Limits on Speculative Module-level Parallelism in Imperative and Object-oriented Programs on CMP Platforms

Fredrik Warg and Per Stenström
Department of Computer Engineering
Chalmers University of Technology

Motivation/Objectives

- Module-level parallelism (MLP)
  - Targets function/method/procedure calls
  - Easy to identify modules
  - No inter-thread control dependences
  - How much speedup can you get with only MLP?
  - Is MLP a good match for a CMP or multithreaded core?
- Imperative vs. object-oriented programs
  - More frequent use of modules in OO codes
  - Does data encapsulation mean fewer dependences on global data?

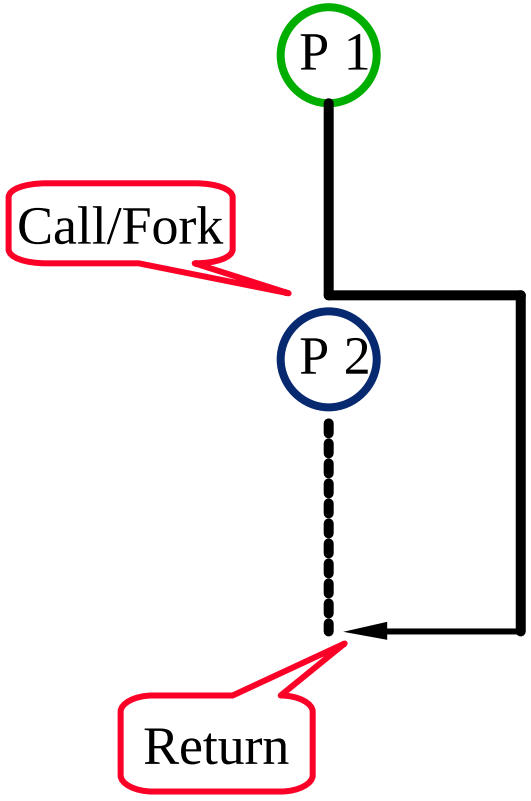
Results

- No significant difference in parallelism between imperative and object-oriented programs
- Inherent module-level parallelism typically below 4 $\Rightarrow$ CMP or multithreaded core sufficient
- Module-level speculation is very sensitive to speculation overheads $\Rightarrow$ low-overhead (hardware) support needed to exploit MLP
- A simple return value predictor does fairly well

Outline

- Methodology
  - Execution & architectural models
  - Experimental setup
- Experiments
  - Inherent MLP
  - Data dependences & value prediction
  - Processing resources and speculation overhead
- Conclusions

MLP Execution Model



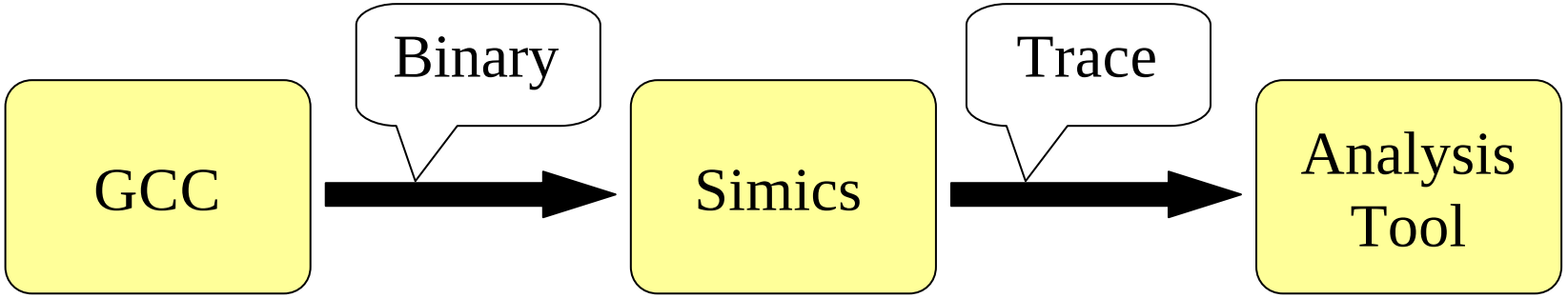
- At every call
  - Create new thread for continuation
  - New thread is more speculative than old
- At every return
  - Thread is finished. Commit if least speculative.

