ECIT | The Institute of Electronics, Communications and Information Technology

Queen's University Belfast

Advanced Multithreading Architecture with Hardware based Thread Scheduling

Ye Lu
Sakir Sezer
John McCanny

FPL 2010 in Milan

Project supported by the EU Programme for Peace and Reconciliation
Invest Northern Ireland
Department for Employment and Learning

ECIT | Overview

- Introduction to multithreading
- Architecture overview
- Thread scheduler
- Pipeline control
- Synthesis and Simulation results
- Conclusions

Introduction

- Hardware support for effectively execute multiple threads on a single core
- Taxonomy
  - Fine Grained MT (IMT)
  - Coarse Grained MT (BMT)

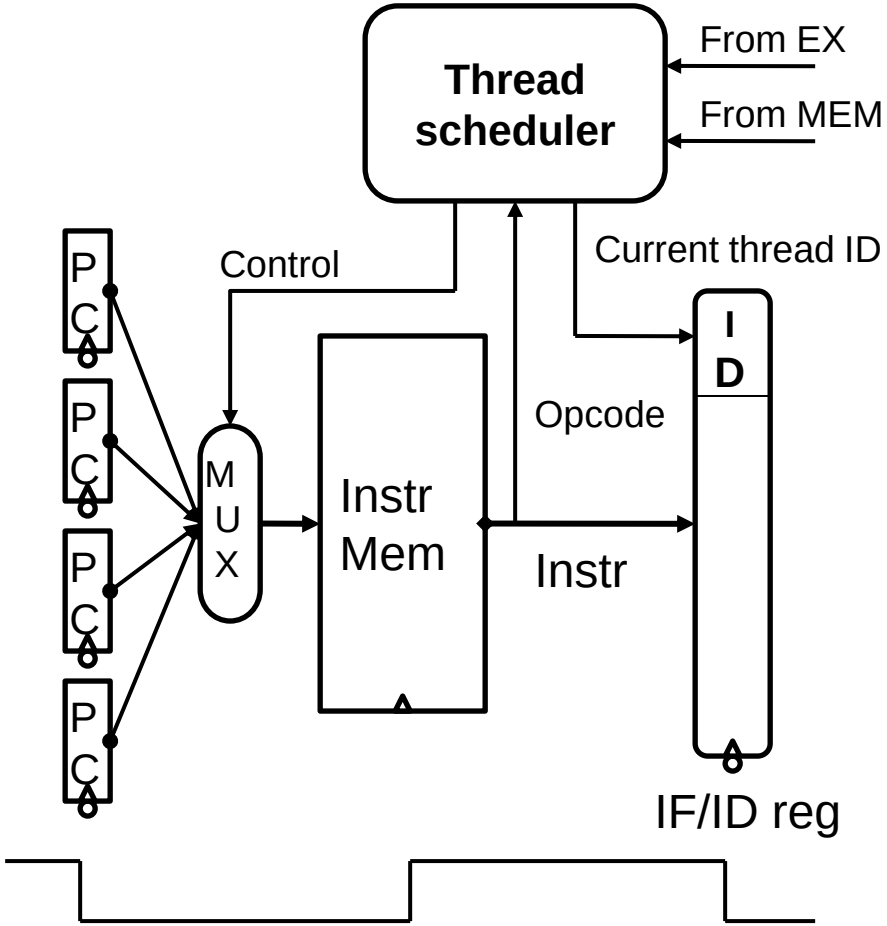
  } Prefered for softcore implementation

  - Simultaneous MT
- Goals of the proposed architecture
  - Reduced hardware overhead
  - Low context switching penalty
  - Maintaining single thread performance

