An Empirical Analysis of Instruction Repetition

Avinash Sodani

Guri Sohi

Computer Science Department

University of Wisconsin-Madison

Oct 5, 1998

Instruction Repetition

Phenomenon

- Instruction executes with **same inputs**

  → produces **same output** as one of its earlier instances

Example

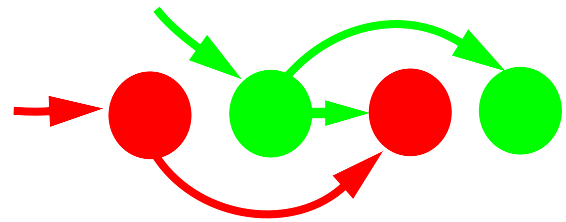
- **Search** function with 2 arguments (**Key**, **List**)
- Called to search different **Key**s in same **List**

  → **List** access and traversal **repeated**

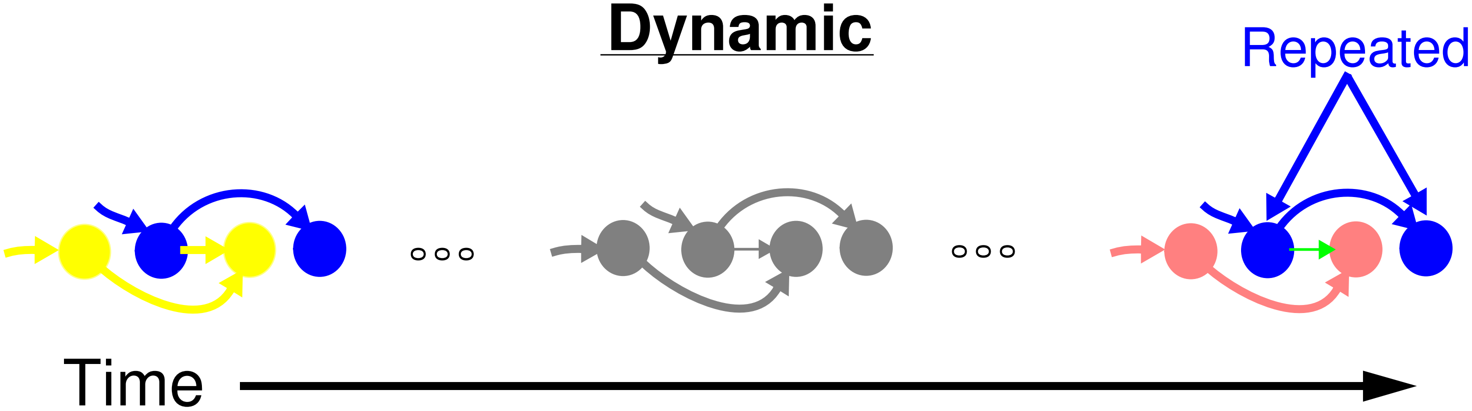
  (same input, same output)

Illustration

Static



Dynamic



Why Do I Care?

Repetition exploited to improve performance

Software

- Dynamic optimizations
    - Eliminate repetition: e.g., code specialization

Hardware

- Instruction Reuse
    - Splice out repeated computation from critical path
- Value Prediction
    - Predict repeated values → break critical path

To exploit better → Understand the causes better

BUT, IS IT JUST A BAD COMPILER?

No! It's Not

Then, what is it?

