Performance Evaluation of a Reconfigurable Instruction Set Processor

Mehdipour, Farhad
Faculty of Information Science and Electrical Engineering, Kyushu University

Noori, Hamid
Faculty of Information Science and Electrical Engineering, Kyushu University

Honda, Hiroaki
Faculty of Information Science and Electrical Engineering, Kyushu University

Inoue, Koji
Faculty of Information Science and Electrical Engineering, Kyushu University

他

https://doi.org/10.15017/14879

KYUSHU UNIVERSITY

Performance Evaluation of a Reconfigurable Instruction Set Processor

Farhad Mehdipour, H. Noori, H. Honda, K. Inoue, K. Murakami

**Faculty of Information Science and Electrical Engineering,
KYUSHU UNIVERSITY, Fukuoka, JAPAN**

{farhad@c.csce.kyushu-u.ac.jp}

Outline

- Reconfigurable Instructions Set Processors

- A Combined Analytical and Simulation-Based Model (CAnSO)

  - Model Extraction and Calibration

  - Basic Model Definitions

  - Speedup Formulations

  - Simplification and Calibration

- Experiments

  - Experimental Setup

  - Model Validation

  - Design Space Exploration Using CAnSO

  - Effects of Modifications

- Conclusions and Future Work
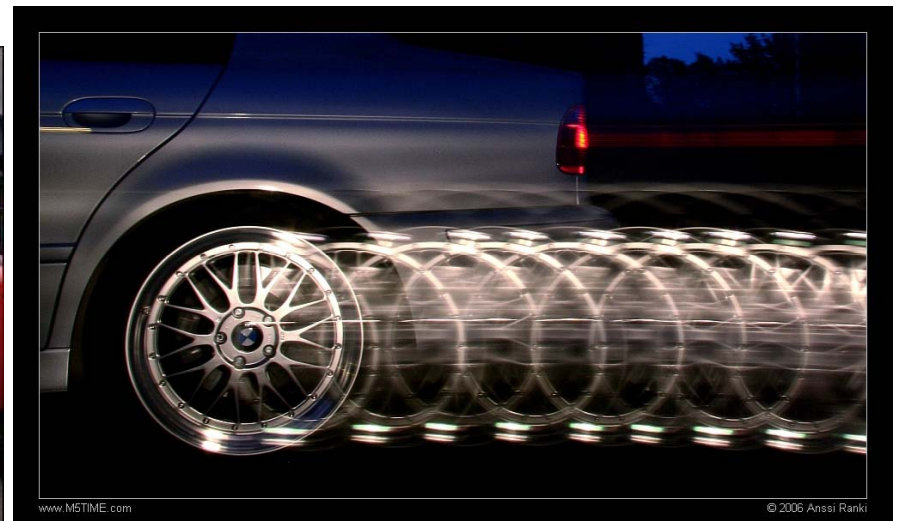
Designing Embedded Systems

- Embedded Microprocessors

- Application-Specific Integrated Circuits (ASICs)

- Application-Specific Instruction set Processors (ASIPs)
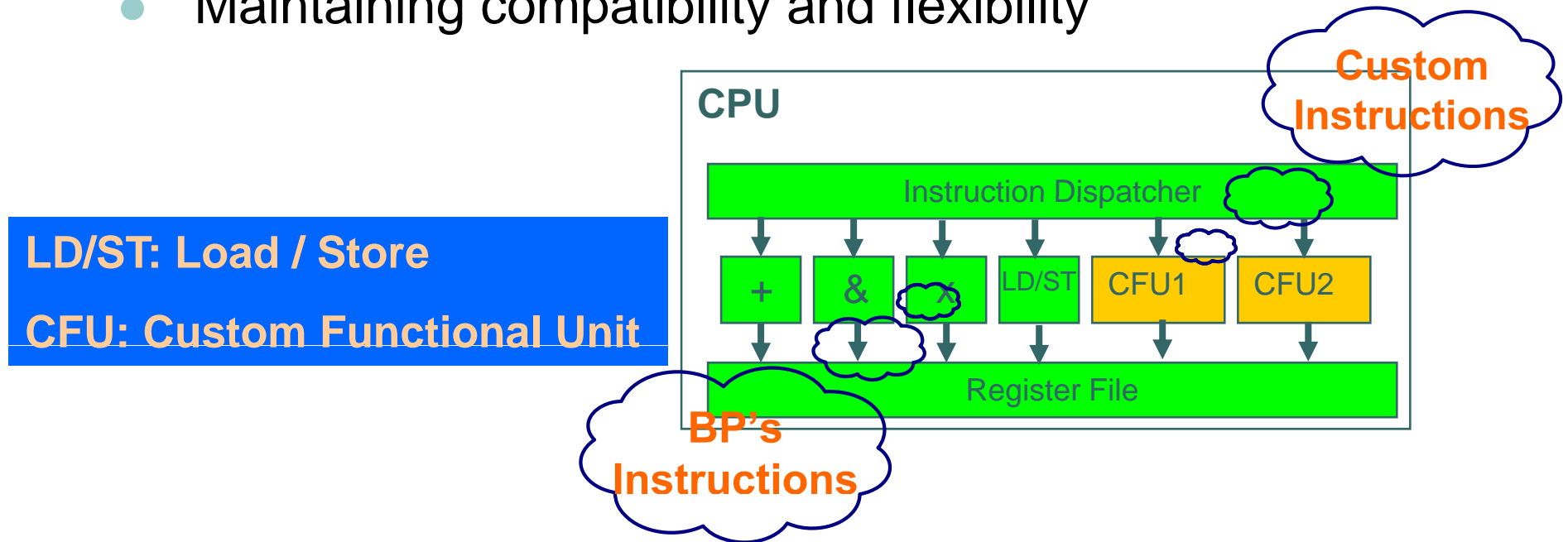
- Extensible Processors
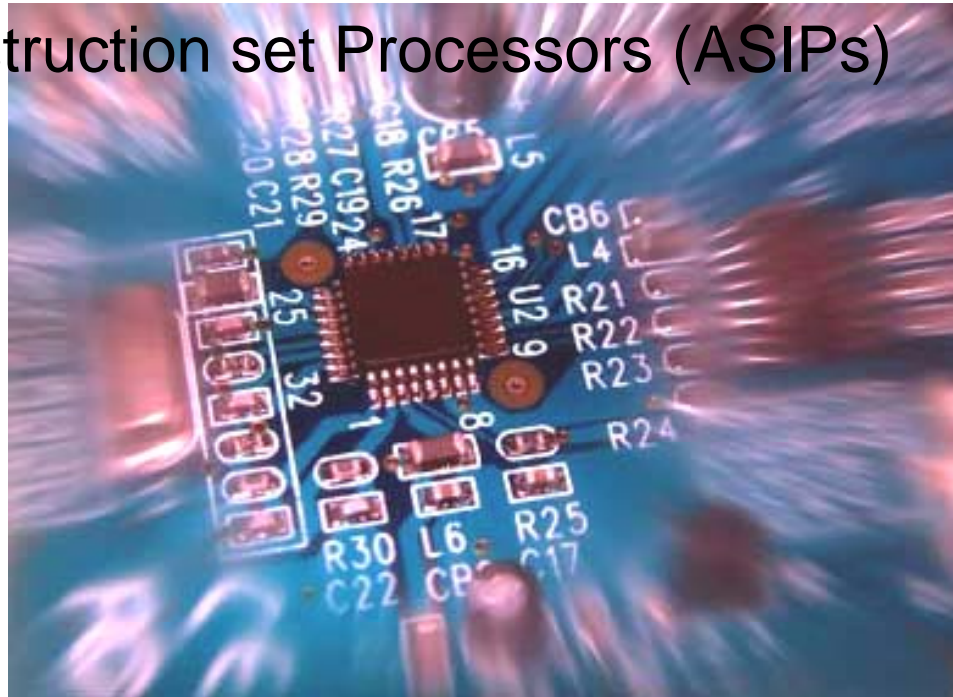
Extensible Processors

○ Mechanism

- Acceleration by using CFU

- a hardware is augmented to the base processor

- Executes hot portions of applications

Extensible Processors

- Base processor (BP)'s fixed instruction set + Custom Instructions
- Goals
  - Improving the performance and energy efficiency
  - Maintaining compatibility and flexibility

