High Performance, Low Power Reconfigurable Processor for Embedded Systems

Mehdipour, Farhad
Computing and Communication Center, Kyushu University

Noori, Hamid
Department of Informatics, Kyushu University

Inoue, Koji
Department of Informatics, Kyushu University

Murakami, Kazuaki
Department of Informatics, Kyushu University

出版情報：International SoC Design Conference, pp.51-55, 2007-10
バージョン：
権利関係：

High Performance, Low Power Reconfigurable Processor for Embedded Systems

Farhad Mehdipour, Hamid Noori, Koji Inoue, Kazuaki Murakami

Kyushu University, Japan

OUTLINE

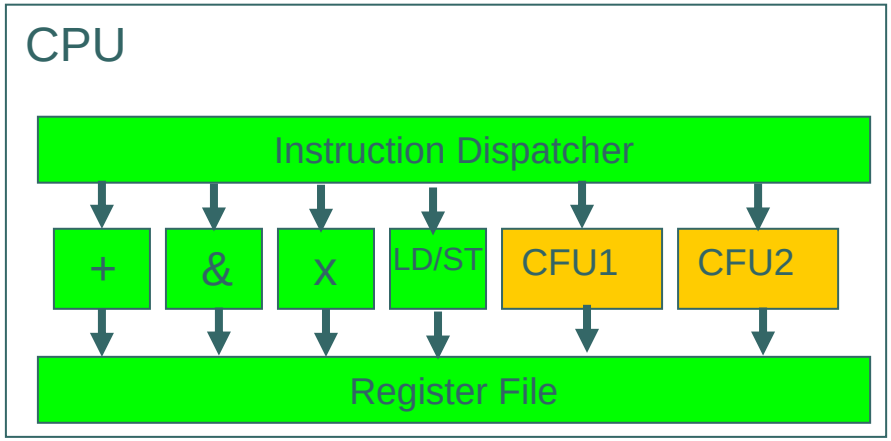
- Introduction
- Motivations for Supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

INTRODUCTION

- Designing Embedded Systems
  - Embedded Microprocessors
  - Application Specific Integrated Circuits (ASICs)
  - Application Specific Instruction set Processors (ASIPs)
  - Extensible Processors



LD/ST: Load / Store

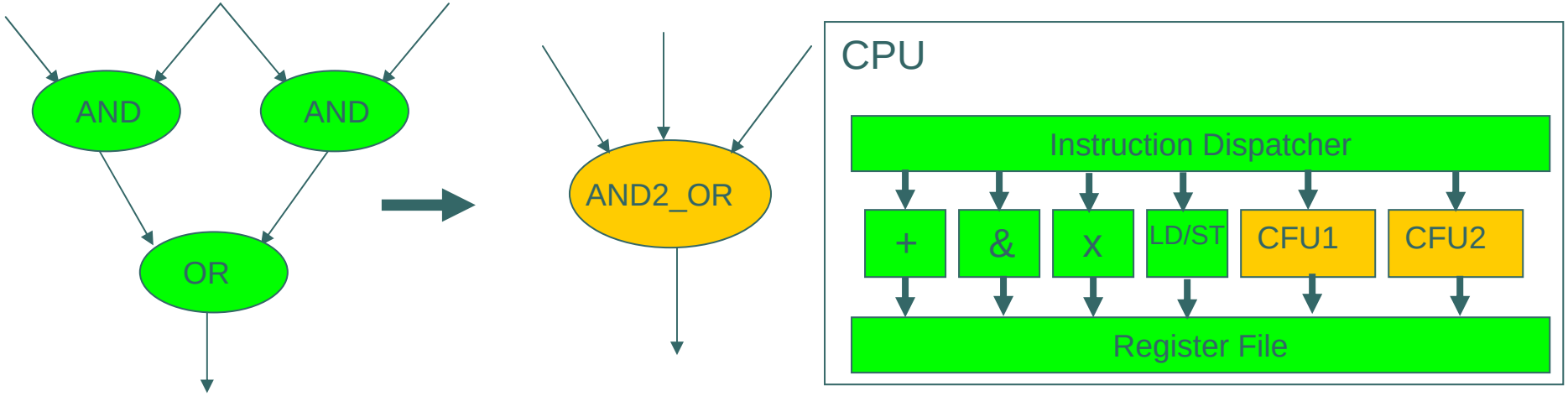
CFU: Custom Functional Unit

GOAL

- Improving the performance and energy efficiency of embedded processors, while maintaining compatibility and flexibility.

EXTENSIBLE PROCESSORS

- Enhancing the performance of a processor in embedded systems
  - Using an accelerator for accelerating frequently executed portions of applications
- Accelerator implementations
  - reconfigurable fine/coarse grain hw
  - custom hardware (such as ASIP or Extensible Processors)



LD/ST: Load / Store

CFU: Custom Functional Unit

CUSTOM INSTRUCTIONS

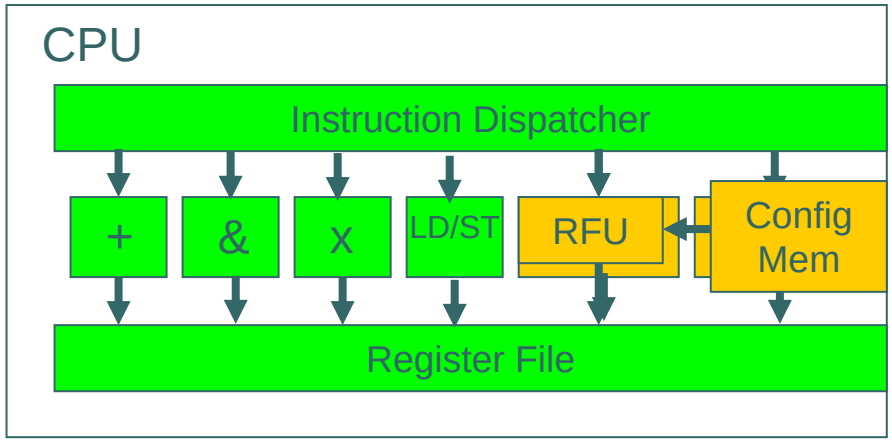
- Critical segments→ Most frequently executed portions of the applications
- Instruction set customization ←→ hardware/software partitioning
- Custom Instructions (CIs) are
  - extracted from critical segments of an application and
  - executed on an Custom Functional Unit (CFU)
- A CI is represented as a Data Flow Graph (DFG)
- CI or DFG generation levels
  - high level or
  - binary level (DFG nodes are the instructions level operations)