Goal of this work

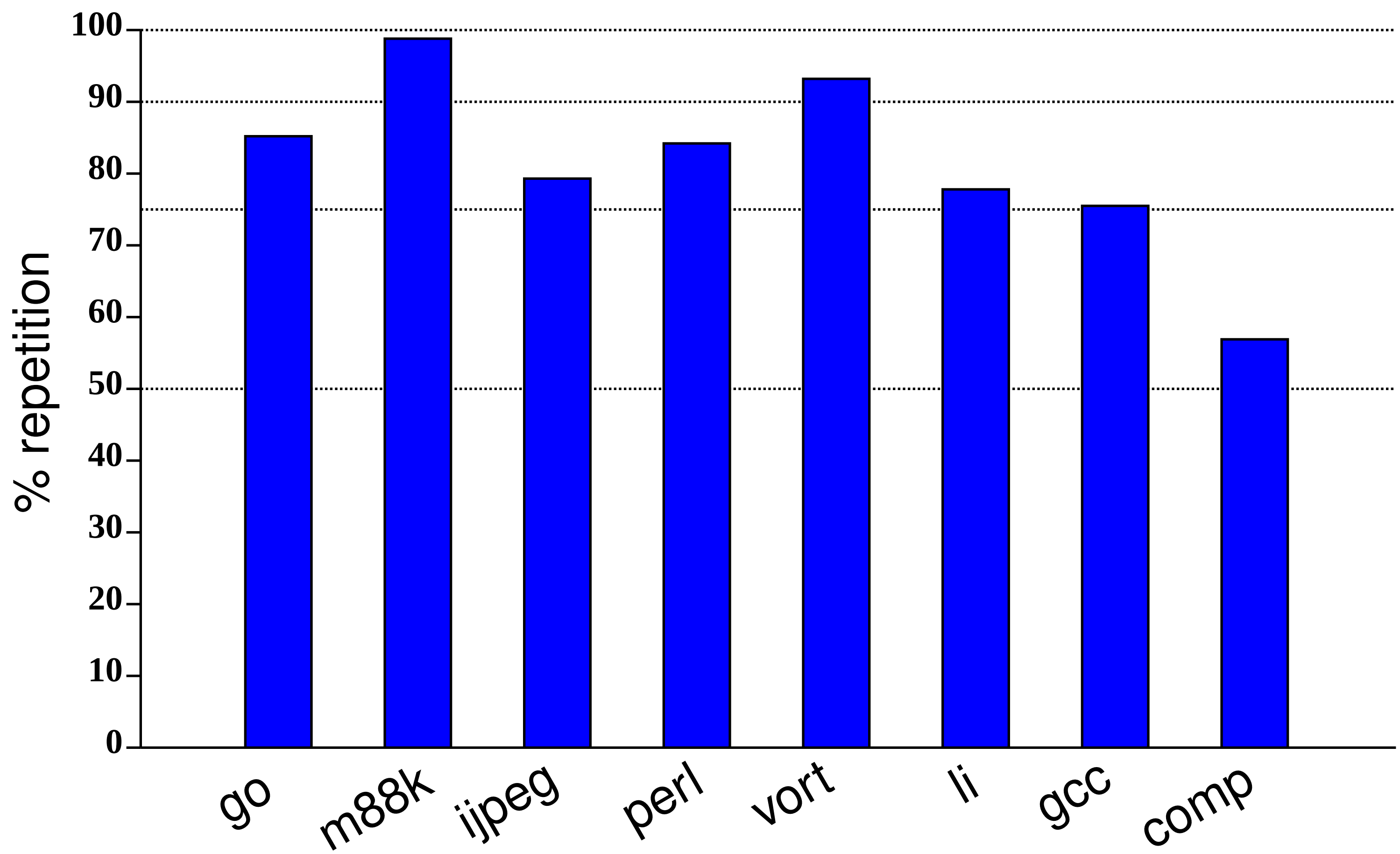
To better understand the causes of repetition

- Perform an ideal study on Specint '95 benchmarks
- Ask
  - What are its statistical characteristics?
  - What kinds of data are causing it?
  - What parts of programs are causing it?

Outline

- ~~Introduction~~
- Statistics on Repetition
- Analysis of Repetition
- Comments of software/hardware exploitability
- Summary

Amount of Repetition



More than 75% of Dynamic Instructions Repeated

Other Statistics

Executed static instructions

- most generate repeated instances
- 20% generate 90% of repetition
- but, repeated with many different inputs (100-1000)

(complete set of results in paper)

Outline

- ~~Introduction~~
- ~~Statistics on Repetition~~
- Analysis of Repetition
- Comments of software/hardware exploitability
- Summary

Causes

- **Due to input data,** e.g.,
    - in *gcc,* same keywords repeated
- **Due to the way we write programs**
    - many overhead instructions, e.g.,
        - loop controls
        - function prologue and epilogue

            → repeated even when inputs are different

Analyses of Repetition

Questions:

- What kinds of data cause repetition?
- What parts of programs get repeated?

