



# Runahead Threads to Improve SMT Performance



**Tanausú Ramírez**



**Alex Pajuelo**



**Oliverio J. Santana**



**Mateo Valero**



*14th Symposium on High-Performance Computer Architecture*

**HPCA'08, Salt Lake City, UT**



# Simultaneous Multithreading

- execute multiple instructions from several threads at the same time



- ▶ threads **share** and also **compete** for important resources in the SMT processor core

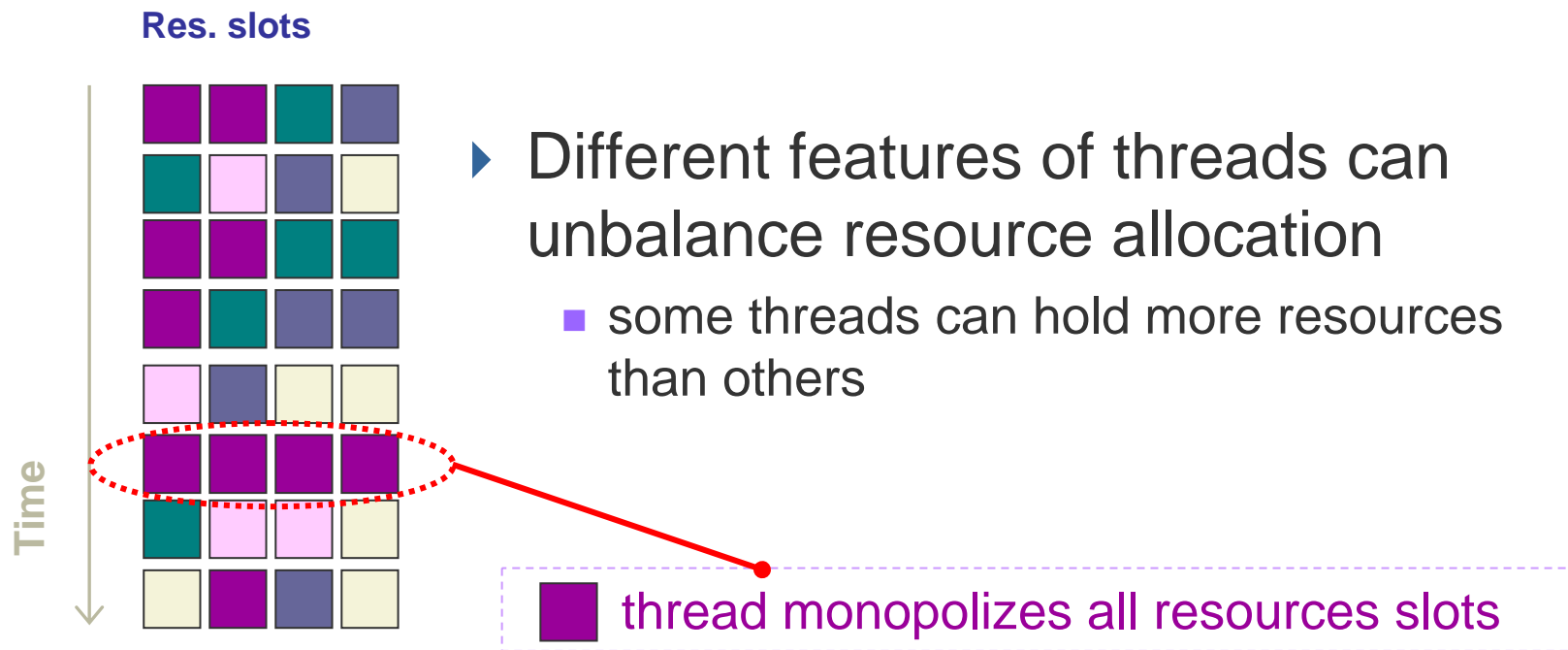
threads





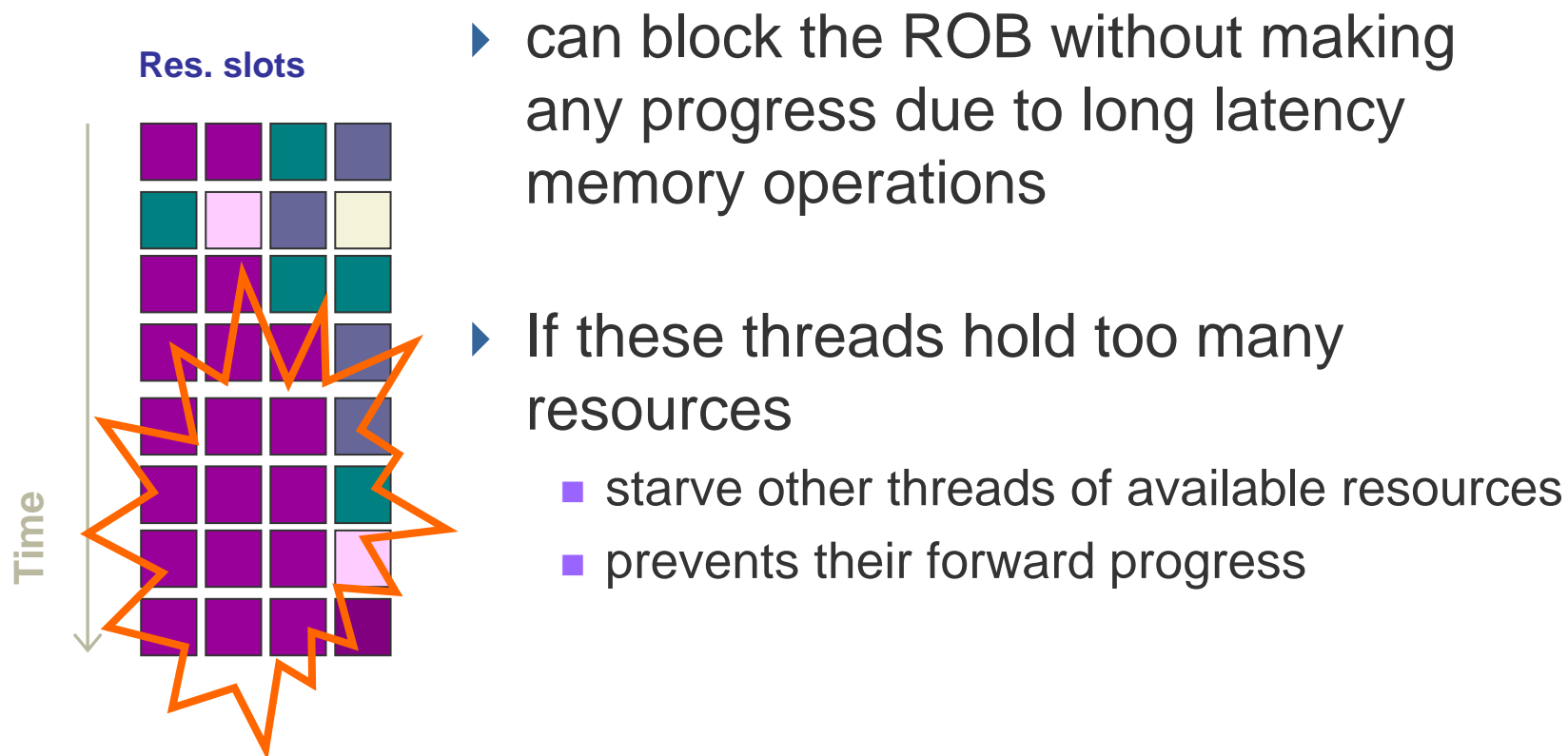
# Resource sharing

- resource conflicts may significantly hinder the benefit of multithreaded execution



