A long and winding road towards predictability...

Eric JENN - IRT Saint-Exupéry
Agenda.

1. The context and the problems
2. Determinism by analysis
3. Determinism by design
4. Conclusion
The context

Who are we?
The context

IRT Saint-Exupery in Technogical Research Institute

- Focus on aeronautic, space, automotive
- Covers aspect related to materials, electrical systems, computing system, communication, artificial intelligence
- Projects co-funded by industry
  - Strongly driven by industrial needs
  - Focused on technological transfer (TRL 4-5, sometimes lower…)
- Work carried out by a composite team of engineers (seconded by their companies), academic researchers, post-docs
The problem

What was the question?
The problem

Dependability and performance

- **Dependability**: the extent to which confidence can be placed on the capability of the system to fulfil its intended purpose

- **Performance**: the efficiency with which a system fulfils its intended purpose
The problem

Emergence of new computation platforms

- Multiple cores (multi + many)
  - Complex cores
  - Heterogeneous cores
- SIMD units
- GPU
- FPGA
- AI accelerators
- Interconnect
- SDRAM
- ...

Master complexity
Use platforms efficiently (reduce margins)
Compliance with certification constraints
Reduce development (incl. V&V) costs
A long and winding road...

How to...

- ensure determinism and predictability?
  - Live with variability?
  - Reduce variability?
- chose effective / reasonable (cost effective) solution?
Predictability by analysis

How to model and analyse a platform?
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
- TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

- How to ensure the completeness of the analysis?

Phase 1 - Modeling

Phase 2 - Analysis
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
- TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

How to ensure the completeness of the analysis?

- Identify “components”, flows of transactions
- Capture elements using AADL
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
- TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

- How to ensure the completeness of the analysis?
  - Identify "components", flows of transactions
  - Capture elements using AADL
  - Translate to Prolog

```
system F
  features
    f1: in out event data port;
    f2: in out event data port;
  flows
    f_f : flow path f1->f2;
end F;
```

AADL

isComponentType('example','PUBLIC','F','SYSTEM','NIL',26).
isFeature('PORT','example','F','f1','IN OUT','EVENT DATA','NIL','NIL','NIL','NIL',28).
isFeature('PORT','example','F','f2','IN OUT','EVENT DATA','NIL','NIL','NIL','NIL',29).
isFlowSpec('PATH','example','F','NIL','f_f','f1','f2','NIL',31).
isComponentTypeEnd('example','PUBLIC','F','F',32).

Transform

Prolog
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
- TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

- How to ensure the completeness of the analysis?
  - Identify “components”, flows of transactions
  - Capture elements using AADL
  - Translate to Prolog
  - Query Prolog

```prolog
?- interference2(Initiator_1, Target_1, Initiator_2, Target_2, Crossings).
Initiator_1 = 'Top.a.a1',
Target_1 = Target_2, Target_2 = 'Top.e.e1',
Initiator_2 = 'Top.b.b1',
Crossings = ['Top.d.f.f2', 'Top.d.f.f1', 'Top.d.d3'] ;

Initiator_1 = 'Top.b.b1',
Target_1 = Target_2, Target_2 = 'Top.e.e1',
Initiator_2 = 'Top.a.a1',
Crossings = ['Top.d.f.f2', 'Top.d.f.f1', 'Top.d.d3'].
```
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
  - TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

- How to ensure the completeness of the analysis?
  - Identify “components”, flows of transactions
  - Capture elements using AADL
  - Translate to Prolog
  - Query Prolog
Identifying interference sources

Modeling for interference analysis

- Processeurs are (very) complex
  - TC277, more than 5000 pages
- Documentation is developer-oriented
- Documentation is not always correct / complete

- How to ensure the completeness of the analysis?
  - Identify “components”, flows of transactions
  - Capture elements using AADL
  - Translate to Prolog
  - Query Prolog

How to improve the docs? (standard format?)
Provide micro-benchmarks?
Use AI techniques?
How to improve accuracy?
Identifying interference sources

Using model checking

- Formal modeling of architecture using LNT and Fiacre
- Two methods
  - Patchcheck
  - SynCheck

```plaintext
property CHECKING_ARBITER_INTERFERENCE(LNT_MODEL, RESULT, i, j, k) is
  "a.bcg" = generation of "$LNT_MODEL.lnt"
  "a.bcg" |=with evaluator4 true*. "REQUEST_ISSUED !REQUEST ($i, $j, $k)". (not "REQUEST_GRANTED !REQUEST ($i, $j, $k)"). {REQUEST_GRANTED ?R:String where R<>'REQUEST ($i, $j, $k)"} >true ...
or
end property
```
Identifying interference sources

