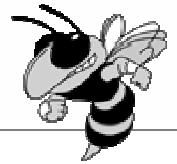
Compiler Optimizations For Highly Constrained Multithreaded Multicore Processors



Xiaotong Zhuang

IBM T.J. Watson Research Center

Agenda

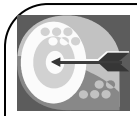
- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Overview of Research

Constraints:
architectural design, power, security, reliability

Optimizing compiler

(with architecture co-design)



Goal:

1. Improve performance

2. Satisfy constraints

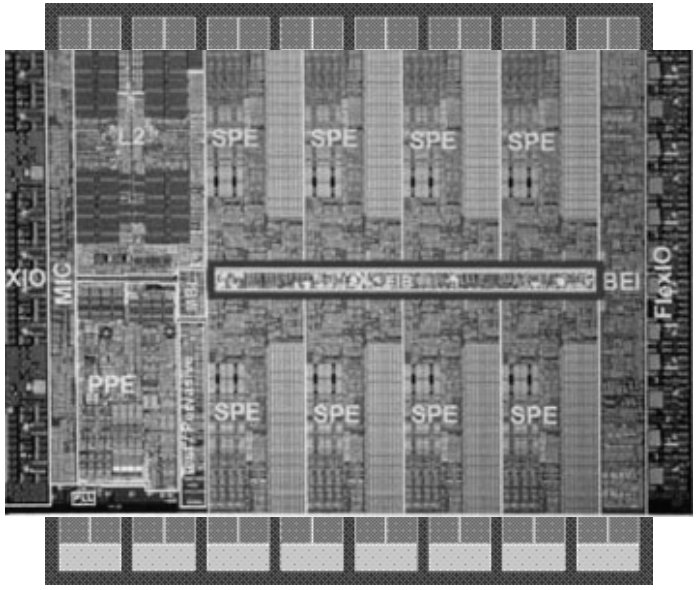
Topics

Compiler Optimizations	PLDI-06, PLDI-04, PACT-03, PACT-02, LCTES-06, LCTES-03, ACM TECSx2, ICDCS-03
Compiler Optimizations + Architectural Support	PLDI-05, ACM TOPLAS, LCTES-04, ACM TECS, LCTES-04, IPDPS-06
Compiler Optimizations for Security, Secure Architecture	ASPLOS-04, MICRO-06, CASES-04 Best Paper, CGO-06, CGO-05, CASES-05
Others	INFOCOM-06, IPDPS-02, IEEE TPDS, IEEE TOC, ICPP-03

Motivation

- Domain specific multicore processors
  - For special applications
  - Specialized, simplified hardware
  - Complexity pushed to the compiler
  - Thread level parallelism

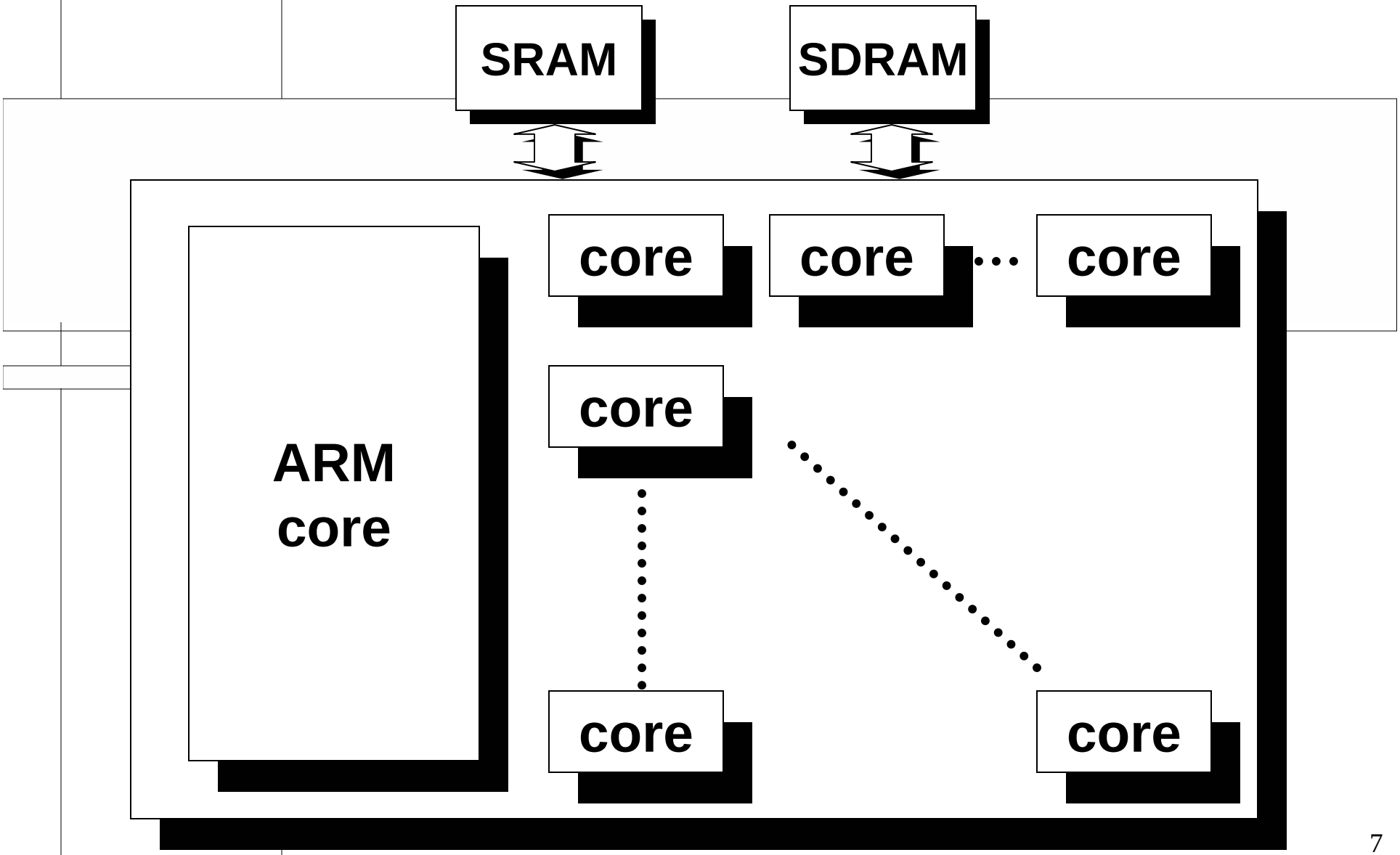
- Examples
  - CELL—1 PPE+8 SPUs
  - Intel's 80 core teraflop processor
  - Cradle CT3616—16 DSPs+8 GPPs
  - ClearSpeed CSX600—96 cores
  - Intel IXP



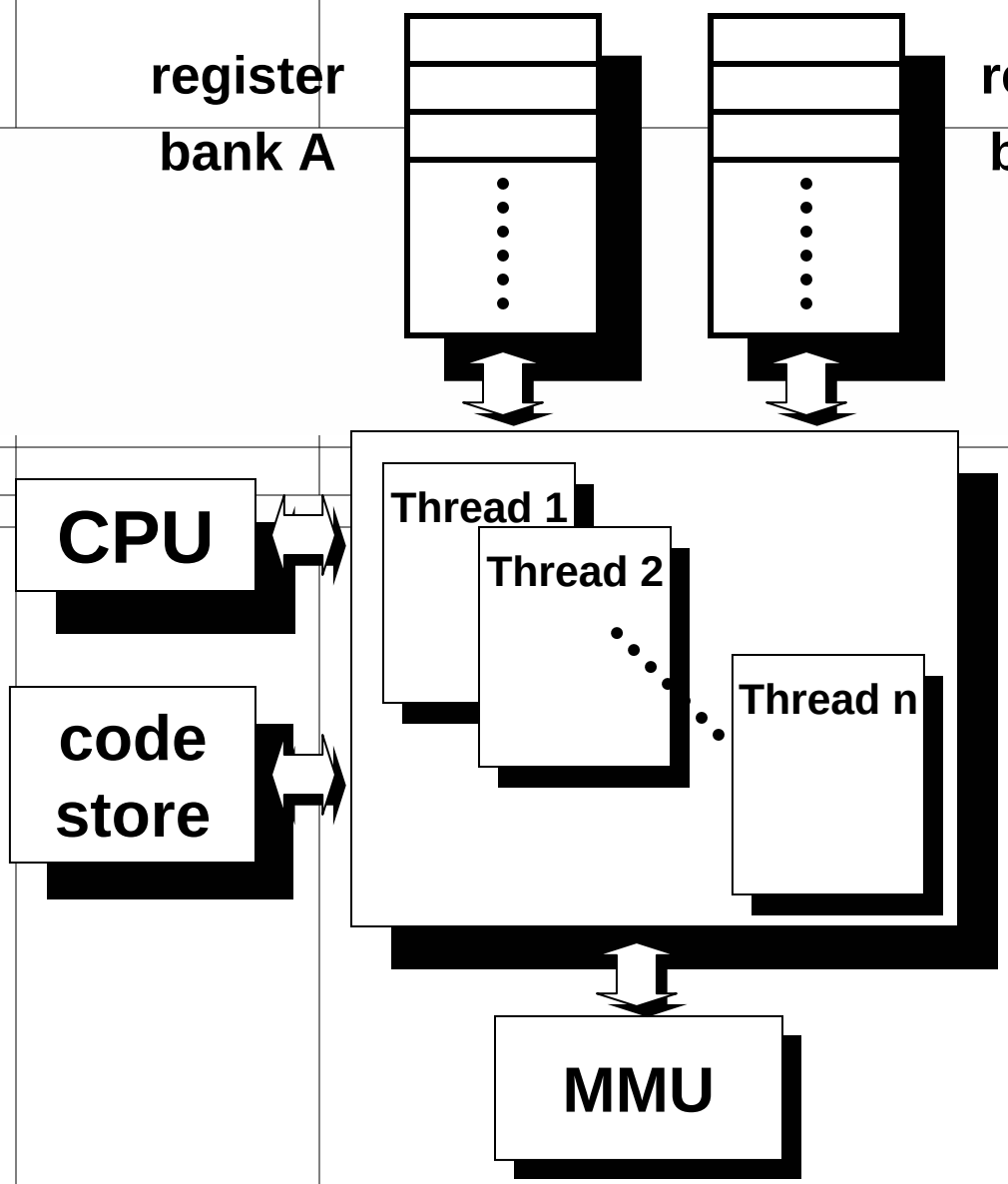
Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

The IXP Processor Model



Packet Processing Core



- No OS, hardware manages threads
- ALU instructions can finish in 1 cycle
- No cache, long memory latency (30~400 cycles)
- Two banks of registers
- Fast context switch

Compiler Challenges

- Code must be highly efficient (1Gb/s => 400 cycle/packet)
- Architectural constraint—register usage
- Resource constraint—not enough registers
  - Large register file is slow and expensive
  - Memory latency is long
  - Functions are often inlined
  - Threads simultaneously active→cannot shared registers
- Service constraint—no OS available

Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Register Allocation Preliminaries

- **GOAL: put variables to registers for faster access**
- **Several variables could be put in the same register if they are active in different places**
- **Some variables might be put in memory (SPILL) when registers are used up**
- **OUTPUT: for each variable, which register or memory location it should be allocated to**