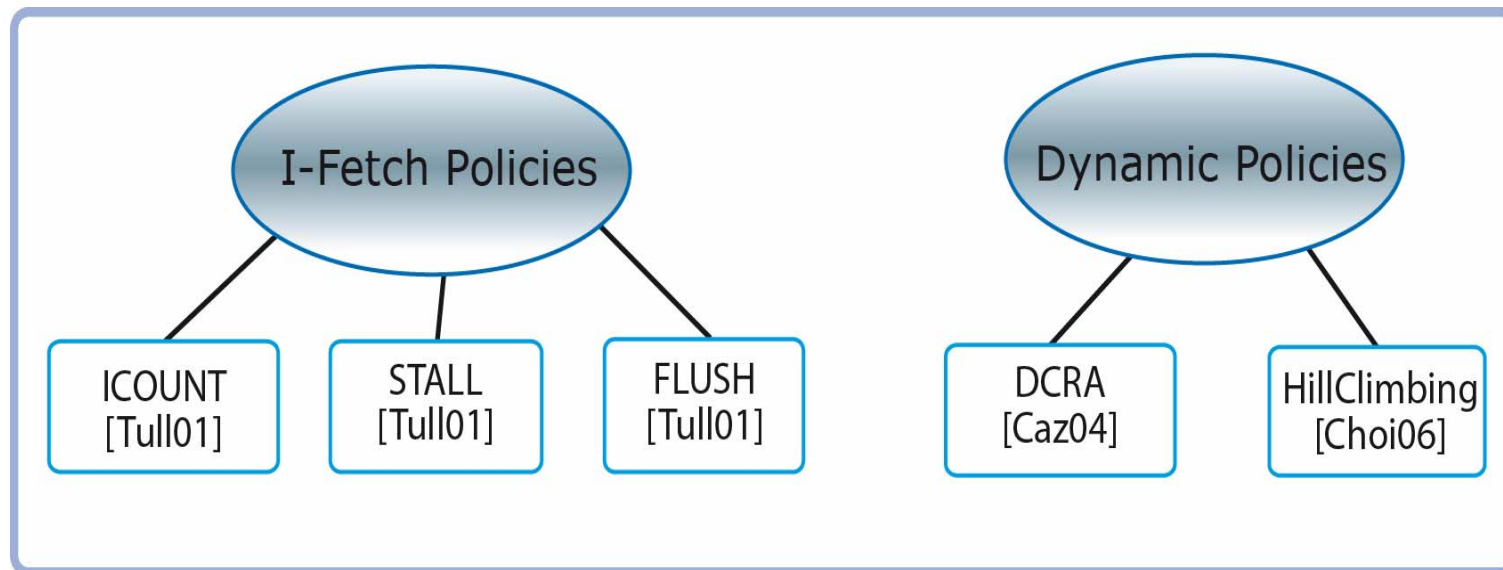
# Worst case

## ■ *Memory-bound threads*



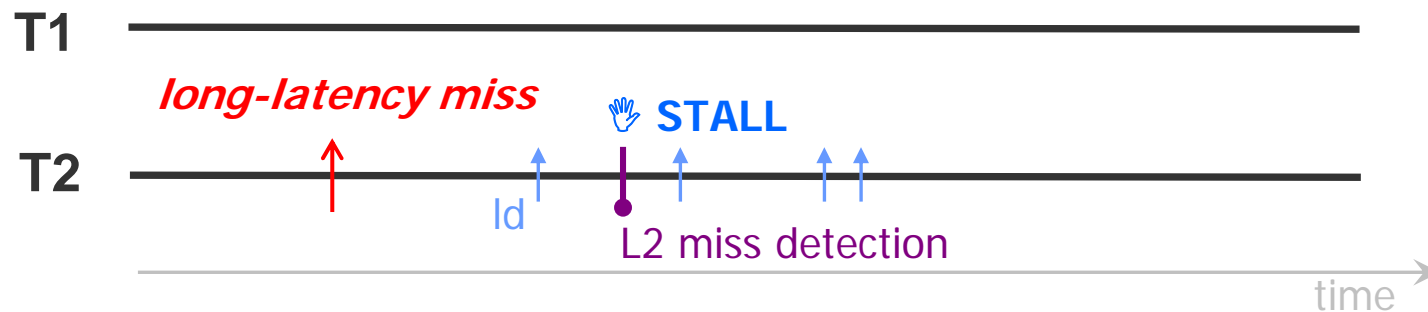
# Current Policies

- Several policies have been proposed
  - ▶ determine how these resources should be shared





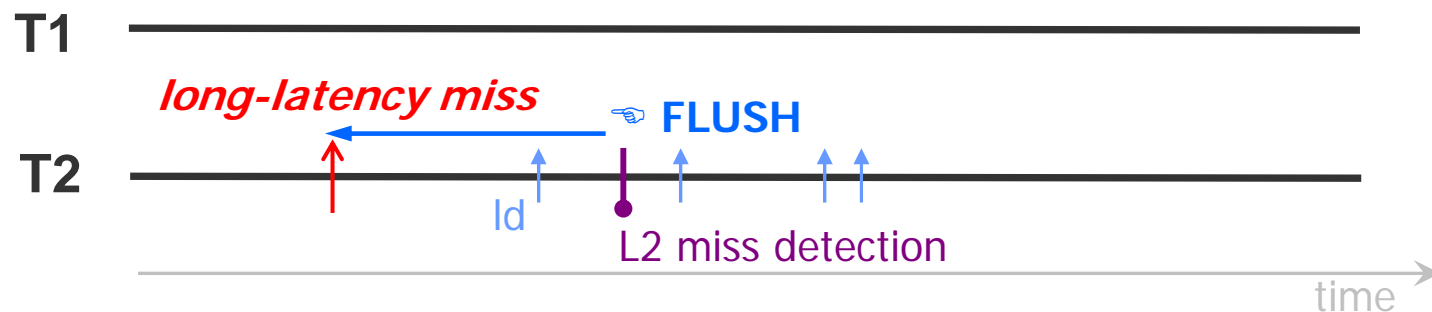
# STALL



- ▶ memory-bound thread is stalled
- ▶ thread (T2) **holds the resources**



# FLUSH



- ▶ memory-bound thread is flushed
- ▶ thread (T2) frees the resources
- ▶ but it is **also stopped**



