Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation by Wei Qin and Sharad Malik

EE 249 Paper Presentation

Wenchao Li

Fall 2007

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

Introduction

- Microprocessor = *Instruction-set-architecture (ISA)* + *Microarchitecture*.
- Instruction-set-simulator emulates the *functionality* of programs.
- Microarchitecture simulator provides *performance* metrics.

Introduction contd.

- Important characteristics for a high quality microarchitecture simulation framework:
  - Efficient
  - Expressive
  - Declarative
  - Productive

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

Motivation

- Growth in application-specific processors demands a *retargetable* modeling framework that is capable of accurately capturing complex process behavior and generating efficient simulators.
- Simulation techniques in t microarchitecture modeli not mature.

Yea, dammit, platform-based design again!

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

Prior Work

- Operation-centric:
  - nML, ISDL, EXPRESSION.
- Hardware-centric:
  - MIMOLA, HASE, SystemC, Asim, Liberty;
  - UPFAST.
- Other formalism:
  - LISA
  - BUILDABONG
  - SimpleScalar

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

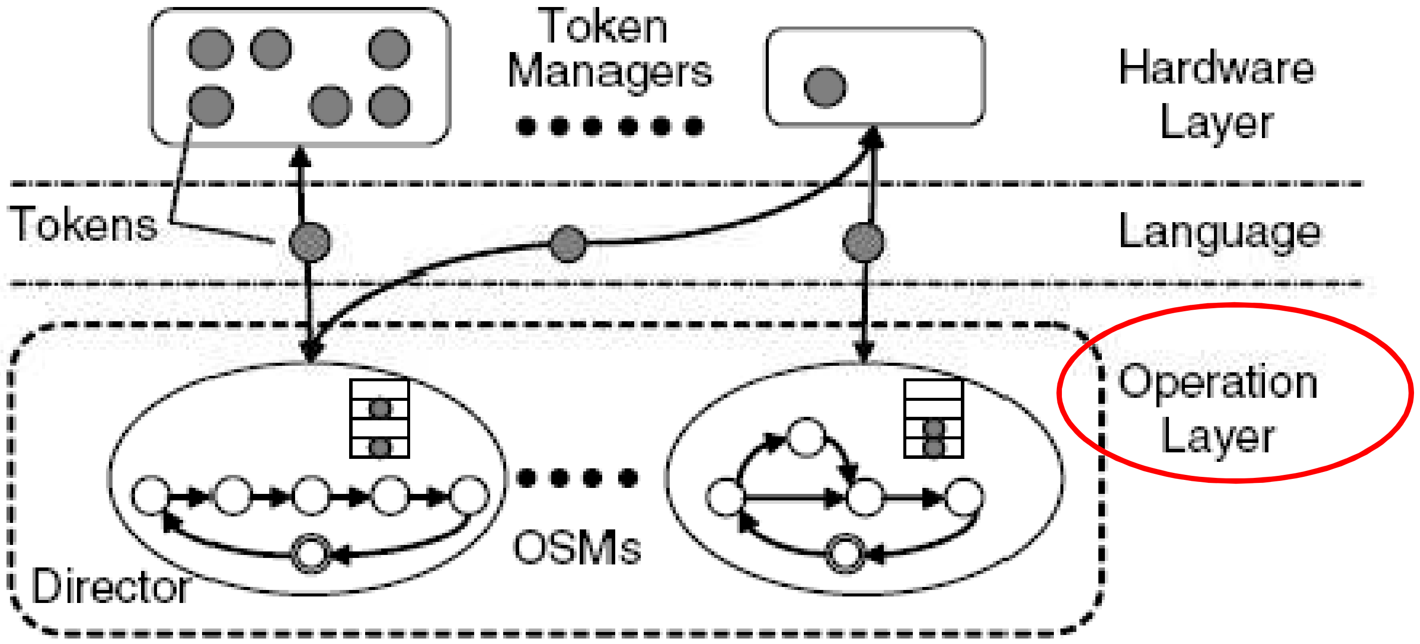
OSM

- Why the heck do we need another model?
  - Microprocessor specifications can be partitioned into:
    1. Operation layer
    2. Hardware layer
  - Existing frameworks focuses on one or the other, or has limited flexibility.
  - So?