Adaptive Extensible Processors

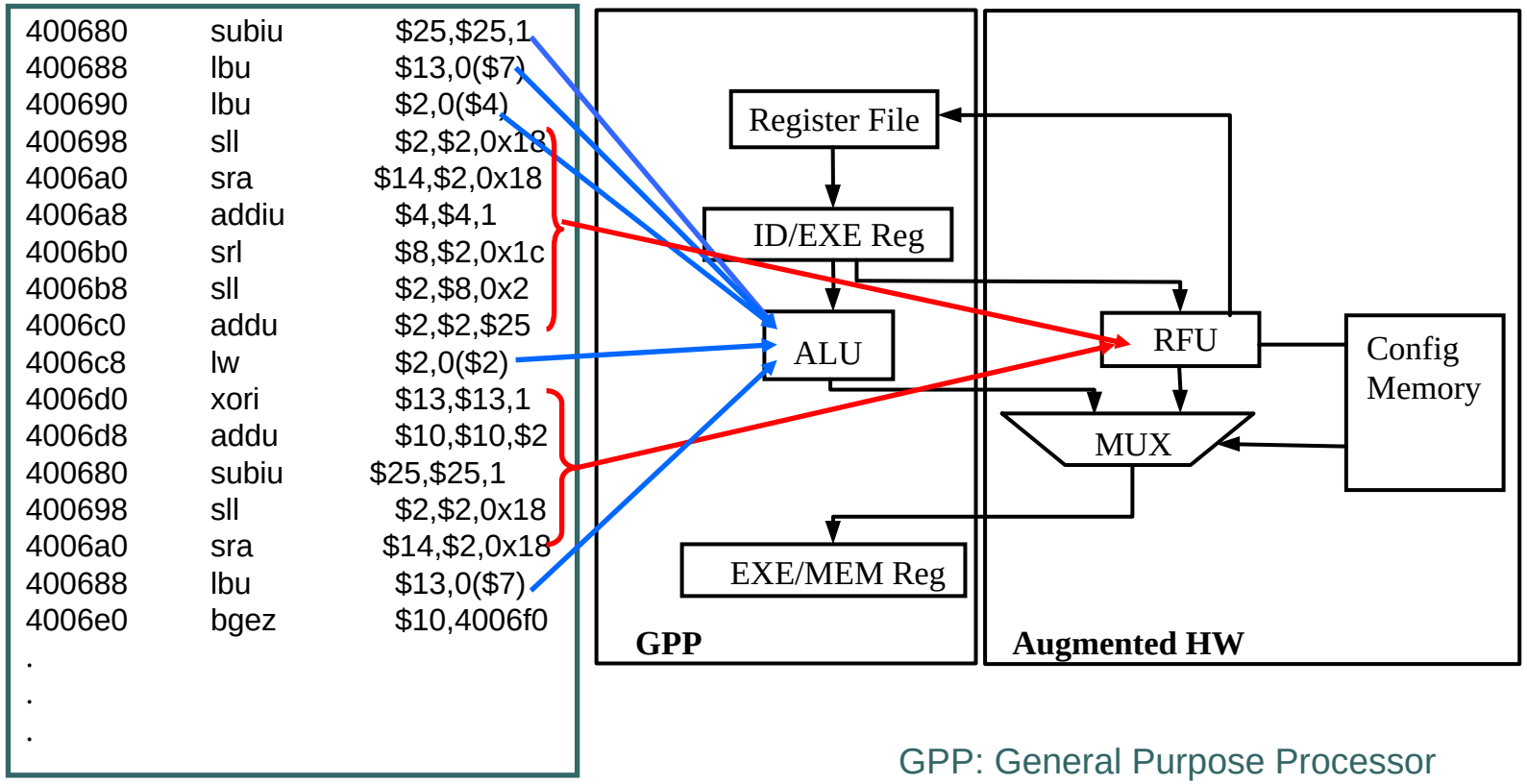
- Issues of Extensible Processors
  - High NRE (Non-Recurring Engineering) and manufacturing costs
  - Long time-to-market
- Adaptive Extensible Processor
  - Adding and generating custom instructions after fabrication
  - Using a reconfigurable functional unit (RFU) instead of custom functional unit

CFU: Custom Functional Unit

RFU: Reconfigurable Functional Unit



How Extensible Processor Works



Hot Basic Block

GPP: General Purpose Processor

RFU: Reconfigurable Functional Unit

OUTLINE

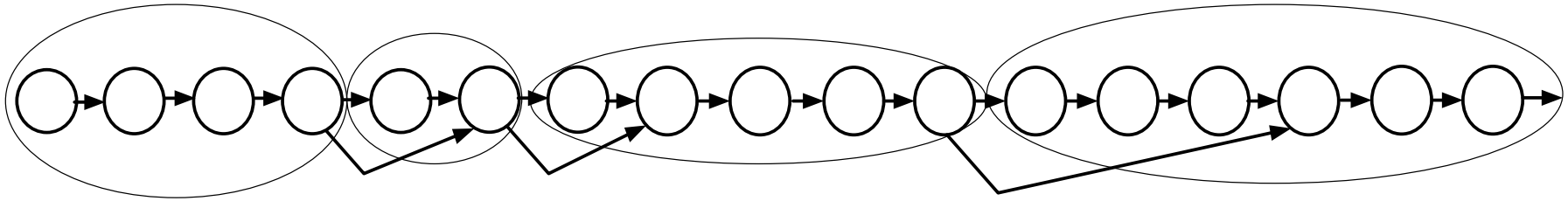
- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

CONTROL DATA FLOW GRAPH: DEFINITION

- CDFG → The DFG containing control instructions (e.g. branch instruction)
- The sequence of execution changes due to the result of a branch instructions
- Types of CDFGs:
  - CDFGs containing at most one branch instruction (last instruction) → accelerator does not need to support conditional execution
  - CDFGs containing more than one branch instruction → accelerator should support conditional execution

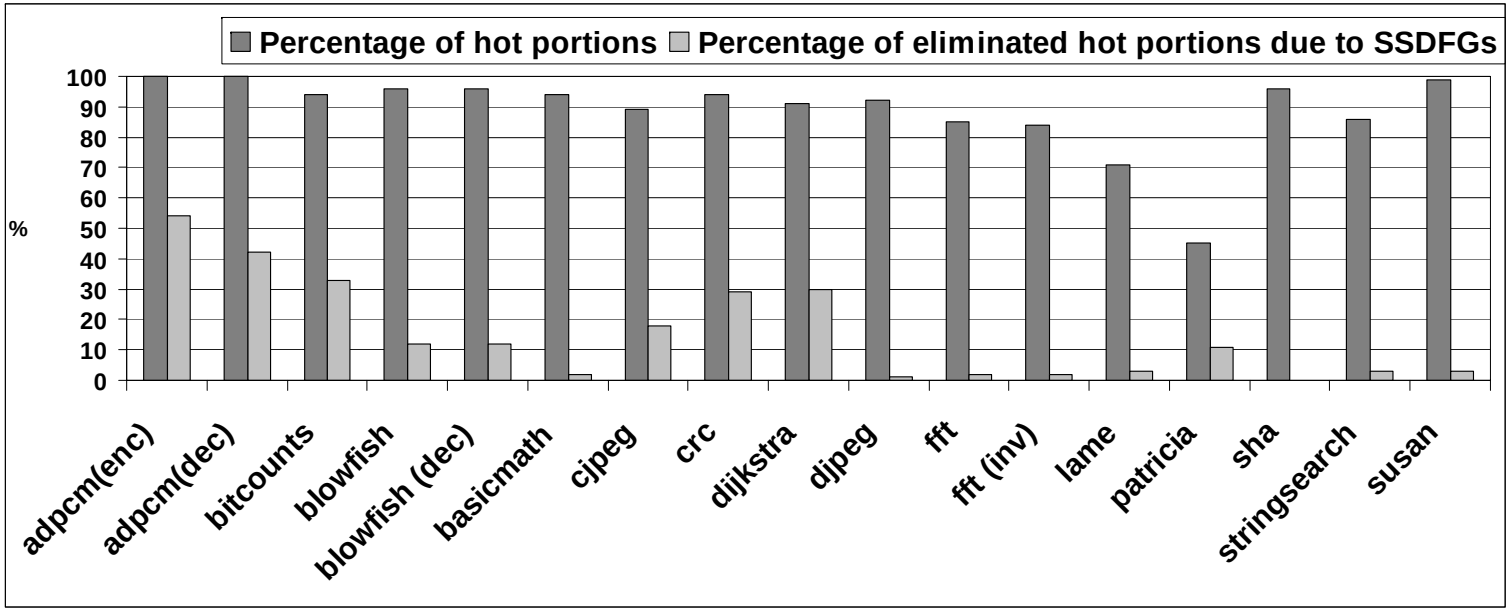
WHY NEED TO SUPPORT CONTROL INSTRUCTIONS- MOTIVATIONS (1/2)