# DCRA and HillClimbing

- control dynamically the resources allocation to prevent resource overuse
  - ▶ DCRA uses several resource counters to check resource allocation
  - ▶ HillClimbing resource scheduler is based on improving a performance function
- both **stall** a thread when it consumes all of its given amount of resources





# Overall Problem

- These techniques stall threads during memory intensive periods
  - ▶ loss performance opportunities of memory-bound threads to unfairly benefit fast threads
    - threads will advance too slowly
    - available MLP is not exploited
  - ▶ ***fail to reach the potential performance achievable in SMT processors***





# Our Proposal

## ■ Goal

- ▶ improve performance of SMT processors without harming memory-bound threads
  - neither let threads excess in the use of resources
  - nor keep on stopping threads

## ■ We propose **Runahead Threads (RaT)**

- ▶ new utilization of the Runahead execution as a different memory-aware SMT policy





# Talk Outline

## ■ Introduction

- ▶ The problem of sharing resources
- ▶ Current Solutions

## ■ Our Proposal: Runahead Threads

- ▶ Runahead Execution
- ▶ Runahead Threads

## ■ Evaluation

- ▶ Framework
- ▶ Comparison

## ■ Conclusions





# Runahead Background

## ■ *Runahead Execution*

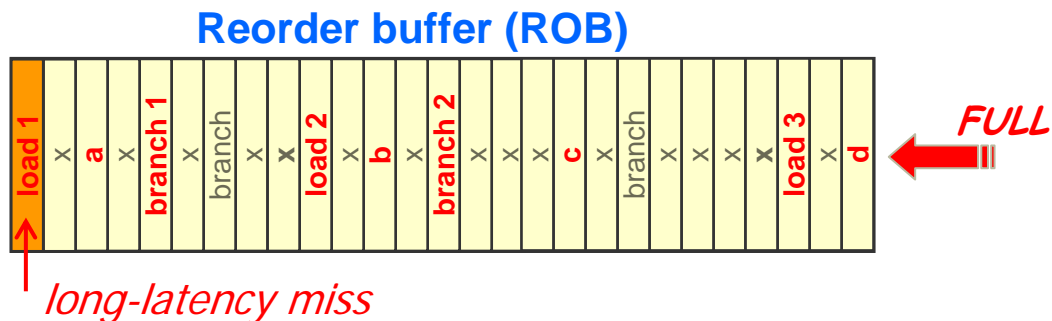
- ▶ alternative mechanism to large instruction windows for single-thread processors
- ▶ executes speculative instructions under L2 miss trying to generate useful prefetches



- J. Dundas and T. Mudge, *Improving data cache performance by pre-executing instructions under a cache miss*, ICS-11, 1997.

- O. Mutlu et al., *Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors*, HPCA-9, 2003.

# Enter in Runahead Execution

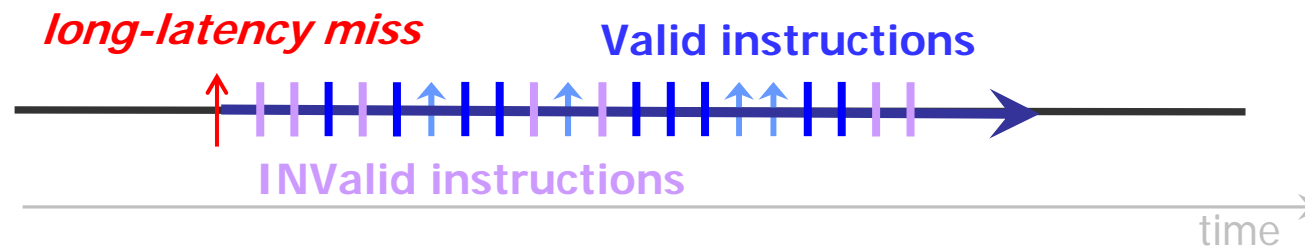


- When L2 miss reaches head of ROB
  - ▶ checkpoint architectural state and
  - ▶ enter in *runahead mode*





# During Runahead mode ...

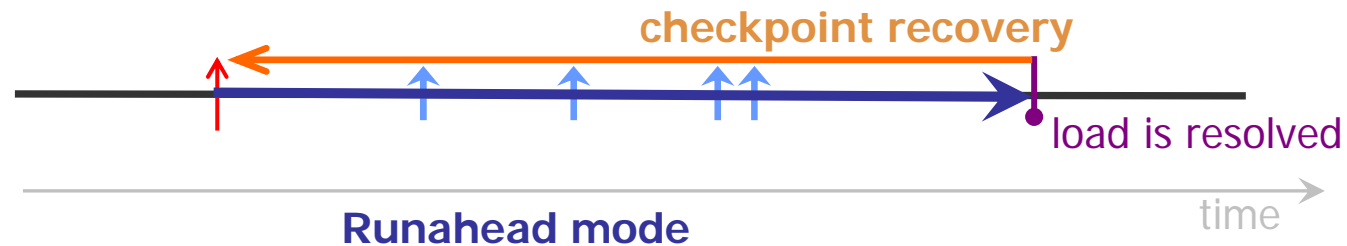


- Instructions are speculatively executed
  - ▶ L2-miss dependent instructions are marked INValid and dropped
  - ▶ Valid instructions are pre-executed to generate prefetches





# Finishing Runahead mode



- Leave runahead when L2 miss resolves
  - ▶ flush pipeline
  - ▶ restore state from checkpoint and,
  - ▶ resume to normal execution