ECIT | Architecture overview

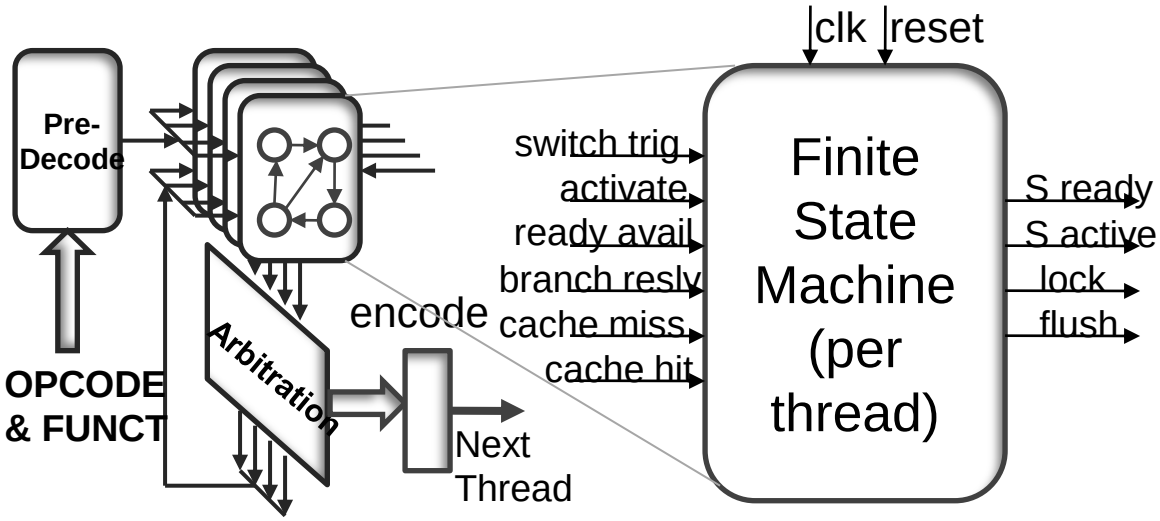
- Advanced thread management
- Zero context switch penalty
- “Temporary execution”
- 5 pipeline stages
- Supporting 4 threads each core
- MIPS ISA for prototyping

Thread scheduler



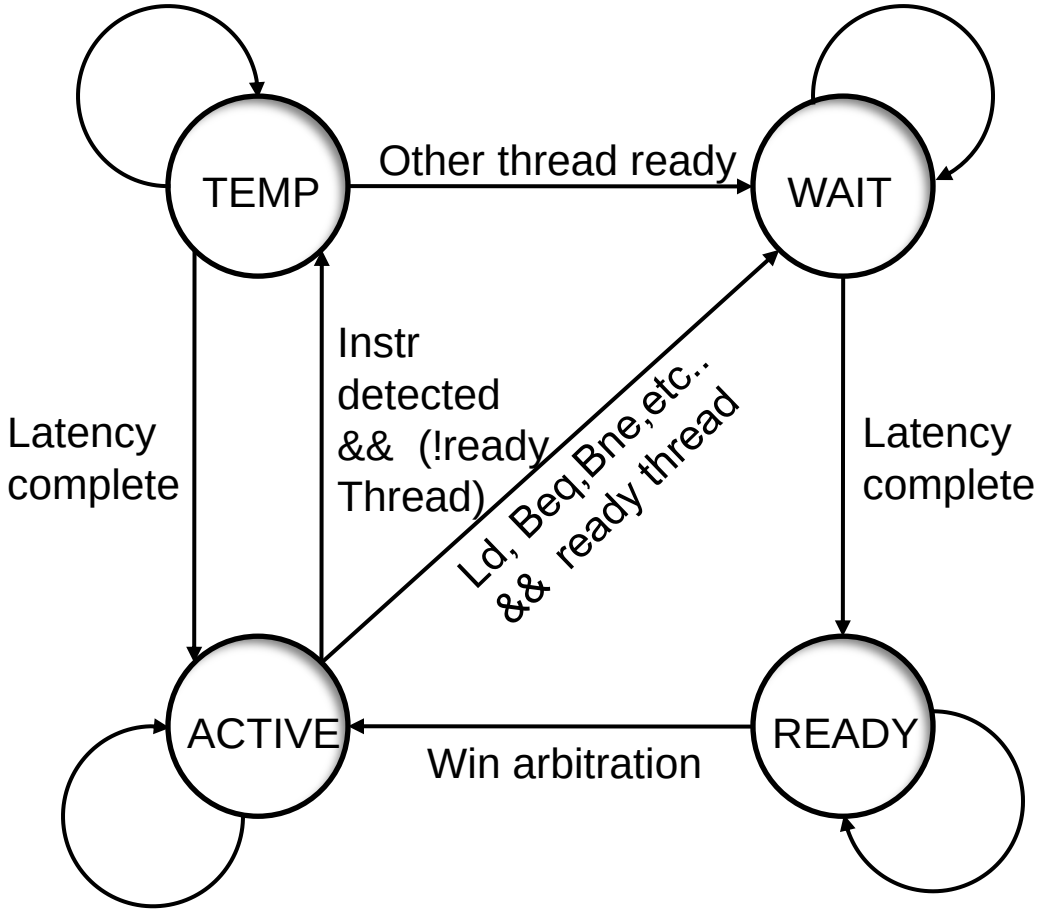
- Pre-decodes the instruction
- Controls the thread switching
- Maintains the thread state
- Tracks the status of execution and memory access stages
- Desides the next active thread