LD/ST: Load / Store

CFU: Custom Functional Unit

CPU

Instruction Dispatcher

+ & × LD/ST CFU1 CFU2

Register File

Custom Instructions

BP's Instructions

Custom Instructions

- Instruction set customization ← → hardware/software partitioning (Identifying critical segments in applications)
- Custom Instructions (CIs) are
  - extracted from critical segments of an application and
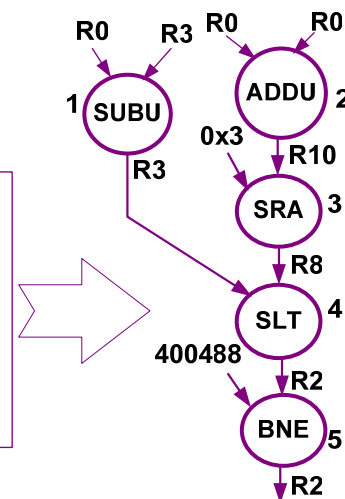 - executed on a Custom Functional Unit (CFU)

> **Critical segments:**
> Most frequently executed (Hot) portions of the applications

A CI can be represented as a DFG

A Custom Instruction

```
1: SUBU   R3, R0, R3
2: ADDU   R10, R0, R0
3: SRA    R8, R10, 0x3
4: SLT    R2, R3, R8
5: BNE    R0,400488, R2
```
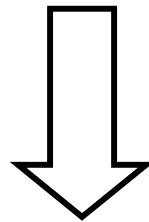
Extensible Processors

- Drawbacks:
  - Lack of flexibility
  - Long time and cost of designing and verifying
  - Many issues associated with designing a new processor from scratch:
    - longer time-to-market and
    - significant NRE (Non-Recurring Engineering) costs

- Solution
  - Using a Reconfigurable Functional Unit (RFU) instead of fixed architecture CFU
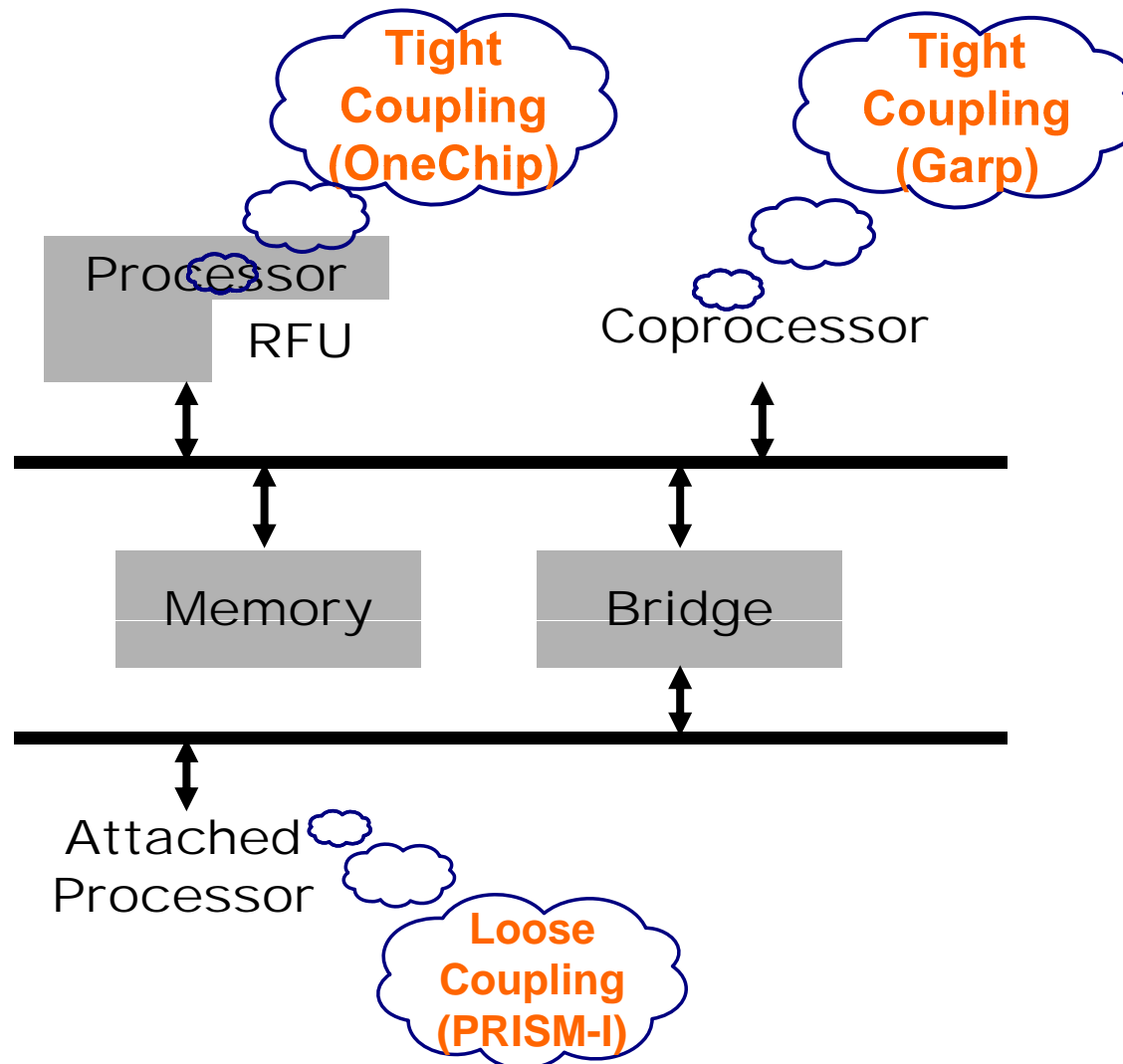
Reconfigurable Processors

Microprocessor $+$ **Reconfigurable Logic**
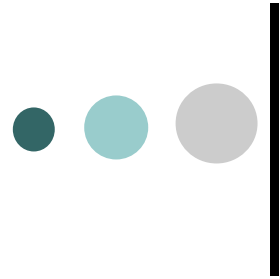
$\Downarrow$

Reconfigurable Processor

Processor coupling

Tight Coupling (OneChip)

Tight Coupling (Garp)