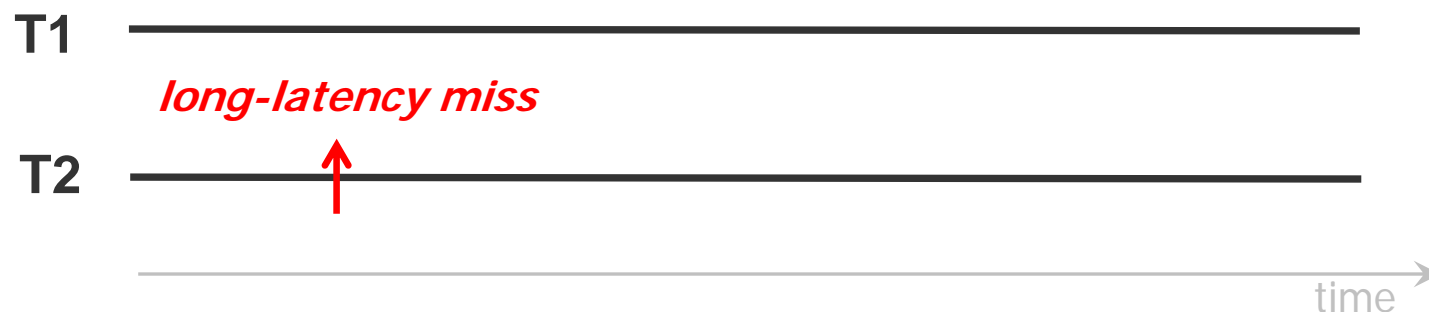




# Our SMT approach

## ■ Target

- ▶ improve the performance of memory-bound threads in SMT processors



## ■ Can Runahead be useful ?

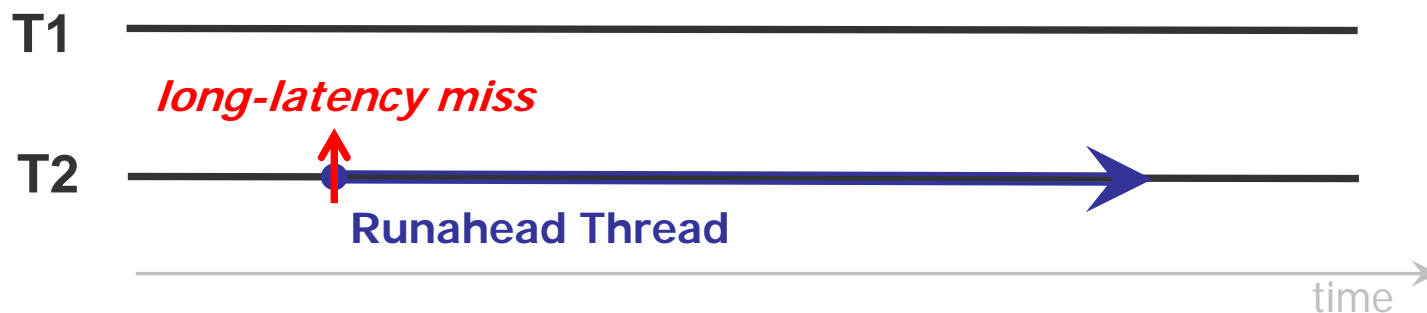
- ▶ YES, it's a good choice to achieve this goal





# Runahead Threads (RaT)

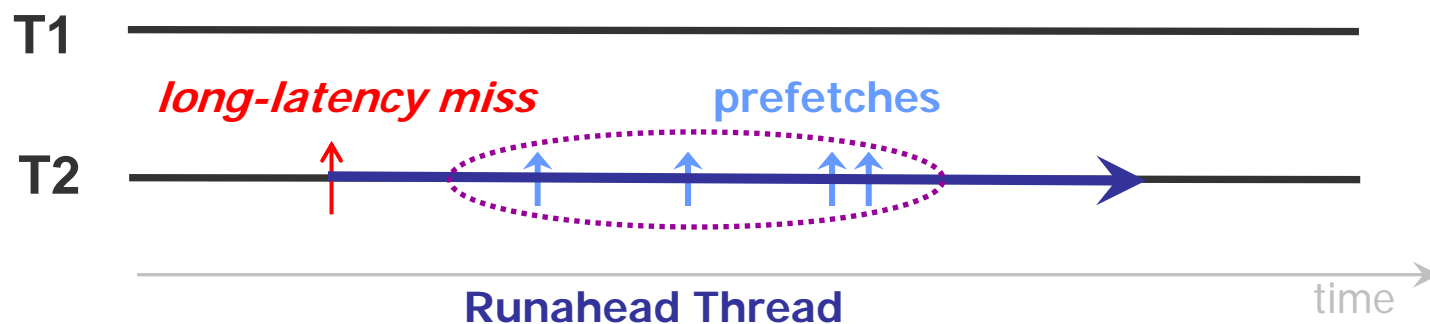
- Manage Runahead for several threads
- How?
  - ▶ turn a thread into a ***Runahead Thread***
    - thread executes speculatively the program path





# Prefetching effect

- RaT allows memory-bound threads going speculatively in advance doing prefetching



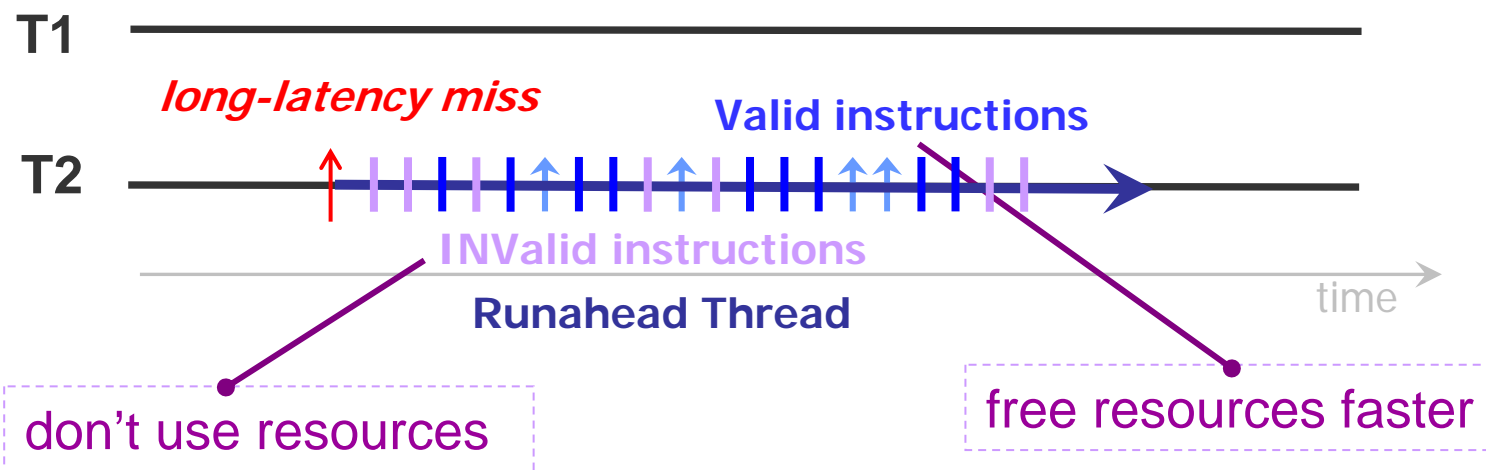
- ▶ threads do useful processing instead of stalling several cycles due to resource contention
  - prefetches increase the MLP
  - each thread is faster without disturbing the other threads.