- Quantitative analysis approach using applications of Mibench
- DFG extraction process
  - Short distance control instructions → small size DFGs (SSDFG)
- SSDFGs do not offer noticeable speedup → have to be run on the base processor



WHY NEED TO SUPPORT CONTROL INSTRUCTIONS- MOTIVATIONS (2/2)

- Analysis on 17 application of Mibench
- *bitcount*
  - almost 92% of application is hot
  - 32% out of 92% of hot portions do not worth to be accelerated due to the SSDFGs
- *fft*, *fft(inv)* and *sha*
  - include few branch instructions
  - supporting conditional execution results in no considerable speedup

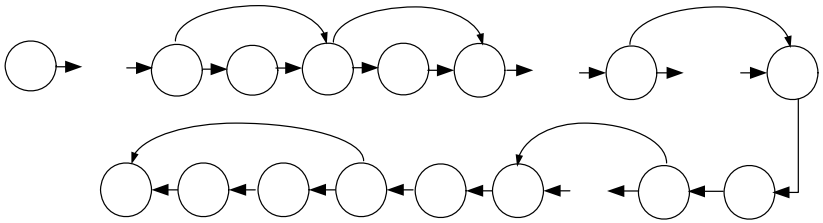


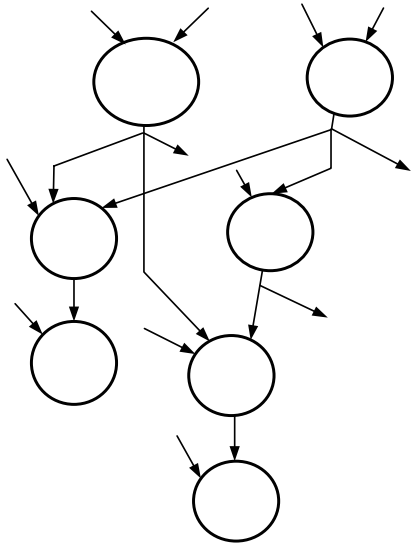
OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

BASIC REQUIREMENTS

- DFG nodes receive their input from a single source
- CDFG nodes can have multiple sources
- The correct source is selected at run time according to the results of branches