ECIT | Hardware Design



- Pre-decoding logic detects the instruction
- Finate state machine manages the thread state
- Arbitration logic decides the next active thread

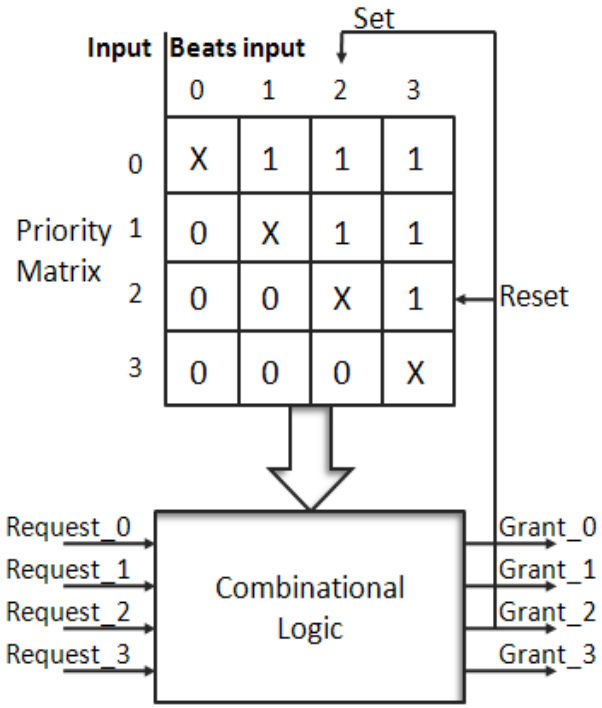
FSM State Diagram



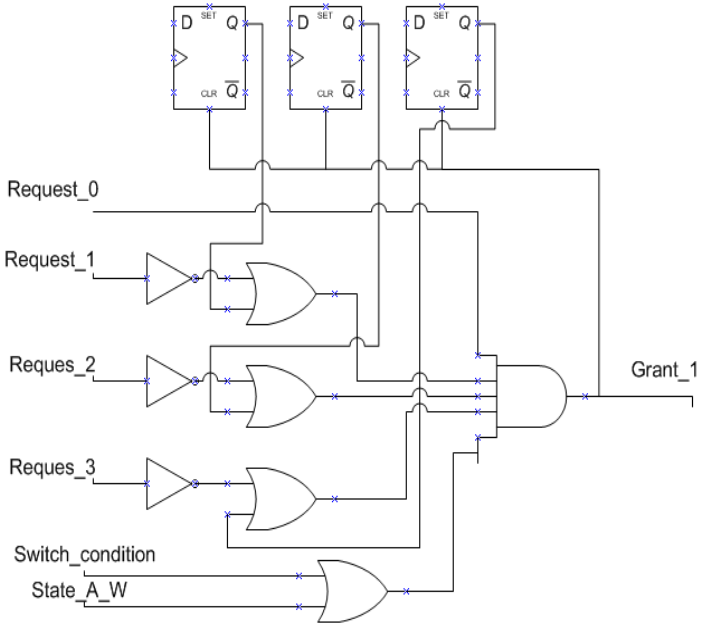
- ACTIVE : Thread is executing
- WAIT : Thread is waiting for branch decision or memory access
- READY : Thread is ready to be executed
- Temp: Instruction detected but no alternate thread