# Alleviate resource contention

- RaT are much less aggressive than normal ones with the important SMT resources



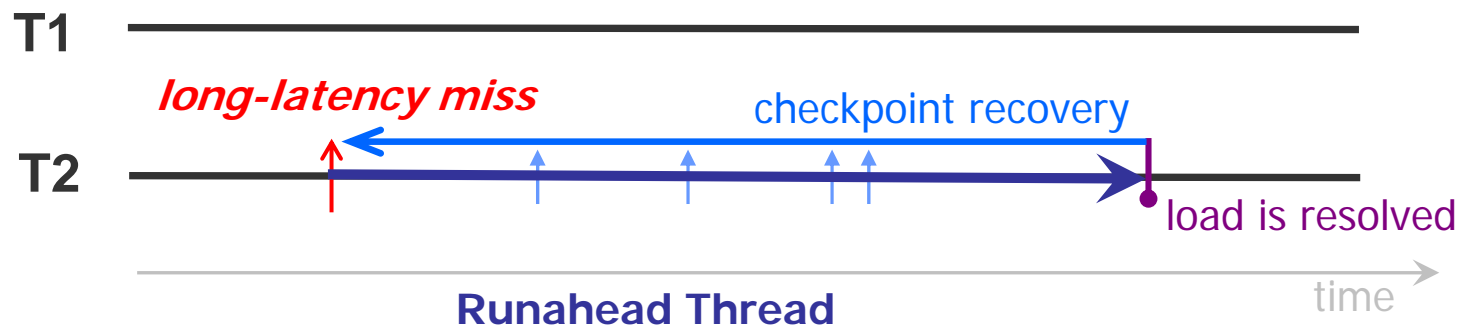
- ▶ uses as few resources as possible with fast resource release





# RaT benefits

- RaT transforms a resource hoarder thread into a light-consumer thread



- ▶ improving memory-level parallelism
- ▶ reducing resource contention





# RaT implementation details

## ■ Architectural state

- ▶ each thread saves its architectural state with checkpoints

## ■ Register control

- ▶ each thread has its own INV bit vector to track register validation

## ■ Don't execute Floating-point operations

- ▶ not needed to compute effective addresses

## ■ Synchronization

- ▶ ignore this kind of instructions





# Talk Outline

## ■ Introduction

- ▶ The problem of sharing resources
- ▶ Current Solutions

## ■ Our Proposal: Runahead Threads

- ▶ Runahead Execution
- ▶ Runahead Threads

## ■ Evaluation

- ▶ Framework
- ▶ Comparison

## ■ Conclusions





# Framework

- Improved SMTSIM exec-driven simulator
  - ▶ enhanced memory model
  - ▶ simulation checkpoints
  - ▶ *FAME* Methodology
- SPEC 2000 workloads with 2 and 4 threads
  - ▶ ILP, MEM, and MIX
- SMT model
  - ▶ complete resource sharing organization
    - issue queues, ROB, the registers, the functional units and the caches





# Framework

## ■ Baseline processor parameters

Processor core	
Processor depth	10 stages
Processor width	8 way (2 or 4 contexts)
Reorder buffer size	512 shared entries
INT/FP registers	320 / 320
INT / FP /LS issue queues	64 / 64 / 64
INT / FP /LdSt unit	6 / 3 / 4
Branch predictor	Perceptron
Memory subsystem	
Icache	64 KB, 4-way, pipelined
Dcache	64 KB, 4-way, 3 cyc latency
L2 Cache	1 MB, 8-way, 20 cyc latency
Caches line size	64 bytes
Main memory latency	400 cycles





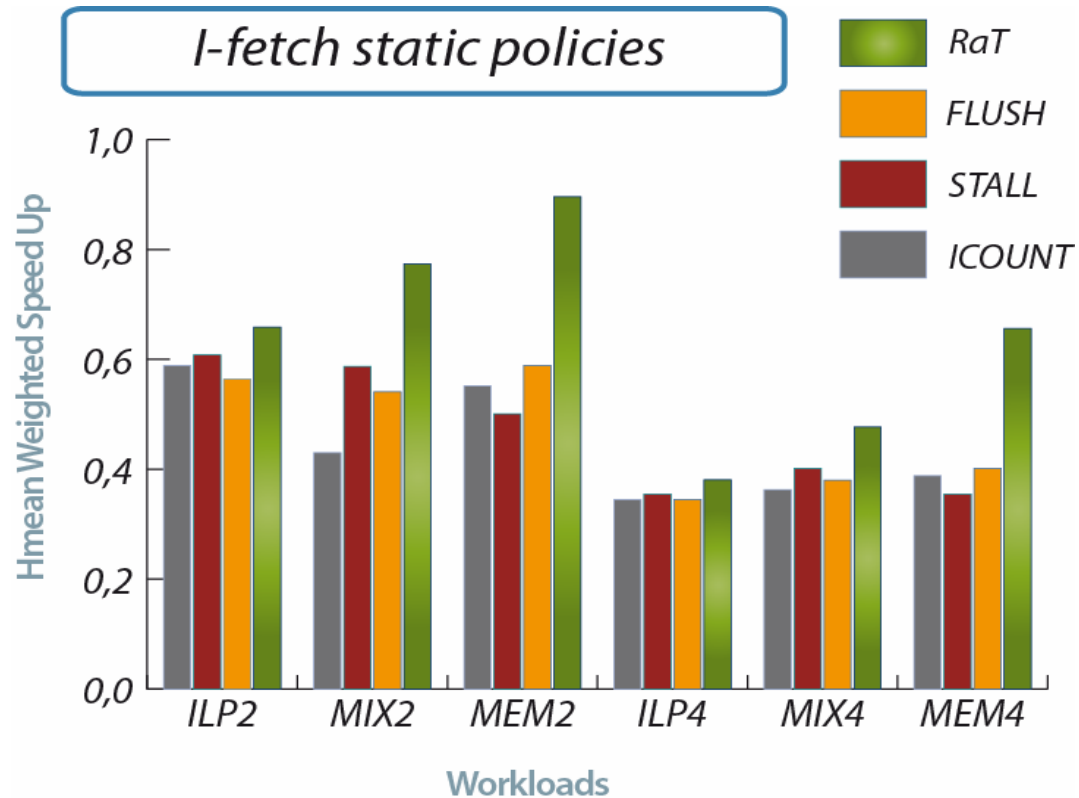
# Evaluation

- Evaluate RaT regarding state-of-the-art
  - ▶ RaT vs. I-fetch Policies
    - ICOUNT, STALL and FLUSH
  - ▶ RaT vs. Dynamic control policies
    - DCRA and HillClimbing
  
- Metrics
  - ▶ Fairness-performance balance (Hmean)
  - ▶ Energy-delay<sup>2</sup> (ED<sup>2</sup>)
    - relates the processor power consumption to its performance





