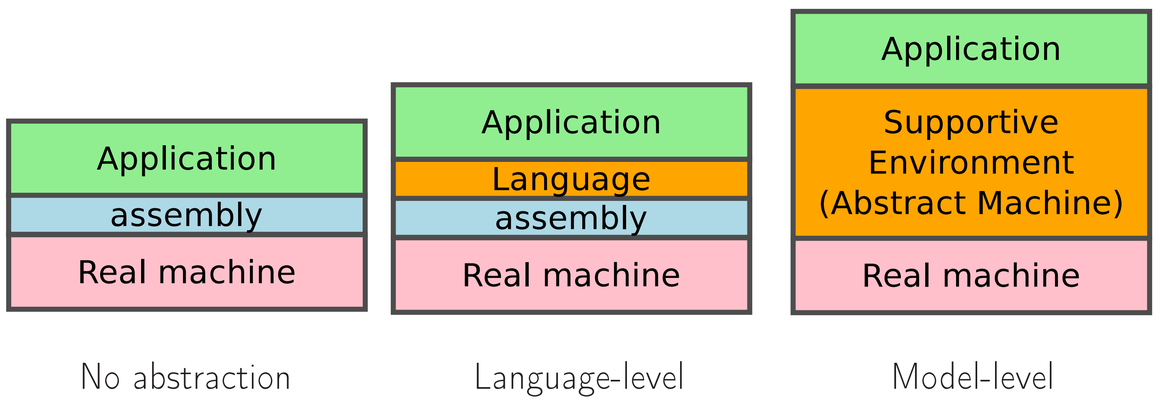
Programming-Model Centric Debugging for OpenMP

Kevin Pouget*, Jean-François Méhaut* and Miguel Santana†
*Université Grenoble Alpes, INRIA, LIG and †STMicroelectronics

5-8 juillet 2016 — COMPAS — Conférence d'informatique en Parallélisme, Architecture et Systèmes — Lorient France

Programming Model and Supportive Environments

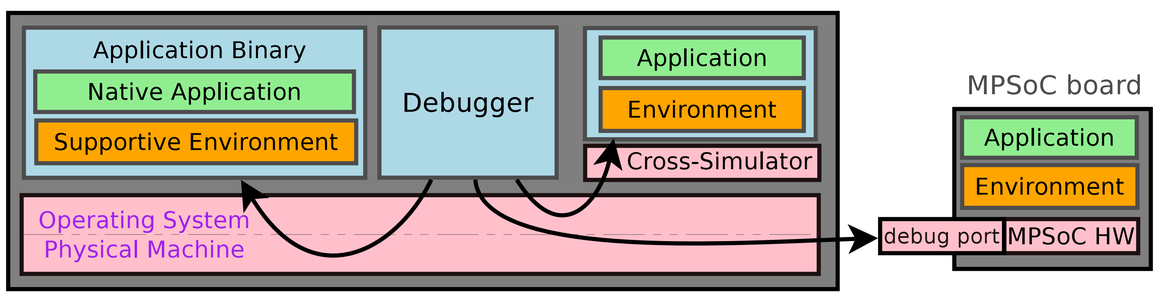
A model is an **abstract machine** providing **certain operations** to the programming level above and **requiring implementations** for each of these operations on all of the architectures below.



No abstraction — Language-level — Model-level

Interactive Source-Level Debugging

- Step by step execution
- Memory and processor inspection
- Static knowledge from compiler
- Dynamic knowledge from OS



OpenMP Programming

- Compiler-assisted parallelization
- Based on preprocessor `#pragma` directives
- Multiple implementations: GNU Gomp, Intel OpenMP, ...

