# MorphCore

AN ENERGY-EFFICIENT MICROARCHITECTURE FOR HIGH PERFORMANCE ILP AND HIGH THROUGHPUT TLP

# The Paper

Authors

- Khubaib
- M. Aater Suleman
- Milad Hashemi
- Chris Wilkerson
- Yale M. Patt

Published for MICRO 2012

Presented by Georgijs Vilums

# Agenda

Background and Motivation
- Workloads
- Current Designs

Design of MorphCore

Design Evaluation
- Performance
- Power Usage

Paper Evaluation

Discussion

# Background and Motivation

WORKLOADS AND CURRENT DESIGNS

# Most common Workloads

## SINGLE THREAD

Instructions are fetched from a single stream

- Parallelism arises between instructions

Desired Characteristics

- High Performance
- Low Latency
- Energy Efficiency

## MULTIPLE THREADS

Instructions can be fetched from multiple streams

- Parallelism between threads can also be exploited

Desired Characteristics

- High Throughput
- Energy Efficiency

# Overview: Out-of-Order-Execution

Want to execute instructions in any order, as long as semantics stay the same

- Can skip waiting for independent instructions
- Less cycles wasted stalling

Core components

- RAT: Prevents register name conflicts
- RS: Instructions wait for their operands to become ready
- Scheduler: Chooses any instruction with ready operands for execution

Independent instructions can execute in any order, exploiting ILP

# Overview: InOrder SMT

Want to execute multiple threads concurrently

- When one instruction has to wait, just execute instructions from another thread

Instruction Queues

- An SMT-Core has multiple Queues, each filled with instructions from different threads

Wakeup

- Head instruction of any of the queues is selected, provided that it does not wait on operands
- Instructions from each thread execute in order

Thread execution is interleaved, exploiting TLP

# What are the problems?

## OUT-OF-ORDER-EXECUTION

Consumes a lot of energy

Reordering unnecessary when TLP could be exploited

- Non-Ideal throughput when working with multiple threads as work is wasted optimizing ILP
- Wasted energy

## SIMULTANEOUS MULTITHREADING

Low performance when working with small number of threads / single thread

- Does not exploit ILP at all

# Summary

Modern workloads are varied

We want the best of both worlds:

- Exploit ILP when working with a single thread
- Exploit TLP when working with multiple threads

Putting two different cores on one chip comes with a large area overhead

# Agenda

Background and Motivation
- Workloads
- Current Designs

Design of MorphCore

Design Evaluation
- Performance
- Power Usage

Paper Evaluation

Discussion

# The Best of Both Worlds

DYNAMICALLY CHANGING CORE LAYOUT

# Basic Idea

Core can work both in OoO-mode and InOrder-Mode

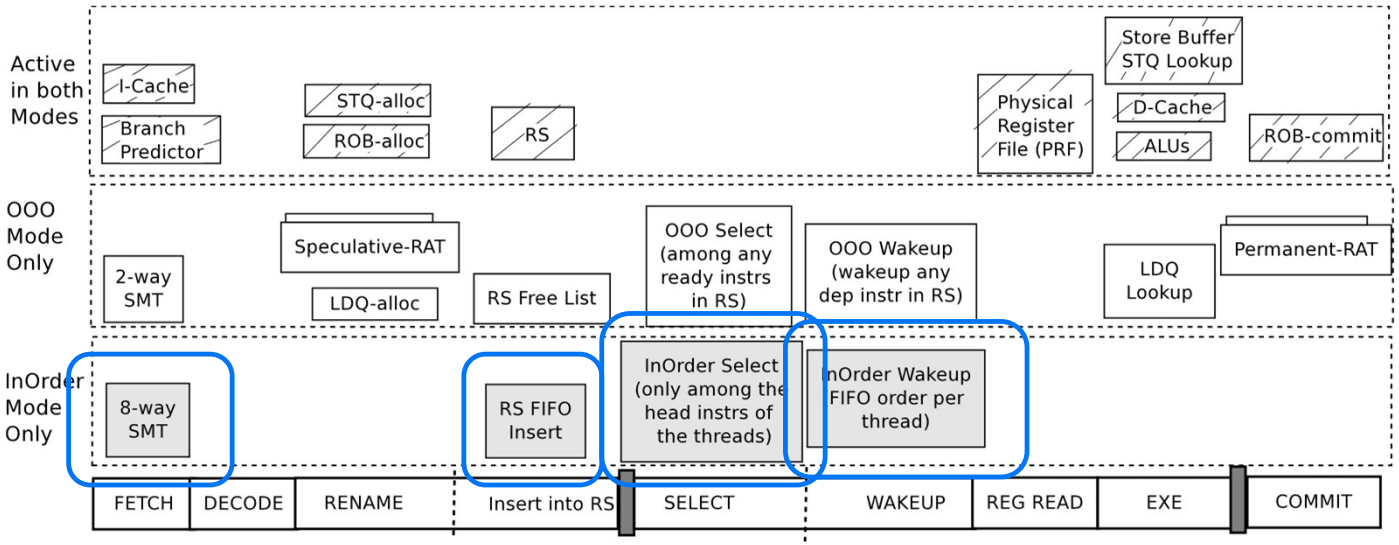
Many Components of an OoO core can also be used when operating as InOrder core

