Predictable Computation and Time-Aware Semantics for Time-Coordinate Computation

Applications in Cyber-Physical Systems (CPS)

Intel, Oregon, December 3, 2015

David Broman

Associate Professor, KTH Royal Institute of Technology

Assistant Research Engineer, University of California, Berkeley

Agenda

Part I

Time-aware systems design – research challenges

Part II

Programming with time – a research initiative

Part I

Time-aware systems design – research challenges



Time-Aware Systems that need Time-Coordinated Computation - Examples

Cyber-Physical Systems (CPS)

Automotive (systems of systems)

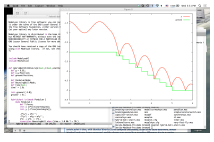
Industrial Automation

Aircraft (traditional or autonomous)

Satellites

Medical Equipment

Time-Aware Development Systems



Physical simulations (Simulink, Modelica, etc.)

Measurement equipment

Time-Aware Distributed Systems

Time-stamped distributed systems (E.g. Google Spanner)

Telecommunication

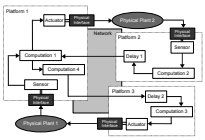
Model-Based Correct-by-Construction

Model

$$\begin{aligned} J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

Equation-based models

Virtual Testing:
Time-aware simulation



Various models of computation (MoC)

Physical prototyping

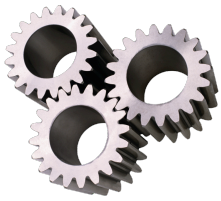
Modeling

Hardware in-the-loop simulation

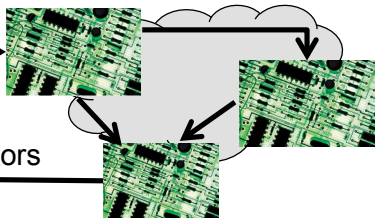
Modeling

Compiling/ synthesizing

System



Sensors



Actuators

Physical system (the plant)

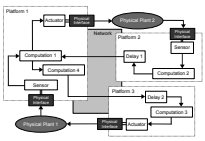
Cyber system: Computation (embedded) + Networking

Model-Based Correct-by-Construction

Model

$$\begin{aligned} J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

Virtual Testing:
Time-aware simulation



Physical prototyping

Compiling/ synthesizing

Modeling

Model fidelity problem

"Ensuring that the model accurately represents the real system"

Research Challenge:
Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

The main challenge concerns guaranteeing correct timing behavior.

Sensors

David Broman
dbro@kth.se

Part I
Time-aware systems design – research challenges

Part II
Programming with time – a research initiative

Modern Systems with Many Processor Platforms

Aerospace

Modern aircraft have many computer controlled systems

- Engine control
- Electric power control
- Radar system
- Navigation system
- Flight control
- Environmental control system

etc…

Automotive

Modern cars have many ECU (Electronic Control Units)

- Airbag control
- Door control
- Electric power steering control
- Power train control
- Speed control
- Battery management.

etc.. Over 80 ECUs in a high-end model (Albert and Jones, 2010)

Mixed-Criticality Systems

Issues with too many processors

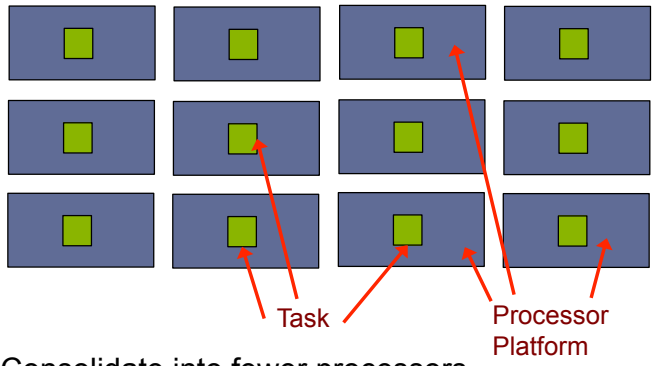
- High cost
- Space and weight
- Energy consumption

Required for Safety

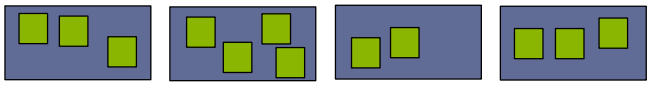
- Spatial isolation between tasks
- Temporal isolation between tasks (necessary to meet deadlines)