# Evaluation - Fairness

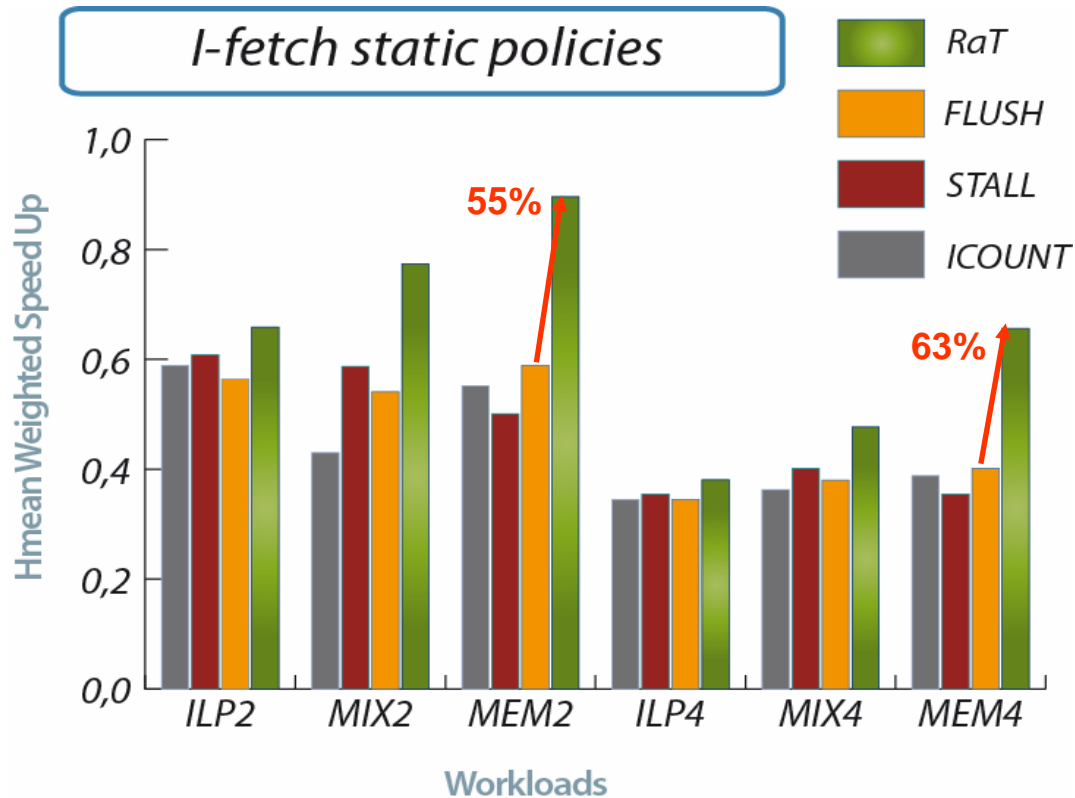


- RaT performs better than flush and stall





# Evaluation - Fairness



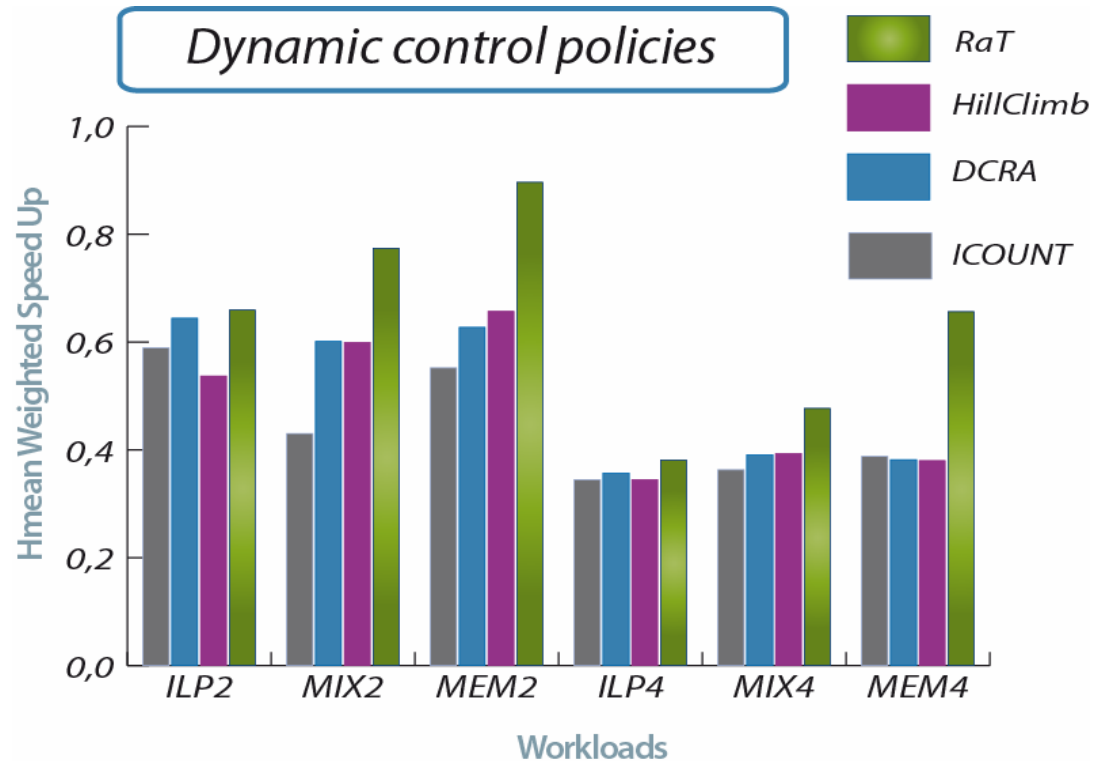
► Significant for MEM workloads.

