How Model-Based Design Simplifies the Debugging of Many-Core Systems

Iuliana Bacivarov

Computer Engineering and Networks Laboratory, ETH Zürich

1st International Workshop on Multicore Application Debugging (MAD) 2013, 14-15 November 2013, München, Germany
Acknowledgements

- **team**
  Devesh Chokshi, Wolfgang Haid, Kai Huang, Shin-Haeng Kang, Pratyush Kumar, Devendra Rai, Lars Schor, Hoeseok Yang, Prof. Lothar Thiele

- **projects**
  EU-SHAPES, EU-PREDATOR, EU-COMBEST, EU-ARTISTDESIGN, EU-PRO3D, EU-EURETILE, nano-tera Extreme, nano-tera UltrasoundToGo
Current Embedded Systems are Complex

parallel applications

dynamic workloads

dynamic mapping

performance, real-time, power, and temperature

many-tile/many-core hardware
Debugging is Hard!
“Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.”

---- Wikipedia

“Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another.”

---- Wikipedia
Problems with Parallel Programming

What it Feels Like to Use the synchronized Keyword in Java

http://ptolemy.eecs.berkeley.edu/presentations/06/
Problems with Parallel Programming

- Threads are wildly nondeterministic
- The programmer’s job is to prune away the non-determinism by imposing constraints on execution order (e.g., mutexes)

“Humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings among even simple collections of partially ordered operations.”


- Nontrivial software written with threads, semaphores, and mutexes is incomprehensible to humans
- … and doesn’t deliver a rigorous, analyzable, and understandable model of concurrency.

http://ptolemy.eecs.berkeley.edu/presentations/06/
Key Concepts in Model-Based Design

- Models are composed to form designs.
- Models evolve during design.
- Specifications are executable models.
- Deployed code is generated from models.
- Modeling languages have formal semantics.
- Modeling languages themselves are modeled.

- For general-purpose software, this is about
  - Object-oriented design

- For embedded systems, this is about
  - Time
  - Concurrency
The Good News

Model-Based Design enables a ‘correct by design’ execution
Distributed Application Layer: model-based design & separation of concerns
Proposed by Kahn in 1974 as a general-purpose scheme for parallel programming
- READ: destructive and blocking
- WRITE: non-blocking
- FIFO: infinite size

Unique attribute: **determinate**

**Deterministic** model of computation
- Focus on causality, not order (implementation independent)
- Functional behavior is independent of timing (execution time, communication time, scheduling)
- Data-driven scheduling: processes run whenever they are ready
Application Specification: MPEG2 KPN

- Kahn process network
- Unique attribute: **determinate**

```plaintext
01 procedure INIT(ProcessData *p)
02     initialize();
03     end procedure
04
05 procedure FIRE(ProcessData *p)
06     fifo->READ(buf, size);
07     manipulate();
08     fifo->WRITE(buf, size);
09     end procedure
10
11 procedure FINISH(ProcessData *p)
12     cleanup();
13     end procedure
```
Execution Scenarios Specification

- Each application can:
  - START
  - STOP
  - PAUSE
  - RESUME

Application / run-time environment can request a scenario change.
Architecture Specification

Hierarchical architecture

e.g., Intel SCC
Application-to-Architecture Mapping

(mapping 1)

scenario1

(c1 c2)

(c3 c4)

(mapping 2)

scenario2

(c5 c6)

(c7 c8)

NoC

(c9 c10)

(c11 c12)

(c13 c14)

(c15 c16)

scenario1

scenario2

(mapping 1)

(mapping 2)

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2

scenario1

scenario2
Hierarchical Mapping Optimization – via Problem Decomposition

State-based decomposition

Architecture-based decomposition


From Specification to Analysis and Simulations

- automatic generation of different system ‘views’
  - analysis
  - functional simulation
  - cycle-/instruction-accurate simulation
  - execution on hardware

system specification

--

functional simulation

simulation/execution

MPA analysis model

Runtime System

- provides an implementation of the programming interface
  - inter-process communication (distributed memory)
  - multi-processing mechanisms
  - services to manage processes and channels at runtime

Inter-Process Communication

- shared vs. distributed memory
- on Intel SCC, RCKMPI lib. for inter-core communication
- one listener thread per core for all incoming traffic
- virtual buffer at sender to limit traffic
Multi Processing

- on top of Linux kernel – processes mapped onto POSIX threads
- data-driven execution – no global scheduler required

```c
void *producer_thread
    (void *arg) {
    Process *p = (Process*) arg;
    while (!p->stopped) {
        p->fire();
    }
}
```
Runtime Manager

- specified as a process network
  - one master process: manages dynamic execution
  - one slave process per core: manage processes and channels
Synthesis Backend

- **target platforms**
  - functional simulation on Linux
  - multi-cluster system:
    - each Linux server forms one cluster with multiple cores
    - Inter-cluster communication with MPI
  - Intel SCC
  - QUonG platform (INFN)

- mapping optimization
- runtime-manager synthesis
- process network synthesis

```plaintext
fire()
    read(...);
    ...
```

Process A --> core 1
Process B --> core 2
Process C --> core 2

M
S
S
main
(for each core)
Makefile
process wrappers
Deployment

DAL is available:
www.tik.ee.ethz.ch/~euretile/dal.php