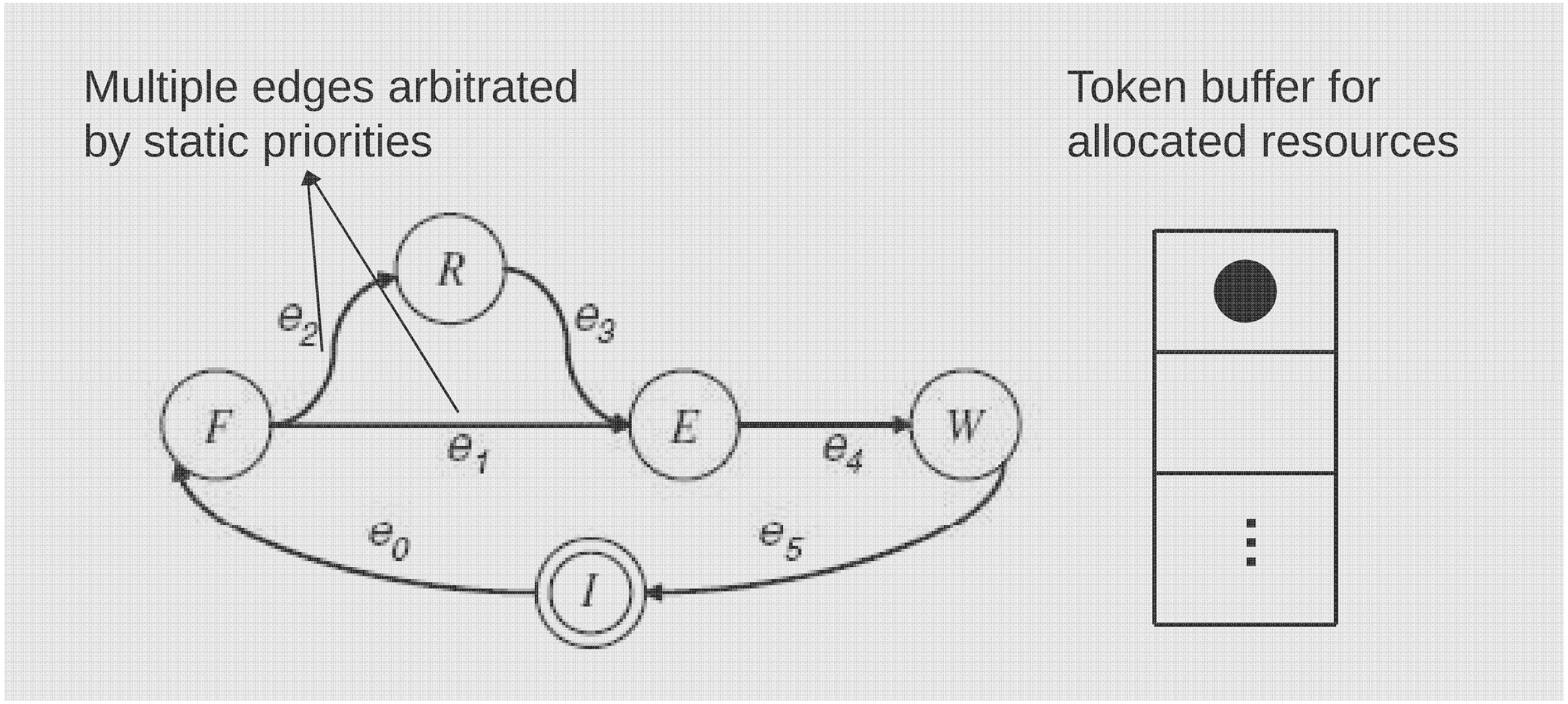
OSM

- OSM aims to provide clean and formal semantics to distinguish these two layers and at the same time model complex interactions between the two. This also helps to *orthogonalize* design considerations.