- InOrder is simpler, requires less logic
- Smaller overhead than implementing an entire second core optimized for InOrder

Switch core from OoO to InOrder when many threads available

Back to OoO when threads block / are terminated

# General Architecture



# Fetch and Decode

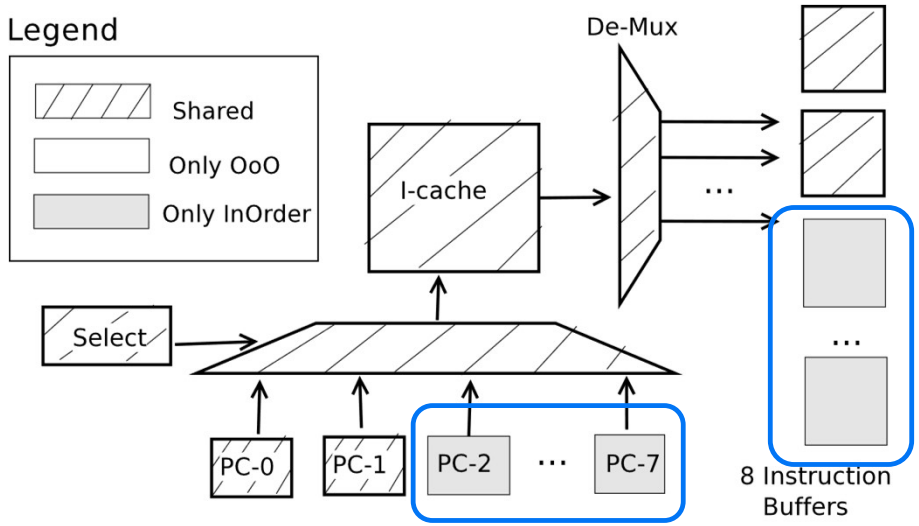
Want to fetch from more instruction streams

Additional Logic:

- Program counters
- Branch history registers
- Instruction Buffers
- Larger Multiplexer

Note: Multiplexer on critical path

- Lower maximum clock rate



# Rename

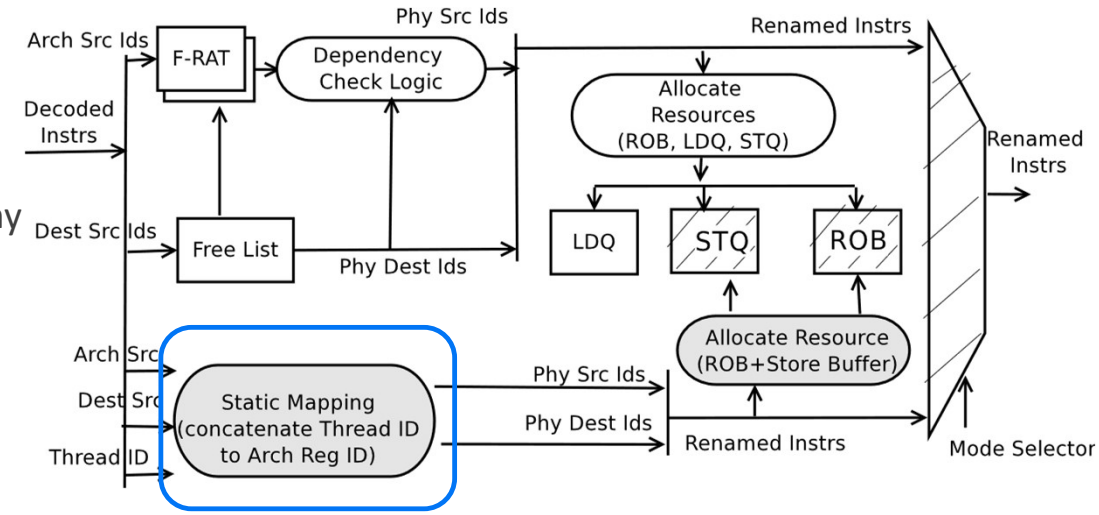
Need a location for storing register data of each thread