Federated Approach

Each processor has its own task



Consolidate into fewer processors

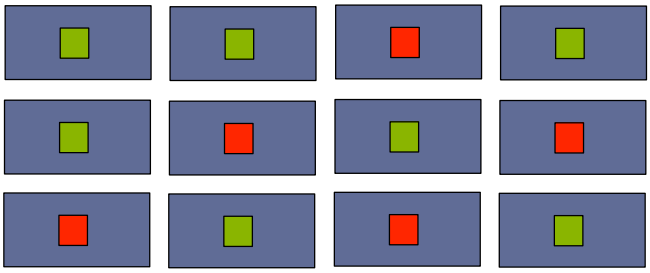


Mixed-Criticality Systems

Issues with too many processors
- High cost
- Space and weight
- Energy consumption

Federated Approach

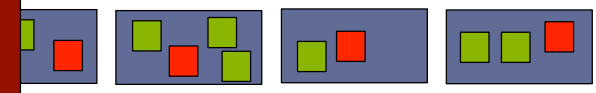
Each processor has its own task



Required for Safety
- Spatial isolation between tasks
- Temporal isolation between tasks (necessary to meet deadlines)

…but such safety requirements are only needed for highly critical tasks

olidate into fewer processors



Mixed-Criticality Challenge

Reconcile the conflicting requirements of:
- Partitioning (for safety)
- Sharing (for efficient resource usage)

(Burns & Davis, 2013)

Overview of Research Areas

Research Objective: Develop model-based *methodologies, algorithms, and time-aware software tools* based on a *correct-by-construction* approach.

Modeling Languages
Expressive, Analyzable, and Extensible

Time-Aware Programming Languages and Compilers

Area 1: Modeling Language Research
- *Embedding Domain-Specific Languages*, Modelyze [PADL'12, EECS report'12]
- *Hybrid Co-Simulation* [EMSOFT'13, HSCC'15]
- *EOO Modeling Semantics* [Modlica'06,GPCE'06, SNE'09]
- *Model-based machine learning* (new direction) [ESE'15]

Area 2: Time-aware Programming Language Research
- *Time-Aware Programming Constructs* [RTAS'13, NIST'15]
- *Timing Analysis* [RTAS'13, CAV'15]

Area 3: Predictable Systems Research
- *Precision Timed (PRET) machines* [ICCD'13, RTAS'13]
- *Predictable Memory Systems* [RTAS'14]
- *Clock Synchronization* [ISPCS'14]

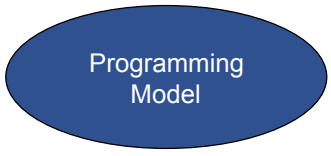
Part II

Programming with time – a research initiative

Programming Model and Time

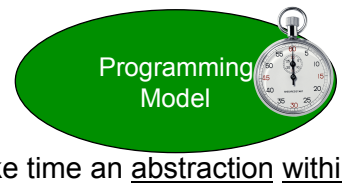
Timing is not part of the software semantics
Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

Traditional Approach

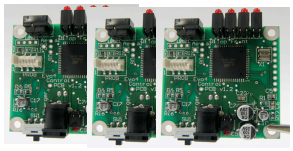


Timing Dependent on the Hardware Platform

Our Objective



Make time an abstraction within the programming model



Enable timing **portability**, where timing requirements are verified by the compiler.

Some Previous Work

Low-level Language Support

- Real-Time Concurrent C (Gehani & Ramamritham, 1991)
- Real-time Euclid (Kligerman & Stoyenko, 1986)
- Ada real-time support, see e.g. (Burns & Wellings 2009)
- POSIX.1b real-time extensions
- Synchronous languages, ESTEREL (Berry & Gonthier), LUSTRE (Caspi et al., 1987), SIGNAL (Benveniste & Guernic, 1991)
- Real-time Java (RTJS)
- Modula for real-time (Wirth, 1977)
- PRET programming, (Lickly et al., 2008)
- PRET-C (Andlam et al., 2010)

High-level Language Support

- Giotto (Henzinger, Horowitz, and Kirsch, 2003) and the embedded machine (Henzinger & Kirsch, 2007)
- PTIDES (Zhao et al., 2007), (Eidson et al., 2011)

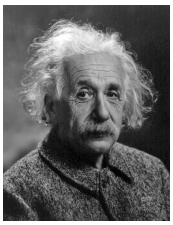
Modeling Languages and Tools

- Modelica (Modelica Association, 2014)
- Simulink (Mathworks)
- Modelyze (Broman & Siek, 2012)
- Ptolemy II (Eker, 2003)
- Labview (National Instruments)

Verification and Formalizations

- Process Algebras with time (Hennesy & Regan 1995)
- Timed automata (Alur & Dill, 1994), UPPAAL (Larsen, Pettersson, Yi, 1997)