The OSM Model

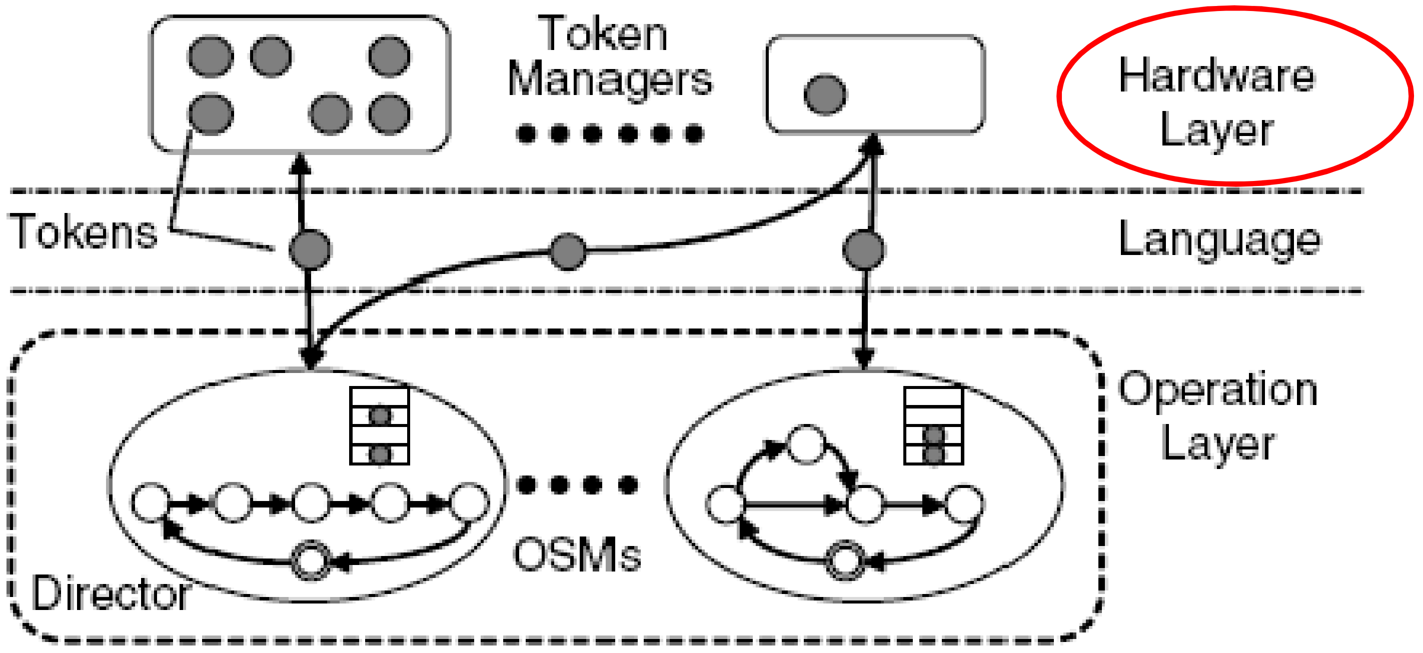


Operation Layer


Multiple edges arbitrated by static priorities
Token buffer for allocated resources
R
e_2
e_3
F
e_1
E
e_4
W
e_0
I
e_5
⋮

- Multiple state machines are coordinated by a *director* to avoid non-determinism.