- RaT performs better than flush and stall





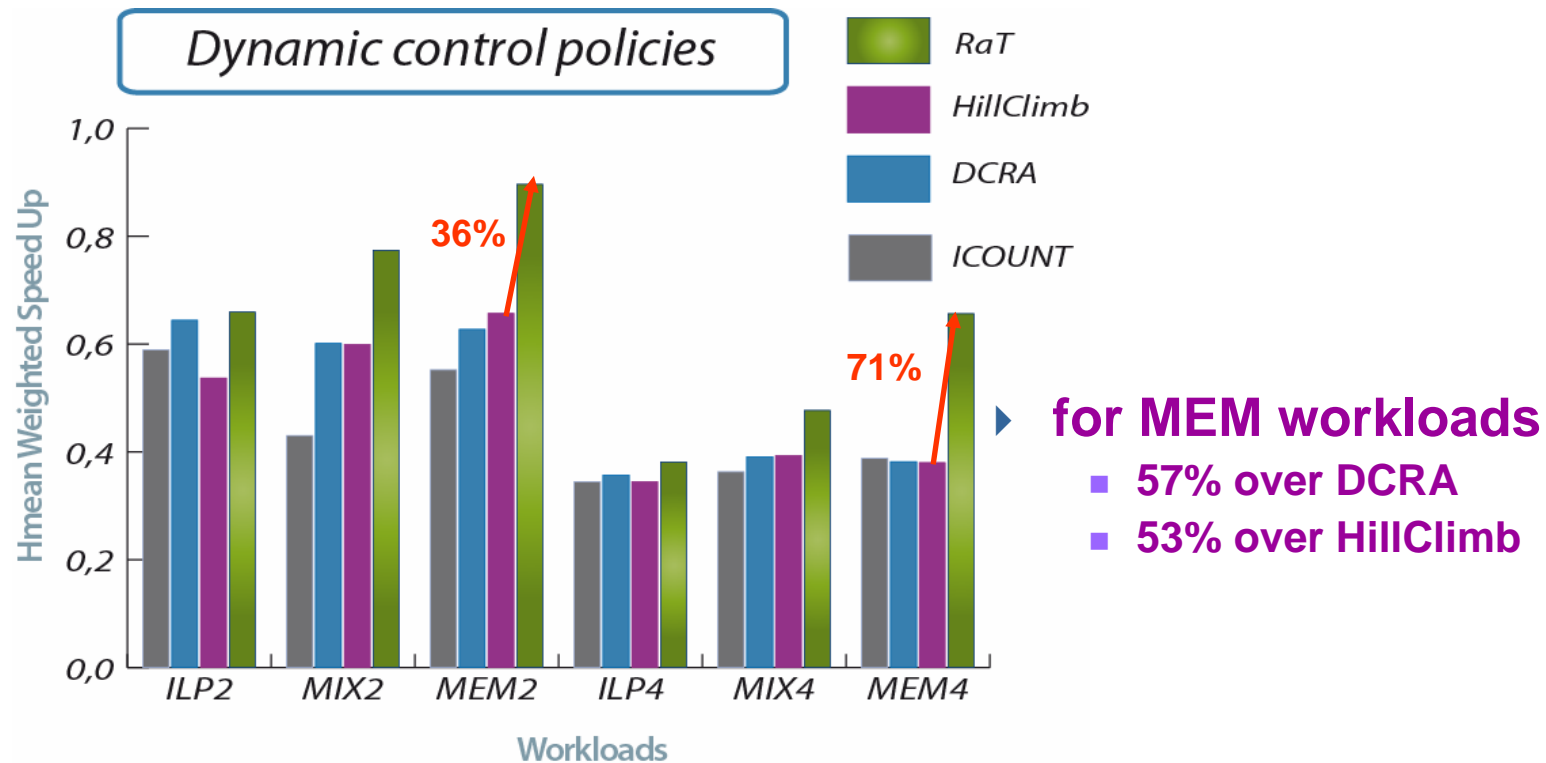
# Evaluation - Fairness



- RaT again achieves better fairness regarding dynamic control policies



# Evaluation - Fairness



- RaT again achieves better fairness regarding dynamic control policies



# RaT Sources of Improvement

- Improvement comes from two sources:

- ▶ ***Prefetching effect***

- current policies don't do it

- ▶ ***Alleviate resource contention***

- runahead threads release resources to other threads

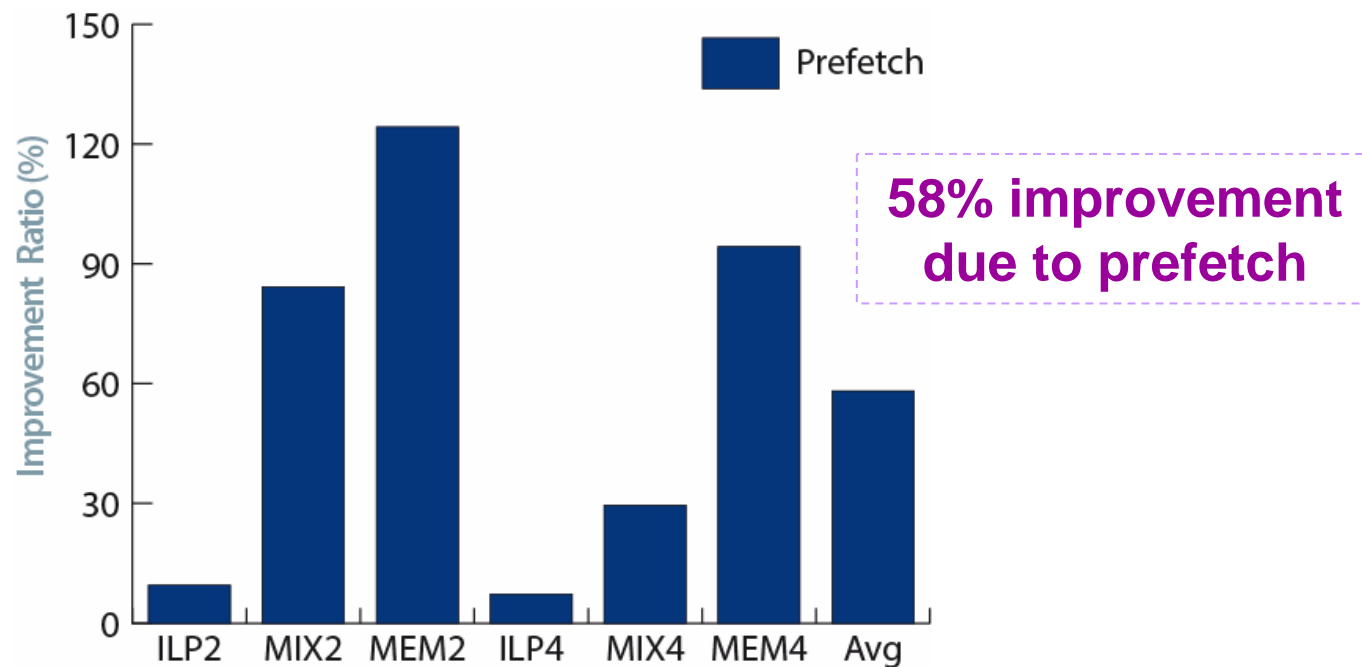




# Prefetching effect evaluation

## ■ Prefetching effect

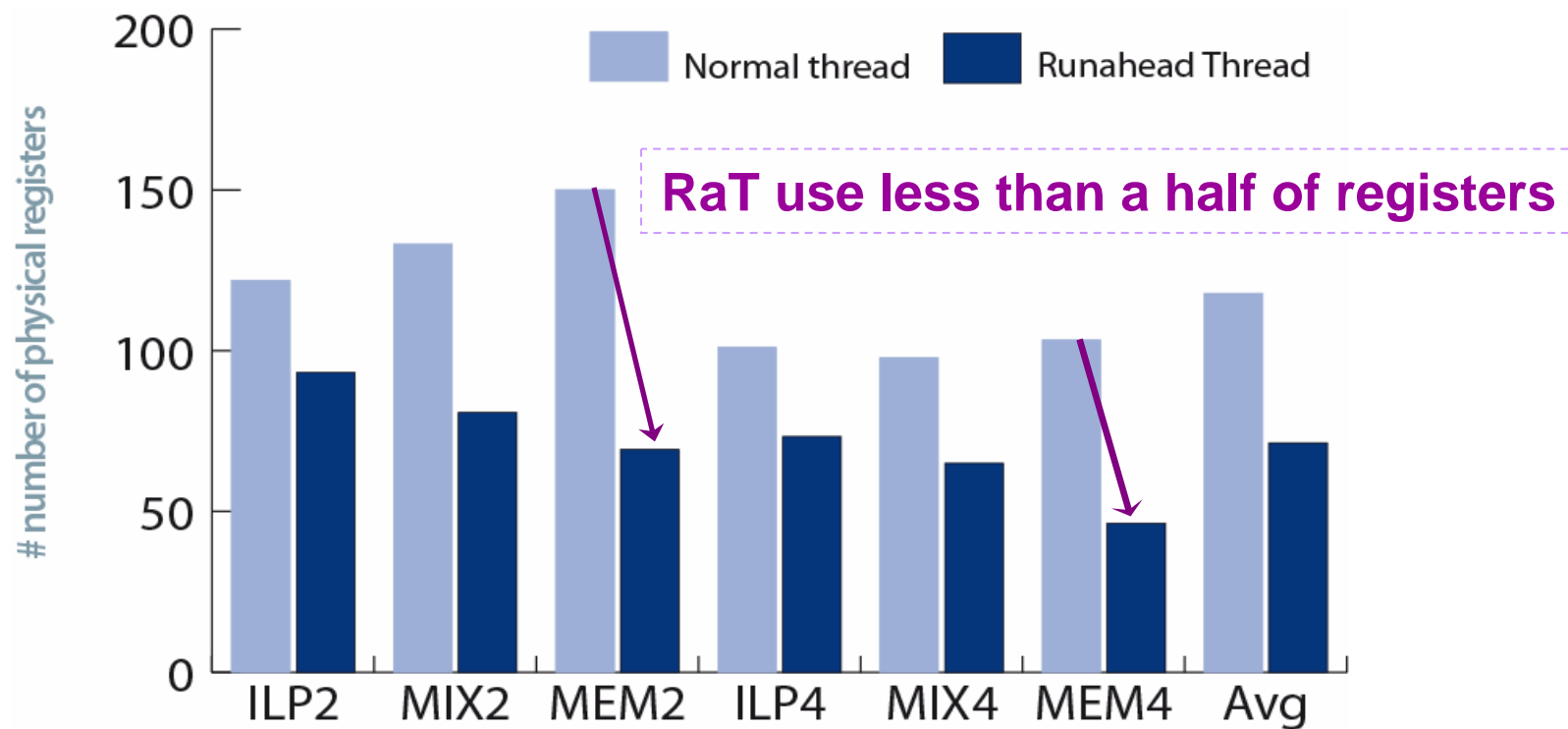
- ▶ Comparative RaT vs. RaT w/o prefetching



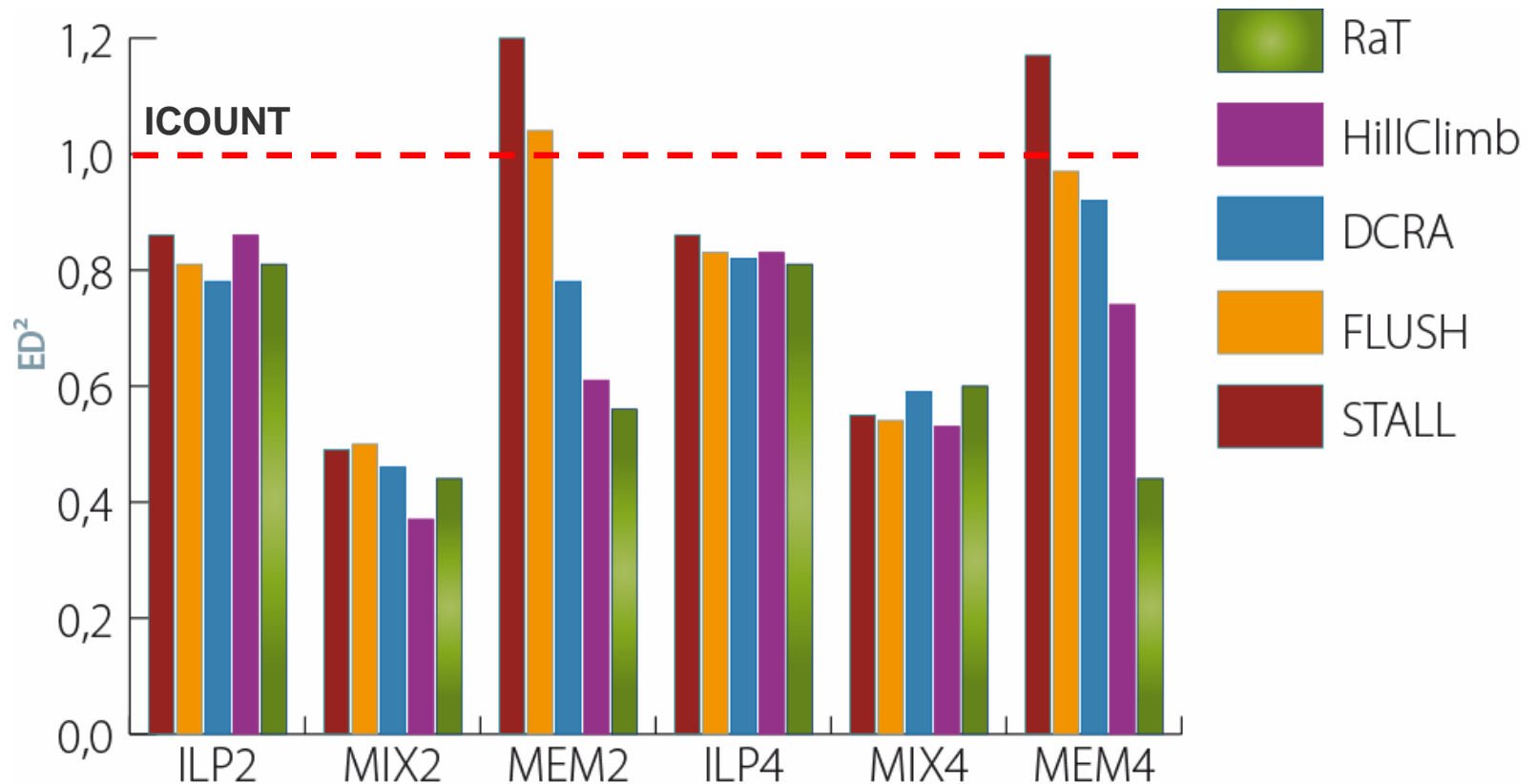


# Reduce resource contention

- An Example: Register File
  - ▶ RaT allows smaller register file



# Evaluation - ED<sup>2</sup>



- RaT has a good efficiency ratio of the performance gain and the additional energy consumed



# Talk Outline

## ■ Introduction

- ▶ The problem of sharing resources
- ▶ Current Solutions

## ■ Our Proposal: Runahead Threads

- ▶ Runahead Execution
- ▶ Runahead Threads

## ■ Evaluation

- ▶ Framework
- ▶ Comparison

## ■ Conclusions





# Conclusions

- We propose ***Runahead Threads*** as an alternative to resource contention in SMT
  - ▶ avoid the resource monopolization by memory-bound threads
  - ▶ improving the performance without prejudicing fast threads
  
- RaT shows significant performance advantage over state-of-the-art techniques
  - ▶ alleviate resource contention
  - ▶ prefetching





# Runahead Threads to Improve SMT Performance



**Tanausú Ramírez**



**Alex Pajuelo**



**Oliverio J. Santana**



**Mateo Valero**



*14th Symposium on High-Performance Computer Architecture*

**HPCA'08, Salt Lake City, UT**