Precision Timed Infrastructure: Languages, Compilers, and Hardware with Ubiquitous Notion of Time

June 28, 2013

David Broman
broman@eecs.berkeley.edu

EECS Department
University of California, Berkeley, USA
and
Linköping University, Sweden

PRET Infrastructure at Berkeley

David Broman  Edward A. Lee
Jian Cai       Aviral Shrivastava
Hokeun Kim    Chris Shaver
Yooseong Kim  Michael Zimmer

Agenda

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Design Challenges
Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Design Challenges

Cyber-Physical Systems (CPS)

Industrial Robots
Power Plants
Aircraft
Cyber-Physical Co-Design Problem

Rapid development of CPS with high confidence of correctness is a co-design problem.

The design of

Physical system
(the plant)

influence each other

The design of

Cyber system:
Computation (embedded) + Networking

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Design Challenges
**Challenge #1:**
Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

The main challenge is to guarantee correct **timing behavior**.
Programming Model and Time

Timing is not part of the software semantics

Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

Traditional Approach

Our Objective

Make time an abstraction within the programming model

Timing is independent of the hardware platform (within certain constraints)

What is PRET?

PRET = PREcision-Timed


PRET Infrastructure

- PRET Language (Language with timing semantics)
- PRET Compiler (Timing aware compilation)
- PRET Hardware (Computer Architecture)
Detecting missed deadlines

<table>
<thead>
<tr>
<th>Missed deadline</th>
<th>Hard task</th>
<th>Firm task</th>
<th>Soft task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic consequence</td>
<td>Result is useless, but causes no damage</td>
<td>Result has still some utility</td>
<td></td>
</tr>
</tbody>
</table>

**Predictable timing**
- Guarantee correctness (WCET)

Processor frequency

**Task**
(clock cycles)

**Deadline**
(measured in e.g., ns)

**Time**

**Precision of timing**
- Level of nano seconds

**Repeatable timing**
- Same platform: Testability
- Changing platform: Portability

**Worst-Case Execution Time (WCET)**

**Measurement-based approach**
- Cannot guarantee to find WCET
- Applicable for any task

**Worst-case execution time (WCET)**

**Static program analysis approach**
- Upper bound of WCET
- Cannot handle any task (conservative)

**Challenges**
- To make it **safe**: upper_bound ≥ WCET
- To make it **tight**: minimize (upper_bound – WCET)
- Scalability: to handle large and complex programs

**WCET overview**
(Wilhelm et al., 2008)
What is our goal?

“Everything should be made as simple as possible, but not simpler“
attributed to Albert Einstein

Execution time should be as short as possible, but not shorter

Objective: Minimize for area, memory, energy, and execution time for non real-time tasks.
Challenge: Still guarantee to meet all timing constraints.

No point in making the execution time shorter, as long as the deadline is met.

No point in making the execution time shorter, as long as the deadline is met.

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Design Challenges

PRET Infrastructure Vision

Modeling Languages

Simulink/Stateflow (Mathworks)
Modelica (Modelica Associations)
Ptolemy II (Eker et al., 2003)
Modelyze (Broman and Siek, 2012)

Programming Languages

Real-time Concurrent C (Gehani and Ramamritham, 1991)
Real-Time Euclid (Klingerman & Stoyenko, 1986)

Assembly Languages

The assembly languages for today's processors lack the notion of time
The good news
Fortunately, electronics technology delivers highly reliable and precise timing

The bad news...
The chip architecture introduces highly non-deterministic behavior (e.g., using caches, pipelines etc.).

Rethink the ISA
Timing has to be a correctness property not only a performance (quality) property

PRET Machine
• Repeatable and predictable execution time (instructions)
• Repeatable memory access time
• Timing instructions for handling missed deadline detection

The good news
Fortunately, electronics technology delivers highly reliable and precise timing

The bad news...
The chip architecture introduces highly non-deterministic behavior (e.g., using caches, pipelines etc.).

