PREcision Timed (PRET) Architecture

Isaac Liu
Advisor - Edward A. Lee

Dissertation Talk
April 24, 2012
Berkeley, CA
Acknowledgements

• Many people were involved in this project:
  - Edward A. Lee - UC Berkeley
  - David Broman - UC Berkeley
  - Ben Lickly - UC Berkeley
  - Hiren Patel - University of Waterloo
  - Jan Reineke - Saarland University
  - Stephen Edwards - Columbia University
  - Sungjun Kim - Columbia University
  - Matt Viele - Drivven Inc.
  - Gerald Wang - National Instruments
  - Hugo Andrade - National Instruments
  - And many more…
Cyber Physical Systems

Two key characteristics of physical processes

- Inherently Concurrency
- Uncontrollable passage of time

Key Challenges [Sangiovanni-Vincentelli, 07]:

- Composability
- Timing Predictability
- Dependability
Composability

IMA - Integrated Modular Avionics

Example Federated Architecture
CPUs: 3
I/O Modules + Network Interface Modules: 5
Physical Comm Channels: 4

Example IMA Architecture
CPUs: 1
I/O Modules + Network Interface Modules: 4
Physical Comm Channels: 1

[Sangiovanni-Vincentelli, ee249 lecture 1]

"Precision Timed Architecture", Isaac Liu
How long does it take to execute the following code?

```c
for (i = 1; i < n; i++)
    if ( a[i] > b[i] )
        c[i] = c[i-1] + a[i];
    else
        c[i] = c[i-1] + b[i];
```

Let’s assume we know $n$.

Branch predicted correctly?

Cache Hit? Miss?

Data Dependency

Out of order execution? Multithreading?

Assume branch mispredict, cache miss?
Timing Anomalies

[Lundqvist et al., 99]

<table>
<thead>
<tr>
<th>FU₁</th>
<th>FU₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

FU₁: In-Order Resource
FU₂: Out-of-Order Resource

ΔC = -2

[Wenzel et al., 05]

<table>
<thead>
<tr>
<th>FU₁</th>
<th>FU₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

FU₁ and FU₂: In-Order Resources

ΔC = -1

Branch Condition Evaluated

Cache Hit

A → Prefetch B - Miss → C

Cache Miss

[Reineke et al., 06]

[Pentium III Time]

[Engblom, 03]
Challenges in WCET Analysis

• “However, both the precision of the results and the efficiency of the analysis methods are highly dependent on the predictability of the execution platform. In fact, the architecture determines whether a static timing analysis is practically feasible at all and whether the most precise obtainable results are precise enough.”

(Emphasis added) [Wilhelm, 03]

Heckmann et al., The influence of processor architecture on the design and the results of wcet tools, IEEE 03
Contribution

- Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.
Architecture Improvements

Cache

Pipelines

Superscalar Out of Order

Multicore

"Precision Timed Architecture", Isaac Liu

[Courtesy of Sami Yehia, Thales]
Execution Time Variance

"Future applications, including safety-critical and active-safety ones, need shorter latencies and time determinism - reduced jitter - to increase performance."

[Sangiovanni-Vincentelli, 07]
Related Work

• **Modifying Modern Processors**
  - **Superscalar** [Rochange et al., 05], [Whitham et al., 08]
  - **VLIW** [Yan et al., 08]
  - **Multithreading** [Kreuzinger et al., 00], [El-Haj-Mahmoud et al., 05]
  - **SMT** [Barre et al., 08], [Mische et al., 08], [Metzlaflf et al., 08]

• **WCET Analysis**
  - **Pipeline Analysis** [Schneider et al., 99], [Ferdinand et al., 01],
    [Lagenbach et al., 02], [Kirner et al. 09] ...
  - **Cache Analysis** [Heckmann et al., 03], [Reineke et al., 07] ...