Dual-bank Register Constraint

- Dual-bank Constraint
  - Only for ALU instructions
  - Two source operands must come from different banks
  - Fetching operands in parallel allows 1 cycle latency for all ALU instructions

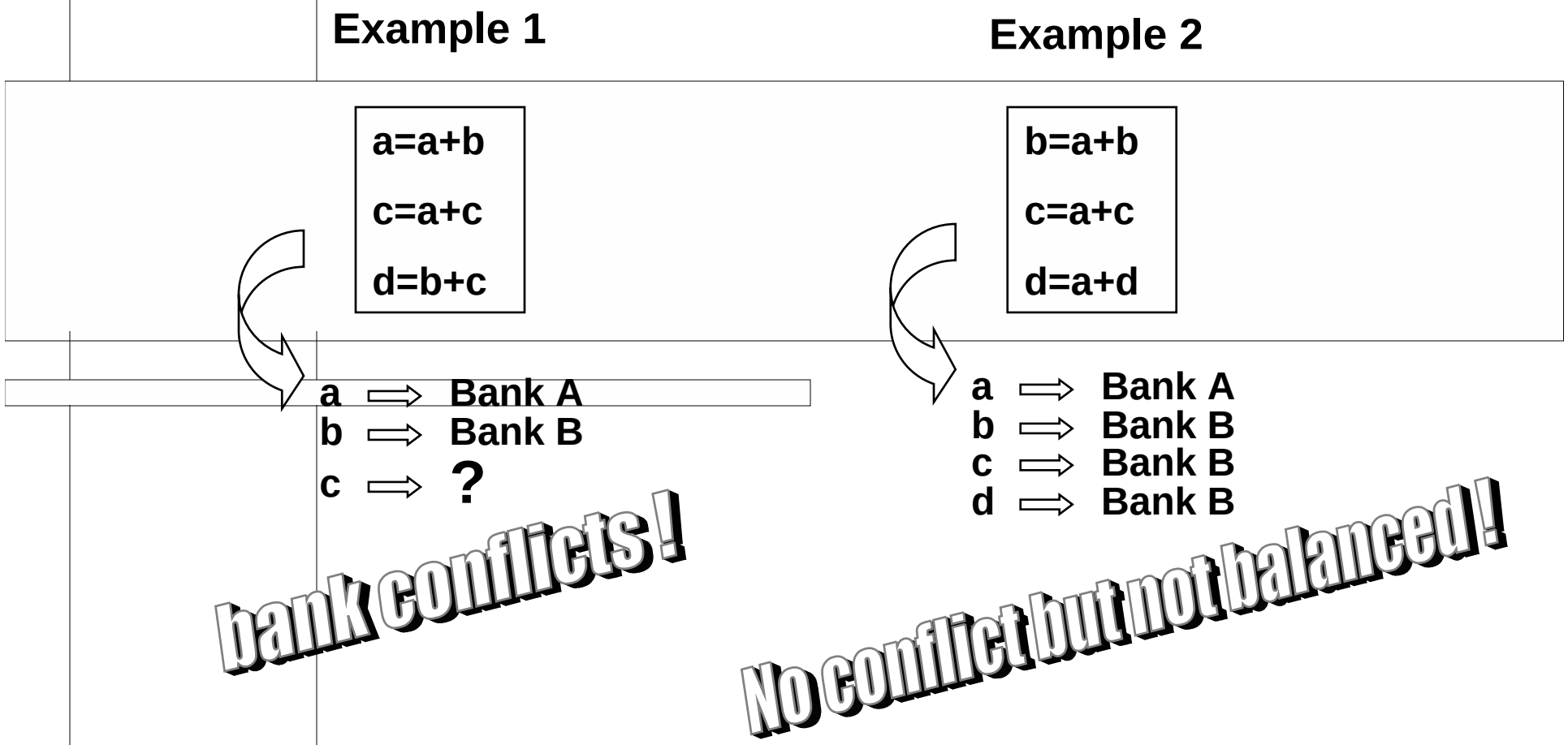
c = a + b

a ⇨ bank A, b ⇨ bank B

OR

a ⇨ bank B, b ⇨ bank A

Two Issues with Dual-bank Register Assignment

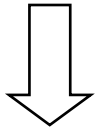


- Intel's assembly tool leaves these problems to the user !!

Register Conflict Graph (RCG)

Register Conflict Graph (RCG)

- Each variable is a node on the graph
- If two variables appear as SOURCE OPERANDS in at least one ALU instruction, they are connected with a CONFLICT EDGE



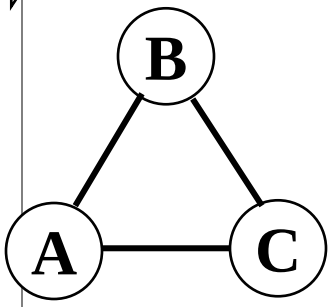
The two end nodes of a conflict edge should be in different banks

Register Conflict Graph (RCG)—Examples

Example 1

```
a=a+b
c=a+c
d=b+c
```

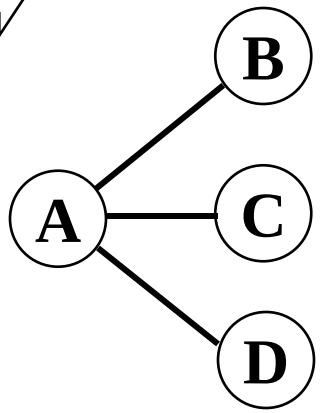
RCG



Example 2

```
b=a+b
c=a+c
d=a+d
```

RCG



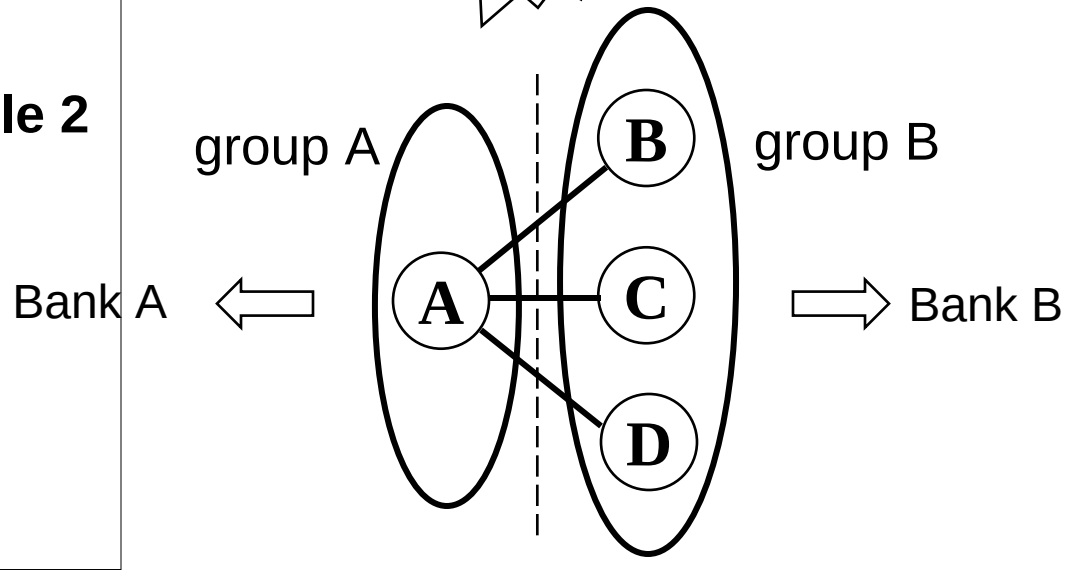
No-Conflict Law

No-Conflict Law:

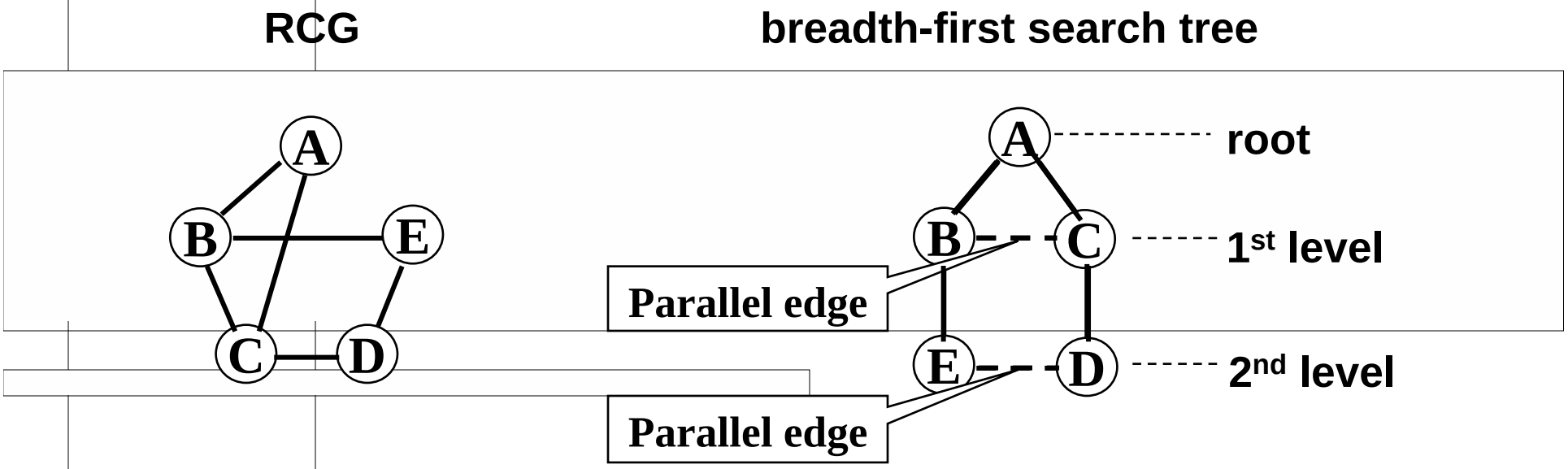
RCG is conflictless iff ***RCG is bipartite*** iff ***No odd-length cycles***

Example 1

Example 2



Detect Odd Cycles up to Certain Length



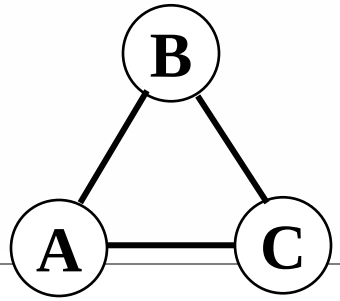
- Parallel Edge: edge between two nodes at the same level
- A level k parallel edge=>there is odd cycle of length up to 2k+1
- Each node should be a root once; complexity: O(|N|×(|N|+|E|))

Break Odd Cycles with Variable Splitting

code

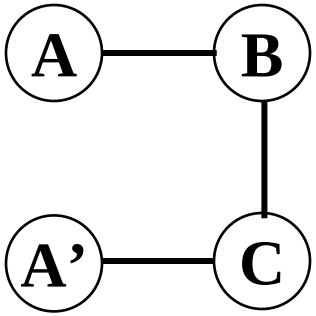
```
…=A op B
…=A op C
…=B op C
```

RCG



- **Inserting MOV can split "A" and break the cycle**
- **Cost: one MOV instruction**