Architectural Models

- Ideal machine
  - Single-issue in-order processor
  - Perfect memory system (1 cycle access)
  - No communication or context switch overhead
  - **Perfect prediction on data and return values**
  - **Unlimited processing resources**
  - **No speculation overhead**

We will look at these!

Experimental Setup



- The compiler produces annotated binaries of the C and Java programs
- A system-level instruction-set simulator runs the programs sequentially and generates traces
- A custom analysis tool is used to simulate parallel speculative execution

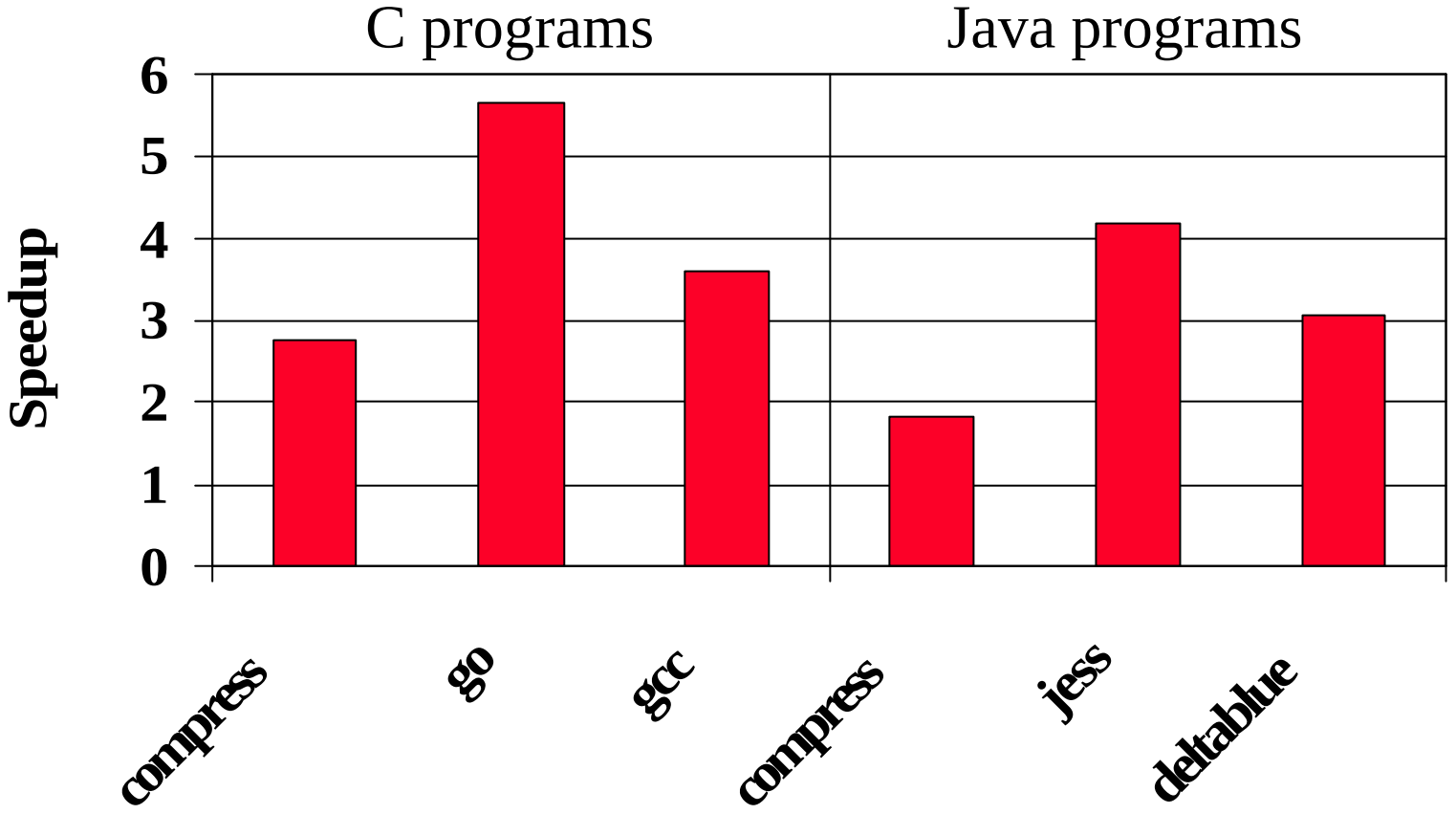
Benchmarks

Name	Description	#Mod.
C Applications		
Compress	Unix compression util.	21k
Go	Plays the game of go	1.1k
M88ksim	A chip simulator	0.5k
Gcc	GNU C Compiler	54.5k
Java Applications		
Db	Simple database	4.9k
Compress	Compress Java version	31.5k
Jess	Expert system	25.8k
Neuralnet	Neural network	2.6k
Idea	Encryption/decryption	12k
Deltablue	Constraint solver	12.5k