Using model checking

- Formal modeling of architecture using LNT and Fiacre
- Two methods
  - Patchcheck
  - SynCheck

Without interference

- REQUEST_ISSUED IR1
- REQUEST_SELECTED IR1
- REQUEST_GRANTED IR1

With interference

- REQUEST_ISSUED IR1
- REQUEST_SELECTED IR2
- REQUEST_SELECTED IR1
- REQUEST_GRANTED IR2
- REQUEST_GRANTED IR1
Identifying interference sources

Using model checking

- Formal modeling of architecture using LNT and Fiacre
- Two methods
  - Patchcheck
  - SyncCheck
Identifying interference sources

Using model checking

- Formal modeling of architecture using LNT and Fiacre
- Two methods
  - Patchcheck
  - SyncCheck

1st LTS: isolated model – M1

2nd LTS: non-isolated model – M2

Synchronous LTS: M1 \(\otimes\) M2

How to build the model?
How to trust the model?
Identify interferences

Using simulation

- Using the VLAB virtual platform
  - Timing modeling is not extremely accurate (this is not a cycle accurate simulator)
  - Identification of multiple transactions in the same time interval.

Not that accurate
Does not really identify interferences
Reduce interferences

Identify time-sensitive code

- Example: early empirical cache-sensitivity analysis
- QEMU model
- First, provide cache usage data
Reduce interferences

Identify time-sensitive code

- Example: early empirical cache-sensitivity analysis
- QEMU model
- First, provide cache usage data
- Application
  - Xilinx ZCU102 Board, with a single core cortex-r5
  - Running Polybench collection

Average Overhead: 12%

Average Relative Error: 3.27%
Reduce interferences

Identify time-sensitive code

- Example: early empirical cache-sensitivity analysis
- QEMU model
- First, provide cache usage data
- Application
  - Xilinx ZCU102 Board, with a single core cortex-r5
  - Running Polybench collection
- Second, provide user-level data

```c
void consumerThread()
{
    unsigned int array[SIZE][SIZE], copy[SIZE][SIZE] = (0);

    while(1)
    {
        ReadData();
        for (unsigned int i = 0; i < SIZE; i++)
        {
            for (unsigned int j = 0; j < SIZE; j++)
            {
                copy[i][j] = array[i][j];
            }
        }
        SendResults();
    }
}
```

\[
\text{copy}[i][j] = \text{array}[i][j]; \quad \text{copy}[j][i] = \text{array}[j][i];
\]

Lines [76;80] -> 88% hits, 12% misses
Lines [74;82] -> 86% hits, 14% misses
Lines [70;86] -> 50% hits, 50% misses

PoC limited to caches

Provide user-level data about interferences
Reduce interferences

On synchronous code

- **On Lustre code**
  - Application to INRIA’s LOPHT tool (KAIROS)
  - Reduction of DDR page changes

Each call induce
- Instructions pipeline break (at core level)
- Usually a DDR page change

- Linear code instructions
- DDR page change only when code size oversize a page
- Require a timed-triggered OS
Reduce interferences

On synchronous code

- **On Lustre code**
  - Application to INRIA's LOPHT tool (KAIROS)
  - Reduction of DDR page changes
  - Reduction of flash-level interferences using prefetching (on-going work)
Reduce interferences

On synchronous code

- **On Lustre code**
  - Application to INRIA’s LOPHT tool (KAIROS)
  - Reduction of DDR page changes
  - Reduction of flash-level interferences using prefetching (on-going work)

- **On PsyC code** (ASTERIOS)
  - Prevent simultaneous accesses to resources

![Diagram showing profile analysis and critical areas](image)
Static WCET analysis

Building the model

- A tedious activity...

<table>
<thead>
<tr>
<th>Instructions vs cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mtc0x0c00,d15</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 movh.a a2,0x7000</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 stw [a2] 0, d8</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 stw [a2] 15, d8</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 stw [a2] 32, d8</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 ldw d8,[a2] 64</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 add d7, d8, d5</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 mov d0,0</td>
<td>FE</td>
<td>DE</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 mtc0x0c00,r0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Better documentation?