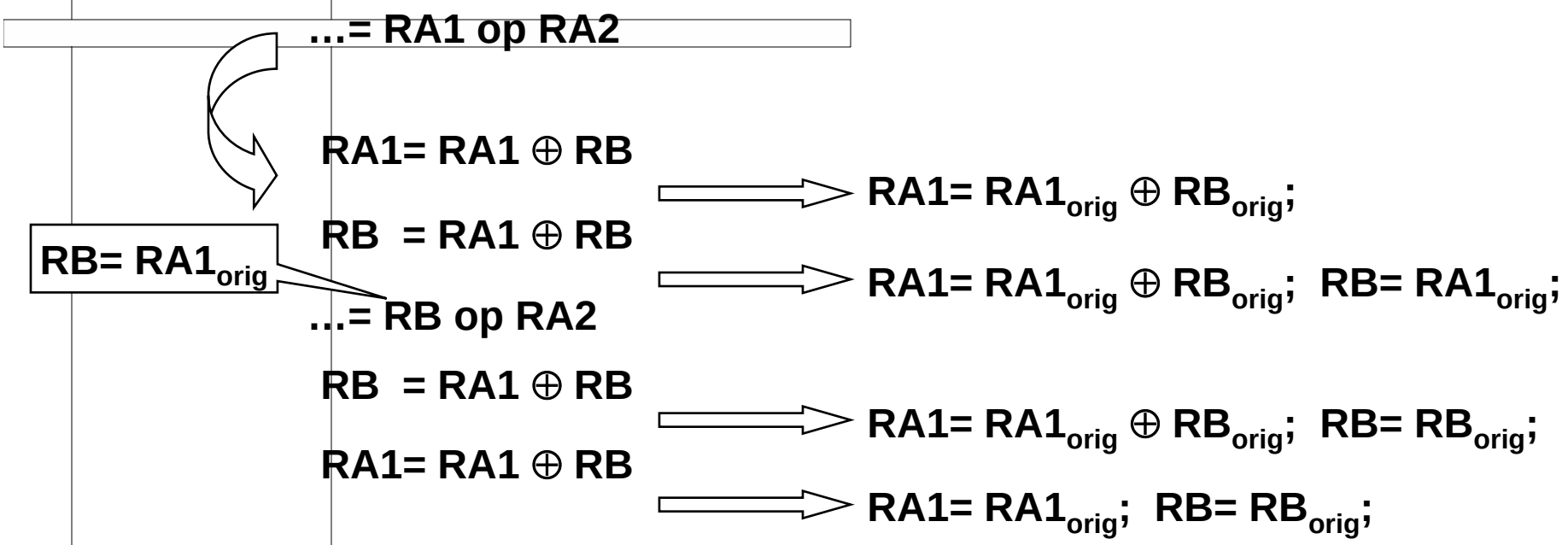
```
…=A op B
A'=A
…=A' op C
…=B op C
```



In-place Bank Exchange

- Require extra registers, which may not be available.

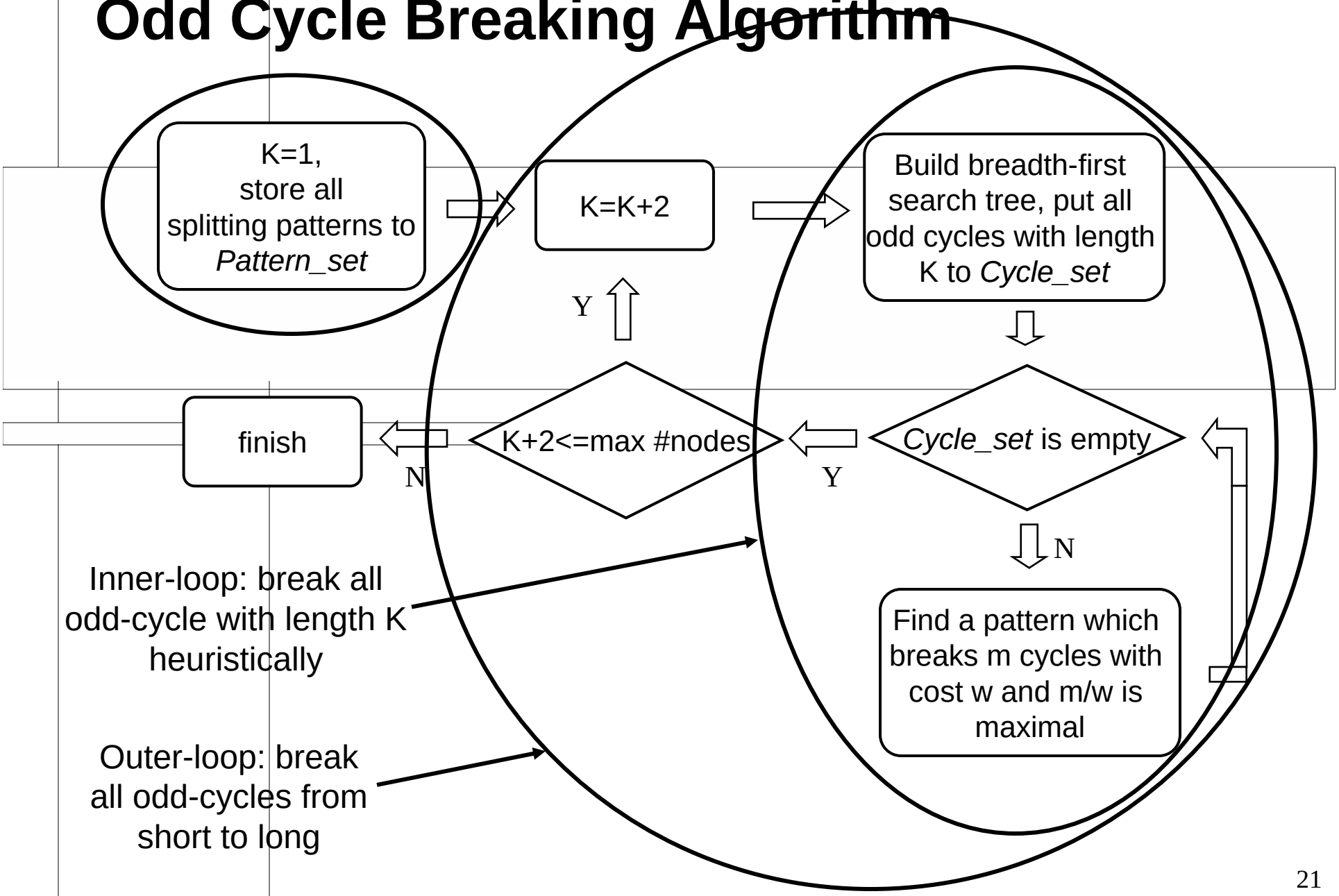
- Our approaches:
  - If no register, try to free one through rematerialization
  - Last resort: in-place bank exchange



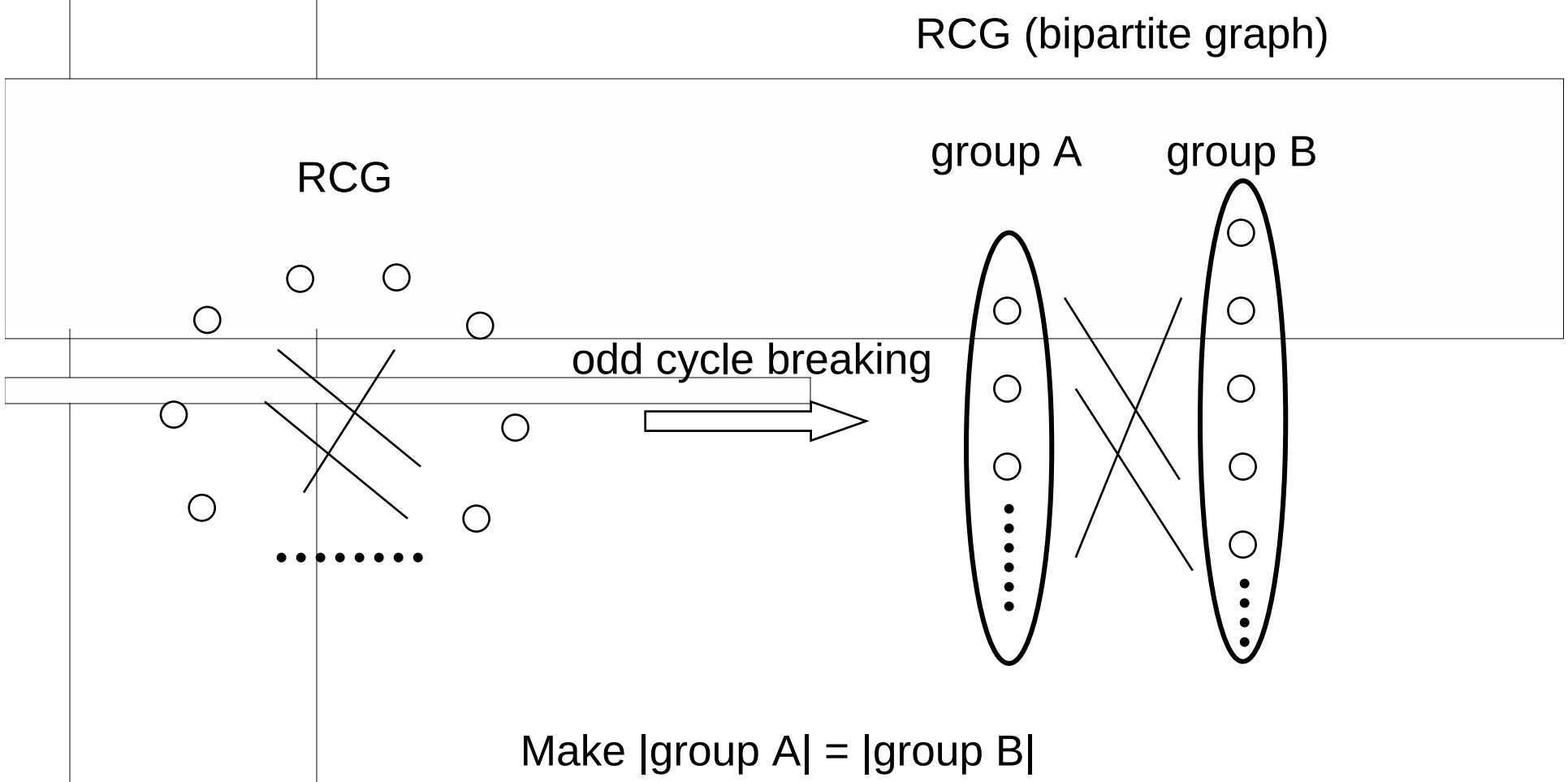
Breaking Odd Cycles

- Breaking odd cycles with minimal cost is very expensive (NP-complete)
- ILP solver—long compilation time
- A heuristic solution that gives good results quickly