Inherent MLP

- Purpose: Finding the upper bound for MLP
- Architectural model:
  - Ideal machine

Inherent MLP

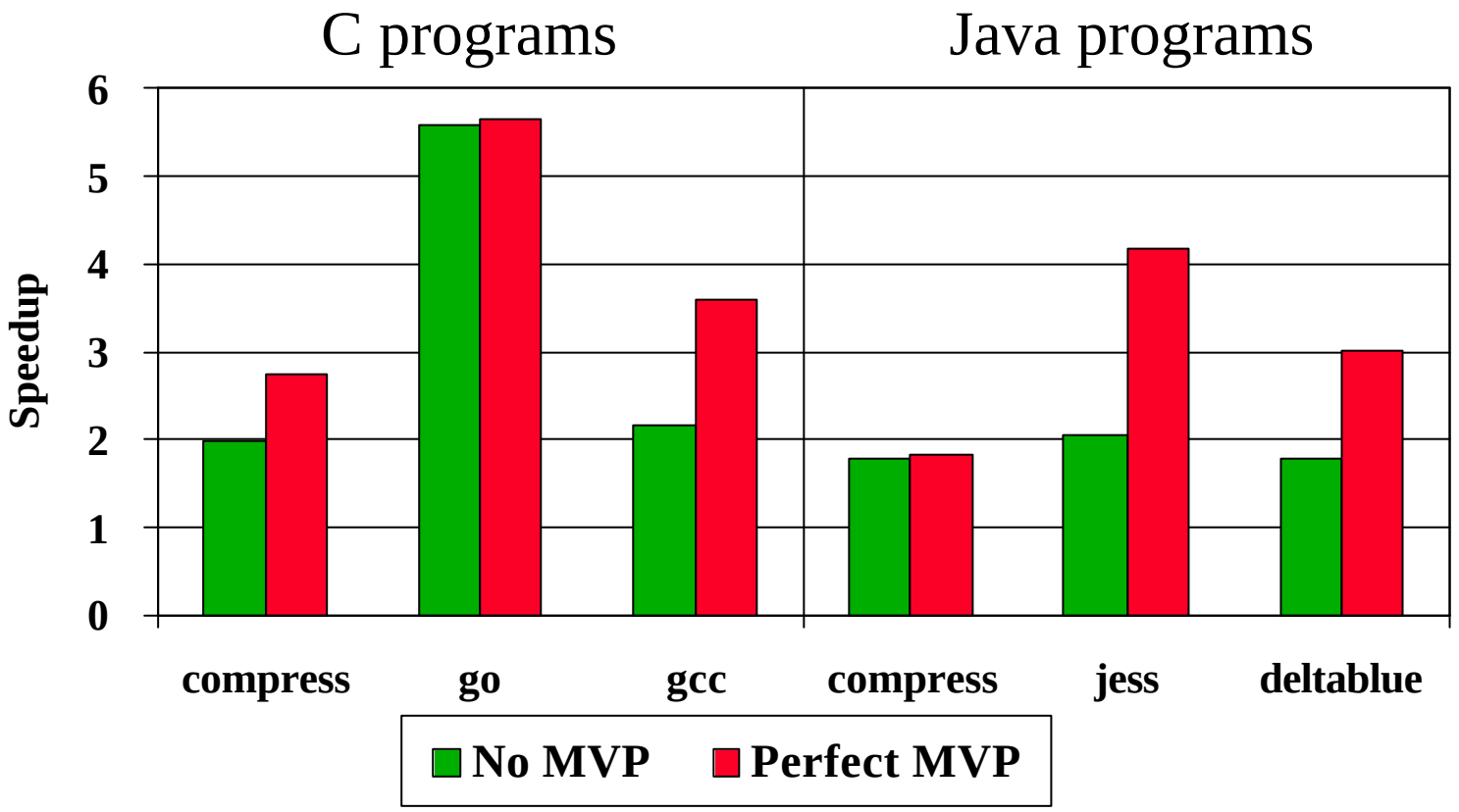


- **MLP can typically yield a speedup of at best 2-6**

Data Dependences