Processor

RFU

Coprocessor

Memory

Bridge

Attached Processor
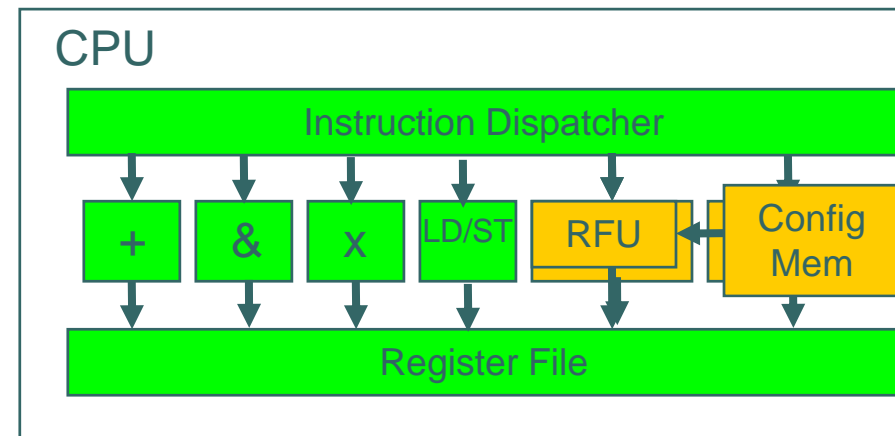
Loose Coupling (PRISM-I)
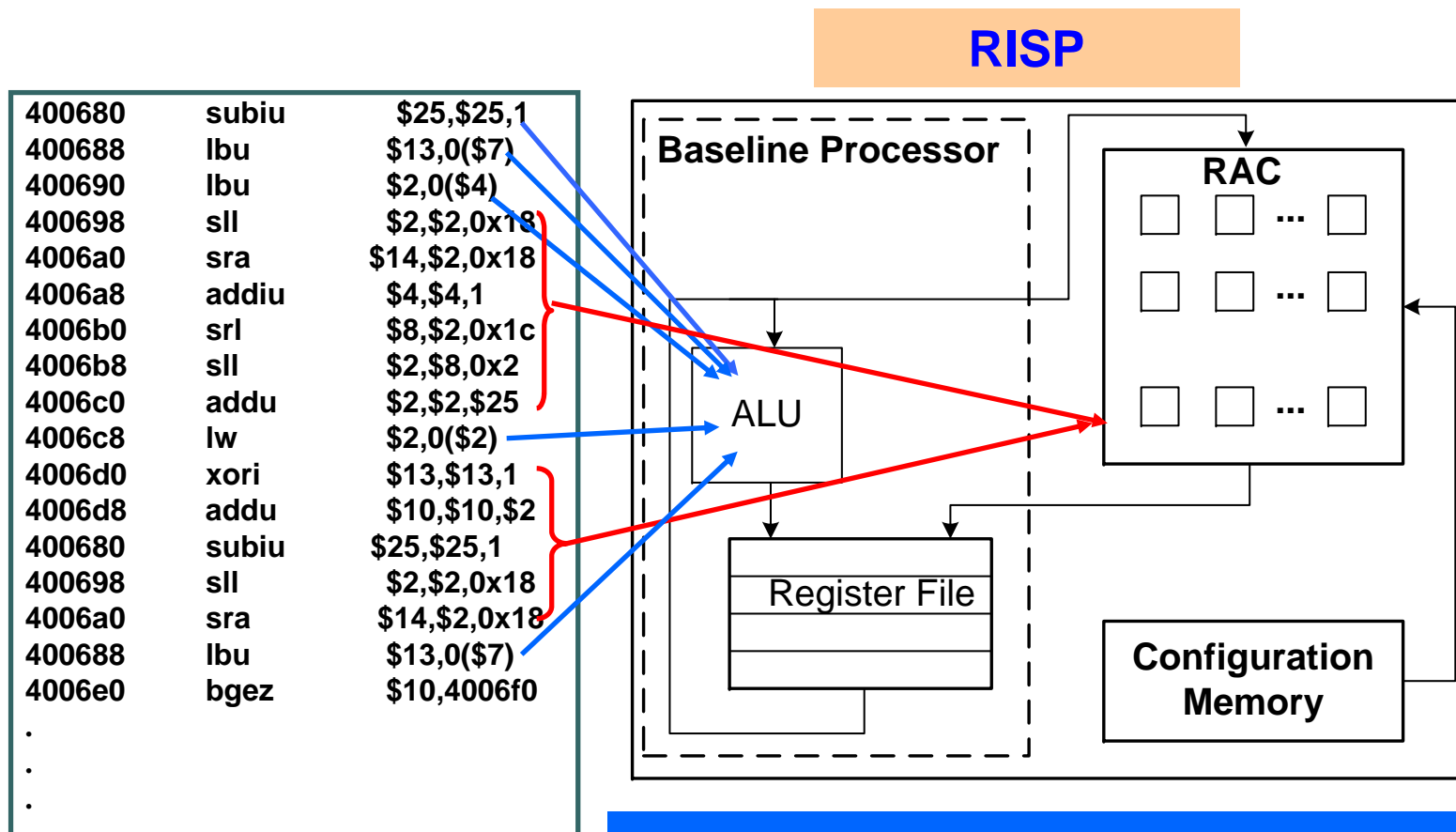
Reconfigurable Instruction Set Processors (RISPs)

○ Adding and generating custom instructions after fabrication

○ Using a reconfigurable FU(RFU) instead of custom FU

**CFU: Custom Functional Unit**

RFU: Reconfigurable Functional Unit

CPU

Instruction Dispatcher

| + | & | x | LD/ST | RFU | Config Mem |

Register File

How a RISP Works

400680	subiu	$25,$25,1
400688	lbu	$13,0($7)
400690	lbu	$2,0($4)
400698	sll	$2,$2,0x18
4006a0	sra	$14,$2,0x18
4006a8	addiu	$4,$4,1
4006b0	srl	$8,$2,0x1c
4006b8	sll	$2,$8,0x2
4006c0	addu	$2,$2,$25
4006c8	lw	$2,0($2)
4006d0	xori	$13,$13,1
4006d8	addu	$10,$10,$2
400680	subiu	$25,$25,1
400698	sll	$2,$2,0x18
4006a0	sra	$14,$2,0x18
400688	lbu	$13,0($7)
4006e0	bgez	$10,4006f0

RISP

Baseline Processor

RAC

ALU

Register File

Configuration Memory

A Hot Basic Block

GPP: General Purpose Processor

RAC=RFU: Reconfigurable Accelerator

RISP Benefits and Drawbacks

Benefits

+
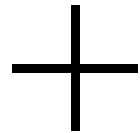
- Specialized datapath
- Shared hardware
- Higher Speedup
- Less power consumption

## Drawbacks

—

- More area
- Difficult to use

Performance Evaluation of a RISP

- Performance evaluation of a RISP challenges
  - designing of a RISP architecture
  - optimizing an existing arch. for an objective function

- For a designer
  - obtaining optimum system configuration is desirable
  - a performance analysis in terms of the performance metrics (speedup, area and so on) is required