• **Stack Based Architecture**
  - **Java Optimized Processor** [Schoeberl, 06]
Precision Timed Architecture

Summary of architectural features:

<table>
<thead>
<tr>
<th>Traditional</th>
<th>PRET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep out-of-order pipelines</td>
<td>Thread-interleaved pipelines</td>
</tr>
<tr>
<td>(Instructional level parallelism)</td>
<td>(Thread level parallelism)</td>
</tr>
<tr>
<td>Caches</td>
<td>Scratchpads</td>
</tr>
<tr>
<td>(Hardware replacement policy)</td>
<td>(Software controlled replacement)</td>
</tr>
<tr>
<td>Best effort DRAM Controller</td>
<td>Predictable DRAM Controller</td>
</tr>
</tbody>
</table>

Pipelining

LD R1, 45(r2)
DADD R5, R1, R7
BE R5, R3, R0
ST R5, 48(R2)

Unpipelined

The Dream

The Reality

Memory Hazard
Data Hazard
Branch Hazard

Edwards, RePP 09
Interleaved Pipeline

- Denelcor, HEP (1981), Lee and Messerschmitt, DSP (1987), CDC 6000 (1961)...

Also called Fine Grained Multithreading!

"Precision Timed Architecture", Isaac Liu
Thread Interleaved Execution

**Thread 0: GCD with conditional branches**

```assembly
add r0, r1, r2
sub r1, r0, r1
ldr r2, [r1]
sub r0, r2, r1
cmp r0, r3
beq end
blt less
sub r0, r0, r1
b gcd
cmp r0, r1
add r0, r1, r2
ldr r2, [r1]
```

**Thread 1: Data dependent code**

```assembly
add r0, r1, r2
sub r1, r0, r1
ldr r2, [r1]
sub r0, r2, r1
cmp r0, r3
```

**Thread 2**

```assembly
cmp r0, r1
add r0, r1, r2
blt less
ldr r2, [r1]
b gcd
```

**Thread 4**

```assembly
add r0, r1, r2
sub r1, r0, r1
ldr r2, [r1]
sub r0, r2, r1
cmp r0, r3
```

**Example Code**

```assembly
cmp r0, r1
beq end
blt less
sub r0, r0, r1
b gcd
```

**Memory Access**
Interleaved Pipeline

Trade-offs:
- Need enough concurrency to utilize processor
- Favor throughput over latency

However...
- Simpler WCET analysis (Timing Predictability)
- Interference free multiple context execution (Composability)
- Simple pipeline design (Energy, Cost...)
- Improved throughput and clock rate (Performance)
Memory Hierarchy

Use Scratchpads instead of Caches!

"Precision Timed Architecture", Isaac Liu
Scratchpads

Trade-offs:

• Need explicit management from the software (compiler/programmer)

However...

• Simpler WCET analysis (Timing Predictability)
• Customize to workload (Performance)
• Simple circuit design (Energy, Cost...)

"Precision Timed Architecture", Isaac Liu
Main Memory

DRAMs:

Two key problems:

• Bank Conflicts
• DRAM Refresh

"Precision Timed Architecture", Isaac Liu
Provides four independent and predictable resources

Allows for predictable refreshes

[Reineke, CODES 11]

Dedicated refresh commands vs refreshes through reads.

(refresh latencies not to scale)
Main Memory

Trade-Offs:

- Shared memory on scratchpad
- Longer average memory latencies

However...

- Predictable access latencies (Timing Predictability)
- Better throughput and latency when fully utilized (Performance)

Reineke et al., PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, CODES 11
PTARM

Thread-Interleaved Pipeline

Scratchpads
BootROM
DRAM Controller
Addr. Mux
I/O Bus

UART Gateway
DVI Controller
LED Registers

Integrated Logic Analyzer

xcvlx110t

RS232
DVI Transmitter
On Board LEDs
DDR2 DRAM Memory Module

Download at http://chess.eecs.berkeley.edu/pret

"Precision Timed Architecture", Isaac Liu
Pipeline Performance

WCET Benchmarks Instruction Throughput (higher is better)

WCET Benchmarks Latency (lower is better)

"Precision Timed Architecture", Isaac Liu
DRAM Performance

Varying Interference

![Graph showing latency vs. interference]

- 4096B transfers, conventional controller
- 4096B transfers, PRET controller
- 1024B transfers, conventional controller
- 1024B transfers, PRET controller

Varying Bandwidth

![Graph showing latency vs. transfer size]

- Conventional controller
- PRET controller

Reineke et al., PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation, CODES 11
Contribution

• Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.
  - Use architectural techniques that provides composability and timing predictability

• Expose “time” in the Instruction Set Architecture.