We perform three levels of analysis:

- Global Analysis: - - - - - - - - - - - Whole programs
- Local Analysis: - - - - - - - - - - - - Within functions
- Function-level Analysis: - - - - - Function arguments

All analyses performed dynamically

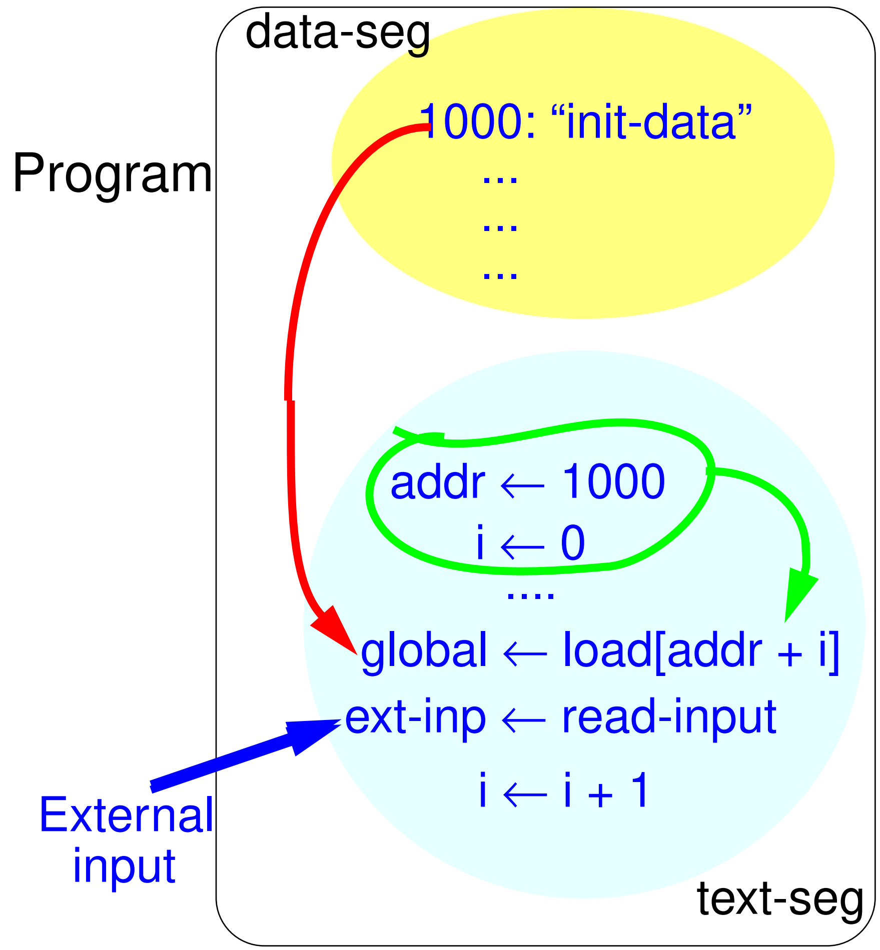
Global Analysis

Distribution of Repetition based on <u>kinds of input data</u>

<u>Sources of input data</u>

- External input
- Compile time initialized data
- Internal input

Hardwired in programs



Global Analysis: Methodology

- Tag data-item with its source category
- Propagate tag to dependent instructions

  ➔ trace inst. slices for each source category
- Determine repetition on inst. slices
- To chose a category at intersection points of slices

  ➔ rule External > Initialized > Internal

Global Analysis: Results

Gcc	internal input	initialized data	external input
Overall	60	25	15
Repeated	65	29	6
Propensity	82	88	30