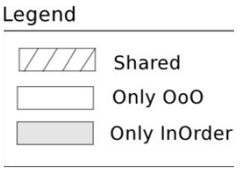
Recall:

- In OoO, the physical register file (PRF) has many more entries than the architecture exposes

In InOrder-mode part of PRF is dedicated to each thread

- Thread ID determines region
- No complicated renaming logic required





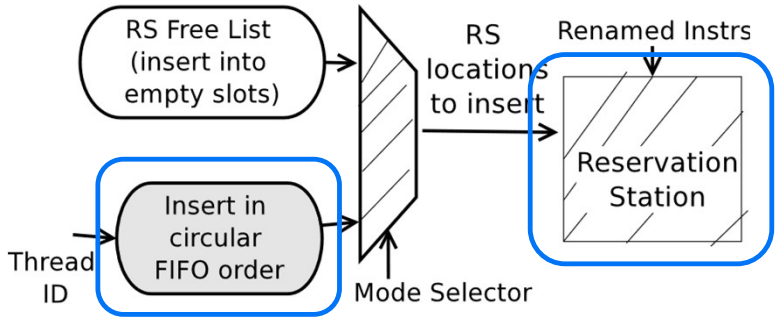

# Dispatch

Recall:

- In OoO, instructions wait in the reservation station (RS) until operands are ready

In InOrder, similar to Rename, each thread is allocated part of the RS

As each thread operates in order, a simple circular FIFO queue determines placement of new instruction in RS



# Wakeup and Select

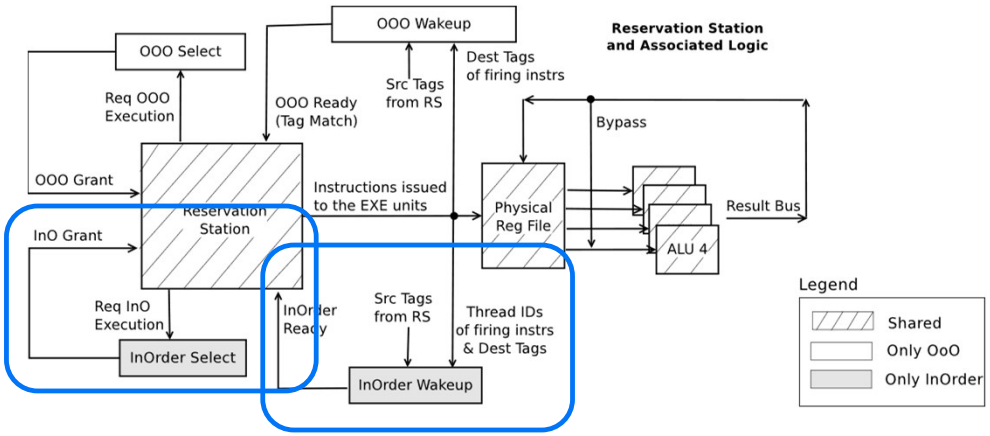
Need to wake up instructions when operands are ready, then select for execution

Recall:

- In OoO, instructions have to monitor broadcasts for relevant operands
- Once operands are ready the instruction can be issued

InOrder Wakeup also keeps track of ready operands for instructions

Only instructions from head of each instruction stream can be selected for execution



# Switching Modes

## OOO TO INORDER

Core monitors the number of active threads

- Threads count as inactive when blocking (IO)

Once number of threads reaches set threshold, switch to InOrder-mode

- Drain Pipeline
- Relocate data into correct partitions in PRF
- Disable unnecessary components