- Performance evaluation models
  - Structural models: includes empirical studies based on measurements and simulations of the target system
  - Analytical models: incorporates a system (usually simplified) structure to obtain mathematically solvable models
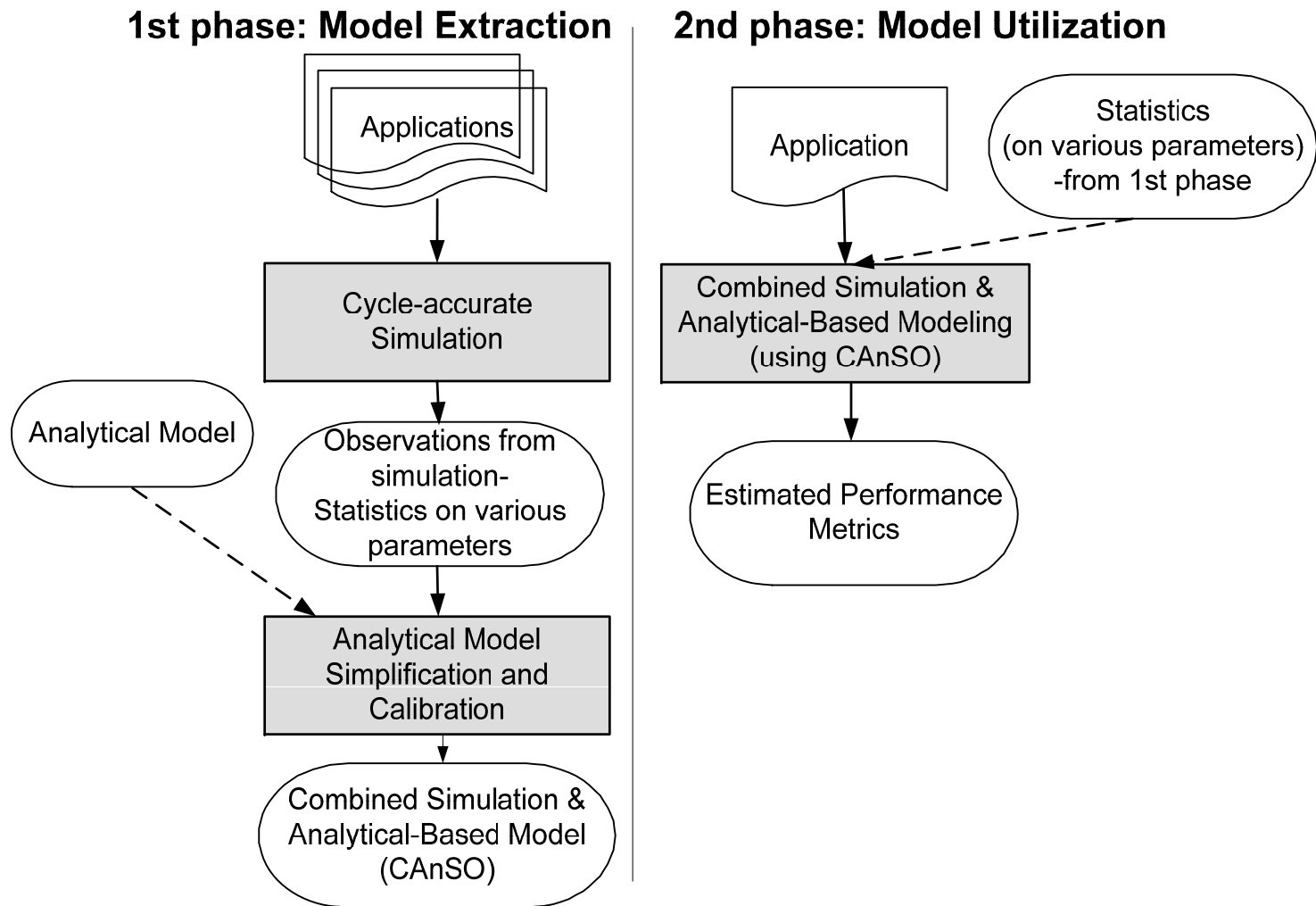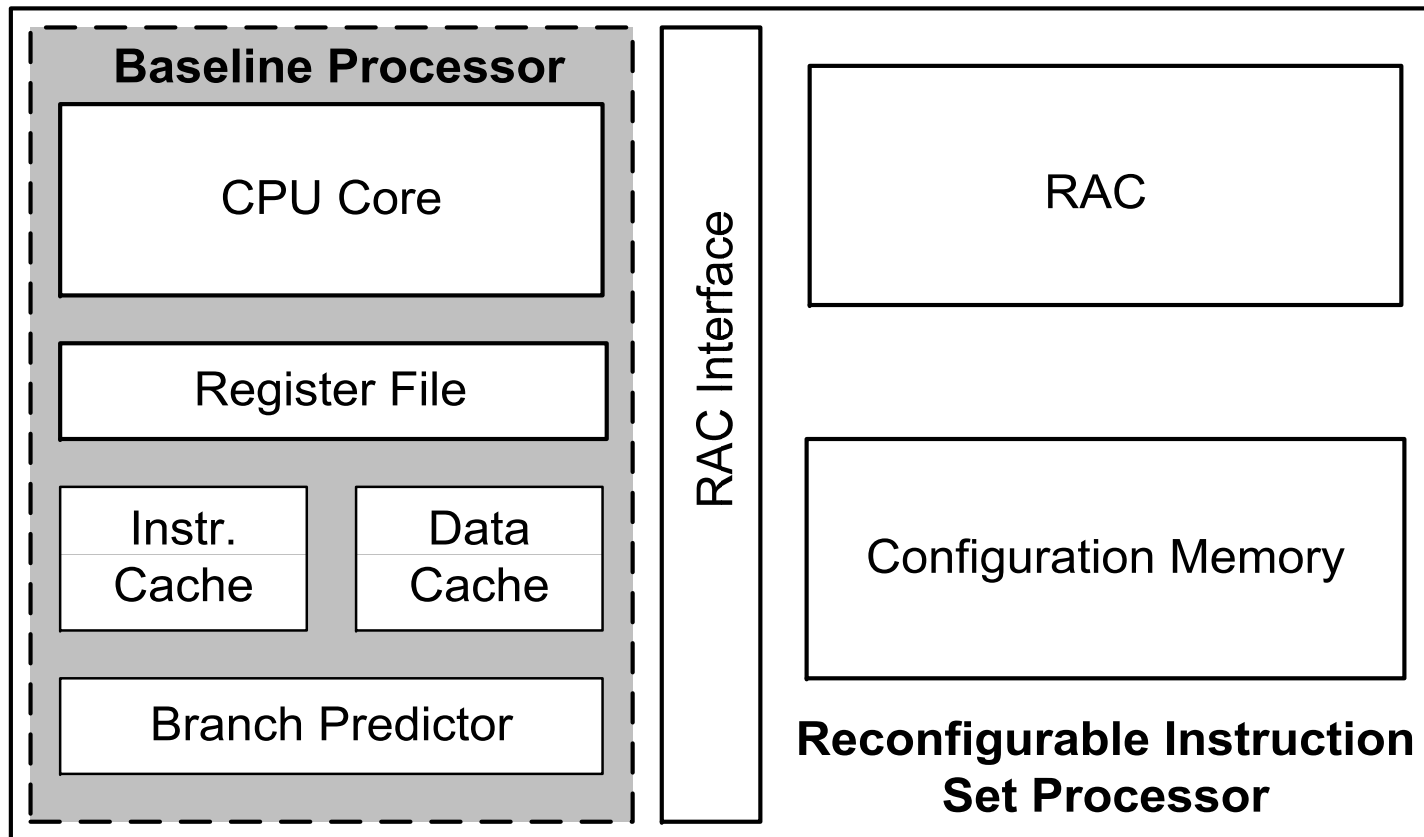
Fraction of Dynamic Instructions in Applications
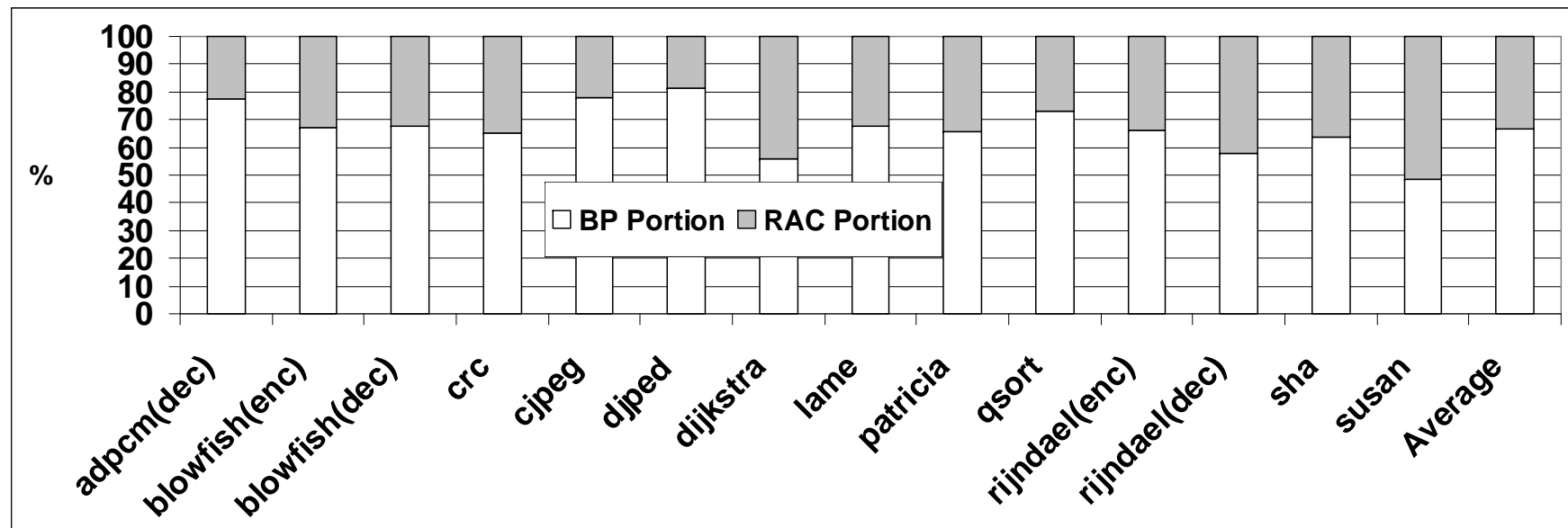
the RAC is responsible for executing almost 30% of dynamic instructions of applications in average

Model Extraction and Utilization

1st phase: Model Extraction

Applications

↓

Cycle-accurate Simulation

↓

Observations from simulation- Statistics on various parameters

Analytical Model ⟶ (dashed)

↓

Analytical Model Simplification and Calibration

↓

Combined Simulation & Analytical-Based Model (CAnSO)

2nd phase: Model Utilization

Application

Statistics (on various parameters) -from 1st phase

↓

Combined Simulation & Analytical-Based Modeling (using CAnSO)

↓

Estimated Performance Metrics

General Template of a RISP

Baseline Processor
- CPU Core
- Register File
- Instr. Cache
- Data Cache
- Branch Predictor

