Compositional Timing in Concurrent, Parallel, and Distributed Real-Time Systems
Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

Thanks to:
- Danica Chang
- David Culler
- Stephen Edwards
- Isac Liu
- Sungjin Kim
- Jeff Jensen
- John Eidson
- Hiren Patel
- Stephen Edwards
- Gage Eads
- David Culler
- Danica Chang
- Jia Zou

Compositional Real-Time Systems
Distributed Real-Time Systems
Concurrent, Parallel, and Compositional Scheduling
Edward A. Lee
Professor, UC Berkeley

In general-purpose computing, concurrency is achieved by relatively coarse-grained time multiplexing of shared resources. In this talk, concurrency is achieved by relatively coarse-grained time multiplexing of shared resources. In suitably-designed multicore architectures, timing can facilitate efficient parallel execution in suitably-designed architectures and in distributed real-time systems.

I then explore an instruction-set architecture that builds on the concept of PREcision Timed (PRET) machines, which introduce temporal semantics into an instruction-set architecture. The approach builds on the concept of PREcision Timed (PRET) machines, which introduce temporal semantics into an instruction-set architecture. I then explore how repeatable timing can facilitate efficient parallel execution in suitably-designed multicore architectures and in distributed real-time systems.
Cyber-Physical Systems (CPS): Orchestrating networked computational resources with physical systems.
Approaching the CPS Challenge

Cyberizing the Physical (CtP): to endow physical subsystems with cyber-like abstractions and interfaces

Physicalizing the Cyber (PtC): to endow software and network components with abstractions that represent their physical properties, such as dynamics in time.

and interfaces that represent their physical properties, such as dynamics in time.
The Problem We Address

Today, correct execution of a program has nothing to do with how long it takes to do anything. Correct execution of a program has nothing to do with the program itself.

We are correcting this by introducing precise and repeatable timing into core computing infrastructure. Timing becomes a property of a program, not just a property of a particular hardware executing the program. Behavior of systems becomes predictable, repeatable, and portable.

- Timing of programs becomes predictable, repeatable, and portable.
- Behavior of systems becomes predictable, repeatable, and portable.
- Timing becomes a property of a program.

```
{ ![τ] \forall \text{notifiers}, ![τ] \forall \text{listeners} //
  ![τ-τ] p x ![τ] x = ![τ] x
} for (τ_0 = 0; τ_0 < 10; ++τ) { // perform the convolution.
```
Problems that Arise Today

- Execution time analysis requires very detailed models of processors.
- Execution time analysis is intractable for many real programs and processors.
- Memory hierarchy (caches) introduce timing variability and timing interference across tasks.
- Network interactions disrupt timing.
- Multicore interactions disrupt timing.

Consequence: System behavior is hard to predict and control.
These problems are not intrinsic to the technology! Electronics technology delivers highly reliable and precise timing... and the overlaying software abstractions discard it.

20,000 MHz (± 100 ppm)

Perform the convolution.

```cpp
{ 
  for (int t = 0; t < 10; t++)
    for (int i = 0; i < 10; i++)
      x[i] = a[i] * b[j-i];
  notify(x[i]);
}
```

... and the overlaying software...
Computing

Lee, Berkeley 8

**PRET Machines**

- Performance with Repeatable Timing = PRET
- Repeatable, Repeatable Timing = PRET
- Precision-Timed Processors = PRET

```java
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    notify(x[i]);
}
```


Make Timing a Semantic Property of Computers: PRET Machines


Wild and Crazy Ideas Track, Design Automation Conference (DAC), June 2007.