(other results in paper)

Repetition

- All categories amenable to repetition
- > 70% fall on slices originating from hardwired values
- < 20% fall on slices originating from external inputs

Repetition occurs due to the way programs are expressed

Local Analysis

Distribution of Repetition within functions

Categories:

Source of input data

- Global + Heap
- Argument
- Internal
- Return values from functions

Type of work

- Prologue
- Epilogue
- Global Address Calc.
- Returns

Methodology same as that in Global Analysis

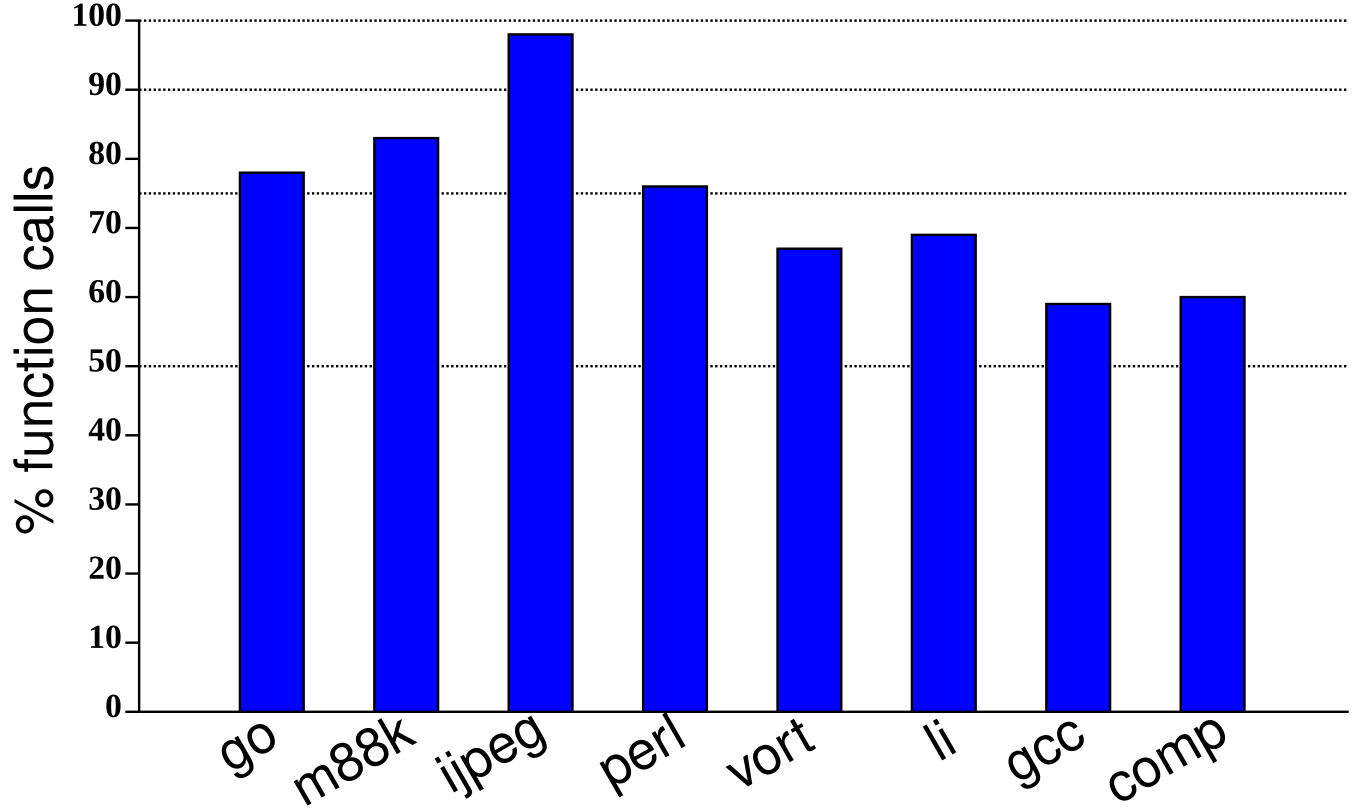
Local Analysis: Results

For <u>GO</u>	Overall %	Repeated %	Propensity %
Global + Heap	54	48	76
Argument	10	10	86
Internals	10	11	99
Return Values	2	2	99
Prologue + Epilogue	6	7	98
Global Address Calc.	16	19	99
Returns	1	1	99