RAC Interface

RAC

Configuration Memory

Reconfigurable Instruction Set Processor

Basic Model Definitions

- Base Processor
  - an in-order general five-stage RISC processor

- RAC
  - a coarse-grained tightly-coupled reconfigurable hardware

- CIs are indexed for direct accessing of the configuration bit-stream

- The content of all registers are sent to the RAC (Shared RF)

- Controlling configurations
  - Hardware-based: starting address of CI and index to the config. Mem. is stored in a CAM for quick retrieval
  - Software-based: starting address of a CI is replaced with a special instruction

- Memory accesses

- Control instructions

Single and Continuous Executions
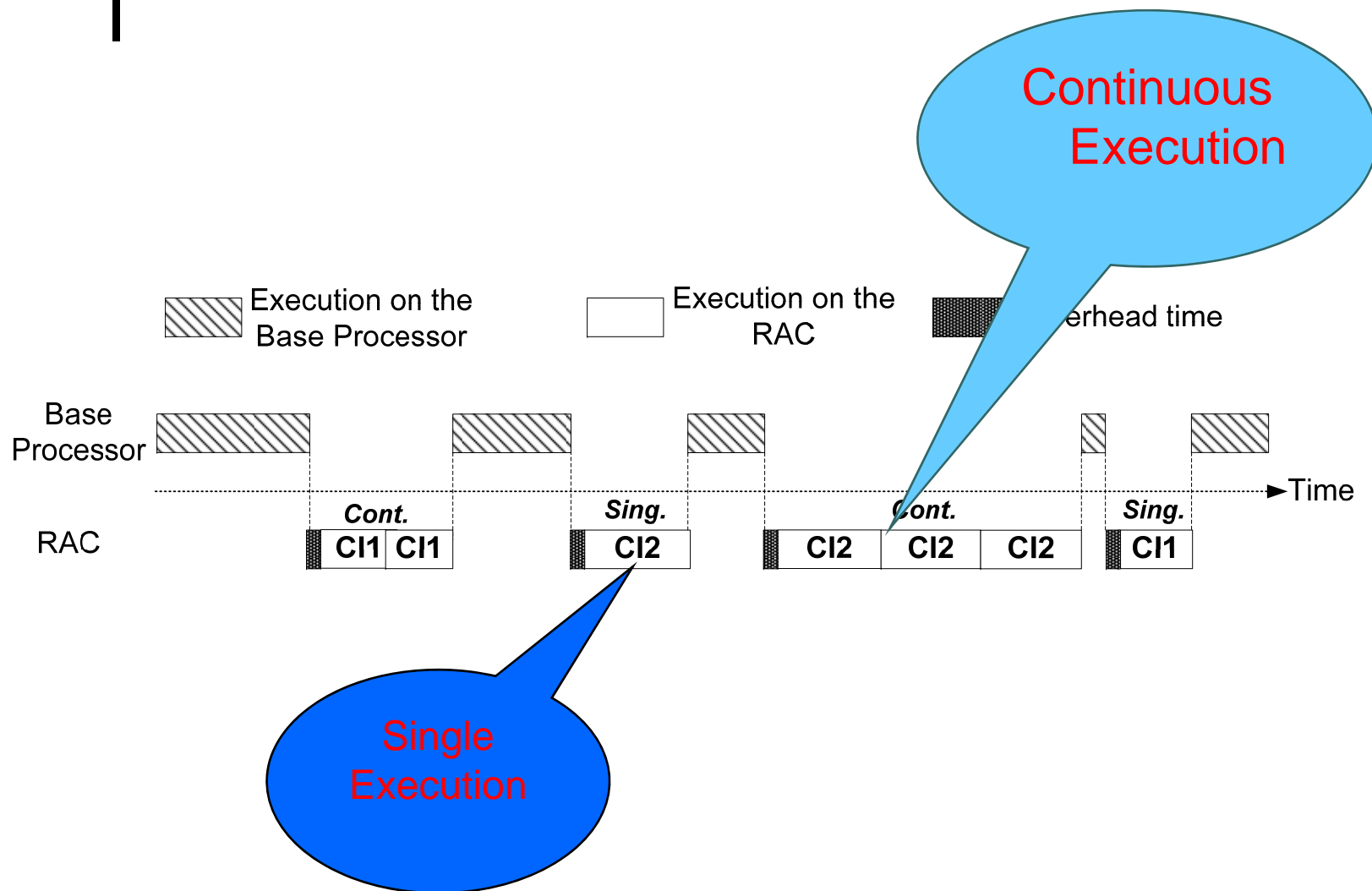
Speedup Formulation

Latency of execution of *Cli* instructions on the BP

Total no. of executions of *Cli*

Fraction of instructions executing on BP

$$f_{RAC} = \frac{\sum_{i=1}^{n_{CI}} \left(\tau_{BP}^i \times O_i \right)}{n_{tcc}}$$

$$f_{BP} = \frac{\left(n_{CC} \sum_{i=1}^{n_{CI}} \left(\tau_{BP}^i \times O_i \right) \right)}{n_{tcc}} = 1 - f_{RAC}$$

Execution time on the BP

Execution time on the RAC

$$s_O = \frac{n_{tcc}}{\left(n_{tcc} - \sum_{i=1}^{n_{CI}} \left(\tau_{BP}^i \times O_i \right) \right) + \psi(\theta, \tau)}$$

Overall Speedup

$$\psi(\theta, \tau) = \sum_{i=1}^{n_{CI}} \left(\sum_{j \in S_i} \left(\theta_{ij} \times (\tau_{RAC} + \tau_{OVH}) \right) + \sum_{j \in C_i} \left((\tau_{RAC} + \tau_{OVH}) + \left((\theta_{ij} - 1) \times \tau_{RAC} \right) \right) \right)$$

frequency of *j*th occurrence of *Cli*

Latency of RAC and the overhead reconfiguration time

The Effect of CI Length

- Large CIs
  - Including more instructions than the no. of available resources in the RAC

- Temporal Partitioning
  - Dividing larger CIs to a number of smaller CIs

$$L = \{k \,|\, k \in \{1,...,n_{CI}\}, l_k > n_{FU}\} \qquad P = \{p_k \,|\, k \in L, p_k = \left\lceil \frac{l_k}{n_{FU}} \right\rceil\}$$

$$m'_{k \in L} = O_i \times p_k, m'_{k \notin L} = m_i$$

$$\theta'_{k \in L} = ((1,...,1),(1,...,1),...,(1,...,1)), |\theta'_{k \in L}| = m'_{k \in L}, \theta'_{k \notin L} = \theta_{k \notin L}$$

$$S'_{i \in L} = \{1,...m'_i\}, S'_{i \notin L} = S_i \qquad C'_{i \in L} = \varnothing, C'_{i \notin L} = C_i$$

Side-Effects

- *Control Instructions*
  - the rate of miss-predicted branches might be reduced → higher speedup

- *Instruction Cache Misses*
  - no need for fetching instructions belonging to the CIs
  - access and miss rates to instruction cache are reduced
  - BP fraction reduces→ speedup increases

> no. of penalty cycles for branch miss-predictions/cache misses

> variation in branch/cache miss-predictions/misses

$$s_O = \frac{n_{tcc}}{\left[n_{tcc} - \sum_{x=\{b,i\}} \delta_{xm} \times p_{xm} + \sum_{i=1} \left(\tau^i_{BP} \times O_i \right) \right] + \psi'(\theta', \tau)}$$