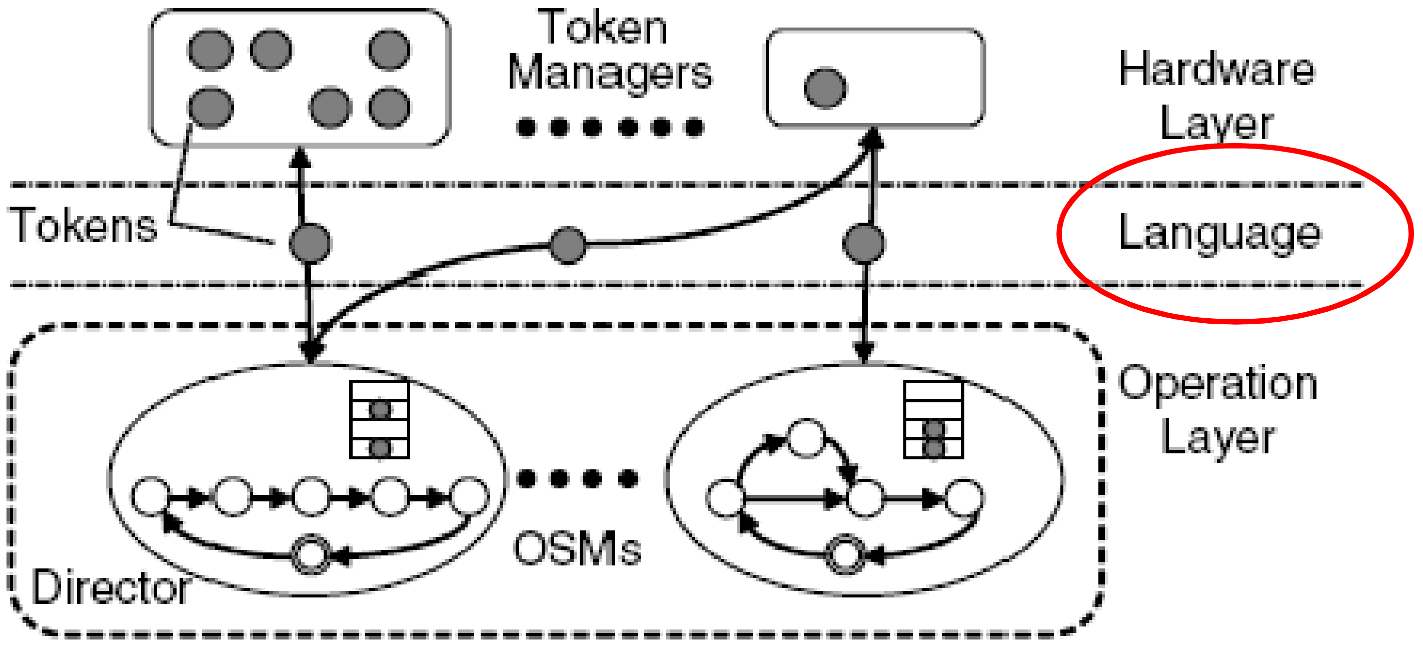
The OSM Model



Hardware Layer

- In the OSM model, we model the *resources as tokens*. A token manager manages one or more closely related tokens. It can grant a token to, or reclaim a token from an OSM upon request. Token managers may check the identity of the requesting OSMs when making decisions.