Odd Cycle Breaking Algorithm



Bank Imbalance

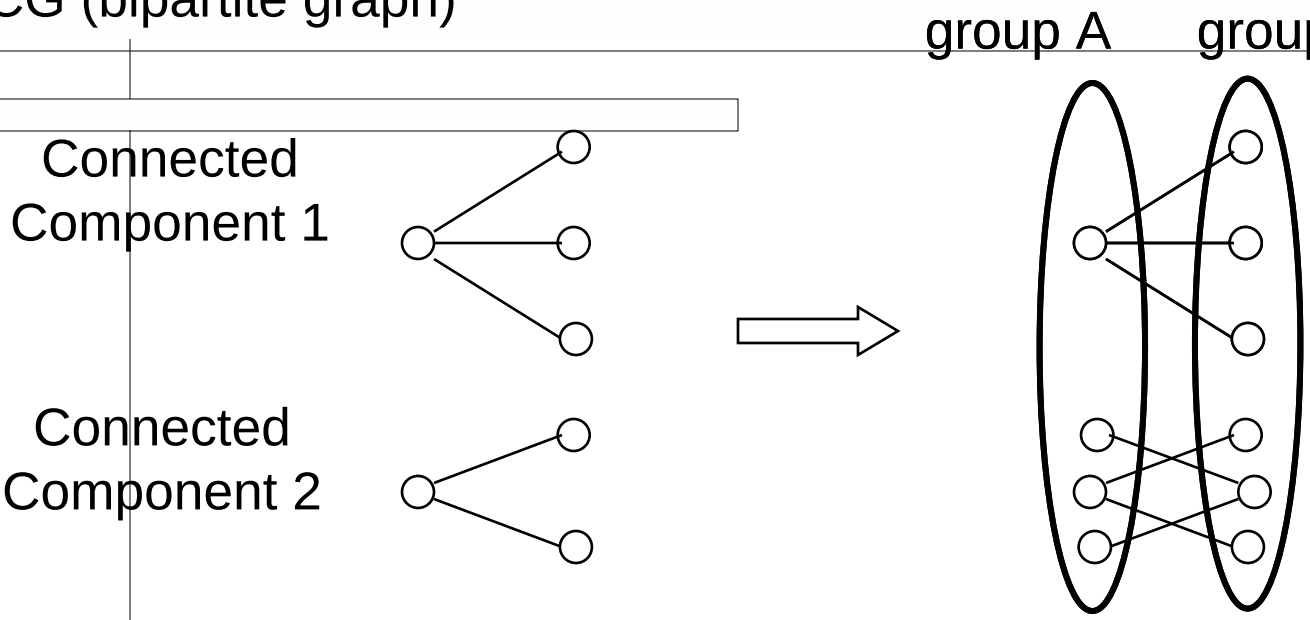


Make |group A| = |group B|

Near-Balancing

- **GOAL: roughly balance the two groups, zero cost !**
- **The graph is likely to be disconnected after cycle breaking.**
- **Each connected component must be bipartite !**

RCG (bipartite graph)



Solving the Balancing Problem

- Suppose the RCG contains m connected components

- The complexity of a naïve but optimal solution is $O(2^m)$, since each connected component could be "flipped" or "not flipped"

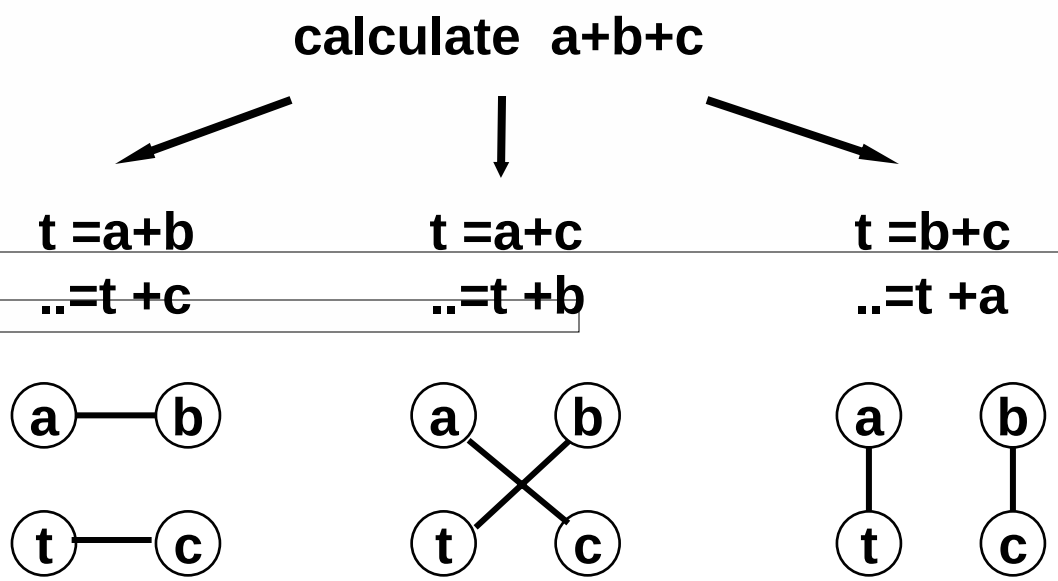
Solving:

For small m => exhaustive search $O(2^m)$

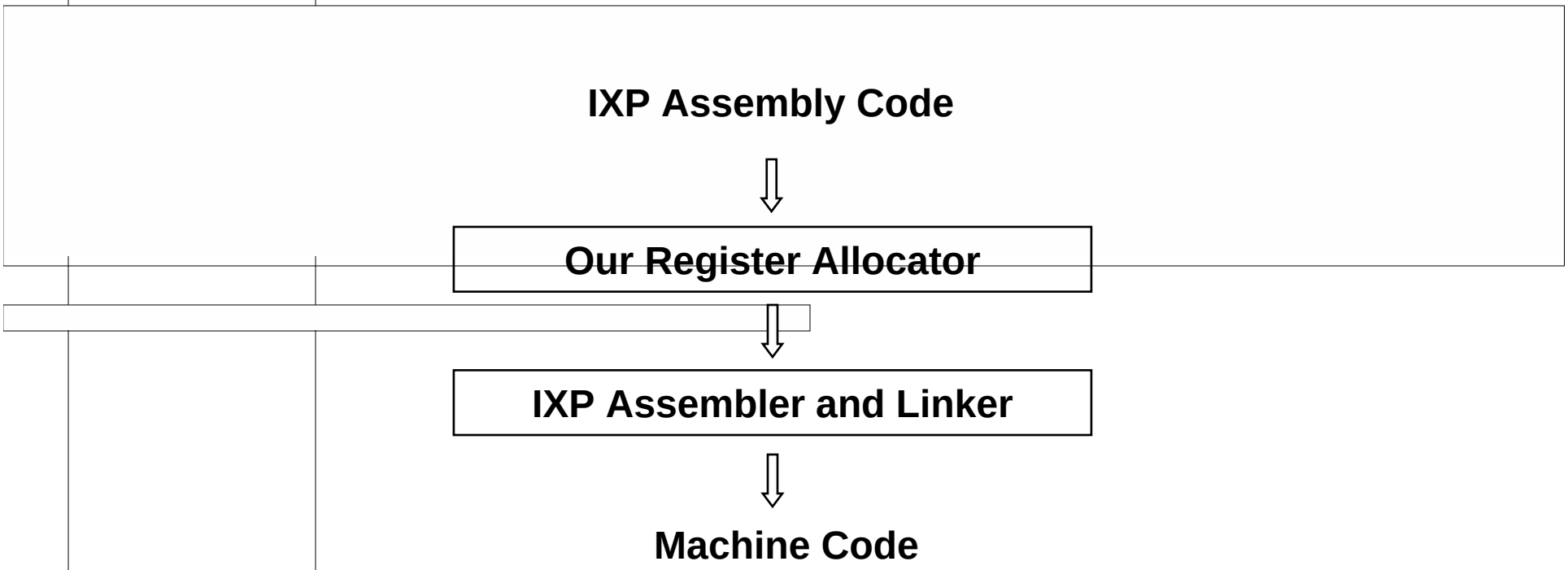
For large m => an approximate algorithm for "subset sum"

- Next, fully balance the two banks with a heuristic algorithm

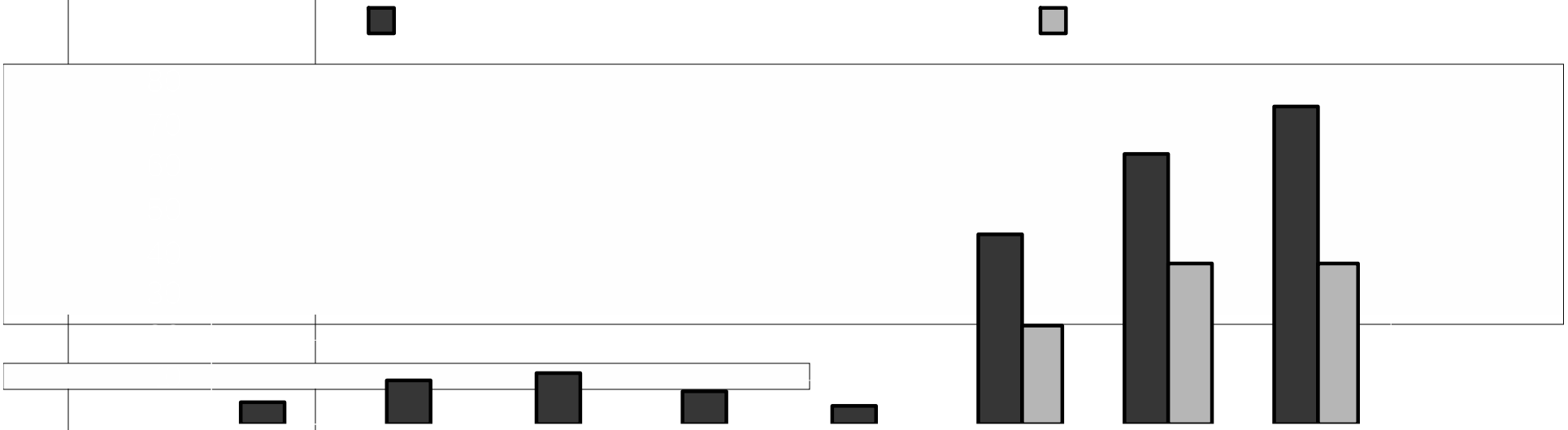
Application of Algebraic Laws



Compilation Flowchart

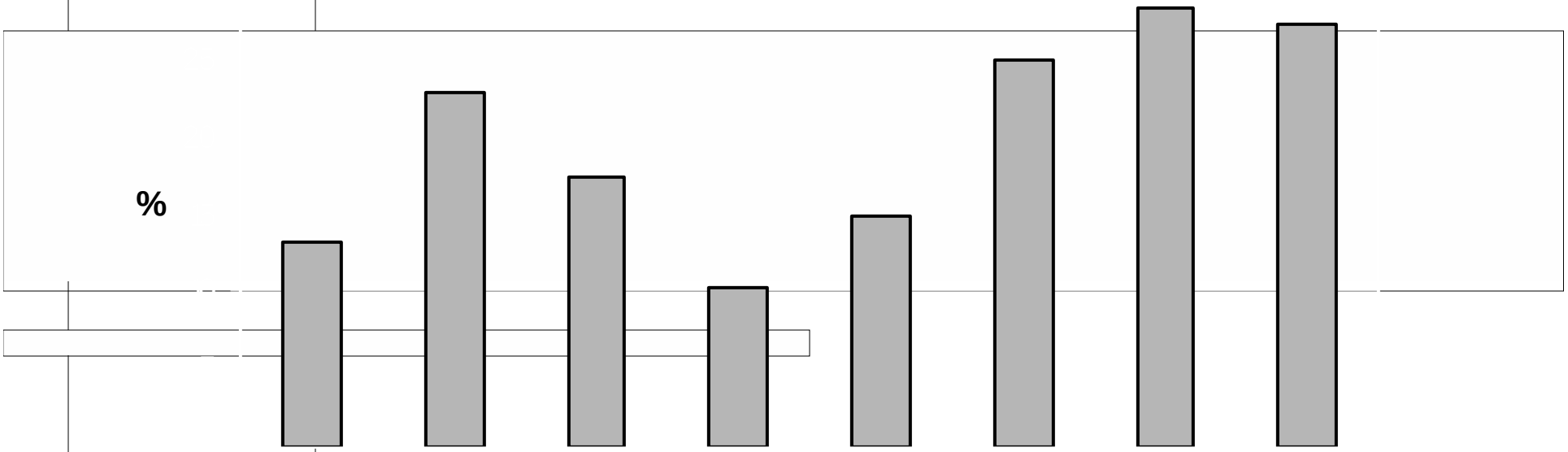


Comparison for Number of Spills (Memory Accesses)