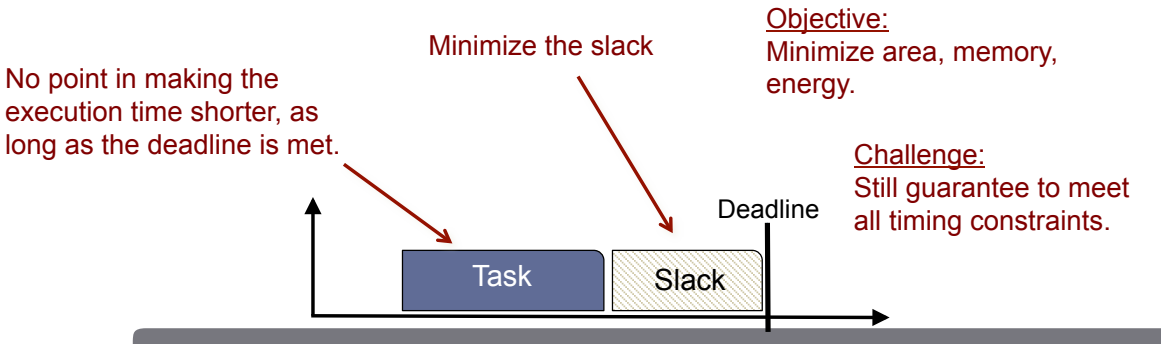
What is our goal?



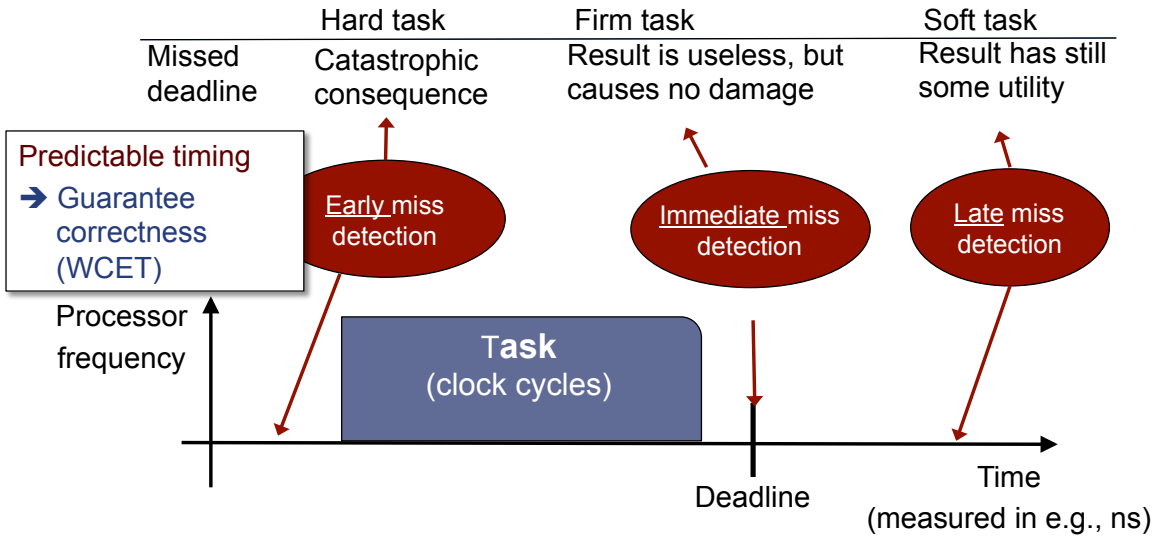
"Everything should be made as simple as possible, but not simpler"

attributed to Albert Einstein

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



Detecting and handling of missed deadlines

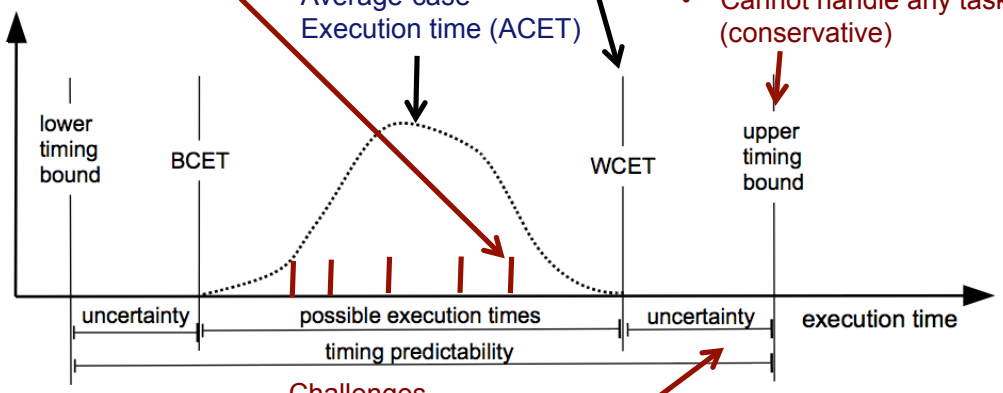


Worst-Case Execution Time (WCET)