## INORDER TO OOO

Once number of active threads drops too low, switch back to OoO-mode

- Drain Pipeline
- Spill registers to memory
- Load active thread registers back into PRF
- Reenable OoO components

# Summary

Not much additional Logic required for implementing InOrder SMT

Many structures from OoO core can be reutilized in a slightly reconfigured way

When operating in order, multiple components which require a lot of power can be disabled (no clock)

Additional logic on critical path decreases maximum possible clock rate

# Agenda

Background and Motivation
- Workloads
- Current Designs

Design of MorphCore

Design Evaluation
- Performance
- Power Usage

Paper Evaluation

Discussion

# Evaluation

PERFORMANCE AND POWER CHARACTERISTICS

# Test Configuration

Machine

- OOO core with fetch width 2 as basis
- Can switch to InOrder-mode with fetch width 8
- OOO-mode with 1 or 2 threads, InOrder-mode with more than 2

Data

- Several workloads using only a single thread (ST)
- Other workloads using multiple threads (MT)

# Points of Reference

## OUT OF ORDER

### OoO-2

- Standard OoO core which can execute two threads concurrently

### OoO-4

- Standard OoO core, with additional hardware to enable the execution of four concurrent threads

### MED

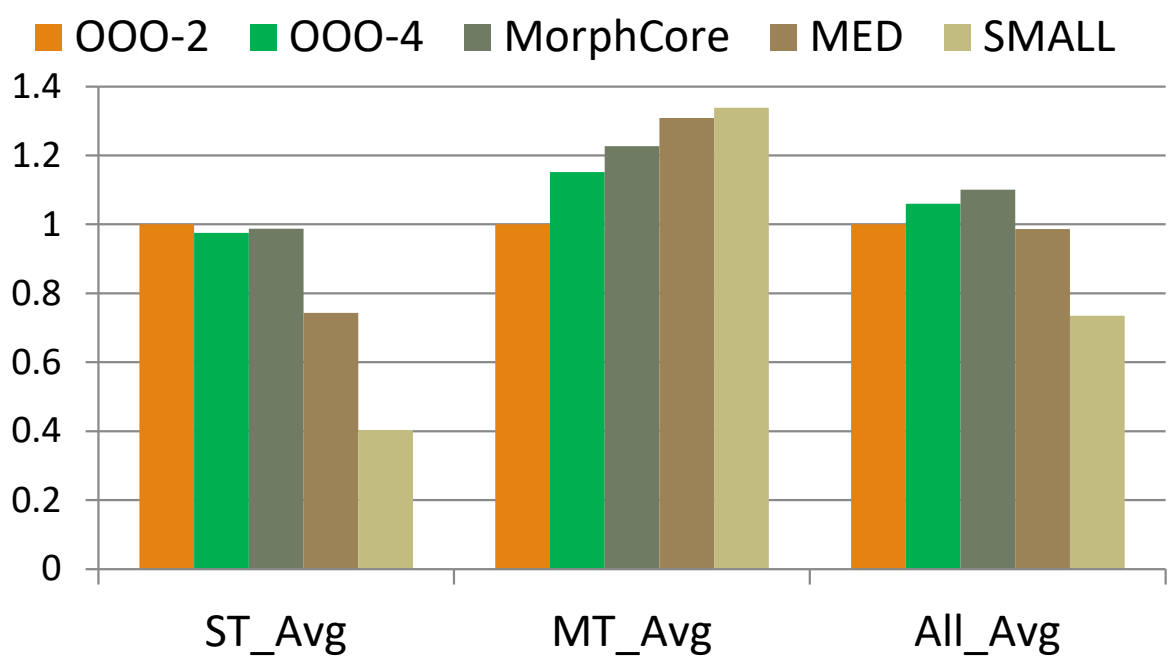
- A cluster of three OoO cores, where each core can execute one concurrent thread