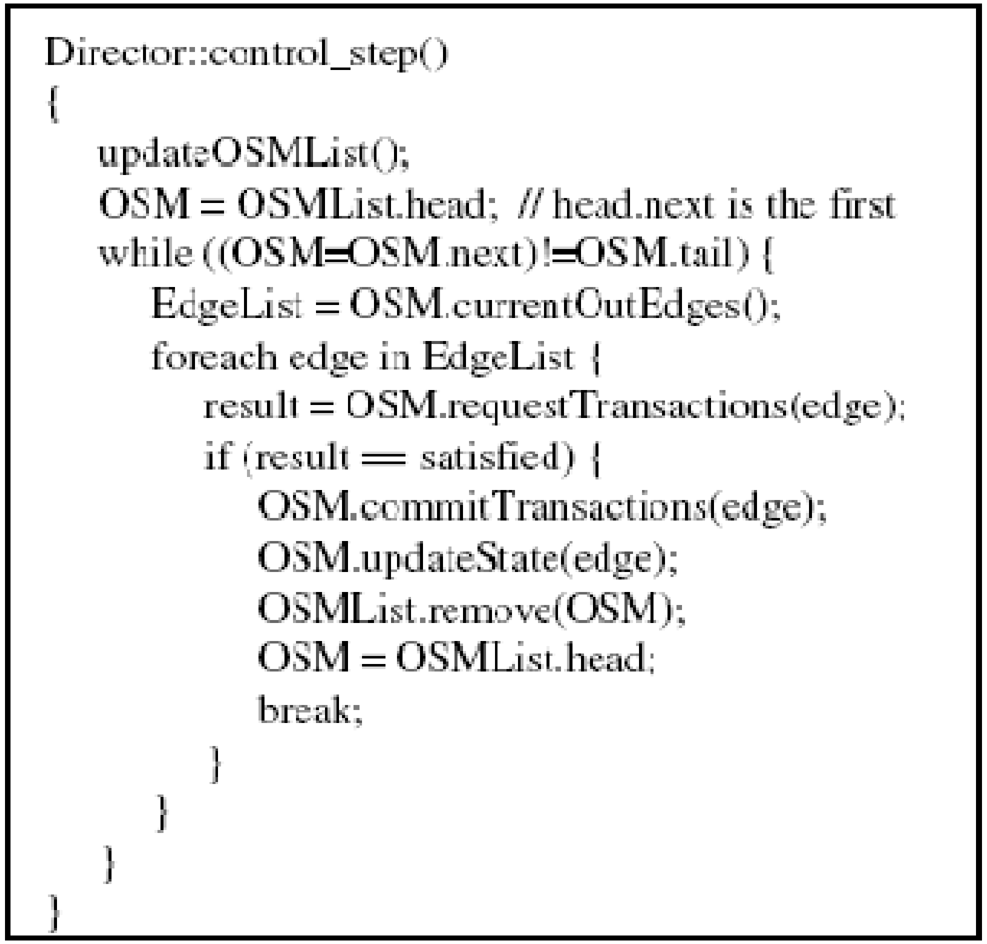
The OSM Model



Language

- Token Manager Interface (TMI).
- Token transactions:
  - *Allocate*: transaction of exclusive resources;
  - *Inquire*: transaction of non-exclusive resources;
  - *Release*: opposite of allocate;
  - *Discard*: reset.

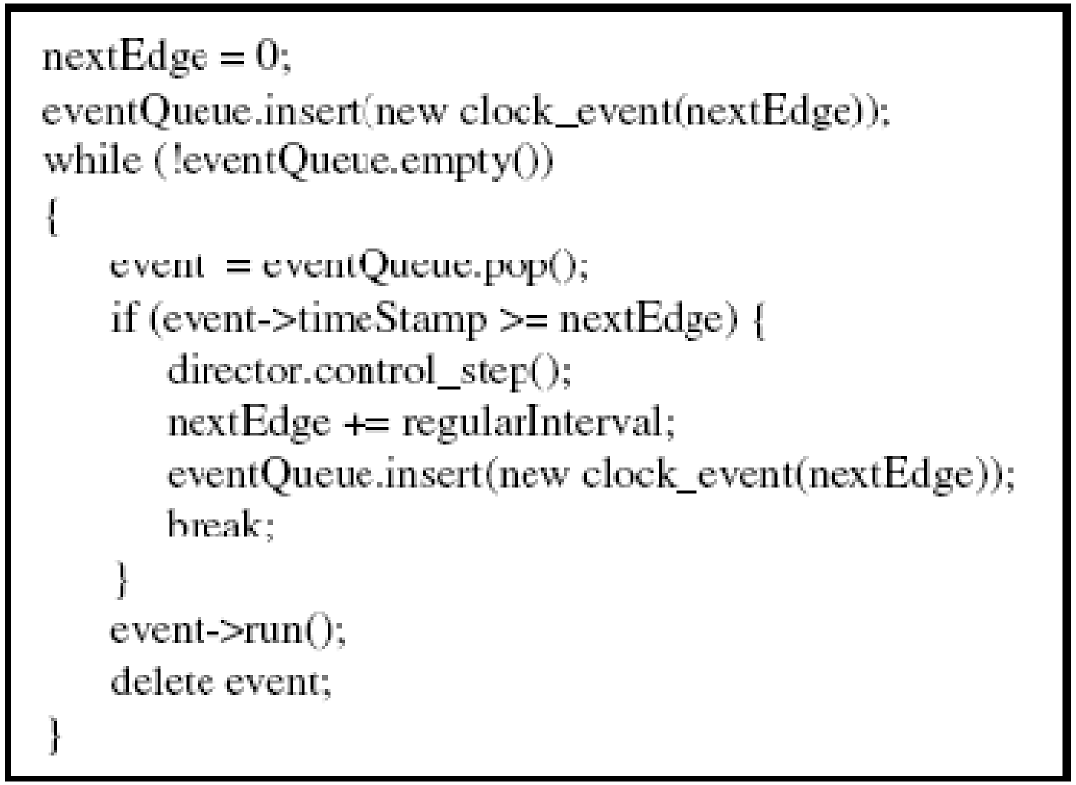
Scheduling OSMs (sequential)



```
Director::control_step()
{
    updateOSMList();
    OSM = OSMList.head;  // head.next is the first
    while ((OSM=OSM.next)!=OSM.tail) {
        EdgeList = OSM.currentOutEdges();
        foreach edge in EdgeList {
            result = OSM.requestTransactions(edge);
            if (result == satisfied) {
                OSM.commitTransactions(edge);
                OSM.updateState(edge);
                OSMList.remove(OSM);
                OSM = OSMList.head;
                break;
            }
        }
    }
}
```