RF's Input/Output Ports

- Register file is shared between BP and RAC

- Additional clock cycles for reading/writing from/to the RF

no. of CI*i*'s inputs

no. of RF's read ports

no. of RF's write ports

$$\tau'_{OVH} = \tau_{OVH} + \max\left(0, \left\lceil \frac{\Delta^i_{CI} - \Delta_{reg}}{\Delta_{reg}} \right\rceil\right) + \max\left(0, \left\lceil \frac{\nabla^i_{CI} - \nabla_{reg}}{\nabla_{reg}} \right\rceil\right)$$
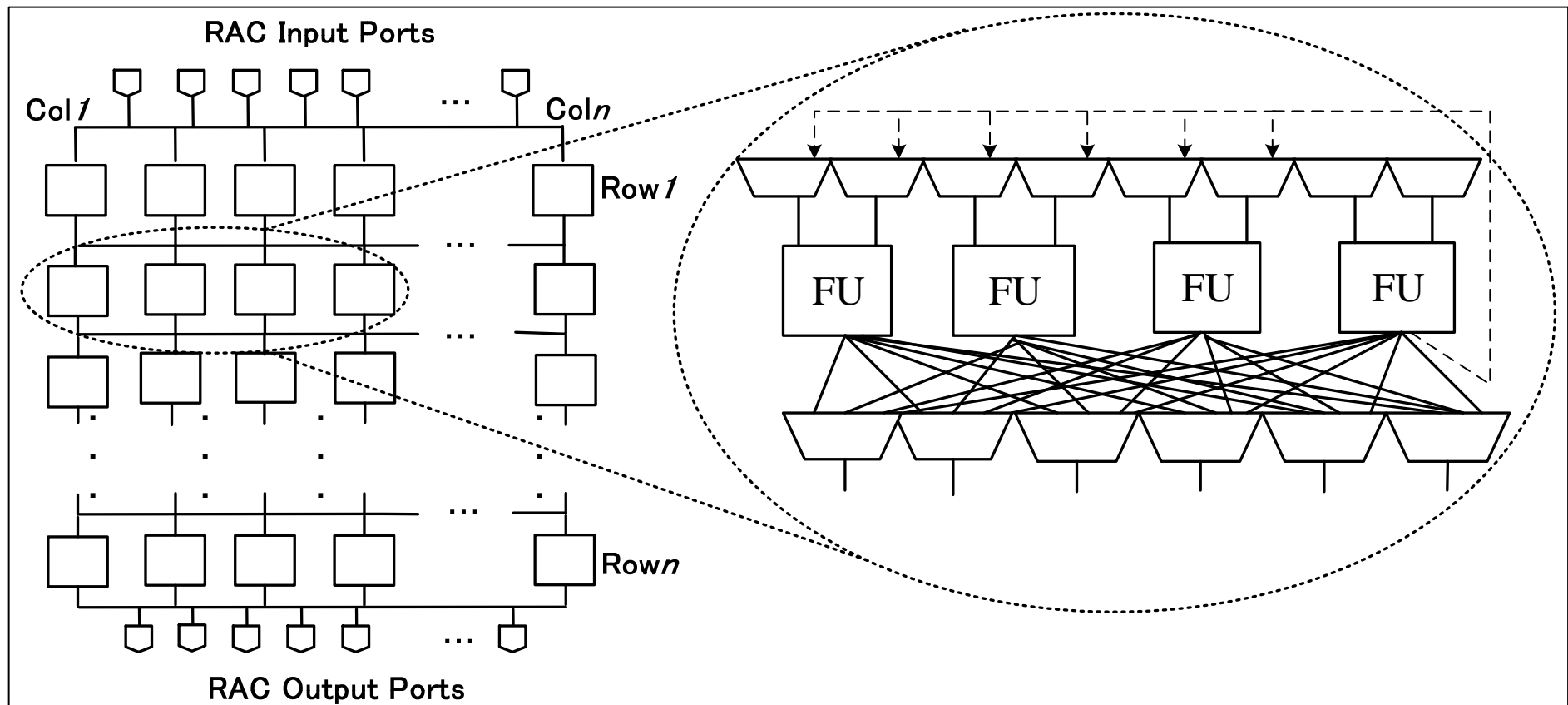
The Assumed RAC Architecture

RAC's Delay

- All FUs in the RAC implement similar operations

- Each mux receives
  - all outputs of the FUs in upper rows and
  - Outputs from its adjacent FUs at the same row

$$\tau_{RAC_h}^{w} = \sum_{i=1}^{h} \tau_{FU} + \sum_{i=1}^{h-1} \tau_{MUX_i}^{k}, \qquad k \in \{0,1,...,w\}$$

$$\psi(\theta,\tau) = \sum_{i=1}^{n_{CI}} \left(\sum_{j \in S_i} \left(\theta_{ij} \times (\tau_{RAC} + \tau_{OVH}) \right) + \sum_{j \in C_i} \left((\tau_{RAC} + \tau_{OVH}) + \left((\theta_{ij} - 1) \times \tau_{RAC} \right) \right) \right)$$
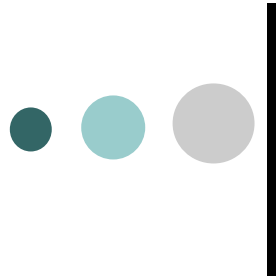
Simplification and Calibration

- Control instructions are not supported

- Reduction in instruction cache accesses as well as cache misses
  - average reduction in access to i-cache is almost 17%
  - average i-cache miss rate is almost 3%.
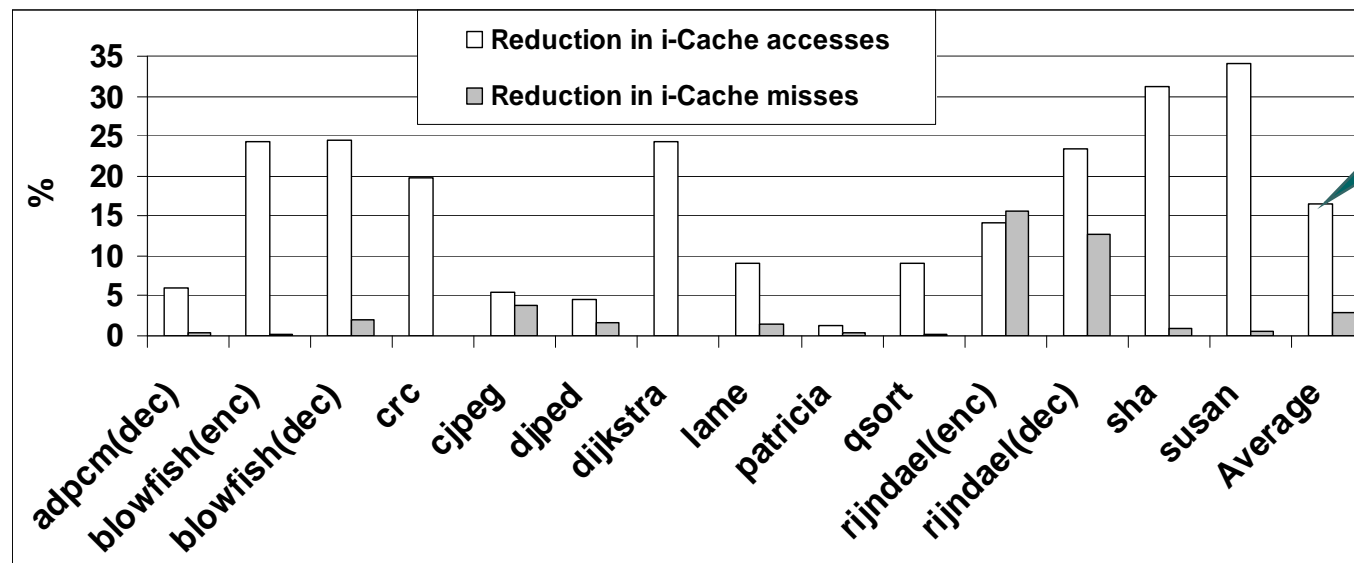
**Average i-Cache
Accesses: 17%
Misses: 3%**