Thread Arbitration

Structure

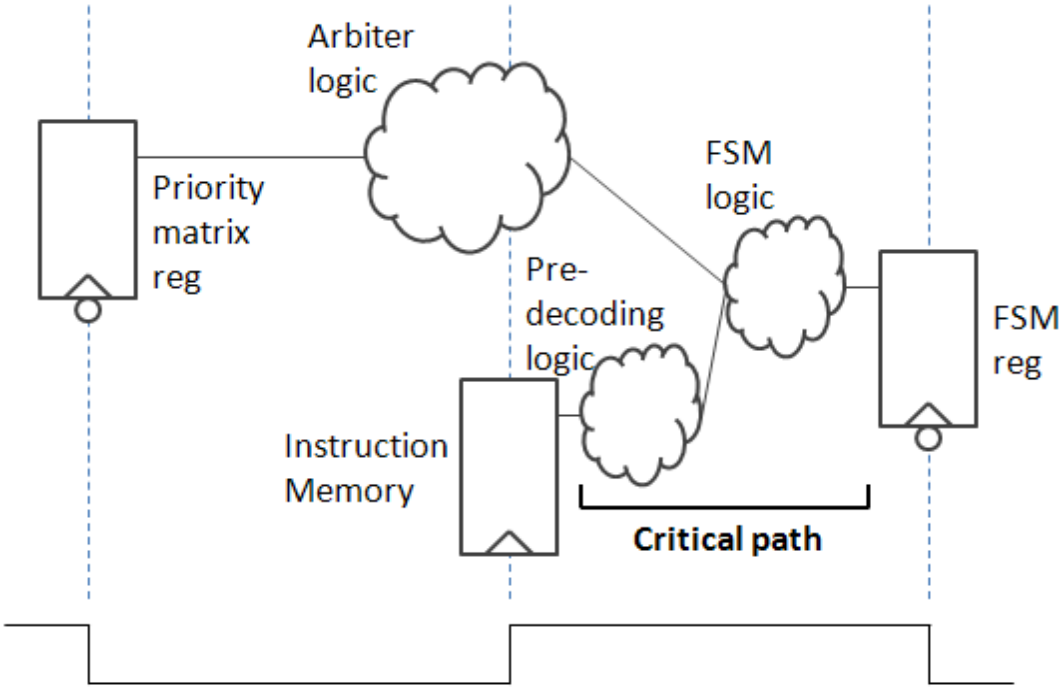


Circuit



- Least recently served scheme
- Matrix arbiter

ECIT | Critical Path

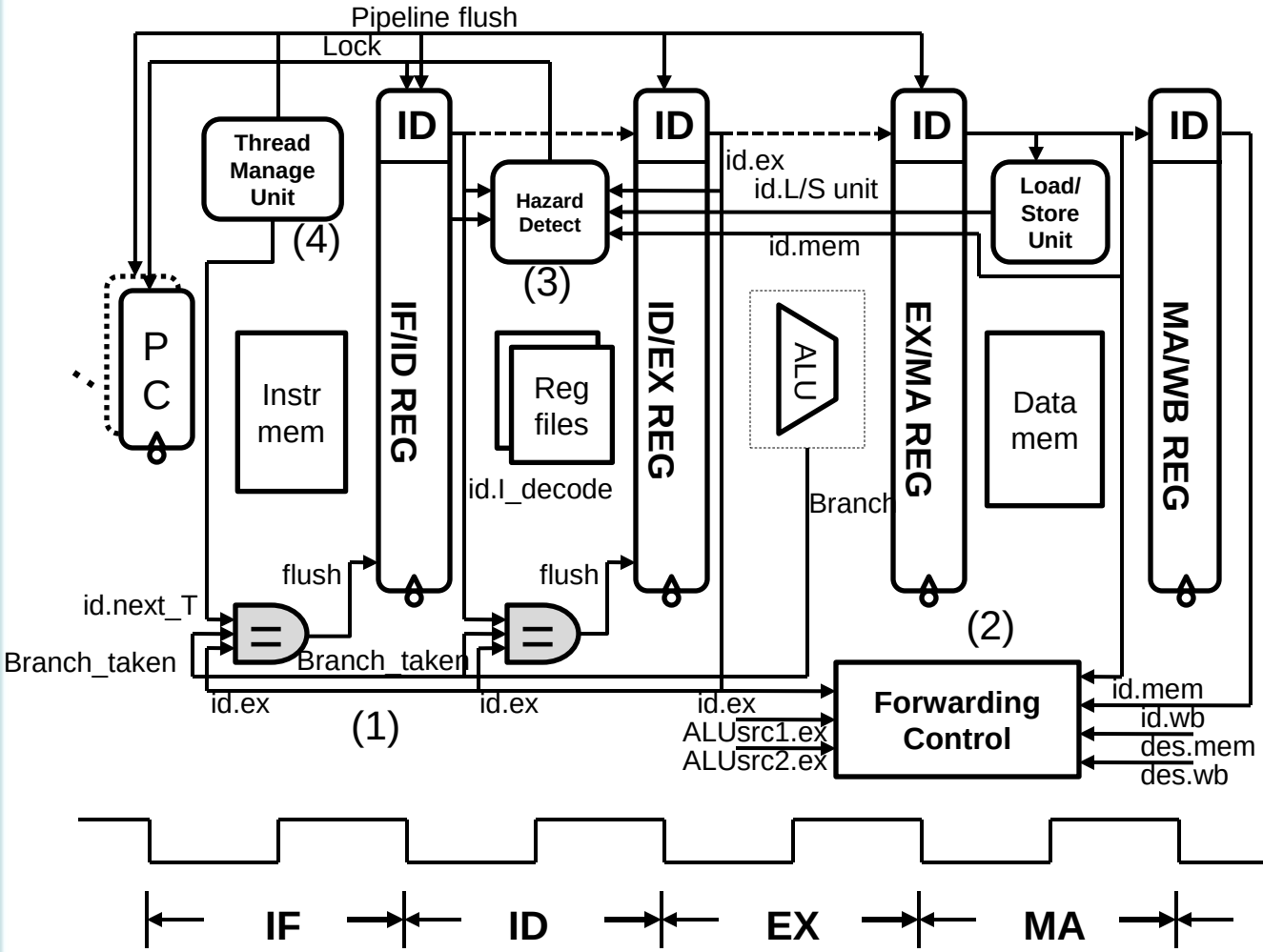


- Date are launched and latched at different clock edges, thus the scheduler is working at doubled clock rate
- Moving the thread arbitration logic out of the critical path and using one-hot encoding to increase the speed

Pipeline Control

- Supporting multiple thread to share the pipeline resource
- Resolving the possible pipeline hazard
  - Control hazard
  - Data hazard
- Using thread tags to distinguish instructions from different threads