Precision networks (TTA/Timed synchronization?)
Multi-core PRET (conflict-free routing?)
Composable timed components (actor-oriented?)
Pre-dicable concurrency (synchronous languages?)
Languages with timing (discrete events? Giotto?)
Predictable memory management (Metronome?)
Memory hierarchy (scratchpads? DRAM banks?)
Deep pipelines (interleaving?)
ISA with timing (repeatable inst. timing? deadline instructions?)
Timing precision with performance: Challenges:

Make temporal behavior as important as logical function.
Pipelining

Pipeline Hazards

An Alternative: Pipeline Interleaving

Stall pipeline:

Thread-interleaved pipeline:

Traditional pipeline:

Instructions behavior of timing

Repeatable

Dependencies result in complex

Timing behaviors
An old idea:

Pipeline Interleaving


Lee, Berkeley 13
Pipeline Interleaving Enables Precise, Regular Timing, and Compositional Timing.

If interrupts are always handled by thread 1, then the timing of threads 2 through n is not affected by I/O. XNOS approach: Activate thread n+1 to handle interrupts. This does affect the timing of other threads, but much more smoothly than standard methods.

If interrupts are always handled by thread 1, then the timing of threads 2 through n is not affected by I/O.
Our solution: PRET Machines

Make temporal behavior as important as logical function.


Wild and Crazy Ideas Track, Design Automation Conference (DAC), June 2007.

Precision networks (TTA, Time synchronization?)

Multi-core PRET (conflict-free routing?)

Composable timed components (actor-oriented?)

Predictable concurrency (synchronous languages?)

Languages with timing (discrete events? Giotto?)

Predictable memory management (Metronome?)

Memory hierarchy (scratchpads? DRAM banks?)

Deep pipelines (interleaving?)

ISA’s with timing (repeatable instructions? deadline instructions?)

Timing precision with performance: Challenges:

Lee, Berkeley 15
PRET principle: all temporary memory is under program control.

Separation of concerns:

- Why is replacement strategy independent?
- Cache is a temporary memory under hardware control.
- Why is it so small?
- Instruction word size.
- Register file is a temporary memory under program control.


Memory Hierarchy
Dynamic RAM vs Static RAM

Static RAM (SRAM) is high capacity: each cell has six transistors
- low access latency
- typically on-chip in lower levels of memory hierarchy

Dynamic RAM (DRAM) is high capacity: one transistor and one capacitor
- low access latency
- typically off-chip in higher levels of memory hierarchy
- with some recent exceptions (with some recent exceptions)
One Possible PRET Architecture

- Hardware thread registers
- SRAM scratchpad
- DRAM main memory
- I/O devices
- Interleaved pipeline with one set of registers per thread shared among threads
- Interleaved hardware thread

Lee, Berkeley 18
What about Main Memory?

Modern DRAMs:

- DDR2: Four pipelined banks
- DDR3: Eight pipelined banks
- DDRn: 2^n pipelined banks

Micron Corp.
The result is non-composable timing.

- Controllers dynamically schedule refreshes.
- Controller might reorder requests to minimize latency.
  - Request to open/closed row within bank?
  - Request to same/different bank?
- Timing of request depends on past requests.
- Timing is hard to predict even for single client.

General-Purpose DRAM Controllers
Predictable DRAM Controllers: Predator (TU Eindhoven) and AMC (Barcelona)

Schedule predefined patterns:
- Latency-rate servers + Buffers to hide interaction
- Round robin (AMC)

Decouple access timing from execution history
- Composable arbitration: Round robin (AMC)
- Decouple access timing from execution history
- Schedule predefined patterns (pessimistic regarding refreshes)
- Still schedule refreshes dynamically
- Allow to determine worst-case access timing

Lee, Berkeley 21
Provides four independent and predictable resources

- Exploit internal structure of DRAM module:
  - Consists of 4-8 banks in 1-2 ranks
  - Share only command and data bus, otherwise independent
  - Cycle through groups in a time-triggered fashion
  - Partition into four groups of banks in alternating ranks

- Successive accesses to successive banks in a group obey timing constraints
- Reads/Writes to different groups do not interfere

- Group 0
- Group 1
- Group 2
- Group 3

Rank 0:
- Bank 0
- Bank 1
- Bank 2
- Bank 3

Rank 1:
- Bank 0
- Bank 1
- Bank 2
- Bank 3
Resulting PRET Architecture
Make timing a semantic property of computers: PRET (Precision Timed) Machines.