"Precision Timed Architecture", Isaac Liu
Fly-by-wire aircraft controlled by software.

They have to purchase and store microprocessors for at least 50 years production and maintenance…

"Precision Timed Architecture", Isaac Liu
Levels of Abstraction

"Precision Timed Architecture", Isaac Liu

[Lee, 08]
ISA with “time”

- Extend Instruction Set with timing instructions that specify and control timing behaviors of code blocks.
  - Assume a “platform clock” synchronous with the execution of instructions
  - Timing instructions use platform clock to control execution time

---

**Diagram:**

- **Deadline of Task**
- **A)** Finish the task, and detect at the end if deadline was missed
- **B)** Immediately handle a missed deadline
- **C)** Continue as long as execution time does not exceed
- **D)** Ensure execution does not continue until specified time

---

"Precision Timed Architecture", Isaac Liu
## Timing Control

### New instruction `get time (gt)`

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gt</code></td>
<td><code>r1, r2</code></td>
</tr>
<tr>
<td></td>
<td>; get time (ns)</td>
</tr>
</tbody>
</table>

---

### New instruction `delay until (du)`

- `adds` `r2, r2, #500` ; add 500 ns
- `adc` `r1, r1, #0` ; add with carry (time in 2 32-bit reg)
- `du` `r1, r2` ; delay until 500ns have elapsed

---

### Where could this be useful?
- Finishing early is not always better:
  - Scheduling Anomalies (Graham’s anomalies)
  - Communication protocols and External synchronization
## Timing Exceptions

### New instruction exception on expire (ee)

```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gt</td>
<td>r1, r2; get time (ns)</td>
</tr>
<tr>
<td>adds</td>
<td>r2, r2, #500; add 500 ns</td>
</tr>
<tr>
<td>adc</td>
<td>r1, r1, #0; add with carry (time in 2 32-bit reg)</td>
</tr>
<tr>
<td>ee</td>
<td>r1, r2; register timer exception</td>
</tr>
<tr>
<td></td>
<td>-- Code block --</td>
</tr>
<tr>
<td>de</td>
<td>; deactivate exception</td>
</tr>
</tbody>
</table>
```

### New instruction deactivate exception (de)

- Immediate deadline miss detection
ISA with "time"

Traditional Approach

- Programming Model
- Timing Dependent on the Hardware Platform

Our Objective

- Programming Model
- Make time an engineering abstraction within the programming model
- Timing is independent of the hardware platform (within certain constraints)

A Timing Requirements-Aware Scratchpad Memory Allocation Scheme for a Precision Timed Architecture [Patel et al. 08]
Contribution

• Propose an architecture that allows for timing predictability and composable resource sharing without sacrificing performance.
  - Use architectural techniques that provides composability and timing predictability

• Expose “time” in the Instruction Set Architecture.
  - ISA extensions to specify temporal properties

"Precision Timed Architecture", Isaac Liu
Real-Time Engine Fuel Rail Simulation

- 1D CFD Simulation - Network of Pipes
- Real-Time requirements: 5.33us
- Common Fuel Rail: 234 nodes

Implement on Xilinx V6 FPGA
Timing Side-Channel Attacks

- Timing exploits:
  - Algorithms
  - Caches
  - Branch Predictors
  - Pipelines...

Root cause: uncontrollable timing side effects!
Summary

• Problem Statement:
  - Conventional methods are limiting the scaling of Cyber Physical Systems design because of its lack of precise timing control and analysis

• Solution:
  - To rethink the design of the bottom layers of abstraction, with emphasis on temporal predictability for Cyber Physical Systems

• Outcome of Research:
  - Precision Timed Architecture (PRET) for timing predictability and composability with ISA extensions
Publications


• Reineke, **Liu**, Patel, Kim, Lee, *PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation*, CODES 11

• Bui, Lee, **Liu**, Patel, Reineke, *Temporal Isolation on Multiprocessing Architectures*, DAC 11


• Lickly, **Liu**, Kim, Patel, Edwards, Lee, *Predictable Programming on a Precision Timed Architecture*, CASES 08
Thank You