Simplification and Calibration

Fraction of Single & Continuous Executions

the ratio of single to continuous execution→ 43%
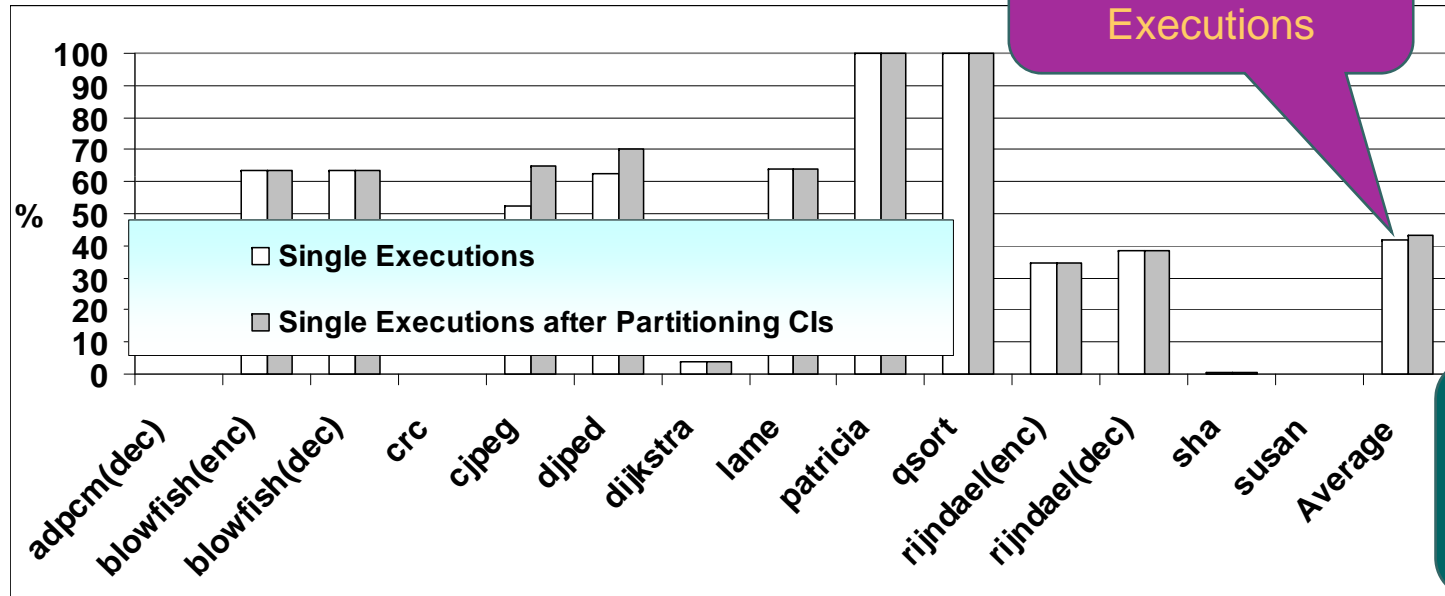
$$s_O = \cfrac{\overset{*}{n_{tcc}}}{\left(\overset{*}{n_{tcc}} - \overset{*}{\delta_{im}} \times \overset{*}{p_{im}} - \sum_{i=1}^{\overset{*}{n_{DI}}}\left(\overset{*}{\tau_{BP}^i} \times \overset{*}{O_i}\right)\right) + \psi'\left(\overset{*}{\theta'}, \tau_{RAC_h}^w + \tau_{OVH}\right)}$$

$$\psi'\left(\overset{*}{\theta'}, \tau_{RAC_h}^w + \tau_{OVH}\right) = \sum_{i=1}^{\overset{*}{n_{CI}}} \overset{*}{O_i} \times \left(\alpha \times \overset{*}{\tau'_{OVH}} + \tau_{h\ RAC}^w\right)$$

Experimental Setup

- Fourteen applications of Mibench
  - automotive, security, consumer, network, telecommunication

- CIs (DFGs) are extracted from applications

- Simplescalar's cycle-accurate simulator is extended to simulate a reconfigurable instruction set processor

- Model Establishment
  - simulating all applications
  - collecting required information
  - model simplification and calibration

~ 4 hours to completion on a PC: Dual Core, Intel 6600@2400Mhz, 2GB RAM
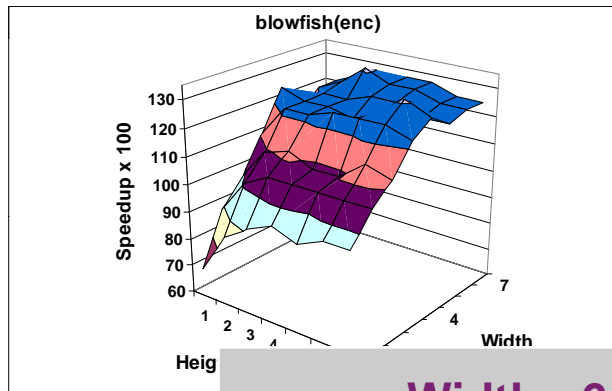
Model Validation
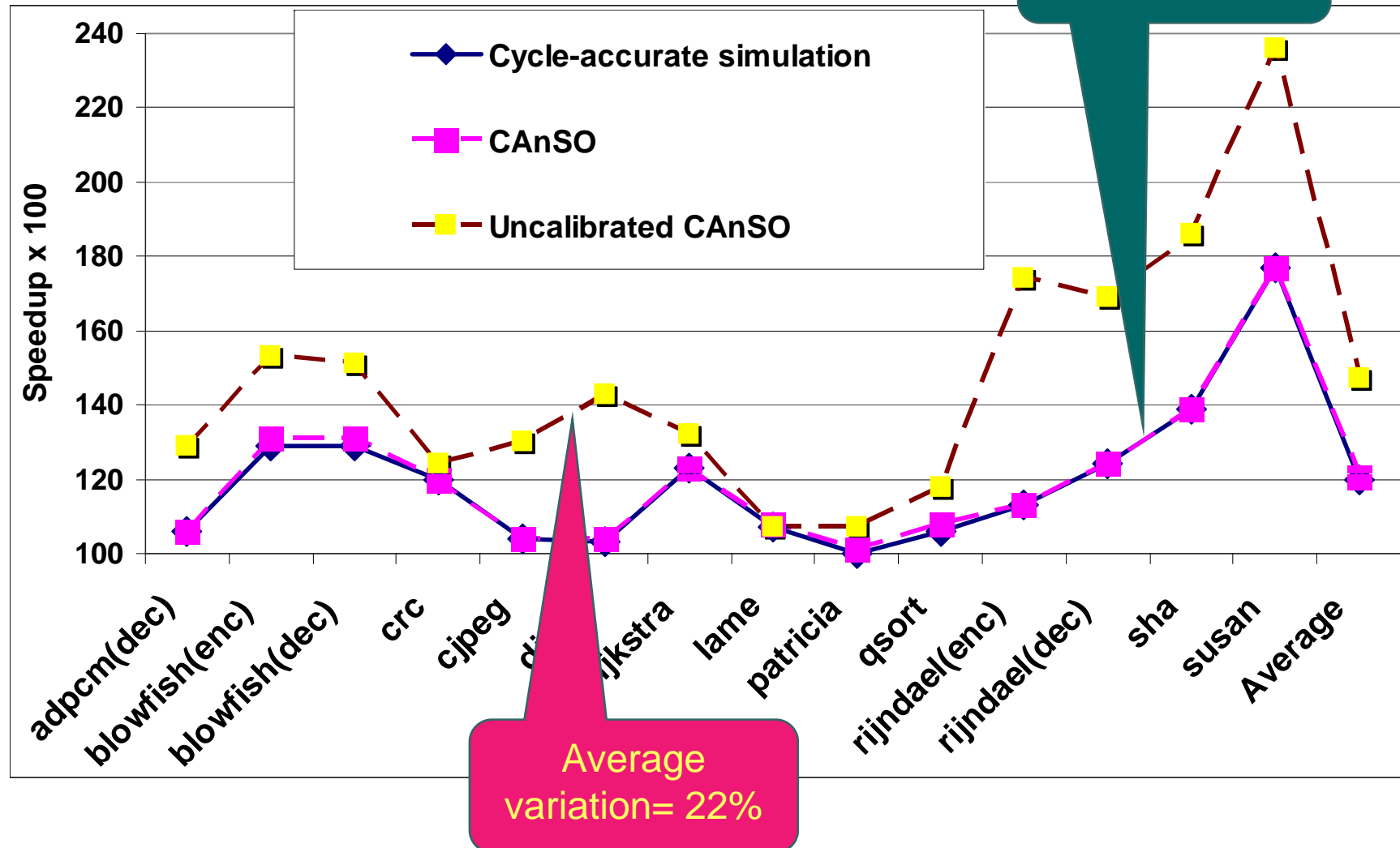
Design Space Exploration Using CAnSO