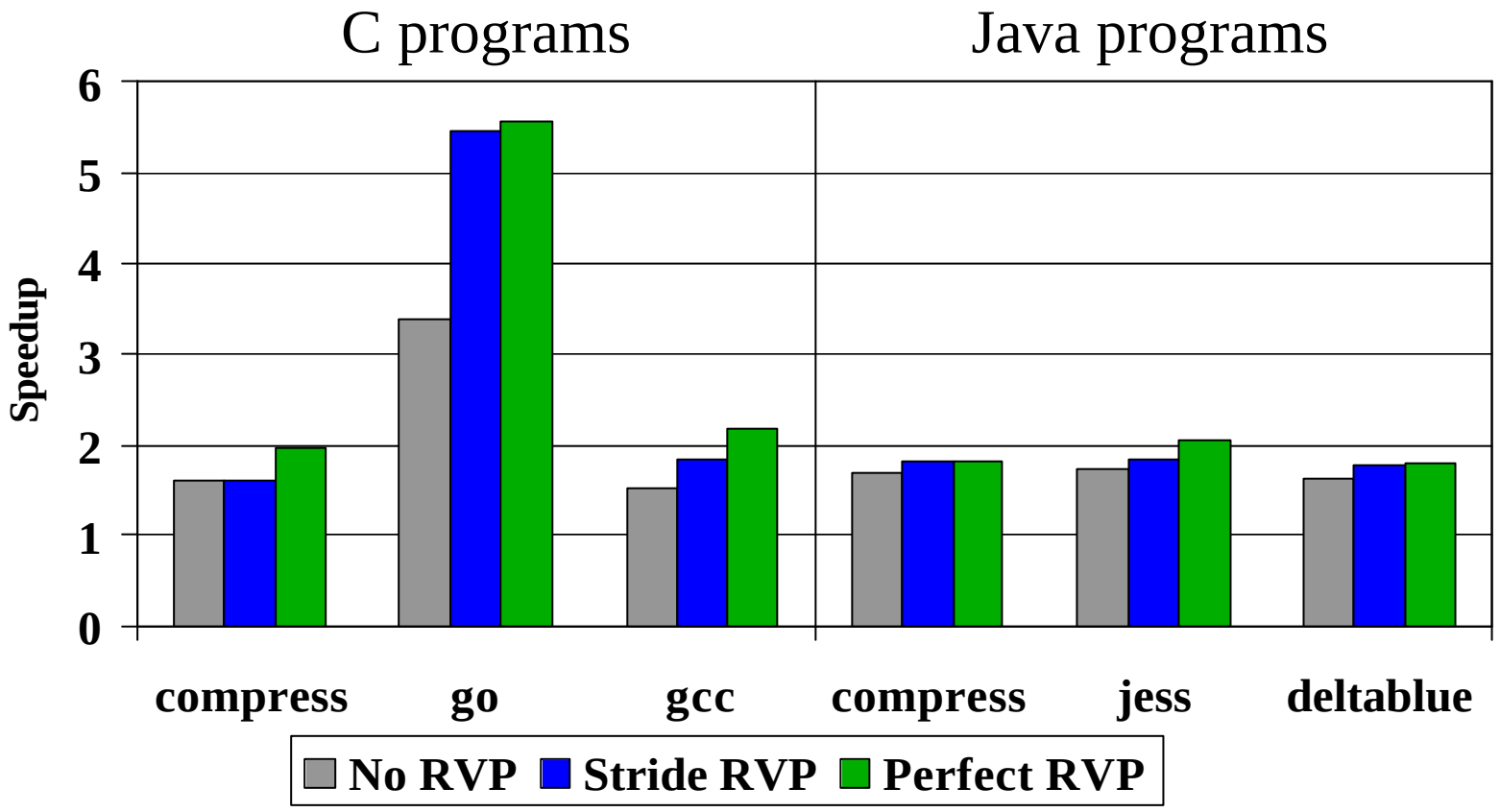
- Purpose: How much data dependences limit speedup, and how value prediction can help
- Change from ideal machine model:
  - Perfect or no memory load value prediction (MVP)
  - Perfect/stride/no return value prediction (RVP)