Please visit http://chess.eecs.berkeley.edu/pret

Questions?
BACKUP SLIDES
Thank You

• Qual Committee
• Edward A. Lee - Berkeley
• Hiren Patel - Univ. of Waterloo
• Martin Schoeberl - Univ. of Denmark
• Stephen A. Edwards - Columbia Univ.
• Ben Lickly, Sungjun Kim
• John Eidson, Marc Geilen, Sami Yehia (Thales), Maarten Wiggers, Jan Reineke, Slobodon Matic, Jia Zou
• Christopher Brooks, Mary Stewart
• My Family
Research Efforts In All Fronts

![Diagram](image.png)

"Precision Timed Architecture", Isaac Liu
Definitions

• Predictability
  - The ability to analyze the execution time

• Repeatability
  - The ability to repeat the execution given the same inputs

• Composability
  - The functional and temporal behavior of an application is the same, irrespective of the presence or absence of other applications

• Robust
  - Small changes in input leads to small changes in output
WCET Analysis

A. Timing-Analysis Framework

Fig. 1. Main components of a timing-analysis framework and their interaction.

Over the last several years, a more or less standard architecture for timing-analysis tools has emerged [11]–[13].

Fig. 1

The following list presents the individual phases and describes their objectives and problems. Note that the first four phases are part of the first building block.

1) Control-flow reconstruction and static analyses for control, data, and value flow;
2) Microarchitectural analysis, which computes upper and lower bounds for the whole program;
3) Loop bound analysis, which computes upper and lower bounds on the number of loop iterations, information which is indispensable to data or instruction caches. This information is, among others, used for a precise data prediction and cache analysis.
4) Global bound analysis, which computes upper and lower bounds on execution time for the whole program. In-formation provided by the loop bound and control-flow analysis concepts. Static cache analyses determine safe approximations to the contents of caches at each program point. Pipeline analysis analyzes how instructions pass through the pipeline accounting for occupancy of shared resources like queues, functional units, etc. Ignoring these dependencies in nested loops.

Since most parts of the pipeline state influence timing, the abstract model needs to closely resemble the concrete hardware. The more performance-enhancing features a pipeline has, the larger is the search space. Superscalar and out-of-order execution times of individual instructions to obtain a bound on the execution time of a basic block. Pipelines increase the number of possible interleavings. The longer the influence of past events lasts. Dynamic branch mispredictions increase the number of possible interleavings. The larger the buffers (e.g., fetch buffers, retirement queues, etc.), the larger the search space. Superscalar and out-of-order execution times of individual instructions to obtain a bound on the execution time of a basic block. Pipelines increase the number of possible interleavings. The longer the influence of past events lasts. Dynamic branch mispredictions increase the number of possible interleavings. The larger the buffers (e.g., fetch buffers, retirement queues, etc.), the larger the search space.

The commercially available tool http://www.absint.de/wcet.htm, implements this architecture.
Computer Architecture

- Typical metrics in processor design for Embedded Systems
  - Performance (Average Case)
  - Power
  - Area (Size)
  - Compiler Support (Developmental Effort)
  - Cost
  - Multiple Context
  - Analyzability
Branch Prediction Anomaly Experiment

for{k=1; k<32; k++) {
    starttimer();
    for(n=0; n < 10000000; n++) // OUTER LOOP
    {
        for(i=0; i < k; i++) // INNER LOOP
        {
            __nop(); // Some compiler-dependent way to get a nop
        }
    }
    stoptimer();
    recordtime();
}
Richard’s Anomalies

Increasing the number of processors

9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:

The optimal schedule with four processors has a longer execution time.
Richard’s Anomalies

Reducing all execution times by 1

9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:

Reducing the computation times by 1 also results in a longer execution time.
Richard’s Anomalies

Removing precedence constraints

9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Optimal 3 processor schedule:

Weakening precedence constraints can also result in a longer schedule.
Progress

Work Completed:
• SPARC instruction set simulator
  - C++ cycle accurate simulator
• PTARM architecture
  - Synthesizable VHDL ARM core
  - VGA controller and Serial Communication

Work in Progress:
• WCET analysis tool (~2 weeks)
• Benchmarking the pipeline (~1 semester)
• Scratchpad allocation with timed programming models (~1 semester)
• Proof of concept workflow (~1 semesters)
Contribution

- Expose “time” in the abstraction layers.
  - ISA extensions to specify temporal properties

- Propose an architecture that allows for timing predictability and composable resource sharing.