- 70% reduction
- Completely avoid spills for 5 benchmarks

Speedup



- Average speedup: 20% purely through compiler optimizations
- Compilation time within a few seconds on a Pentium 4 machine

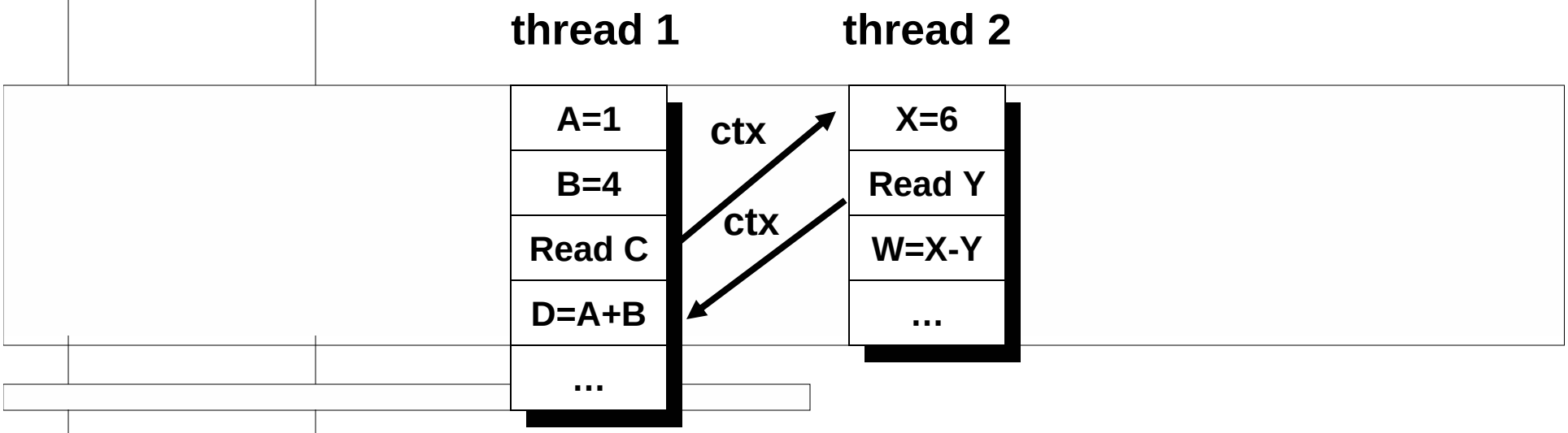
Contributions

- Tackled several hard problems with good, fast solutions
- Achieved 20% speedup through compiler optimizations
- First compiler solution to overcome the dual bank constraint
- Published in PACT-03. This work was later integrated into Intel's new compiler

Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Lightweight Context Switch



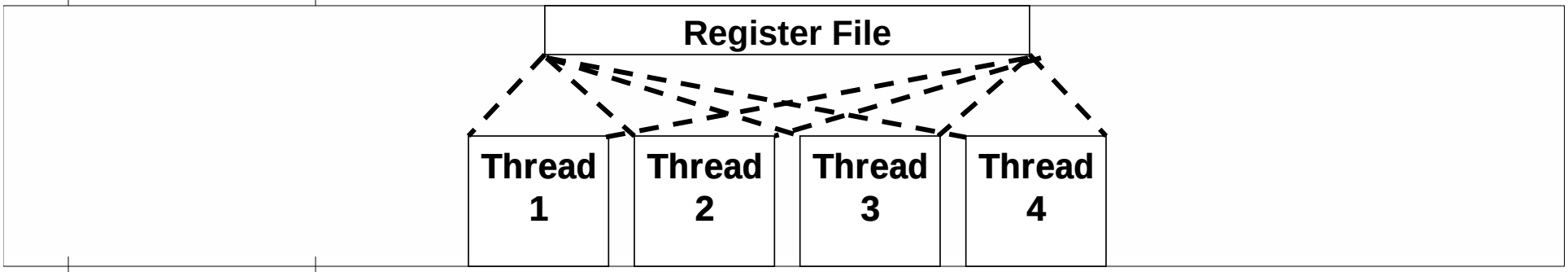
- Context switch only happens for long latency instructions, highly frequent – every 20 cycles
- Thread execution is non-preemptive, predictable; threads are simultaneously active

⇩

1 cycle context switch: only PC is saved

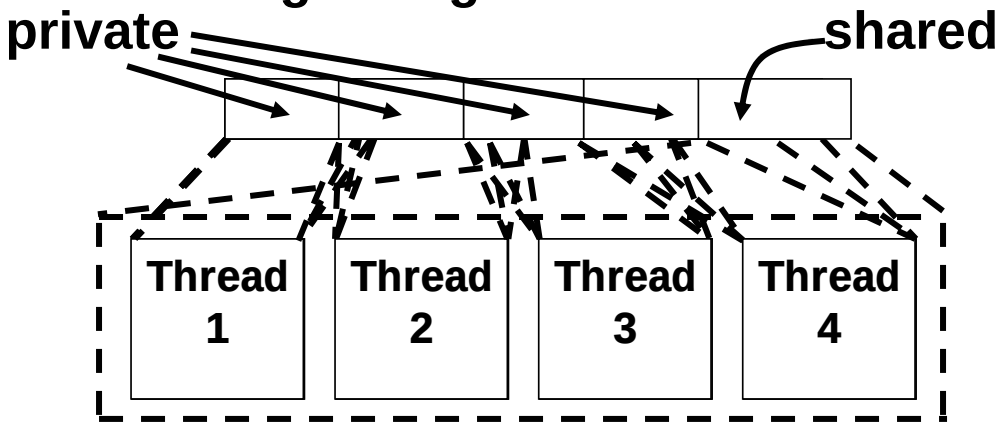
Register Sharing

with traditional context switch



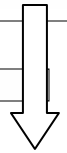
Our approach

with lightweight context switch



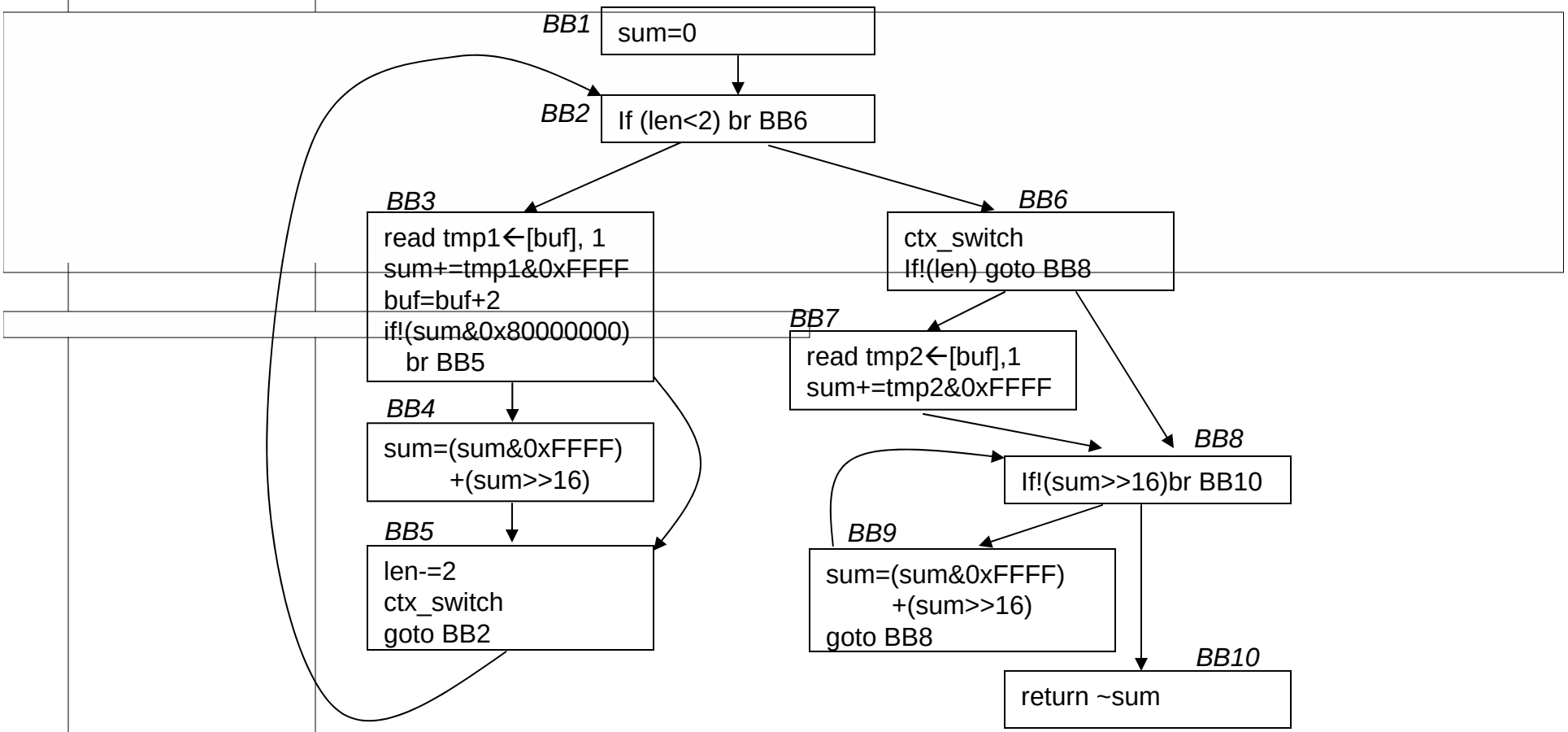
What to Put in Shared Registers ?

Variables in shared registers must not be used across context switches. Upon context switch, they should already be dead i.e. unused.