Challenges:
- Timing with performance: Challenges.
- ISA with timing (repeatable instruction timing? deadline instructions?)
- Deep pipelines (interleaving?)
- Memory hierarchy (scratchpads? DRAM banks?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete event? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Multi-core PRET (conflict-free routing?)
- Precision networks (TTA, Time synchronous?)

Wild and Crazy Ideas Track, Design Automation Conference (DAC), June 2007.


Lee, Berkeley 24
Extending an ISA with Timing Instructions

[LV4] Exact execution:

\[ \text{MTF}\ r1 \]
\[ // \text{Code block} \]
\[ \text{set}\_time\ r1, 1s \]

[LV2] Late miss detection:

\[ \text{delay-unti}\ r1 \]
\[ \text{branch-expired}\ r1, \text{target} \]
\[ \text{set}\_time\ r1, 1s \]
\[ \text{branch-expired}\ r1, \text{target} \]

[LV3] Immediate miss detection:

\[ \text{delay-unti}\ r1 \]
\[ \text{set}\_time\ r1, 1s \]

[LV1] Best effort:

\[ \text{deactivate-exception}\ r1 \]
\[ // \text{Code block} \]
\[ \text{exception-on-expire}\ r1, 1 \]

Timing Instructions

Extending an ISA with...
Example:

```
tryin (500ms) {
    // Code block
}
catch {
    panic()
}
```

This pseudocode is neither C-level nor assembly-level implementation, but is meant to explain an assembly-level procedure to be called, causing the panic procedure to be called, but returning non-zero. The else statement will then be run longjmp, which will return control flow to setjmp, exception 0 will be thrown, and the handler code will exception handler 0

```
deactivate_exception
	// Code block
0 exception_on_expire
set_time r1, 500ms
}
if (setjmp(buf))
jmp_buf buf;

if the code block takes longer than 500ms to run, then non zero when invoked by longjmp()

using setjmp and longjmp to handle timing exceptions.
This realizes variant 3, “Immediate miss detection”

Example:

```
low-level language (like C)
```

Exporting the Timed Semantics to a
Summary of ISA extensions

- **V4** Execute a block of code taking exactly the specified time. MTFD
- **V3** Do [V1], but if the specified time is exceeded, the specified time was exceeded.
- **V2** Do [V1], and then conditionally branch if the specified time [Ip & Edwards, 2006]
- **V1** Execute a block of code taking at least a minimum execution time.

Variants:

- Time may be literal (seconds) or abstract (cycles).
- For V2 – V4, may not impose minimum execution time.
- For V2 – V4, may not impose minimum execution time.
A Brief Word on Multicore PRET Machines

Make temporal behavior as important as logical function.
A preliminary project by Dai Bui shows that control over timing enables conflict-free routing of messages in a network on chip. This means that it becomes possible to have programs on a multicore PRET with compositional timing.

Multicore PRET
A Few of the (Many) Remaining Challenges and Opportunities

- DRAM designs today compromise efficiency even with private banks (e.g., write-after-read latencies).
- Interleaved pipelines may not be the best choice for power optimization.
- Exposing timing properties in programming models (completely absent in today's languages).
- I/O mechanisms that do not disrupt repeatable timing.
- More work needed on multicore.

...
Lee, Berkeley 31

PTIDES: Distributed execution under discrete-event semantics, with "model time" and "real time" bound at sensors and actuators.

Make Timing a Semantics Property of Software Components

A Top Down Approach:
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

PTIDES uses static causality analysis to determine when events can be safely processed (preserving DE semantics).

Assume bounded sensor delay $s$
Assume bounded clock error $e$
Assume bounded network delay $d$

An earliest event with time stamp $t$ can be safely merged when real time exceeds $t + s + d - d^2$. Application specification of latency $d^2$. 

{$\text{PTIDES}$}
... and being explicit about time delays means that we can analyze control system dynamics...

Distributed Embedded Systems

PTIDES: Programming Temporally Integrated Distributed Embedded Systems
First Test Case

- Tunneling Ball Device
  - sense ball
  - track disk
  - adjust trajectory

This device was designed by Jeff Jensen, now at National Instruments.
Tunneling Ball Device in Action
Tunneling Ball Device

Mixed event sequences

Periodic Events

Quasi Periodic Events

Sporadic Events
Second Test Case: Distributed Synchrophasor Measurement & Control

Power swing and Unstability detection

Thanks to Vaselin Skendzic, Schweitzer Engineering
Thanks to Vaselin Skendzic, Schweitzer Engineering

Diagram

Experiment

DP83640 with Renesas demo board, with processing unit (SVP) built

Synchrophasor Vector

Ethernet bridge or 1588

Boundary/transparent clock

Ethernet bridge (IEEE 1588 Boundary Clock)

Primary Measurement Unit (PMU) built with Renesas demo boards

Synchrophasor Vector Processing Unit (SVP) built with Renesas demo board with DP83640

Grid emulator built with National Instruments PXI

Primary Measurement Unit (PMU) built with Renesas demo boards

Synchrophasor Vector

Ethernet bridge (IEEE 1588 Boundary Clock)
The question addressed by the PTIDES project:
If you assume that computers on a network can agree on the current time of day within some bounded error, how does this change how we develop distributed real-time software?

Our answer: It changes everything!

Our approach: Model-based design based on distributed discrete-event (DE) models with synthesis of embedded software.
A question we are addressing at Berkeley: How does this change how we develop distributed CPS software?

Techniques like NTP are precise than older techniques like NTP, within 8ns, far more current time of day to within 8ns, far more precise than older techniques like NTP. This may become routine!

This may become routine!
An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous ethernet.

and synchronous ethernet.
More Generally than PTIDES:
Rethinking Software Components to Admit Time.

Object Oriented vs. Actor Oriented:

Object Oriented:
Things happen to objects

Actor Oriented:
Actors make things happen
Examples of Actor-Oriented Systems

- UML 2 and SysML (activity diagrams)
- ASCET (time periods, priorities, preemption, shared variables)
- CORBA event service (distributed push-pull)
- ROOM and UML-2 (dataflow, Rational, IBM)
- SDL (process networks)
- Open (rendezvous)
- Occam (process networks)
- Real-time systems, OpenTechologies
- SDL (discrete events, C++, RealTechologies)
- Modelica (continuous time, constraint-based, Linkoping)
- VHDL, Verilog (discrete events, Cadence, Synopsys, ...)
- ROOM (continuous time, Cadence, Synopsys, ...)
- ROOM and UML-2 (dataflow, Rational, IBM)
- CORBA event service (distributed push-pull)
- SCADe (synchronous, based on Lustre and Esterel)
- LabVIEW (structured dataflow, National Instruments)
- Simulink (continuous time, The MathWorks)
- Autosar (software components w/ sender/receiver interfaces)
- ASCEt (time periods, priorities, preemption, shared variables)

The semantics of these differ considerably in their approaches to concurrency and time. Some are strongly actor-oriented, while some retain much of the flavor (and flaws) of threads. These differ considerably in their approaches to concurrency and time. Some are loose (ambiguous) and some are rigorous. Some favor (and flaws) of threads. Some are strongly actor-oriented, while some retain much of the flavor.
Actor-Oriented Design

Polyphony II: Our Laboratory for Experiments with Actor-Oriented Models

Director from a library defines component semantics and decomposes component interfaces. Modern type system for software component library.

Visual editor for composing components.
Today, timing is a property only of realizations of computational systems. Tomorrow, timing will be a semantic property of computational models.
This book strives to identify and introduce the durable intellectual ideas of embedded systems as a technology and as a subject of study. The emphasis is on modeling, design, and analysis of cyber-physical systems, which integrate computing, networking, and physical processes.