## IN ORDER

### SMALL

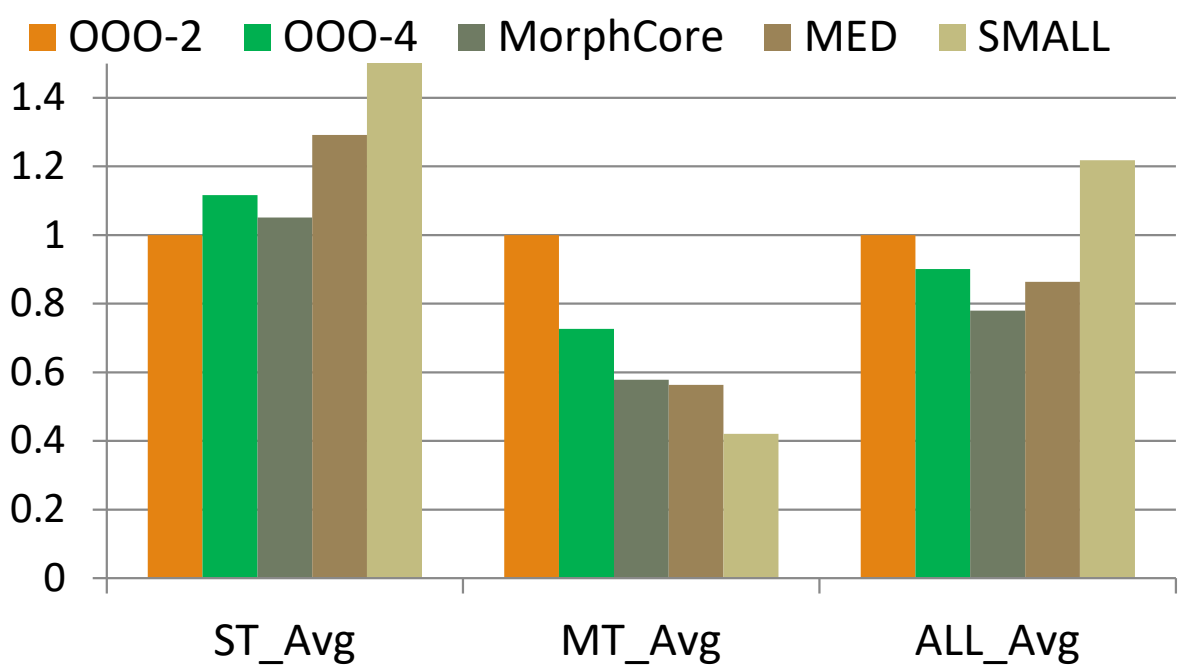
- Cluster of three InOrder cores, each executing two concurrent threads

# Performance




- Almost matches OOO-2 in single-threaded tasks
- Beats OOO-2 and OOO-4 in multi-threaded tasks, beaten by MED and SMALL
- Overall best performance

# Energy-Delay-Squared




- Similar to performance, almost matches OOO-2 in ST, beaten by MED and SMALL in MT
- Again, overall best (lowest) Energy-Delay-Squared

# Agenda

Background and Motivation
- Workloads
- Current Designs

Design of MorphCore

Design Evaluation
- Performance
- Power Usage

Paper Evaluation

Discussion

# Paper Critique

STRENGTHS & WEAKNESSES

# Strengths

## DESIGN

Significant gains in MT performance, efficiency
- Makes large OoO-cores more flexible
- Allows use in devices with stricter power budgets

Changes are transparent to user
- Eases adoption, software does not have to be redeveloped

Already present hardware is repurposed
- Low area overhead
- Less changes to design

## PAPER

Provides well-explained and thorough motivation for the issue

Thorough analysis, comparison to other common and alternative architectures

Performance losses in some areas are acknowledged

# Weaknesses

Flexibility comes at the cost of overhead

- Single-threaded applications suffer a (slight) performance penalty
- ST-workloads are still very common

Might not be flexible enough

- For example, if designed for 1/8+ threads, energy-delay-squared might suffer at 2-7 threads

# Takeaways

Dynamically change between executing…

- … few threads out of order, exploiting ILP
- … many threads in order, exploiting TLP and saving power

Sizeable performance gain in MT-applications

Changes transparent to user

- Makes adoption easier

Additional overhead when executing ST only

- Might be hindering adoption

# Agenda

Background and Motivation
- Workloads
- Current Designs

Design of MorphCore

Design Evaluation
- Performance
- Power Usage

Paper Evaluation

Discussion

# Discussion Starters

Do you think such dynamic core architectures will become more common in the future?

- Why?
- Why not?

Should the mechanism for mode switching be controllable by the programmer?

- What benefits could this bring?
- What could be the negative consequences?

Do you see other issues that the design might have?

# Thank You for your Attention