Data Dependences



- **Data dependences common in OO and imperative apps.**

Return Value Prediction

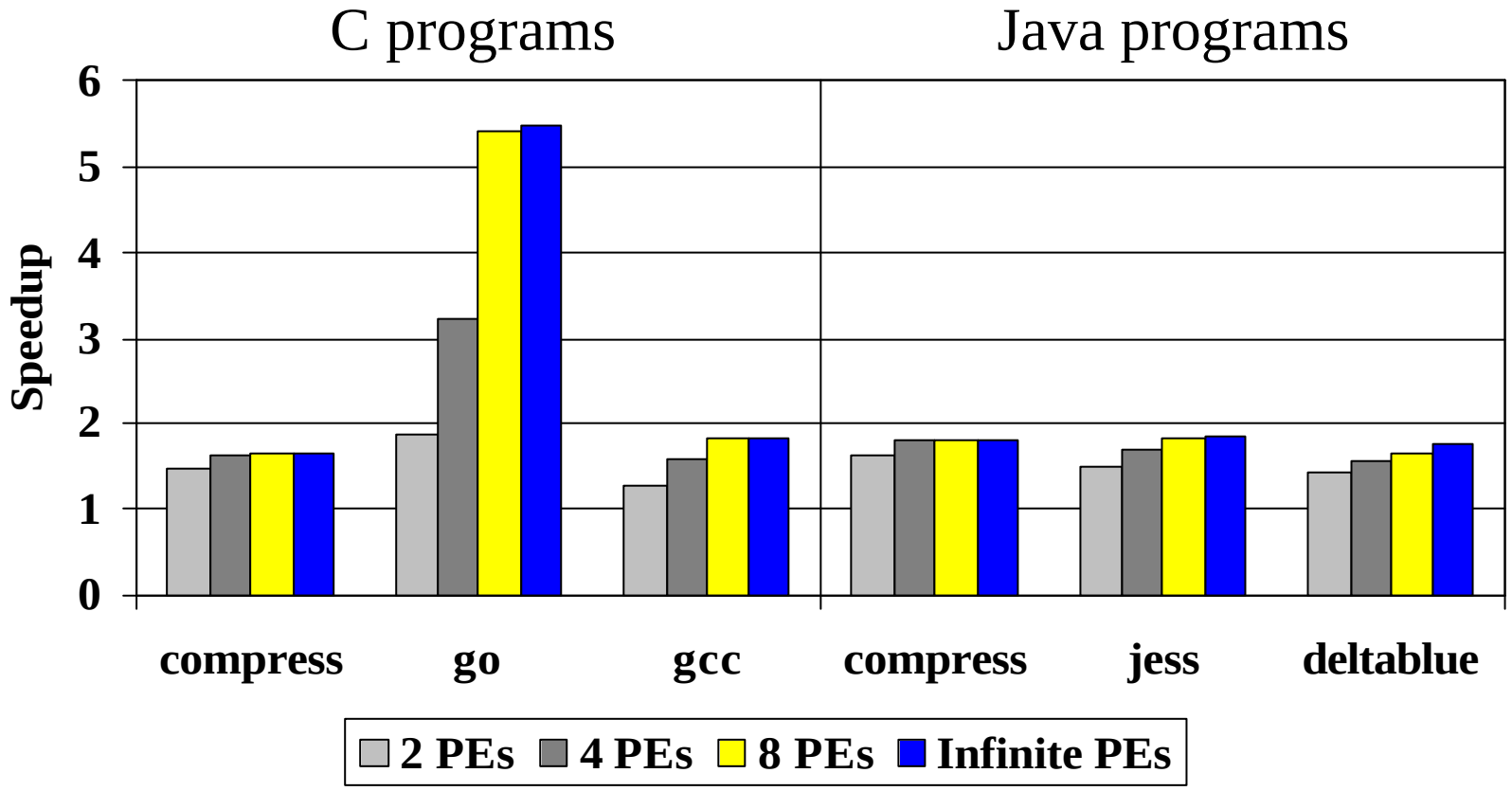


- **A simple stride value predictor does fairly well**

Processing Resources & Overhead

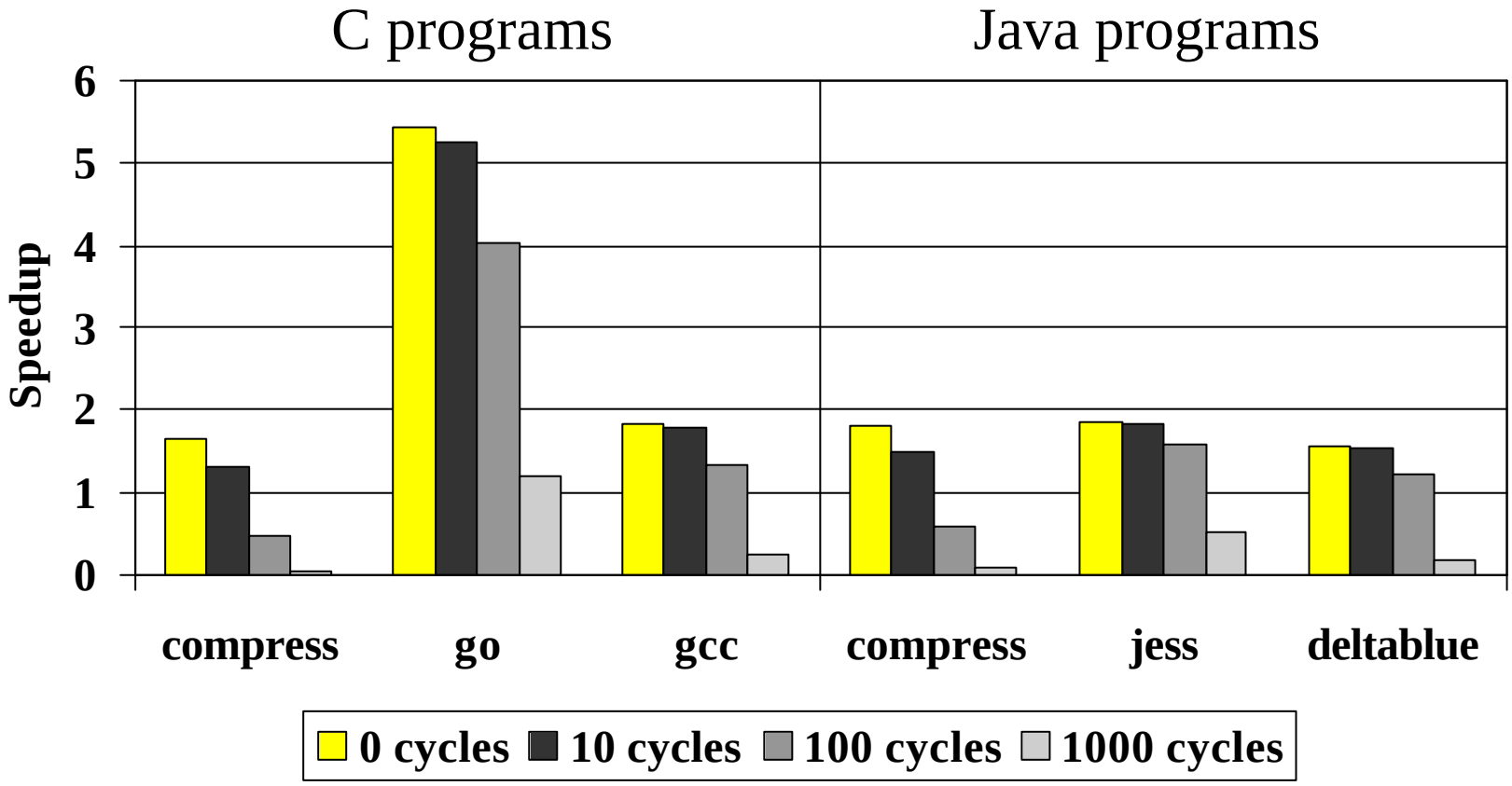
- Limited processing resources
  - Less speculative threads prioritized
  - Threads can be preempted
- Thread-management overhead
  - Extra cycles added for speculation operations: thread start, rollback on misspeculation, and commit

Limited Processing Resources



- **Four PEs are typically enough to exploit available MLP**

Thread-management Overhead



- **Speculation overheads need to be low (<100 cycles)**

Conclusions

- Module-level parallelism is a possible source of parallelism for small chip multiprocessor or multithreaded cores, provided that low-overhead speculation support is implemented
- Module-level parallelism is limited in object-oriented as well as imperative programs, additional sources of parallelism are needed to achieve large performance gains