ECIT | Pipeline Control



(1) Control hazard

(2) Forwarding

(3) Load delay

(4) Pipeline lock & flush

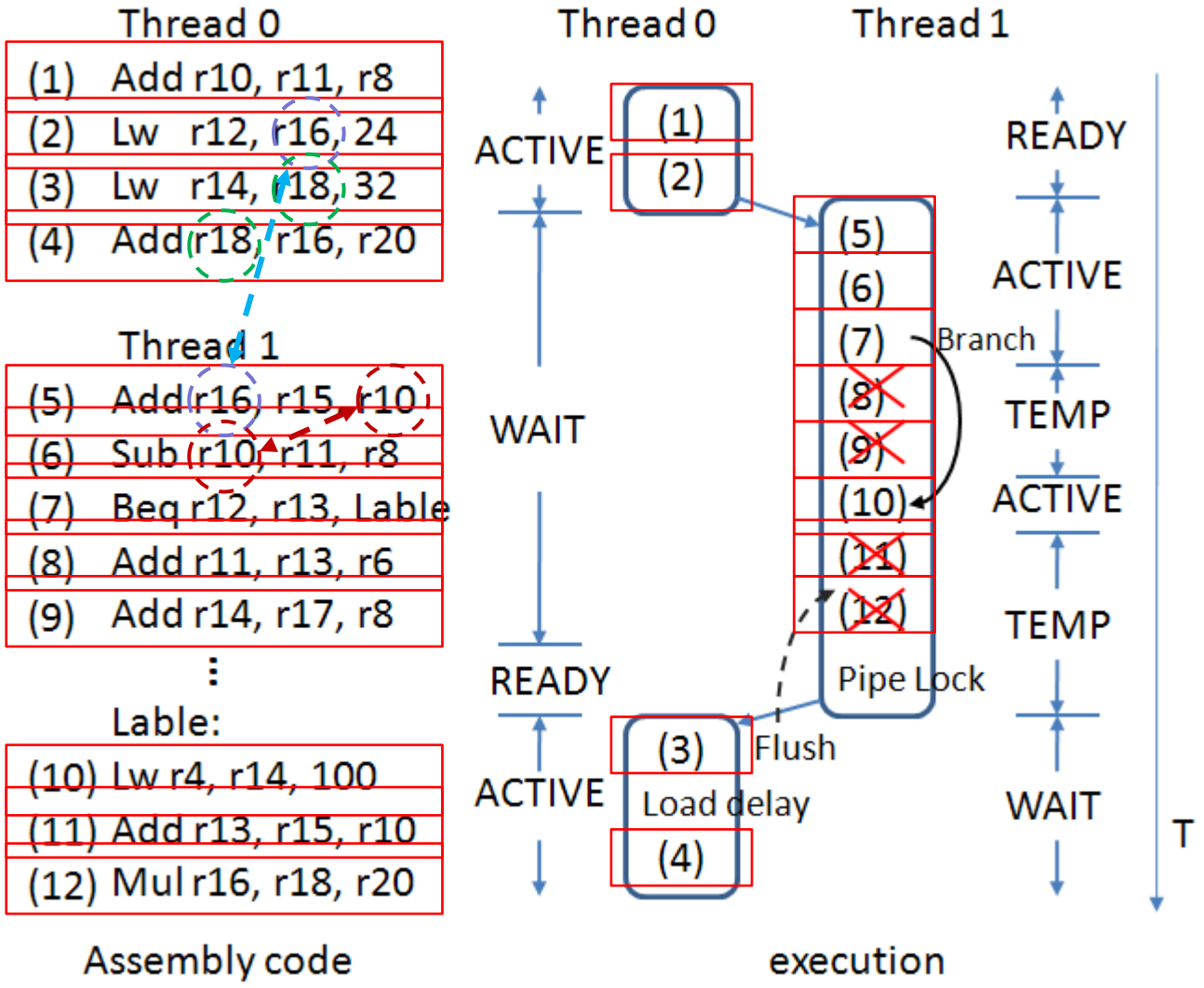
ECIT | Pipeline Control

1. Control hazard
   - ➢ Hiden by the thread switching
   - ➢ When no alternate thread and mis-prediction happens
     - ✓ Exam the ID of instructions
     - ✓ Flush the corresponding pipeline registers
     - ✓ Change the PC
2. Conventional RAW data hazard
   - ➢ No effect to the performance
     - ✓ Exam the ID of instructions
     - ✓ Resolved by the forwarding logic

Pipeline Control

3. Load delay (RAW following load)
   - Hiden by the thread switching
   - When no alternate thread and dependence happens
     - ✓ Can not be resolved by the forwarding path
     - ✓ Stall the PC and IF/ID pipeline register
4. Cache missing without alternate thread
   - Stall the pipeline
   - When any other thread ready, switch to the ready thread, flush the pipeline, recover the context