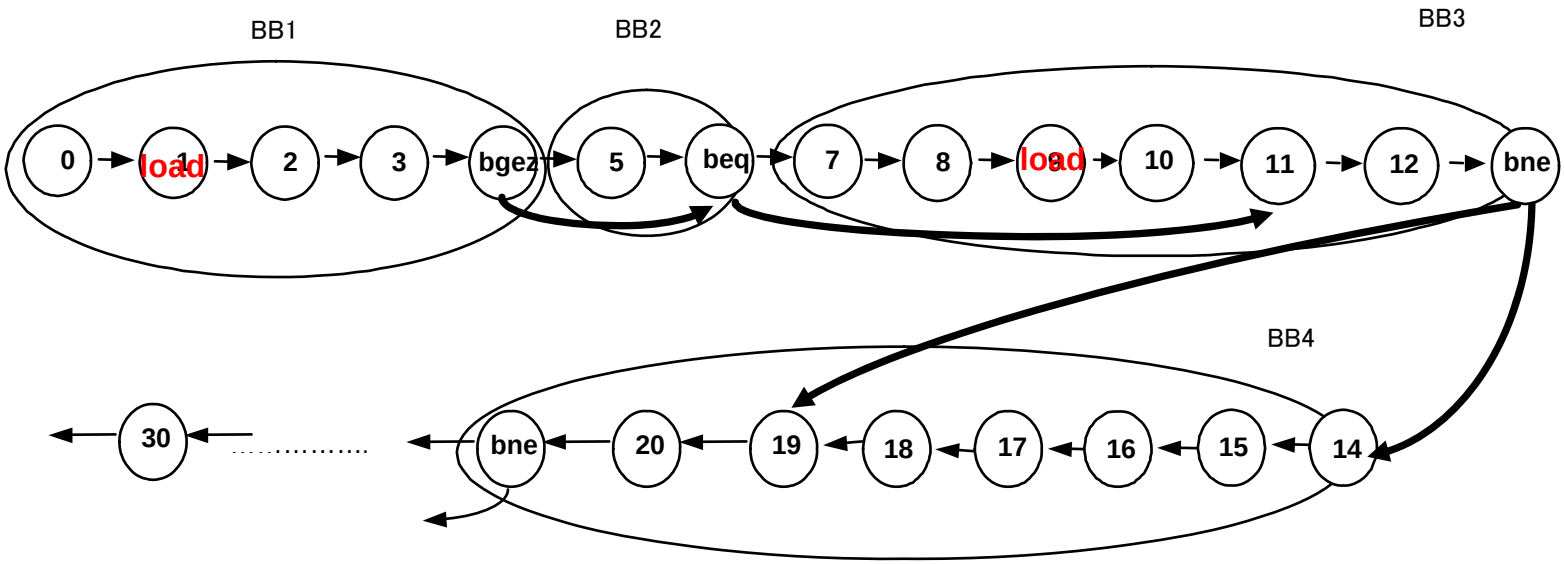




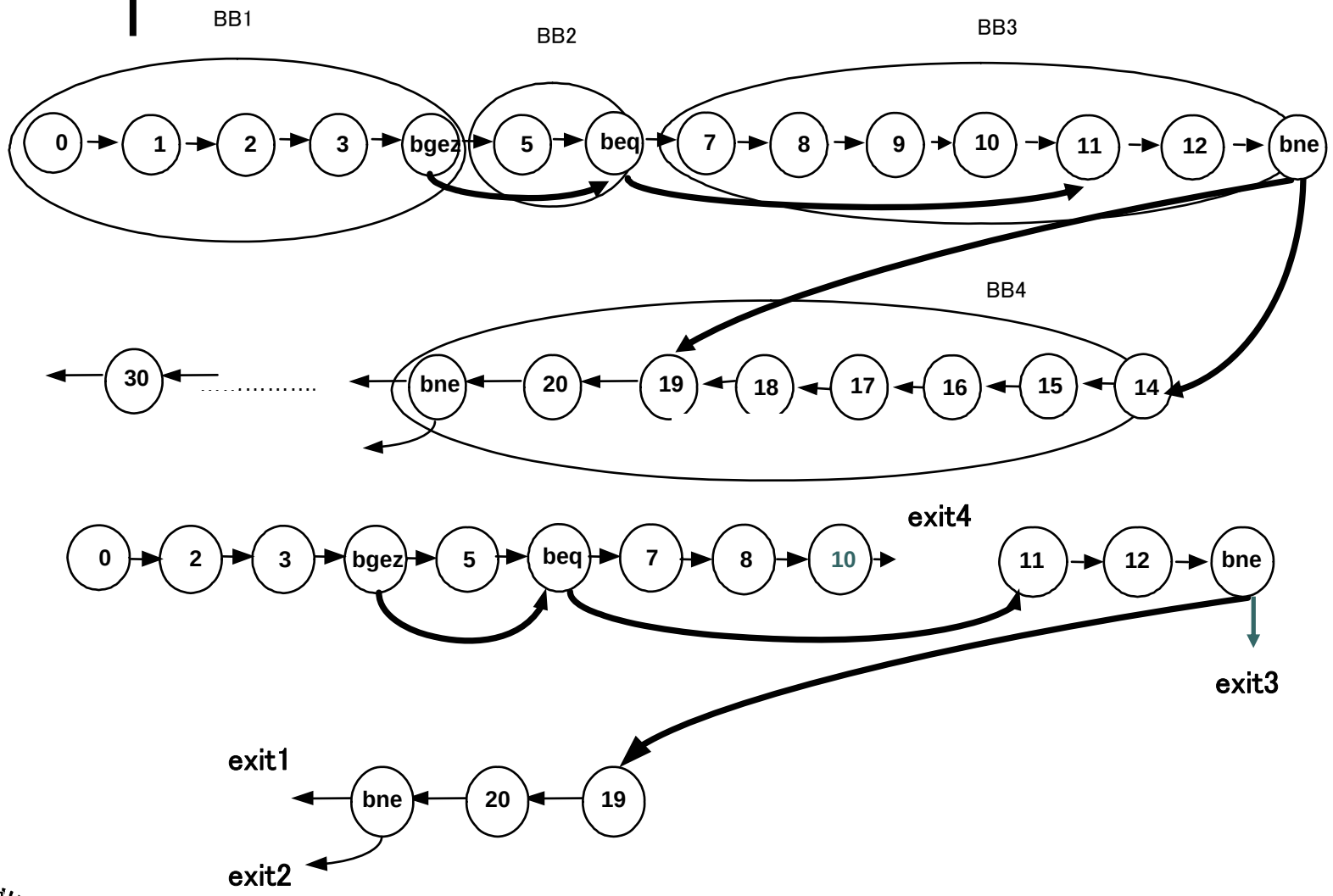
BASIC REQUIREMENTS

- Capability of selective receiving of inputs from both accelerator primary inputs and output of other instructions (FUs) for each node
- Capability of selecting the valid outputs from several outputs generated by accelerator according to conditions made by branch instructions
- Accelerator should be equipped by control path to provide with the correct selection of inputs and outputs for each FU and entire accelerator

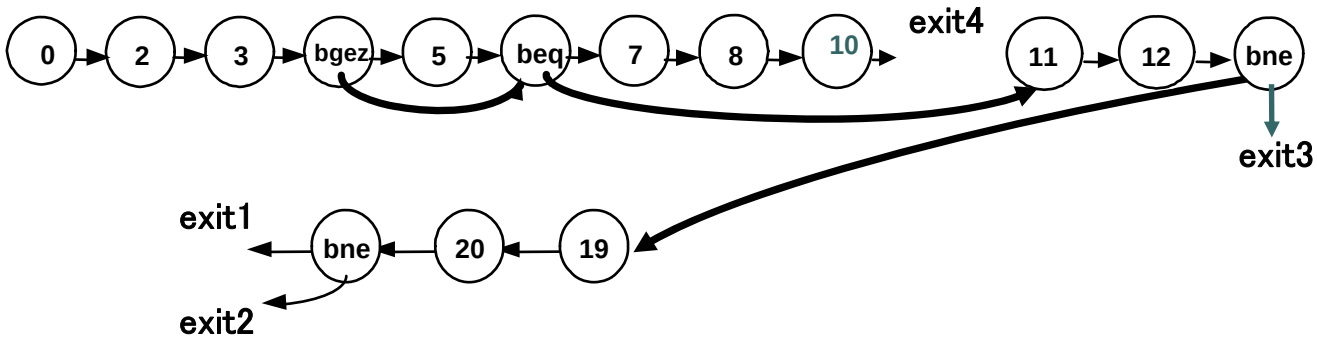
CDFG GENERATION-EXAMPLE (1/3)



CDFG GENERATION-EXAMPLE (2/3)



CDFG GENERATION-EXAMPLE (3/3)



inst. #	*address*	*inst.*	*operands (dest, src1, src2)*
0	400410	addu	R13 R0 R0
1	400418	lw	R23 100 R2
2	400420	addiu	R4 R4 2
3	400428	subu	R3 R2 R11
4	400430	bgez	400440 R3
5	400438	addiu	R13 R0 8
6	400440	beq	400468 R13
7	400448	subu	R3 R0 R3
8	400450	addu	R10 R0 R0
9	400458	lw	R8 R9 0x3
10	400460	slt	R2 R3 R9
11	400468	addu	R8 R8 R9
12	400470	ori	R10 R10 1
13	400478	bne	4004a8 R2
14	400480	addiu	R10 R0 4
15	400488	subu	R3 R3 R9
16	400490	addu	R8 R8 R9
17	400498	sra	R9 R9 0x1
18	4004a0	slt	R2 R3 R9
19	4004a8	ori	R10 R10 2
20	4004b0	subu	R3 R3 R9
21	4004b8	bne	400410 R2
22	4004c0	slt	R2 R3 R9

Code before generating MECI

inst. #	*address*	*inst.*	*operands (dest, src1, src2)*
1	400410	lw	R23 100 R2
0	**400418**	**addu**	**R13 R0 R0**
2	**400420**	**addiu**	**R4 R4 2**
3	**400428**	**subu**	**R3 R2 R11**
4	**400430**	**bgez**	**400440 R3**
5	**400438**	**addiu**	**R13 R0 8**
6	**400440**	**beq**	**400468 R13**
7	**400448**	**subu**	**R3 R0 R3**
8	**400450**	**addu**	**R10 R0 R0**
10	**400458**	**slt**	**R2 R3 R9**
9	400460	lw	R8 R9 0x3
11	**400468**	**addu**	**R8 R8 R9**
12	**400470**	**ori**	**R10 R10 1**
13	**400478**	**bne**	**4004a8 R2**
14	400480	addiu	R10 R0 4
15	400488	subu	R3 R3 R9
16	400490	addu	R8 R8 R9
17	400498	sra	R9 R9 0x1
18	4004a0	slt	R2 R3 R9
19	**4004a8**	**ori**	**R10 R10 2**
20	**4004b0**	**subu**	**R3 R3 R9**
21	**4004b8**	**bne**	**400410 R2**
22	4004c0	slt	R2 R3 R9