Deadlocks from cyclic resource dependency is considered pathological.

Simulating OSMs

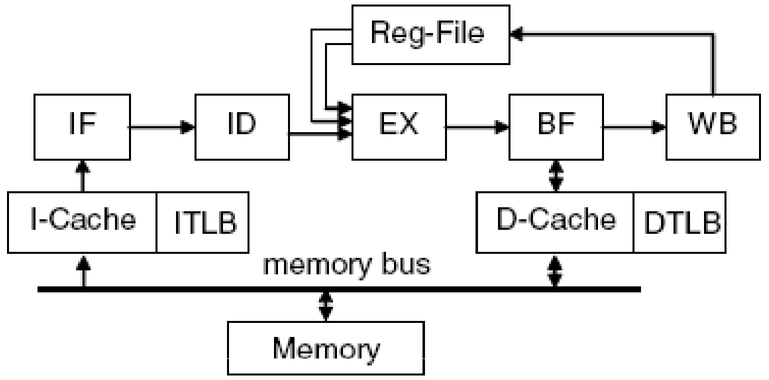


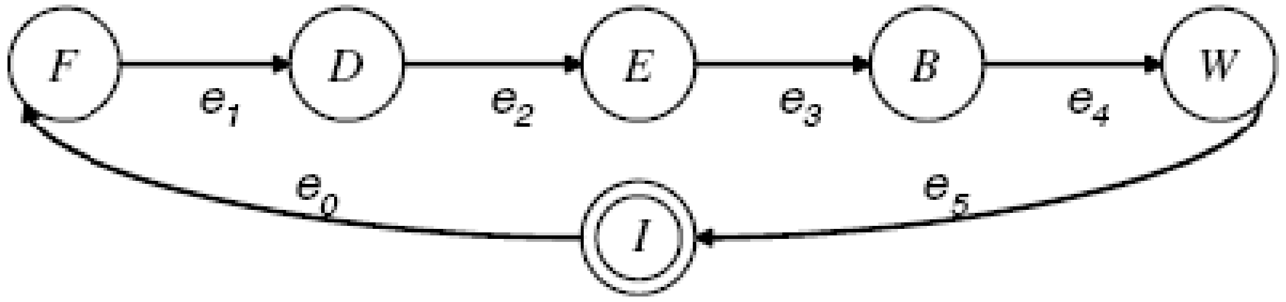
```
nextEdge = 0;
eventQueue.insert(new clock_event(nextEdge));
while (!eventQueue.empty())
{
    event  = eventQueue.pop();
    if (event->timeStamp >= nextEdge) {
        director.control_step();
        nextEdge += regularInterval;
        eventQueue.insert(new clock_event(nextEdge));
        break;
    }
    event->run();
    delete event;
}
```

OSM MoC is embedded inside the DE schduler.

