DISTRIBUTED APPLICATION LAYER

towards seamless programming
of many-tile architectures

Iuliana Bacivarov, Hoeseok Yang,
Devendra Rai, and Lothar Thiele
ETH Zürich, Switzerland

CASTNESS’11, ROME - 17th–18th Jan. 2011
EURETILE in a Nutshell

European Reference Experimental Platform for Brain-Inspired Many-Tile Architectures

- HW prototype: 64 tiles/128 cores
- multiple, dynamic applications: HP numerical and DSP
- many-tile programming environment: scalability, dynamism, efficiency
- fault-tolerance: system-level
EURETILE in a Nutshell

European Reference Experimental Platform for Brain-Inspired Many-Tile Architectures

- HW prototype: 64 tiles/128 cores
- multiple, dynamic applications: HP numerical and DSP
- many-tile programming environment: scalability, dynamism, efficiency
- fault-tolerance: system-level
DOL* Programming Model

- Kahn process network MoC
  - expressiveness: static, single (stream-oriented) applications

- Mapping (incl. scheduling)
  - expressiveness: static mapping optimized at design-time

- Distributed memory architecture

*DOL – distributed operation layer
http://www.tik.ee.ethz.ch/~shapes
How to Specify Dynamic Applications?

Hierarchical spec

- **top-level**: scenarios FSM
- **bottom-level**: DOL KPN
- **control**: API, special proc.

![Diagram showing hierarchical spec with apps #1, #2, and #3, and scenarios connected with transitions t1 to t10.](image)

- **predictability**
- **scalability**
DAL API Provided by Programmer/HdS

01: procedure DAL_INIT(Process p)
02:   //initialize process state
03: end procedure
04:
05: procedure DAL_FIRE(Process p)
06:   DAL_READ(input, size, buf)
07:   //processing
08:   DAL_WRITE(output, size, buf)
09: end procedure
10:
11: procedure DAL_FINISH(Process p)
12:   //clean up, free mem, etc.
13: end procedure
14:
15: procedure DAL_SAVE(Process p)
16:   //save current process status
17: end procedure
18:
19: procedure DAL_RESTORE(Process p)
20:   //restore process status
21: end procedure

01: procedure DAL_START(Application A)
02:   //attach all processes to schedulers
03:   //call DAL_INIT for all processes
04:   //call repeatedly DAL_FIRE for all processes
05: end procedure
06:
07: procedure DAL_STOP(Application A)
08:   //permanently detach all processes from schedulers
09:   //call DAL_FINISH for all processes
10: end procedure
11:
12: procedure DAL_PAUSE(Application A)
13:   //temporarily detach all processes from schedulers
14:   //call DAL_SAVE for all processes
15: end procedure
16:
17: procedure DAL_RESUME(Application A)
18:   //re-attach processes to schedulers
19:   //call DAL_RESTORE for all processes
20: end procedure
How to Implement Dynamic Applications?

- **scalable solution:** hierarchically centralized control
  - clusters divide many-cores
  - local controller controls cluster
  - main controller controls local controllers

**Control:** centralized vs. distributed?

- **centralized control**
  - control tile(s): manage dynamic execution
  - "slave" tiles: execute (statically) mapped applications
  - communication: system command lines

- **scalable solution:** hierarchically centralized control
  - global controller
  - local controllers
Centralized Controller Implementation

- **Centralized control mechanisms**
  - in control tile
    - FSM for task control and migration
    - control API
  - in HdS of “slaves”
    - scheduling and FIFO management
    - “implicit” control tasks
EURETILE in a Nutshell

European Reference Experimental Platform for Brain-Inspired Many-Tile Architectures

- HW prototype: 64 tiles/128 cores
- multiple, dynamic applications: HP numerical and DSP
- many-tile programming environment: scalability, dynamism, efficiency
- fault-tolerance: system-level
Run-Time Mapping of Dynamic Applications

**Idea:** each “application scenario” linked to one (or several) mapping(s)

- **Trade-offs**
  - mapping optimality vs.
  - remapping cost
Run-Time Mapping of Dynamic Applications

Idea: each “application scenario” linked to one (or several) mapping(s)

- Trade-offs
  - mapping optimality vs. remapping cost

- Remapping is needed!
  - … necessary to accommodate new resource-demanding applications
  - … useful for overall optimality
  - … necessary for system-level fault tolerance
EURETILE in a Nutshell

European Reference Experimental Platform for Brain-Inspired Many-Tile Architectures

- HW prototype: 64 tiles/128 cores
- multiple, dynamic applications: HP numerical and DSP
- many-tile programming environment: scalability, dynamism, efficiency
- fault-tolerance: system-level
Faults in EURETILE
Fault-Tolerance at System-Level in DAL

- **remapping**
  - relocate applications/processes on faulty tiles/cores
  - … (mostly) at run-time
Fault! How to Proceed?

**Shutdown/Restart**
1) entire system shuts down
   2) remap with fault info
   3) restart the system

simple procedure
all contexts lost

**Reconfiguration**
1) stop processes on faulty proc
   2) remap with context
   3) restart processes

all contexts preserved
difficult to analyze/predict

VS.
Fault-Tolerance via Dynamic Remapping

- **Dynamic remapping**
  - feasible even in a busy system
  - composable scheduling needed; schedulability at run-time
Fault-Tolerance via Static Remapping

- **Static remapping**
  - move entire mapping to empty cluster
  - easy to determine (mapping topology preserved)
  - no analysis needed (only reconfiguration overhead)

- over-provisioning
Runtime Remapping Decisions

- **Remapping**
  - compact (computation)
  - guarantee performance, constraints

- **Fault recovery**
  1. shutdown and restart
  2. reconfiguration

- **Remapping**
  1. static
  2. semi-dynamic
  3. dynamic
Fault-Tolerance at System-Level in DAL

- **remapping**
  - relocate applications/processes on faulty tiles/cores
  - ... (mostly) at run-time

- **system-level analysis**
  - provide *guarantees*, e.g., guarantee real-time and temperature constraints
  - ... at design-time
EURETILE in a Nutshell

European Reference Experimental Platform for Brain-Inspired Many-Tile Architectures

- HW prototype: 64 tiles/128 cores
- multiple, dynamic applications: HP numerical and DSP
- many-tile programming environment: scalability, dynamism, efficiency
- fault-tolerance: system-level
DAL in EURETILE – Vision

1. Specification of app scenarios & coordination FSM
2. Functional simulation generation
3. Simulation on workstations
4. System synthesis (HdS generation)
5. Simulation/VP or emulation/HW
6. Simulation on workstations
7. Reconfiguration
8. Dynamic mapping adaption (on-line)
9. Performance analysis (off-line)
10. Evaluations on workstation
11. Architecture specification
12. Design space exploration of multi-applications/mappings

- On/off-line mapping spec.
- Calibration data back-annotation test & debug
- Performance data
Thank You!
Questions?

http://euretile.roma1.infn.it
iuliana.bacivarov@tik.ee.ethz.ch