Code after generating MECI

OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

CDFG Temporal Partitioning Algorithms

- CDFGs extracted from various applications
  - have different sizes
  - for some of the CDFGs
    - the whole of CDFG can not be mapped on the accelerator due to the resource limitations of the accelerator
- Resource constraints
  - the number of inputs, outputs, logics and routing resource constraints

CDFG Temporal Partitioning Algorithms

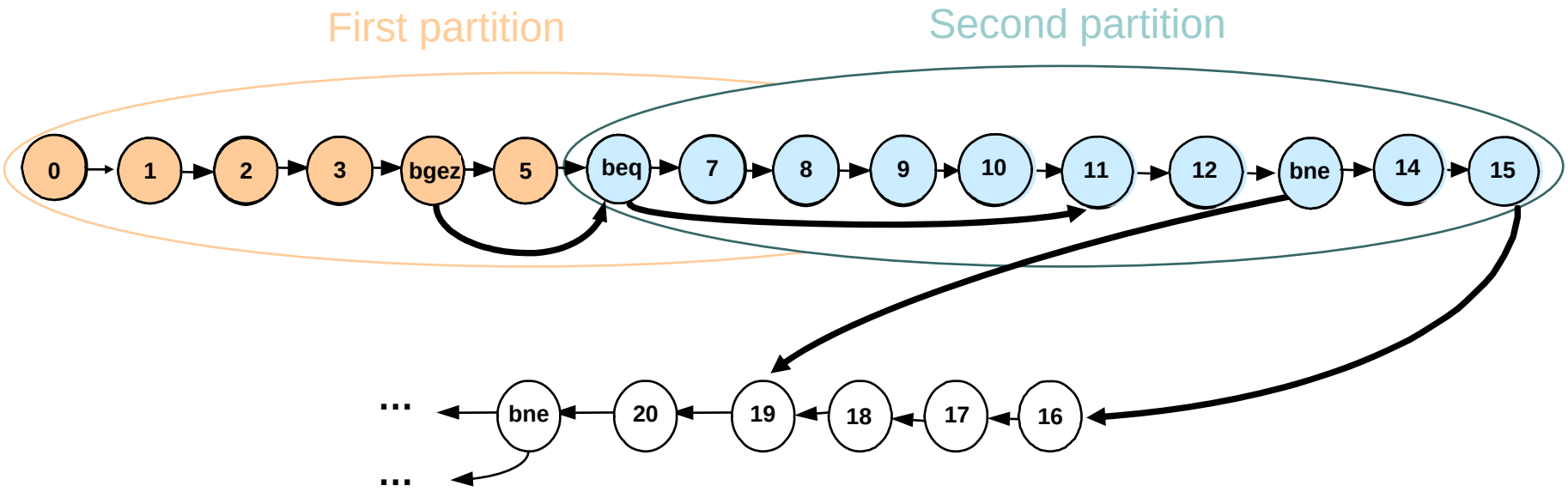
- Temporal partitioning
  - Temporally divides a DFG into a number of smaller partitions
  - each partition can fit into the target hardware
  - dependencies among the nodes are not violated

- Temporal Partitioning algorithms for CDFGs
  - Not-Taken Path Traversing alg. (NTPT)
  - Frequency based TP alg.

TP BASED ON NOT-TAKEN PATHS (NTPT)

- Adds instructions from not-taken path of a control instruction to a partition until
  - violating the target hardware architectural constraints or
  - reaching to a terminator control instruction
- Generating a new partition is started with the branch instructions which at least one of their taken or not-taken instructions has not been located in the current partition
- Terminator instruction→
  - an instruction which changes execution direction of the program
  - an exit point for a CDFG

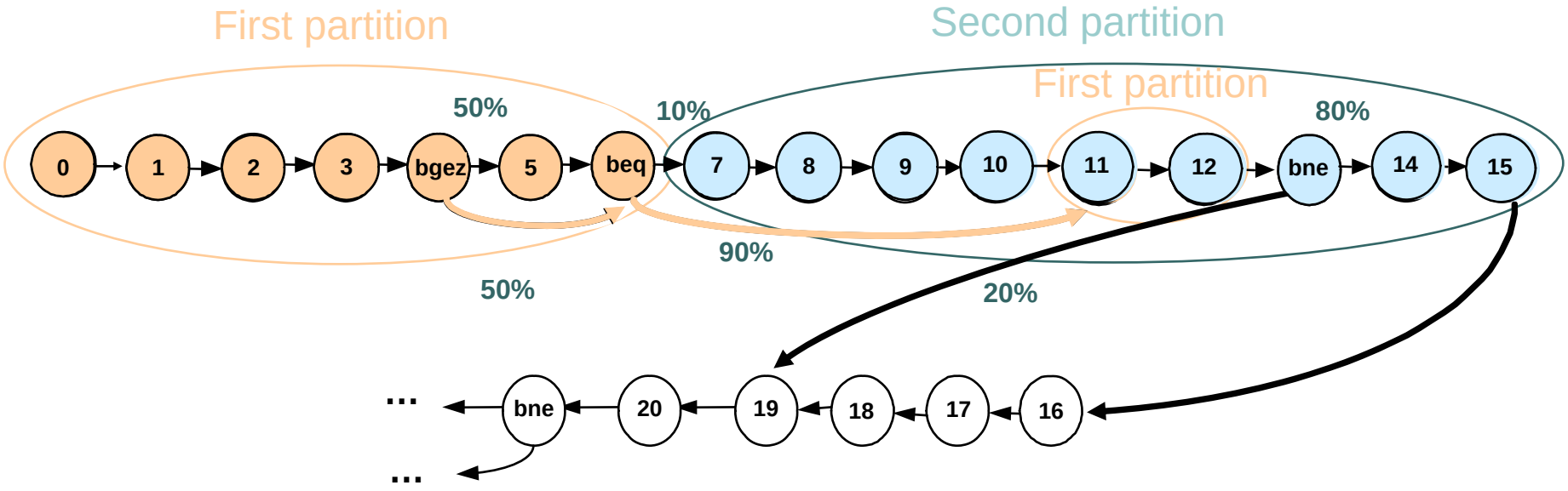
TP BASED ON NOT-TAKEN PATHS (NTPT)