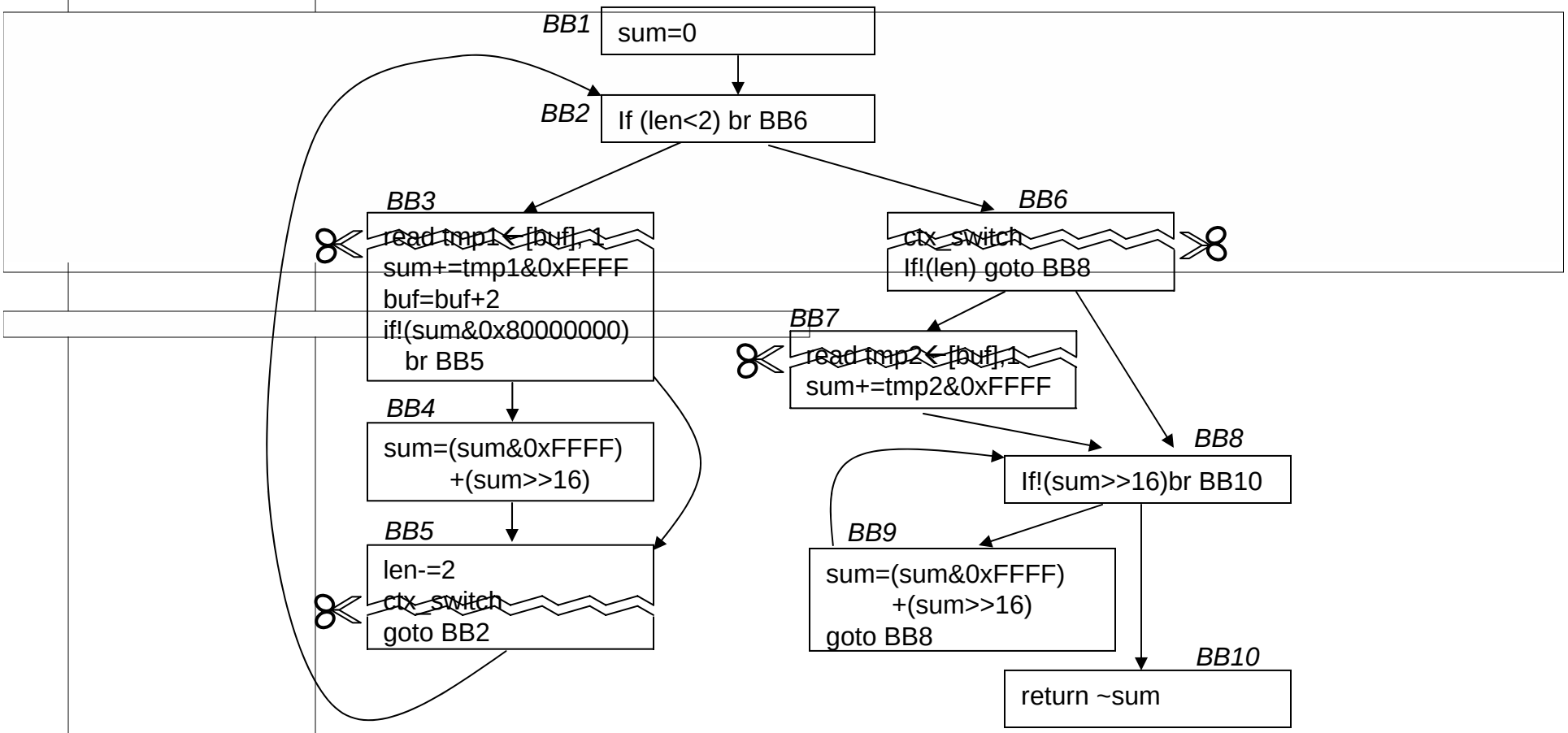


Categorize variables into two types: those used across context switches, and those are not; Allocate them separately.