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

Worst-case execution time (WCET)

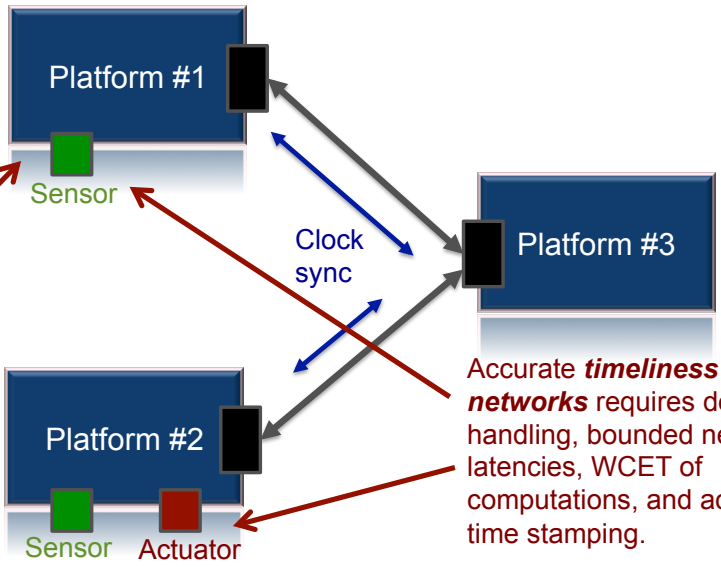
Static program analysis approach
- Upper bound of WCET
- Cannot handle any task (conservative)



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

Time-Coordinated Computing

Timeliness within a platform (soft, firm, and hard deadline management)



Accurate ***time stamping*** requires time synchronization

Accurate ***timeliness over networks*** requires deadline handling, bounded network latencies, WCET of computations, and accurate time stamping.

Programming Problems and Quality Factors

Photo by Kyro

Mobile devices

Error prone to construct correctly timed programs.
A safety and security problem

Simplicity

Programs

Retargeting **heterogeneous** target platforms (processors and OS:s.)

Portability

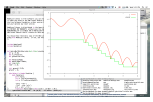
Distributed real-time high performance platforms (RTOS, POSIX API, Linux, Windows)

Unclear timing semantics
Accuracy, resolution, and meaning of time.

Unambiguity

tributed embedded microcontrollers (bare metal implementations)

Overview and Research Goal



Timed and/or synchronous modeling languages (Simulink, Modelica, Ptides, SCADE etc.)

Timed C programs

Source-to-source compilation

C Compiler

Distributed real-time high performance platforms (RTOS, POSIX API, Linux, Windows)

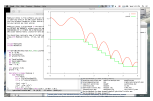
Timing Analysis

C Compiler

Distributed embedded microcontrollers (bare metal implementations)

May act as an **intermediate language**.

Overview and Research Goal



Timed and/or synchronous modeling languages (Simulink, Modelica, Ptides, SCADE etc.)

Timed C programs

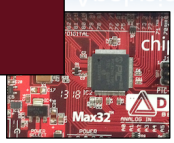
Research goal:
To develop a formally defined *programming model* together with a *compiler tool chain* where time is first class citizen, thus enabling:

- *Simple and correct* programming with time
- *Portability* across processers and OS:s.
- *Unambiguous* meaning of a timed program.

C Compiler

Distributed real-time high
orms (RTOS,
x, Windows)

Timing Analysis



Distributed embedded microcontrollers (bare metal implementations)

Requirements of the program model

Time stamping

Get accuracy and error bounds, depending on underlaying technology (PTP, NTP etc.)

Variable resolution

Express time in ms, us, or ns. Type system should give compile time errors for incorrect usage.

Timed Concurrency

Declarative concurrency for expressing concurrent tasks.

Timeliness and Missed Deadline handling

Express how to handle soft, firm, and hard deadlines, and how to react on misses.

Timeliness and Communication

Sending and receiving data with timing guarantees

Conclusions

Some take away points:

- **Time** *and* **timeliness** are inherently important in systems that interact with the physical reality.
- Two important overall design challenges for time-aware systems are high **model fidelity** and the construction of **mixed-criticality systems**.
- The initiative of **programming with time** aims at making it *simpler* to write *unambiguous* timed programs that are *portable*.

Thanks for listening!