FREQUENCY-BASED TP ALGORITHM

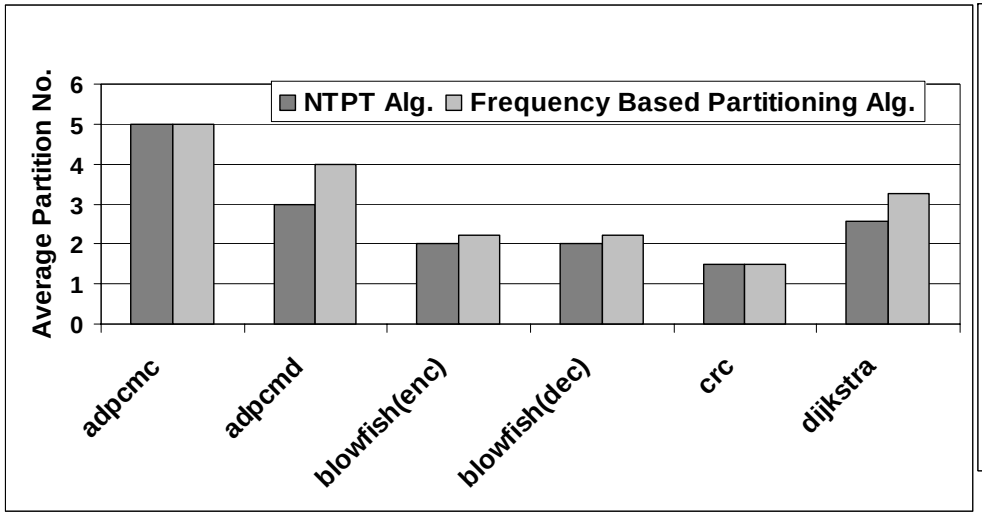
- NTPT algorithm
  - instructions were selected only from not-taken paths of branches
- Frequency-based algorithm
  - execution frequency of taken and not-taken paths is an effective factor in adding them to the current partition
  - a frequency threshold is defined to determine that whether instruction is critical or not
  - one of the taken or not-taken paths or both of them can be critical

FREQUENCY-BASED TP ALGORITHM

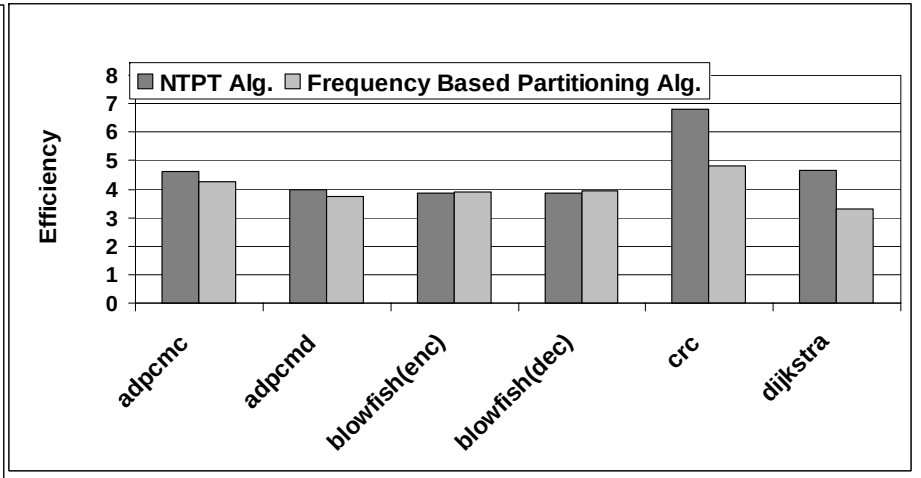


EVALUATING TP ALGORITHMS

Average Partition No.



Efficiency

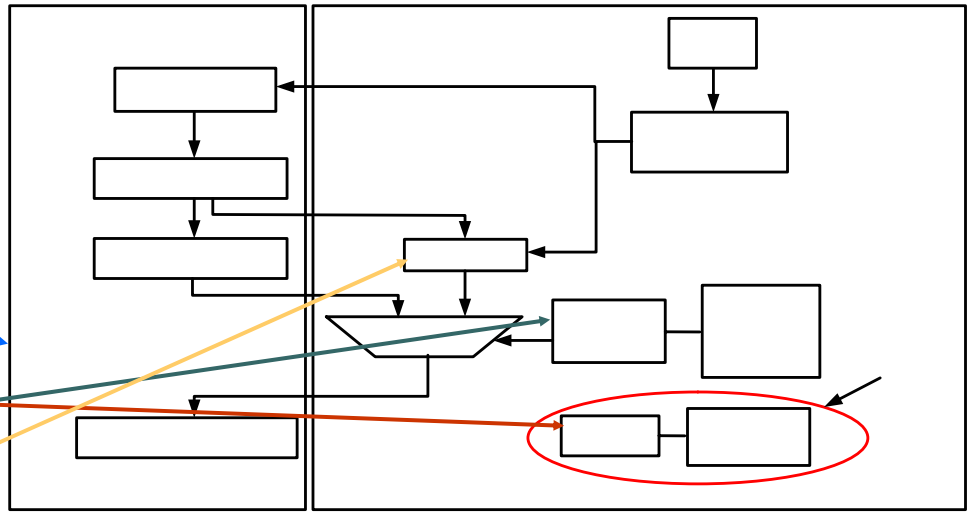


OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

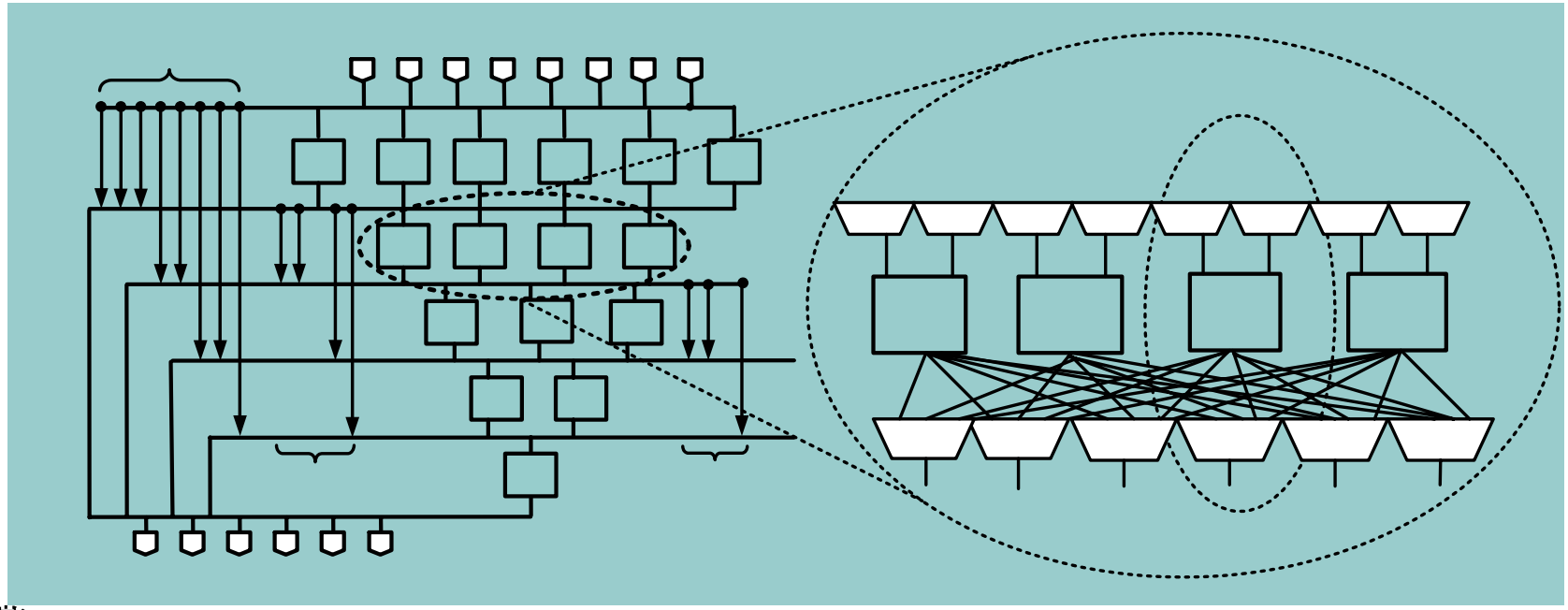
CASE STUDY: EXTENDING AN EXTENSIBLE PROCESSOR TO SUPPORT CONDITIONAL EXECUTION

- AMBER
  - an extensible processor targeted for embedded systems
  - Goal: accelerating application execution
  - Including :
    - a general RISC processor
    - Profiler
    - Sequencer
    - a coarse grain RFU
  - RFU: an accelerator without conditional execution support