Documentation for timing analysis?
How to gain confidence on the tool

- Two levels verification
  - Verifying the ISS
    - ISS generated from NMP files used to decode instructions in OTAWA
    - First step to support a new architecture
    - Compare the “processor state” between TSIM and OTAWA-ISS
      - Register values
      - Memory accesses
  - Verifying the abstract model
    - Is the value/address analysis correctly implemented?
    - Are “semantic instructions” correctly implemented for TriCore

Diagram:

- Binary → Decode the binary → Create CFGs and BBs → Compute time for each BBs → Generate and solve the ILP formula → WCET
- Hardware info → Constraint building → ILP creation → ILP solver invocation
- Identify Instructions → Create CFGs & BBs → Detect loops → Caller relations → Value/address analysis → Prog/Data cache analysis → Branch prediction → Flow facts
Reduce cost of analysis

How to trade-off accuracy and cost

- WCET analyzers based on AI are costly to develop
- Can we focus the effort on the most "pertinent" parts of the component (the one with the highest contribution)?

All instructions have the same, worst-case, latency.

- Functional units latencies
- Multiscalar
- Latencies of code memory areas
- Global buffer (flash-level cache)
- Caches
- Store buffer
- Branch prediction
- On-going work...
Predictability by construction

How to build a predictable platform?
Deploy functions to hardware

Deploy to FPGA

- Leverage SoCs embedding FPGAs (Zynq, Dahlia, etc.)
- Exploit parallelism
- By product of performance enhancement

- Example: OpenMP to HW

Sequential C code → OpenMP annotated C code → SpaceModules → High Level synthesis → Parallel HW Platform Parallel SW architecture

```c
int a = 2;
int b = 12;
#pragma omp target map(from:b)
{
    b = a + 2;
}
printf("b = %d\n", b);
// b = 4
```
Deploy functions to hardware

Deploy to FPGA

- Leverage SoCs embedding FPGAs (Zynq, Dahlia, etc.)
- Exploit parallelism
- By product of performance enhancement

- Example: OpenMP to HW

Sequential C code

OpenMP annotated C code

Parallel HW Platform
Parallel SW architecture

Needs a FPGA...

... usually dedicated to other demanding computations...
Deploy functions to hardware

New issues with parallelism

- Parallelism raises "new" issues with respect to determinism
  - Deadlocks
  - Race conditions...

- Parallelization frameworks generally not designed for safety critical systems...

Provide a “deterministic” version OpenMP

Prevent / detect non-deterministic effects
Determinism by construction

Use the PL to implement a timing-analysis friendly processor

- Split the problem (not every processing have the same level of temporal requirements…)
- Deploy time-critical function on deterministic processors
- Example: Berkeley's FlexPRET
- Combination of FlexPRET and synchronous execution model (Lustre/LOPHT and ForeC)

Berkeley’s FlexPRET (M. Zimmer’s PhD thesis)

Performances?

“Exotic” architecture…
Conclusion

Things are getting clearer…

- But problems are getting more (and more) complicated…

- But there is still quite a lot of work to be done (see all the boxes of the presentation)

- Guidance is still missing for industrial partners: Which approach shall I use to use for what result and with what confidence on the result?…
References
Publications

- **Interference analysis**
  - V. A. Nguyen, E. Jenn, W. Serwe, F. Lang, and R. Mateescu, ‘Using Model Checking to Identify Timing Interferences on Multicore Processors’, in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Available at [https://hal.inria.fr/hal-02462085](https://hal.inria.fr/hal-02462085).

- **WCET analysis**
  - W.-T. Sun, E. Jenn, and H. Cassé, ‘Build Your Own Static WCET analyser: the Case of the Automotive Processor AURIX TC275’, in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Available at [https://hal.archives-ouvertes.fr/hal-02507130](https://hal.archives-ouvertes.fr/hal-02507130).
Publications

- **Parallel applications**
  - V.-A. Nguyen, W. Serwe, R. Mateescu, and E. Jenn, ‘Hunting Superfluous Locks with Model Checking’, in From Software Engineering to Formal Methods and Tools, and Back, vol. 11865, Springer Verlag, 2019, p. 416-432. Available at [https://hal.inria.fr/hal-02314088](https://hal.inria.fr/hal-02314088)
  - R. Leconte, E. Jenn, G. Bois, and H. Guérard, ‘Make Life Easier for Embedded Software Engineers Facing Complex Hardware Architectures’ in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Available at [https://hal.archives-ouvertes.fr/hal-02474476](https://hal.archives-ouvertes.fr/hal-02474476)

- **Deterministic multicores**
  - A. Girault, N. Hili, E. Jenn, and E. Yip, ‘A Multi-Rate Extension of the ForeC Precision Timed Programming Language for Multi-Cores’, presented at the Forum Design Language, Southampton, United Kingdom

- **Simulation**