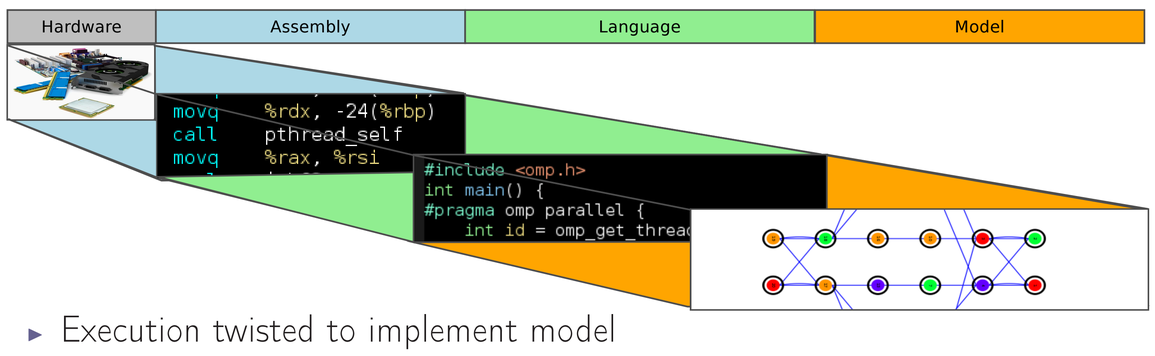
Fork-Join Multi-Threading

- Quick start and stop
- Parallel, single, critical, ... regions
- Automatic for-loop parallelization

Task with Dependencies

- In and out data dependencies ...
- ... on scalar value or array ranges
- Dynamic load balancing

OMP Debugging Challenges



- Execution twisted to implement model
  - outlining, callbacks from the environment...
  - implementation-specific transformations
- Multiple threads executing the same code
- Task-graph information buried inside OpenMP library
- Debuggers only work at language/assembly level

  ⇒ abstract machine state not known

```
(gdb) where
#0 0x00401502 in main._omp_fn.1() at omp_threads.c:45
#1 0xf7bc6f64 in gomp_barrier_handle_tasks ()
#2 0xf7bcad1c in gomp_team_barrier_wait_end ()
#3 0x004009c1 in main._omp_fn.0() at omp_threads.c:34
#4 0xf7bc83fe in gomp_thread_start()
#5 0xf79a34a4 in start_thread()
#6 0xf76e113d in clone()
```

Objective

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

Contribution

Programming-Model Centric 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

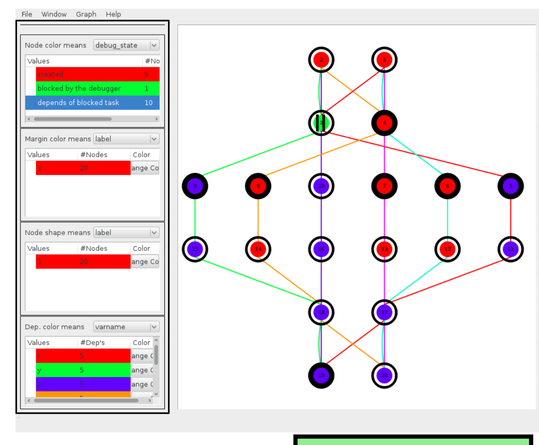
OpenMP Aware Debugging

- Task-graph reconstruction and visualization
  - with interpretation of data dependencies
- Execution control based on OpenMP semantic
  - catchpoints on parallel regions,
  - step-by-step execution of critical regions, ...
- Current region and task identification, parallelism level control, ...

Proof-of-Concept Environment

Temanejo

- task graph visualization
- simple model-level execution control
  - task-by-task stepping
  - task breakpoint and blocking

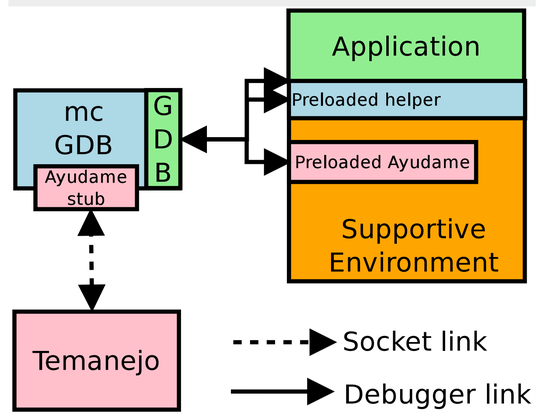


mcGDB

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

GDB — The Gnu Debugger

- Extendable with Python API
- Adapted to low level debugging
- Patches contributed to FSF



Conclusion and Perspectives

New interactive debugging approach

- matching programming and debugging level
- applied to OpenMP fork-join and task programming

Continue with performance debugging

1. extend GDB to support interactive profiling
2. turn towards mcGDB and parallel programming environments

*first.lastname@imag.fr http://www.liglab.fr/ — †first.lastname@st.com http://www.st.com/