PRET Machine
• Repeatable and predictable execution time (instructions)
• Repeatable memory access time
• Timing instructions for handling missed deadline detection

PTARM (ICCD’12)
• Replacing caches with scratchpads
• Use a thread-interleaved pipeline (4 threads)
• Timing instructions (delay until, exception-on-expire)
• Soft core on a Xilinx Virtex 5 FPGA

FlexPRET (work-in-progress)
• Dynamically change no of active threads (1-8)
• RISC-V ISA (Waterman, Lee, Patterson, Asanovi, 2011)
PRET Infrastructure Vision

**Modeling Languages**
- Simulink/Stateflow (Mathworks)
- Modelica (Modelica Associations)
- Ptolemy II (Eker et al., 2003)
- Giotto and E machine (Henzinger et al, 2003)
- Modelyze (Broman and Siek, 2012)

**Programming Languages**
- C with asm inline macros

**Assembly Languages**
- Difficult to compute WCET (e.g., determine loop bounds and infeasible paths)

**PRET ISA**

**PRETIL**
- Expose timing constructs
- Abstracting away memory hierarchy (scratchpad, DRAM etc.)

Our current work-in-progress is an extension to LLVM

PRET Infrastructure Vision

**Modeling Languages**
- Simulink/Stateflow (Mathworks)
- Modelica (Modelica Associations)
- Ptolemy II (Eker et al., 2003)
- Giotto and E machine (Henzinger et al, 2003)
- Modelyze (Broman and Siek, 2012)

**Programming Languages**
- C with asm inline macros

**Assembly Languages**
- Difficult to compute WCET (e.g., determine loop bounds and infeasible paths)

**PRET ISA**

**PRETIL**
- Expose timing constructs
- Abstracting away memory hierarchy (scratchpad, DRAM etc.)

Our current work-in-progress is an extension to LLVM
Meet the final deadline (MTFD). Specify upper timing bound (constraint) of 10µs.

Get time (nanoseconds) since boot time.

Sensing, computation, and actuation

Delay Until (DU) Specifies lower bound of 5µs.

Upper bound, MTFD, static analysis
Lower bound, DU, enforced at runtime

---

**Intermediate Language (ptLLVM) example**

```
1 %t1 = gt i64
2 mt i64 10000
3 ; ...computation...
4 %t2 = add i64 %t1, 5000
5 du i64 %t2
6 fd
```
Precision Timed Compiler (work-in-progress)

**Part I**
Cyber-Physical Systems

**Part II**
Precision Timed Infrastructure

**Part III**
Design Challenges

---

**ptc**
(Precision Timed Compiler)

Timing-aware compilation that includes WCET analysis. Bare metal, not executed on top of a RTOS.

A back-end compiler separated from the LLVM code base. Written in OCaml.

WCET analysis and backend compiler phases are seen as combined problems.

Aim of being modular and easy to extend.
A research platform with clearly separated libraries.

Currently targeting the new FlexPRET processor using the RISC-V ISA.

---

**Part III**
Design Challenges
Design Challenges

Precision Timed Languages

What is the minimal set of **timing instructions** for a PRET intermediate language?

How should **concurrency** be expressed? How can communication with precise timing be part of the languages?

How can we incorporate modeling of **distributed systems**?

How do we include clock synchronization in the infrastructure (e.g., IEEE 1588)

---

Part I

Cyber-Physical Systems

Part II

Precision Timed Infrastructure

Part III

Design Challenges

---

Part I

Cyber-Physical Systems

Part II

Precision Timed Infrastructure

Part III

Design Challenges

---

Design Challenges

Precision Timed Languages

How should **memory hierarchies** be handled?

Two extremes: (1) caches and (2) programmable DMA transfers. Golden middle way?

Precision Timed Hardware

How should **pipelines** be handled?

Thread interleaved pipelines guarantees non-interference at the expense of latency.

Can various threads have different performance and predictability guarantees?

Precision Timed Compilers

How can **Dynamic Voltage & Frequency Scaling (DVFS)** be combined with WCET analysis?
Design Challenges

Precision Timed Languages
Precision Timed Hardware
Precision Timed Compilers

How computer architecture-aware must a PRET compiler be? What, besides the ISA, should be the abstraction?

How can scratchpad memory allocation be combined with WCET analysis?

How can back-end compiler phases (instruction selection, register allocation, etc.) be optimized for worst-case instead of average-case? Can WCET analysis and back-end optimization be combined?

Conclusions

Main takeaway points

For CPS applications, time is a correctness factor – not just a performance (quality) factor

A PRET intermediate language language include timing semantics and abstracts away platform details.

PRET Hardware should give predictable timing behavior and provide hardware support for programming with real-time.

A PRET compiler should guarantee that all timing constraints are fulfilled when executed on a specific platform.

For more information see:
http://chess.eecs.berkeley.edu/pret/

Thank you for listening!