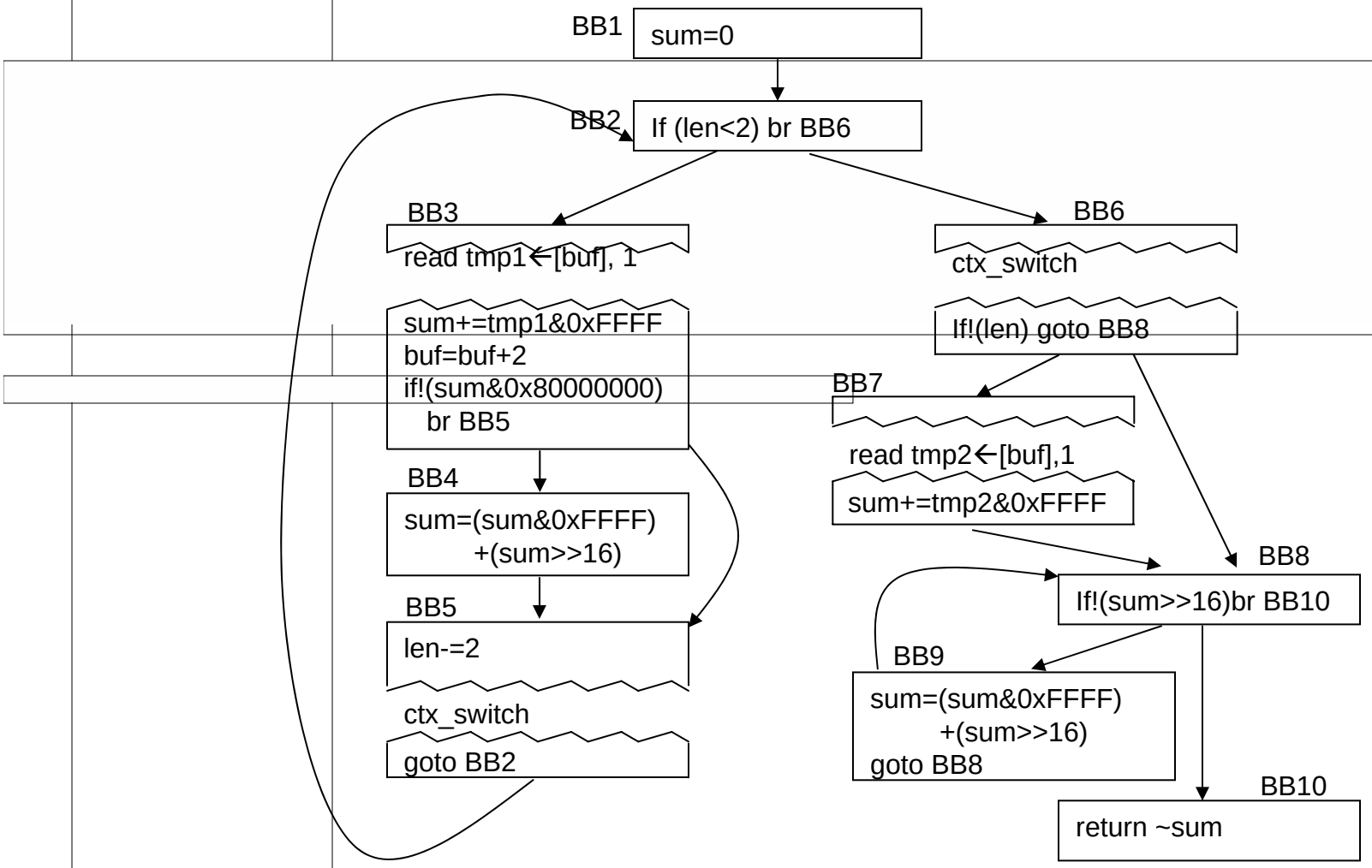
Non-Switch Region (NSR)--Commbench



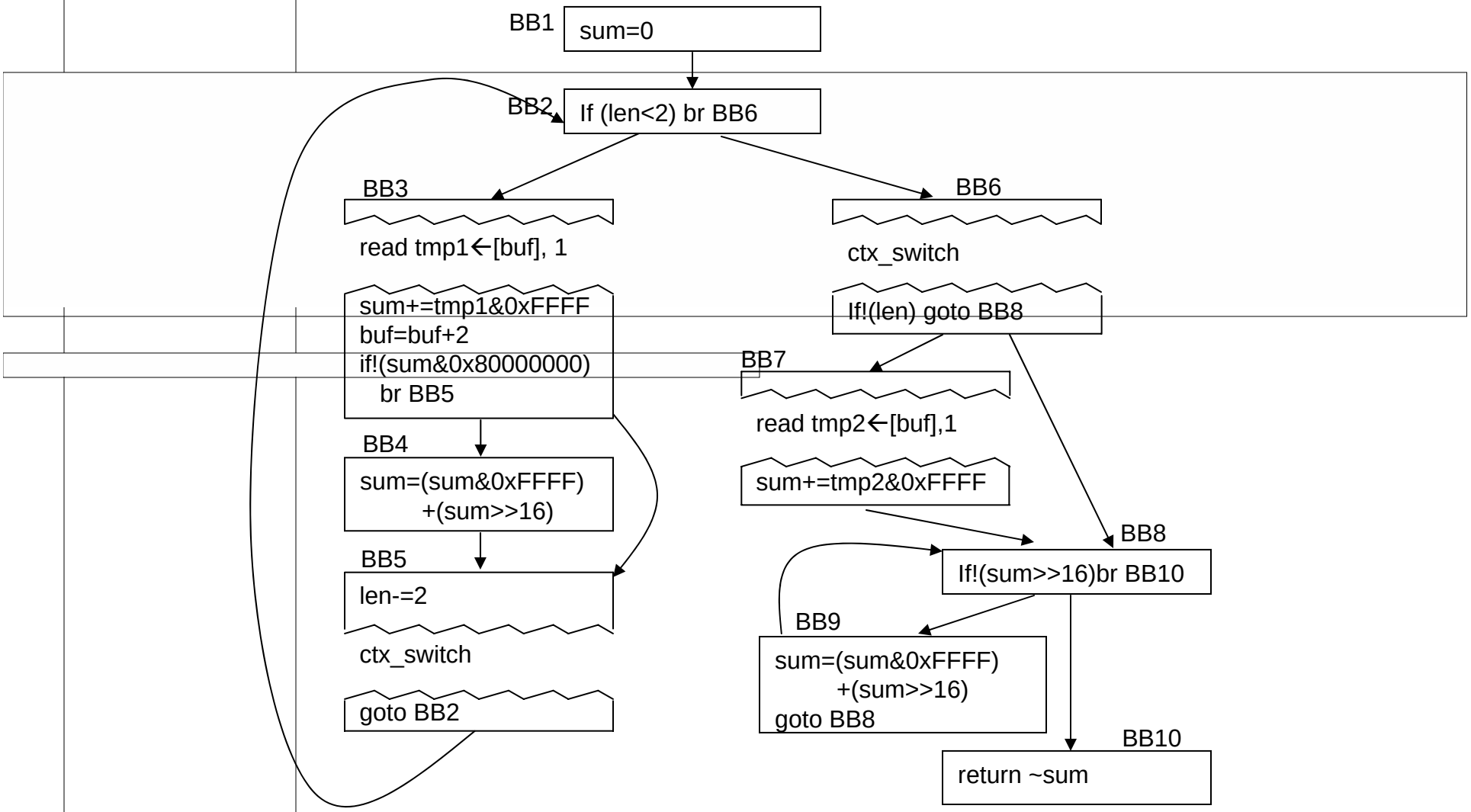
Non-Switch Region (NSR)--Commbench



Non-Switch Region (NSR)--Commbench

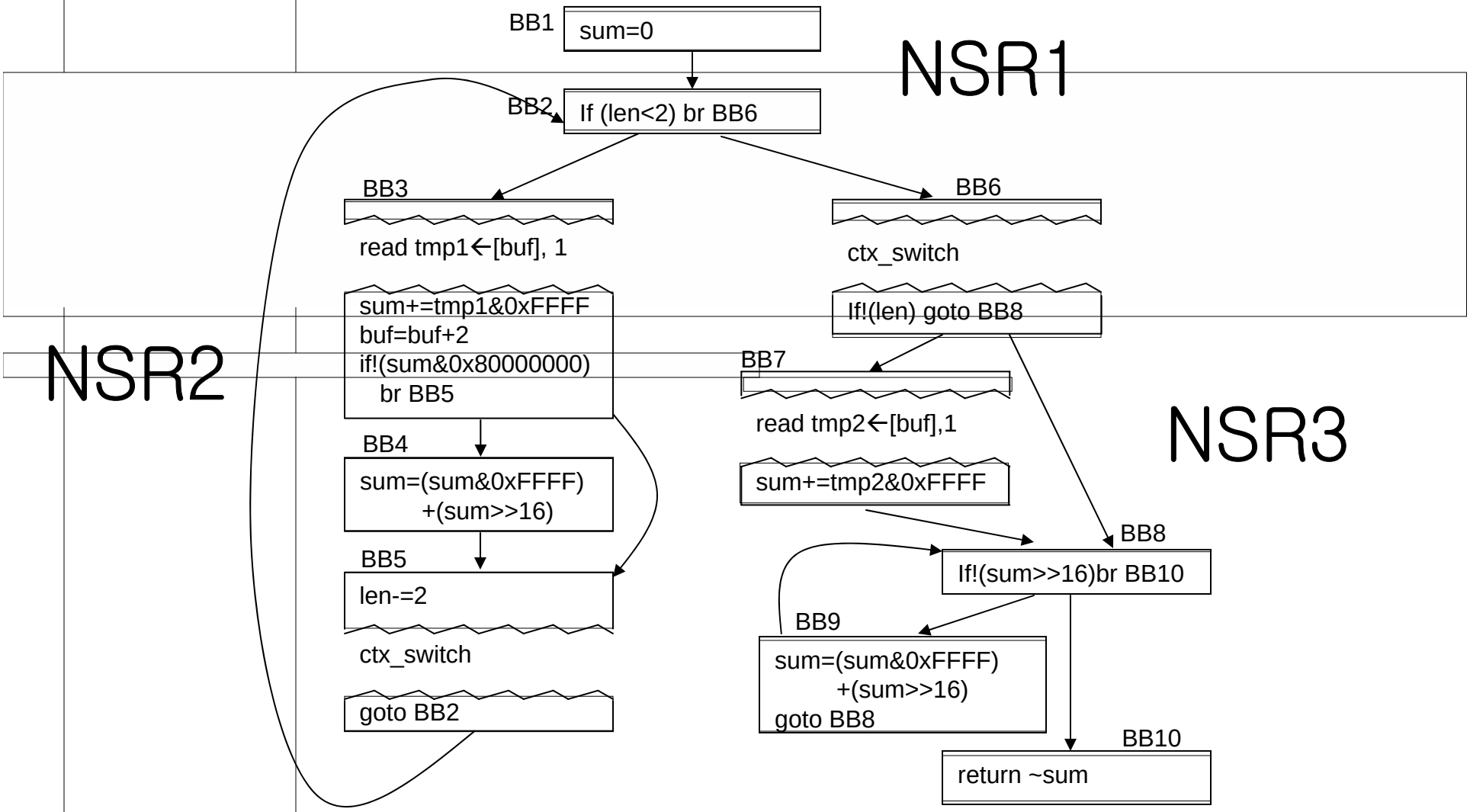


Non-Switch Region (NSR)--Commbench

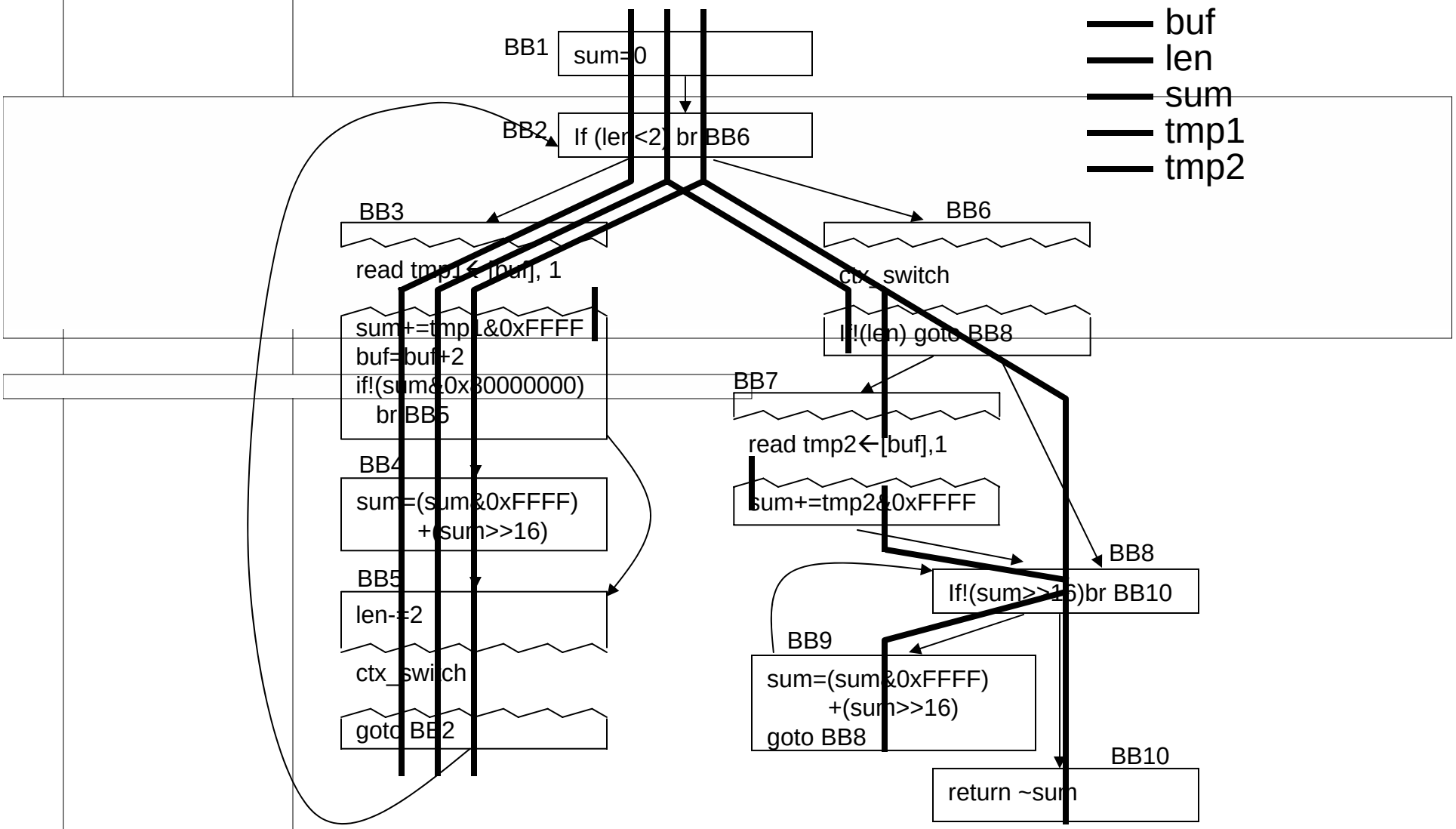


Non-Switch Region (NSR)--Commbench

Each Connected Component Form a NSR

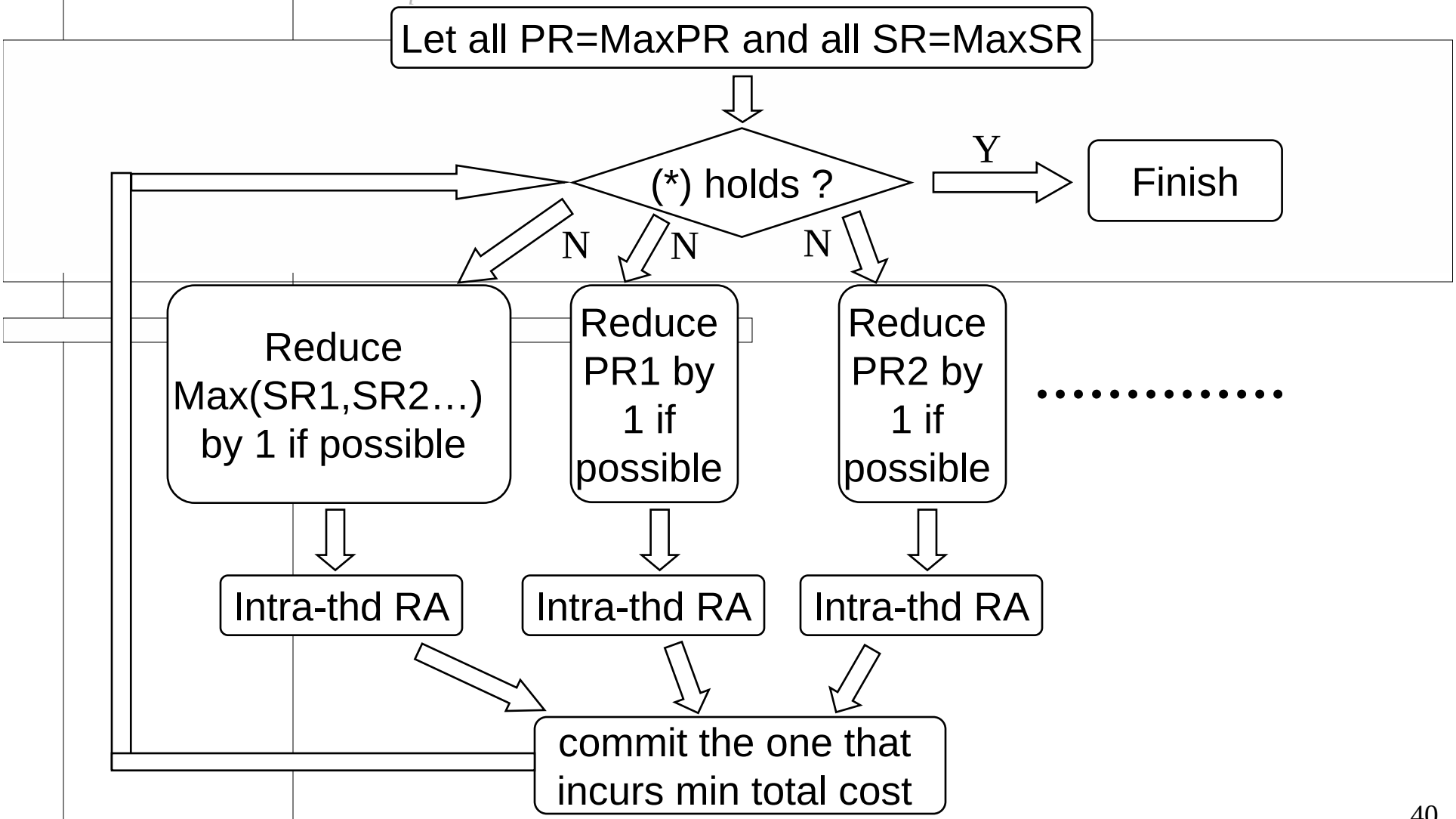


Variable Classification

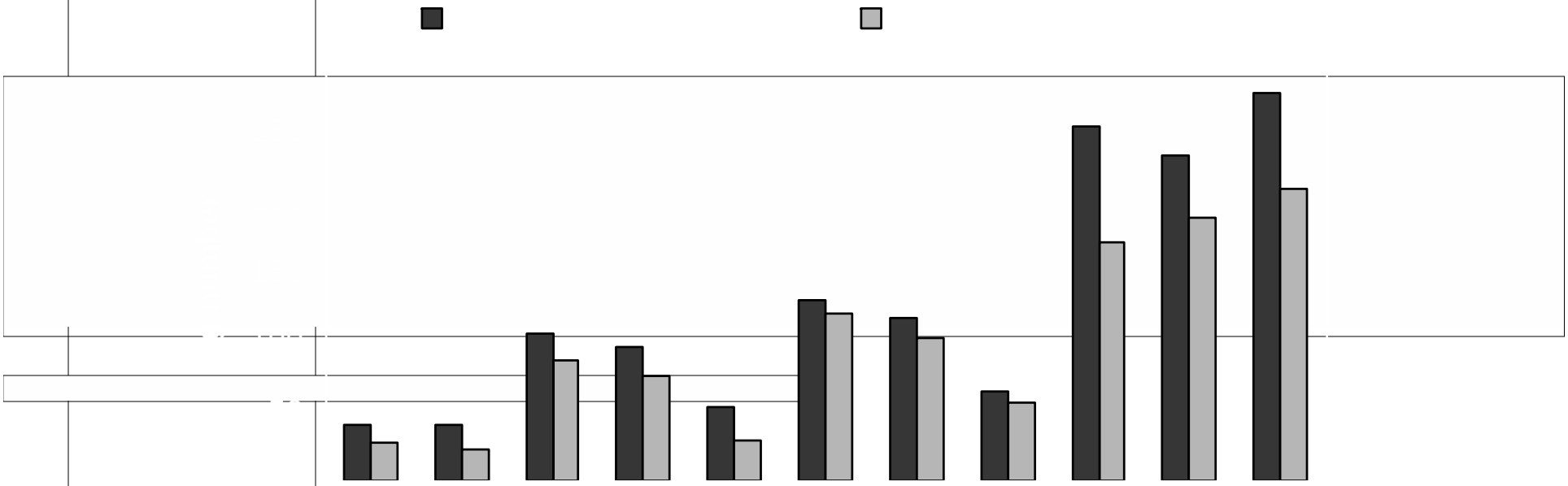


Inter-thread Register Management

$$\sum_i PR_i + Max(SR_1, SR_2 ... SR_{N_{thd}}) \le N_{reg} \quad (*)$$

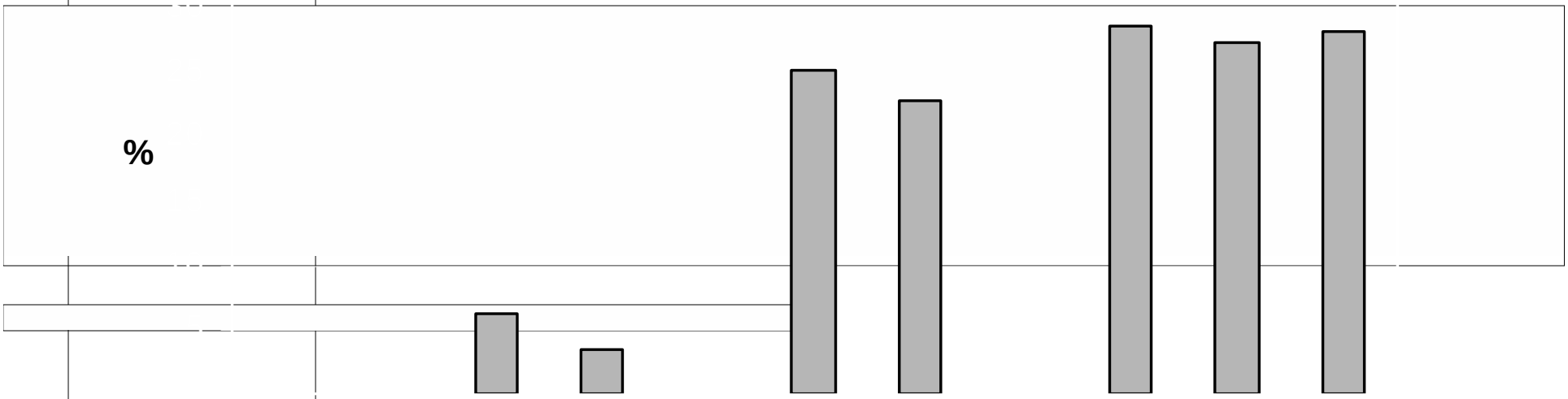


Register Pressure Reduction



- Four threads with identical code
- 24% saving on the number of registers needed

Speedup



- 128 physical registers
- Speedup up to 29%
- Average 20%

Contributions

- Partially sharing registers among threads alleviates registers shortage
- Combined with intra-thread allocation, it gives us around 40% speedup
- Published in PLDI-04, later integrated into Intel's new compiler
- Our recent work on IPDPS-06 adds hardware modifications to achieve more sharing

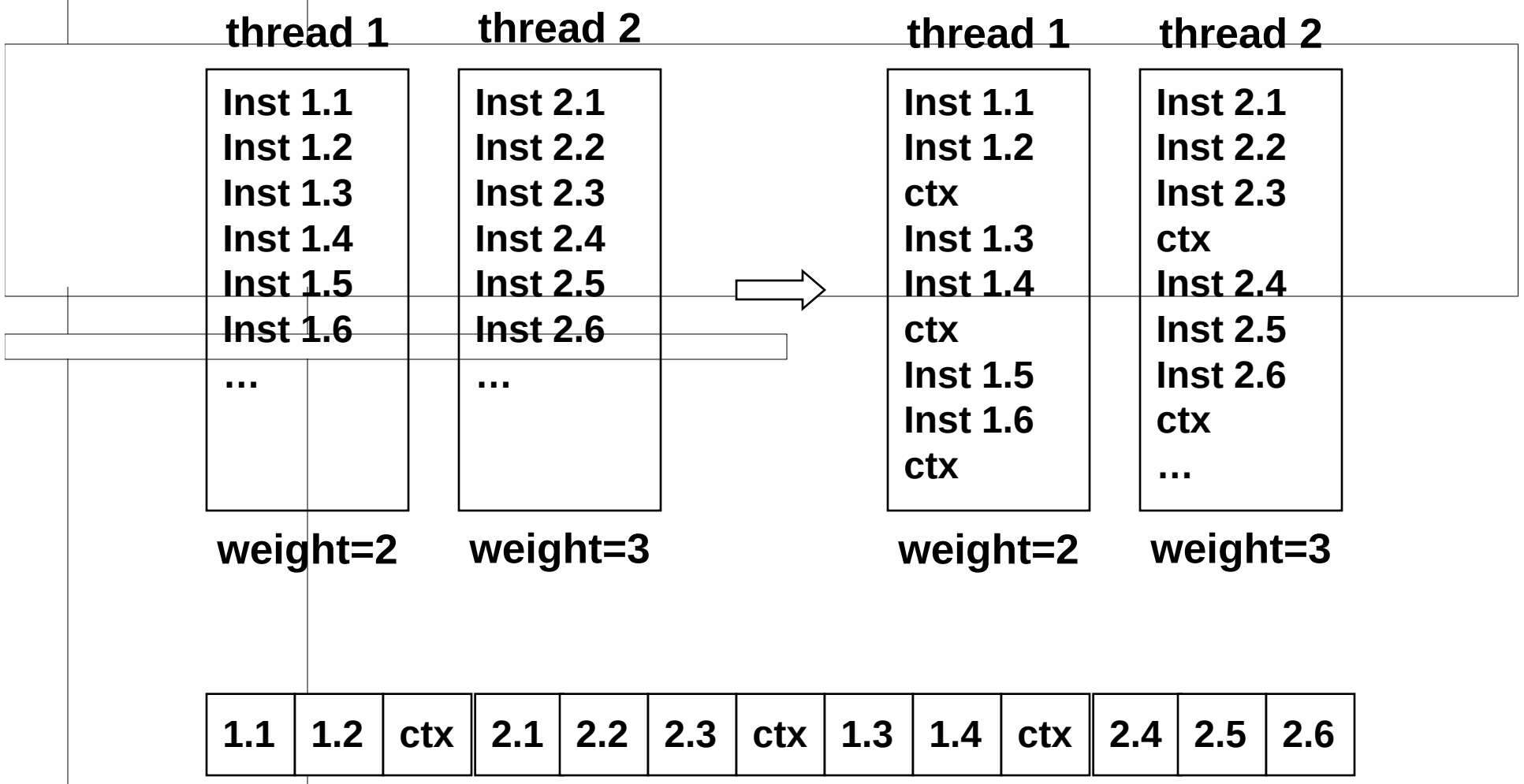
Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Motivation

- CPU cycle wastage (20-30%) due to unnecessary stalls
- Need for better CPU sharing, some threads take more CPU due to less long latency instructions
- Real-time scheduling, packet scheduling

Example — Weighted Round Robin



Main Results

Category	*Constraint*	*Approach*
CPU Scheduling	(Weighted) Round Robin—(W)RR	FCS
	Priority Sharing—PS	FCS
Real-time Scheduling	Rate Monotonic—RM	FCS
	Earliest Deadline First—EDF	DCS
Packet Scheduling	Priority Class—PC	FCS
	First Come First Serve—FCFS	DCS
	(Weighted)Fair Queueing—(W)FQ	FCS

- **Up to 2% slowdown**
- **Code growth <5%**
- **Eliminate unnecessary stalls, 20-30% improvement on CPU utilization**

Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Other Compiler Work

- Parallelize load/store instructions [PACT-02] journal version [ACM TECS]
- Auto addressing mode [LCTES-03]
- Manage hidden registers on ARM [LCTES-04]
- Lower power prefetching [LCTES-04], journal version [ACM TECS]

Other Compiler Work