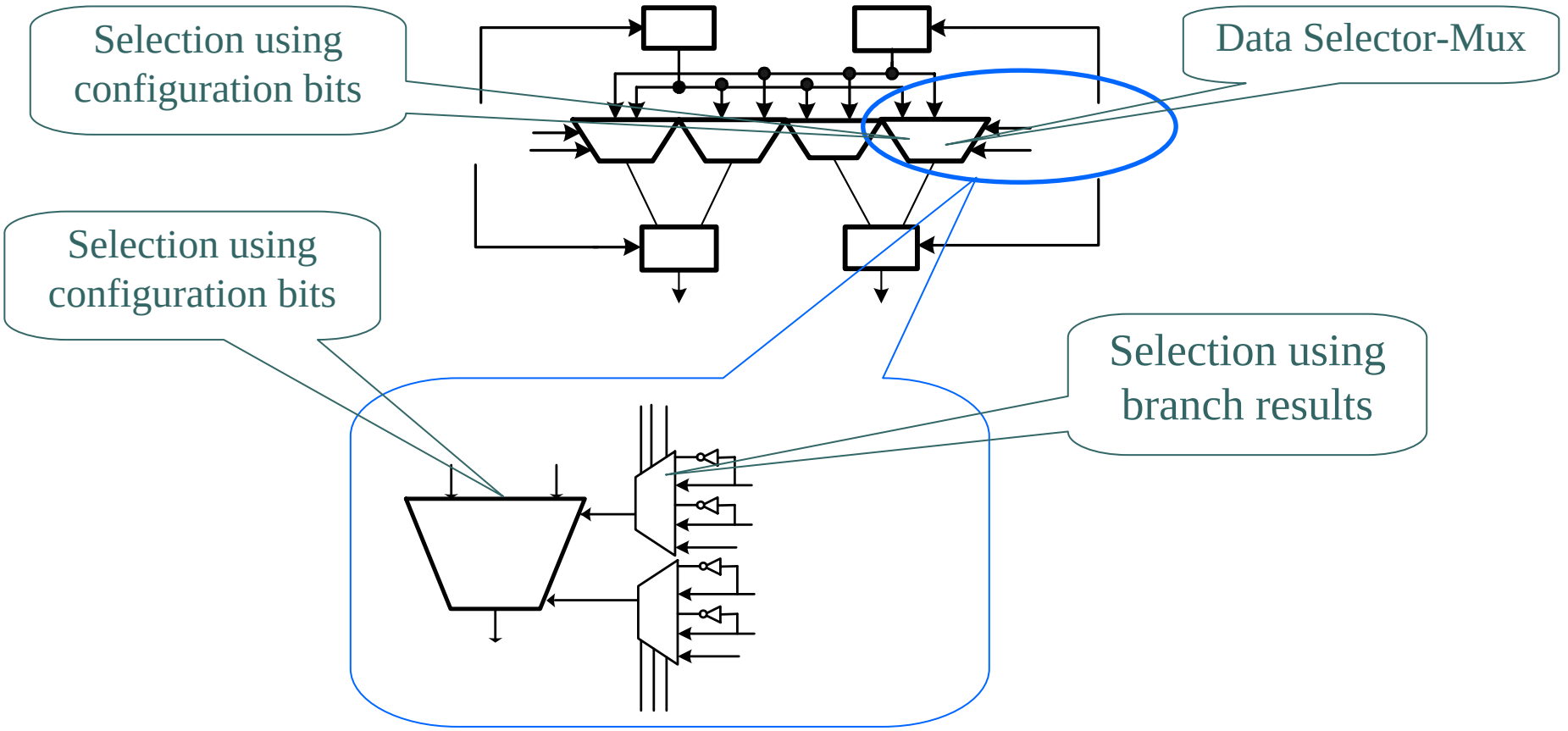
EXTENDING AMBER's RFU (CRFU)

- The selectors of muxes are
  - used for choosing data for FU inputs
  - controlled by the configuration bits
- The outputs of FUs are only applied to the Selector-Muxes in the lower-level rows



CRFU

- Inputs of Selector-Mux (one-bit width) originate from the FUs executing branch instructions



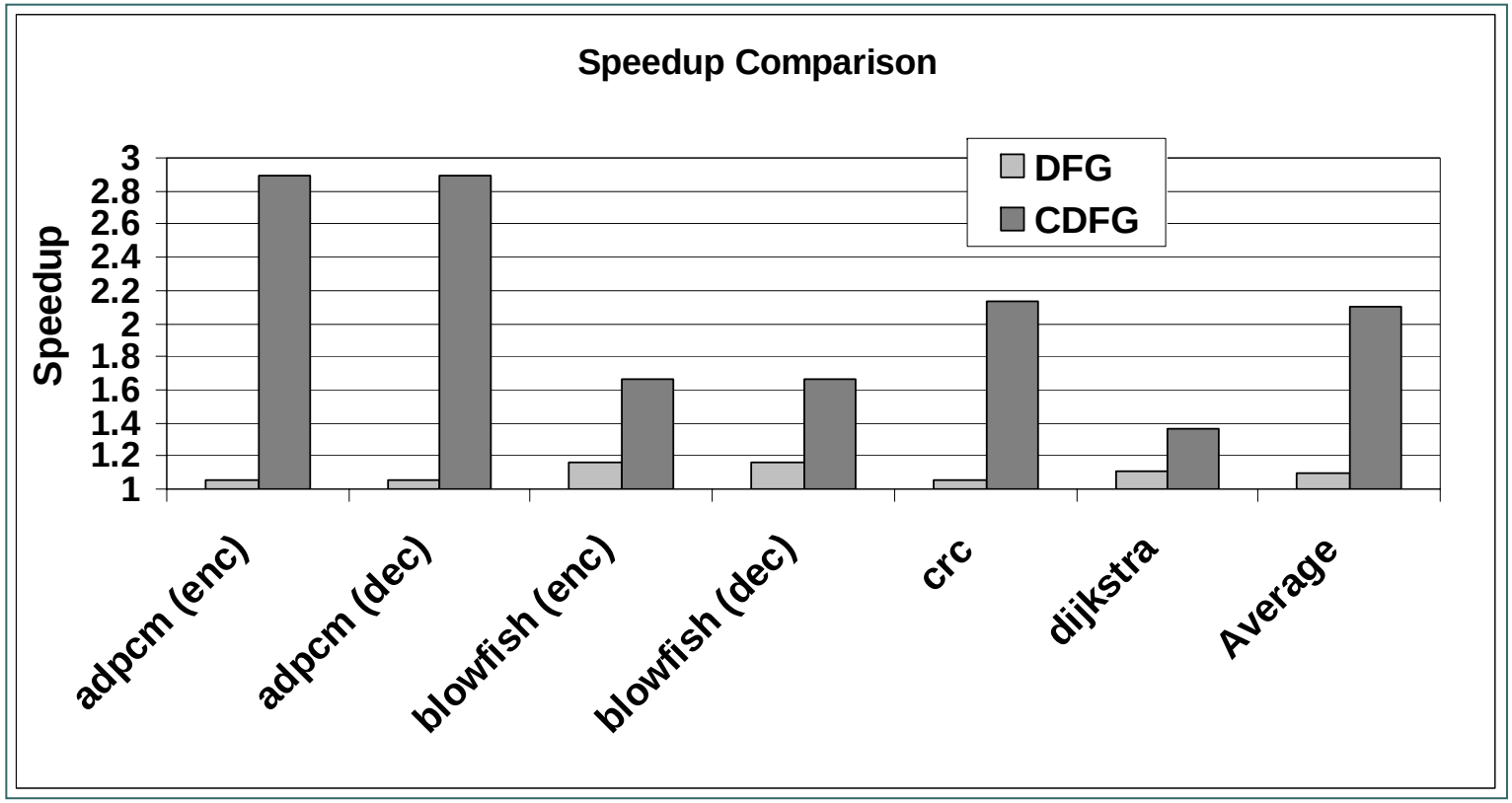
OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