Modeling Hazards

- Structure hazard
- Data hazard
- Variable latency
- Control hazard





OSM of a 5-stage pipelined RISC processor

Outline

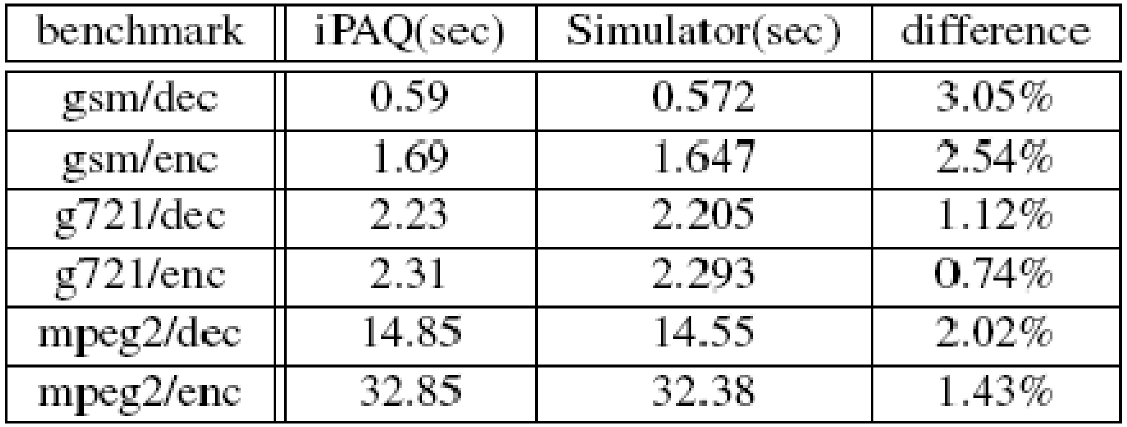
- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies

Case Studies

- StrongARM: a five-stage pipelined implementation of the Advanced RISC Machine architecture.
- PowerPC 750: a dual-issue out-of-order superscalar processor.

StrongARM

- The resulting simulator runs at 650k cycles/sec compared to 550k cycles/sec by SimpleScalar.
- Model validated against an iPAQ-3650 PDA containing a SA-1100.



benchmark	iPAQ(sec)	Simulator(sec)	difference
gsm/dec	0.59	0.572	3.05%
gsm/enc	1.69	1.647	2.54%
g721/dec	2.23	2.205	1.12%
g721/enc	2.31	2.293	0.74%
mpeg2/dec	14.85	14.55	2.02%
mpeg2/enc	32.85	32.38	1.43%

PowerPC 750

- Validated against a SystemC based model using a benchmark mix from MediaBench and SPECint 2000, and found that differences in timing are within 3% in all cases.
- OSM simulator runs at 250k cycles/sec, 4 times of the SystemC model.

Productivity

Lines of code	SA-1100	PowerPC 750
OSM Total	3032	5004
SimpleScalar/SystemC	4633	16000

About 60% of the source code for the OSM model is dedicated to instruction decoding and OSM initialization, which can be automatically synthesized through the use of an architecture description language. Most hardware modules and their TMIs were reused across the two targets.

Outline

- Introduction
- Motivation
- Prior Work
- The Operation State Machine (OSM) Model
- Case Studies
- Conclusion

Conclusion

- Where the paper succeeds:
  - OSM as an efficient retargetable simulator generation framework for different microprocessor architecture including scalar, superscalar, very-long-instruction-word and multi-threaded (add tag).
- Where the paper fails:
  - Architecture description language can be derived by not done (the declarative criterion).
  - Mentioned OSM as ASM but how to do successive refinement is not clear.

Q & A

Thank you.