- Differential encoding and register allocation [PLDI-05], journal version [ACM TOPLAS]
- Compiler scheduling of mobile code in a distributed data intensive environment [ICDCS-03]
- Profile-driven optimizations for server applications [PLDI-06]
- Current project at IBM Research: speculative parallelization for Blue Gene/Q and Power 7

Optimization for Security

- Prevent information leakage through the address bus for secure processors [ASPLOS-04] [CASES-04 Best Paper]
  - Address bus information leakage is a severe problem
  - Propose two solutions to remedy it

- Reduce security overhead, improve security strength through compiler/hardware approaches [CGO-06]
  - Apply to secret sharing [CGO-05]
  - Apply to anomaly detection [MICRO-06] [CASES-05]

Some Other Work

- A highly scalable priority queue [IEEE INFOCOM-06]
- Reduce cache pollution via prefetch filtering [ICPP-03], journal revision [IEEE TOC]
- Low latency broadcasting in massive parallel computers [IPDPS-02], journal version [IEEE TPDS]

Agenda

- Overview of My Research
- Processor Model
- Dual-bank Register Allocation
- Inter-thread Register Sharing
- Thread Management
- Other Work
- Future Plan

Multicore

The number of cores will double every 18 months, with 256-core systems commonplace by 2011

—Anant Agarwal, MIT

- **Program partitioning, speculative parallelization**
- **Aggressive speculation with multiple parameterized compilation versions**
- **On-chip memory organization and data layout optimizations**
- **Compiler scheduling for power and temperature management**

Specialization

- Applications in special domains: multimedia, scientific computing, simulation, physics, chemistry, bio-informatics
  - Specially designed hardware
  - Heterogeneous multicore

- Hybrid optimization according to runtime conditions
  - Compiler generates rough optimization strategies
  - Runtime system fills in the details

Security

- Automatic patch generation for large-scale zero-day worms
  - Record forensic data w/ hardware support
  - Compiler analysis for worm code and system source code
  - Generate the patch automatically
- Compiler/architectural approaches for fast identification of malicious inputs

Questions & Answers

That's All Folks !