- The design of a RAC including different components entails a multitude of design parameters
- Examining 100 design points using 14 applications:
  - Simulation: 17 days
  - CAnSO: 4 hours
- Using CAnSO, re-simulation is not needed after establishing the model

Increasing the width of RAC increases speedup
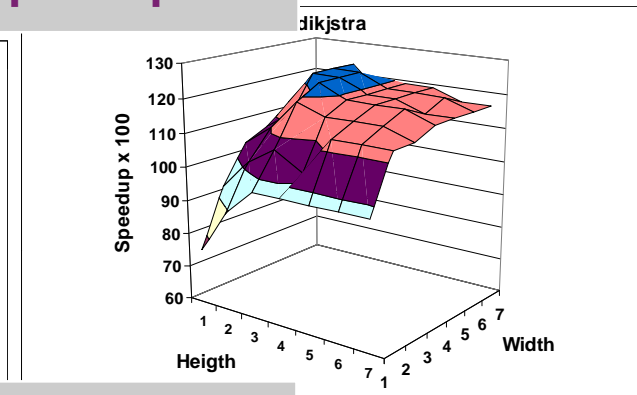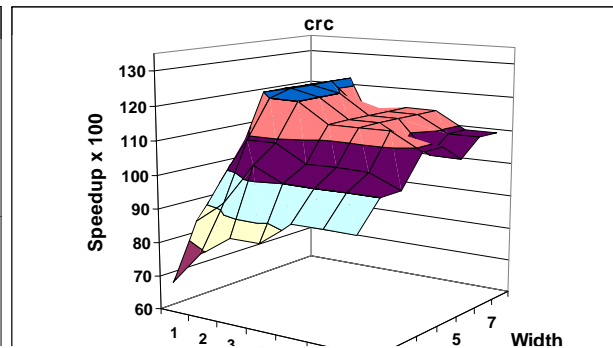
Width> 6: no more speedup is achievable
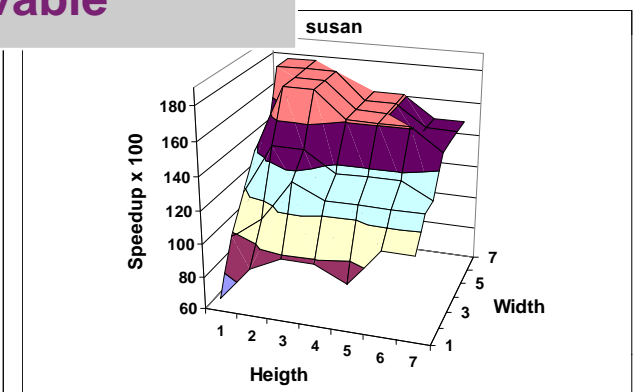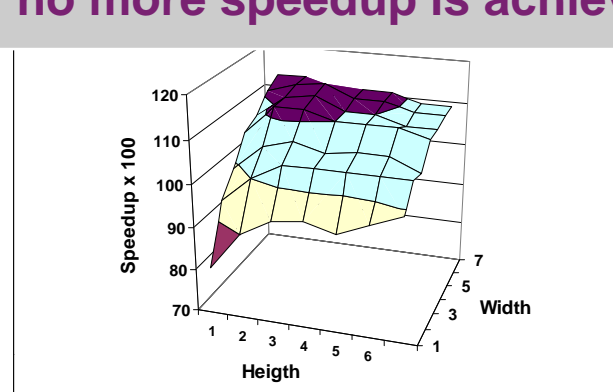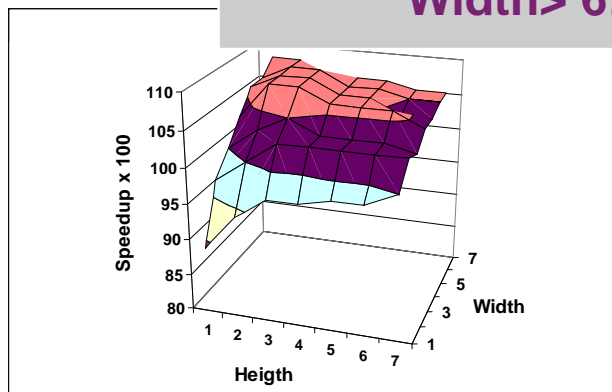
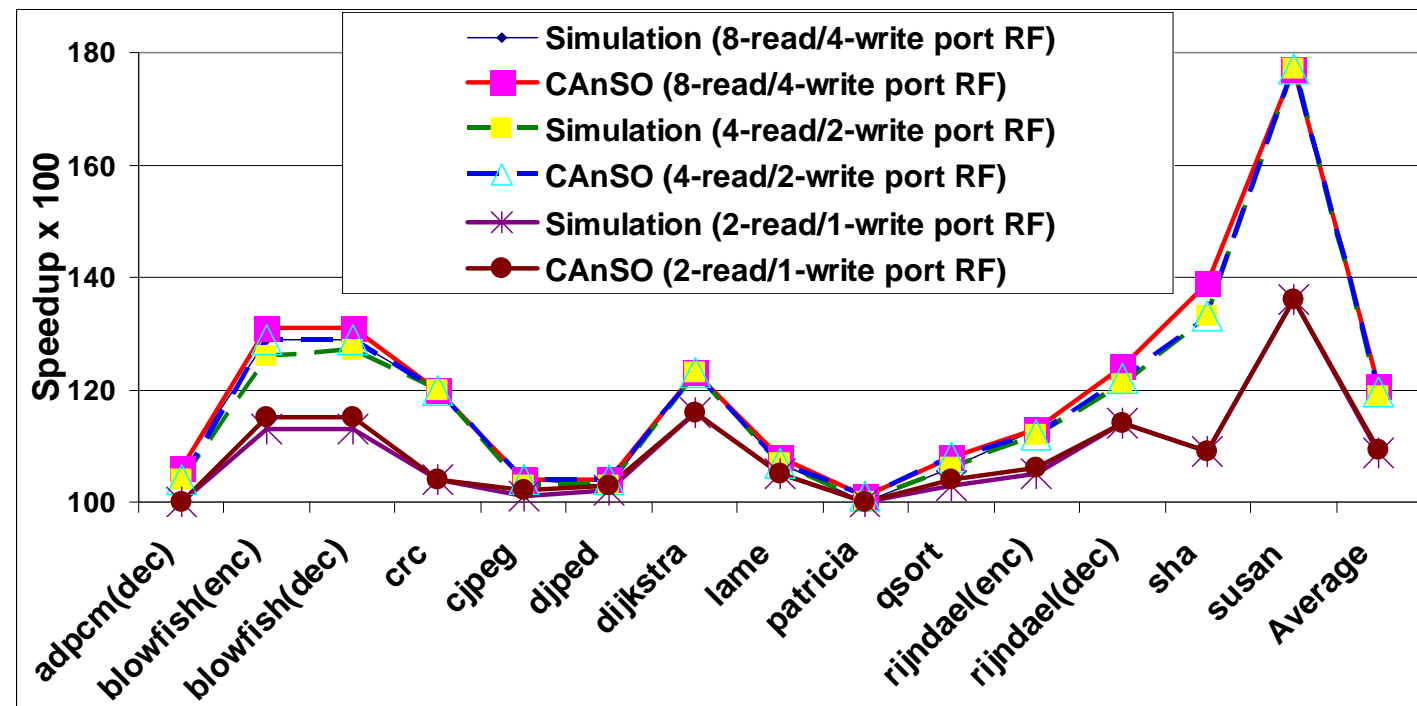**the small heights → very low speedup**

Height> 5: RAC's longer critical path delay → speedup declines

Effect of Modifications

Applying modification to the design→
- Small time is required for repeating the simulation
- Each iteration of the CAnSO takes less than a minute

CONCLUSION

- **Reconfigurable instruction set processors**

- **A combined analytical and simulation-based model (CAnSO)**

- **Suitable for exploring a large design space for the accelerator**

- **Sufficient flexibility in a rapid evaluation of modified target architectures**

- **Substantially reduce the design or optimization time while preserving a reasonable accuracy**

- **Proves less than 2% variation in evaluation results**

- **Uncalibrated CAnSO depicts 22% difference in average**

- **Future work:**
  - **Expanding CAnSO to support control instructions**
  - **Considering more complicated RAC architectures**