ECIT | Example



Results

- Logic overhead

TABLE I
SYNTHESIS RESULTS OF THE THREAD MANAGEMENT UNIT AND PIPELINE CONTROL

Components	Number of ALUTs	Number of Registers
Pre-decoding logic	3	0
FSM (total)	30	16
Arbitration logic	14	6
Control hazard logic	2	0
Forwarding path	78	0
Load delay control	6	0
Total	133	22

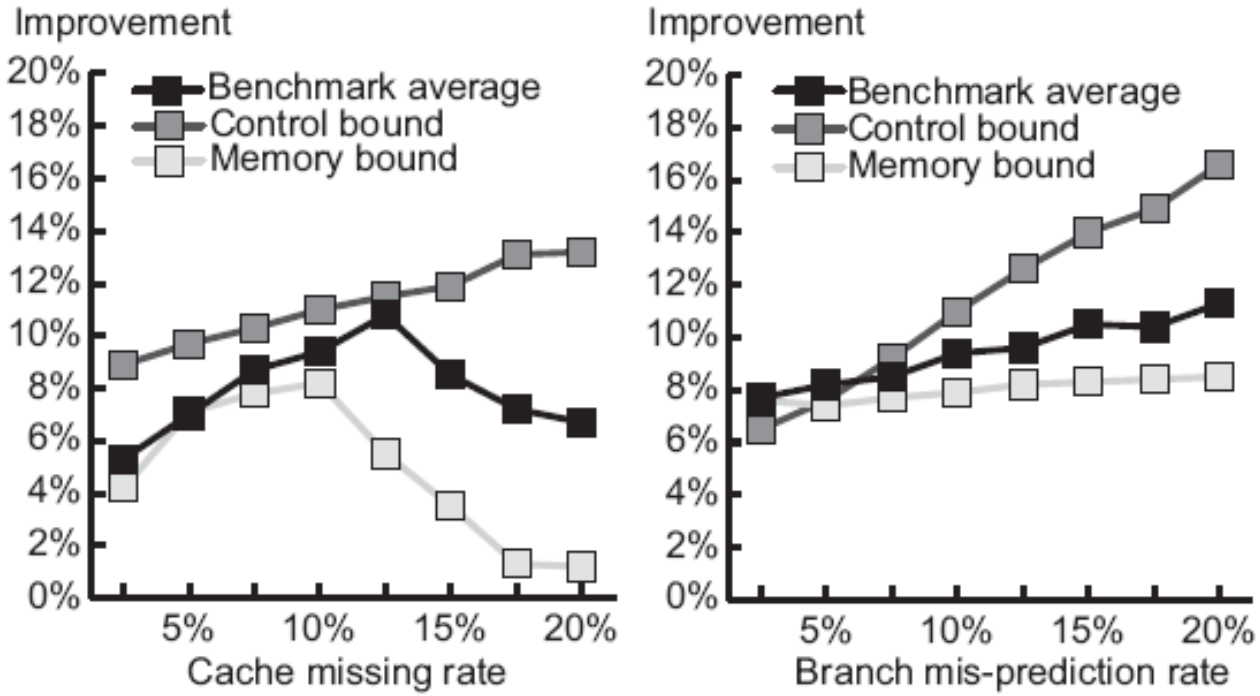
- Clock rate

TABLE II
STATIC TIMING ANALYSIS OF THE THREAD MANAGEMENT UNIT

Item	Value
Interconnection Delay	0.708 ns (45% of total)
Cell Delay	0.774 ns (50% of total)
uTco	0.066 ns (4% of total)
Fmax	324Mhz

ECIT | Performance Evaluation

Compared to a conventional BMT architecture



Reference:

Ye Lu, Sakir Sezer, John McCanny, Design and Analysis of an Advanced Static Blocked Multithreading Architecture, SOCC 2010

Conclusions

- We present a multithreading architecture that manages thread according to the thread state and instruction decoding
- The architecture can eliminate the context switching penalty while maintaining the single thread performance
- We demonstrant its performance and feasibility for softcore implementation

ECIT | Q & A

Thanks for your attention

Questions?