EXPERIMENTAL RESULTS

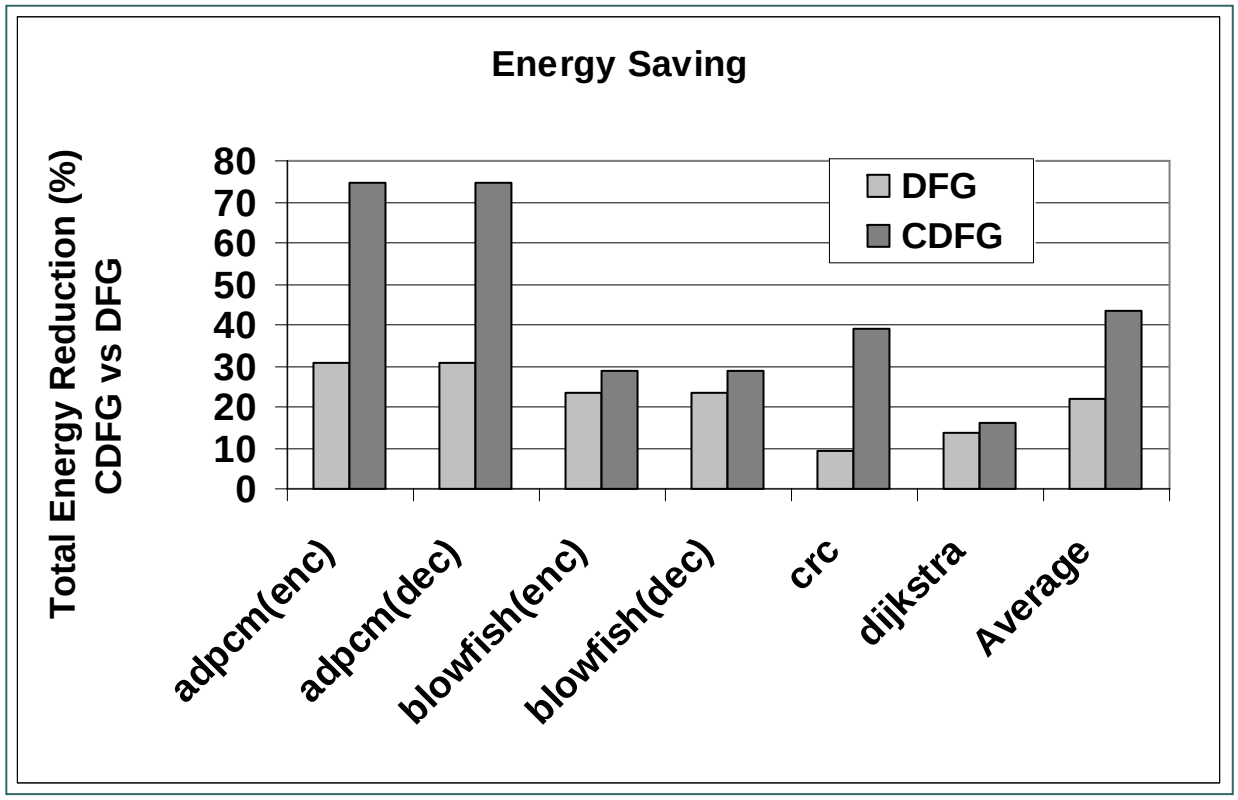
- Synthesis result of the extended RFU
  - Synopsys tools and Hitachi 0.18 μ m
  - area → 2.1 mm^2
- CDFG configuration bit-stream → 615 bits
  - Control signals: 375 bits
- Executing applications on the Simplescalar as ISS →
  - Profiling and extracting hot instruction sequence
- NTPT temporal partitioning algorithm for generating mappable CDFGs

EXPERIMENTAL RESULTS - SPEEDUP



Average Speedup: **2.1 (CDFG)** versus **1.1 (DFG)**

EXPERIMENTAL RESULTS – ENERGY CONSUMPTION



Average Energy Saving: **43% (CDFG)** versus **21% (DFG)**

Experimental Results - Comparison

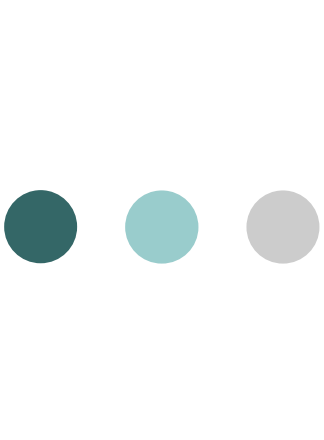
Approach	Base Processor	Accelerator Coupling to Processor	Technology Size (μm)	Accelerator Granularity	Accelerator Delay (# of clock cycles)	Average Speedup	Energy Reduction (%)
Yehia et al.	ARM 1-issue, 250MHz	Tight	0.13	Fine (LUT-based)	1	**1.47**	N/A
Clark et al.	ARM 4-issue, 250MHz	Tight	0.13	Coarse (FU-based)	1	**1.28**	N/A
Ours	**MIPS 1-issue, 300MHz**	**Tight**	**0.18**	**Coarse (FU-based)**	**Variable**	**2.1**	**43.5**

OUTLINE

- Introduction
- Motivations for supporting Control Instructions
- Basic Requirements for Supporting Conditional Execution
- Algorithms for CDFG Temporal Partitioning
- Case study: Extending an Extensible Processor to Support Conditional Execution
- Experimental Results
- Conclusion

Conclusion

- Handling branch instruction & extending DFGs to CDFGs
- Basic requirements for an accelerator featuring conditional execution
- CDFG temporal partitioning algorithms
  - NTPT traverse not-taken path of the branch instructions
  - Frequency-based temporal partitioning algorithm
- Extending AMBER's RFU to support conditional execution
  - Increasing speedup
  - Reducing energy consumption



THANK YOU!