- Most repetition on Global + Heap
- (Pro+Epi)logue + Global Address significant for some programs

Function-Level Analysis

Function calls with ALL arguments repeated



Most (> 99%) have external i/o or read/write global variables

Software Exploitability

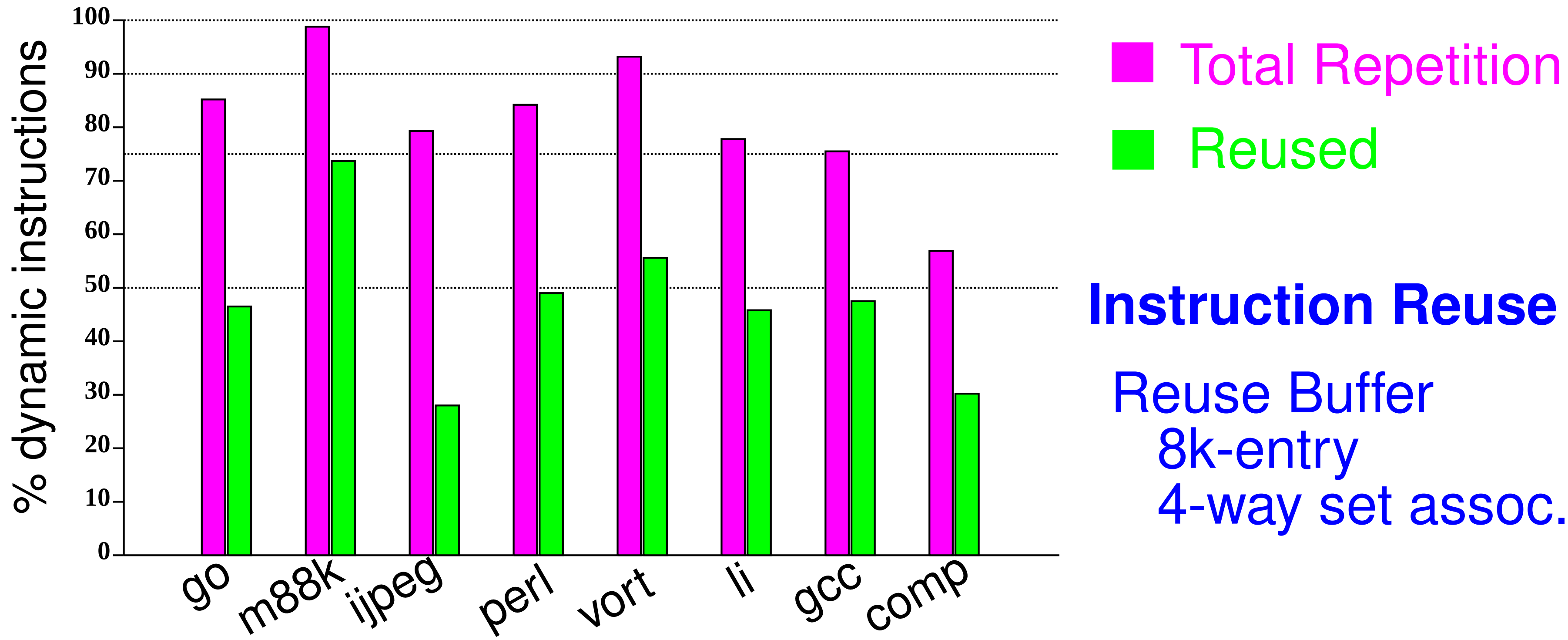
Most repetition due to hardwired values

➔ can it be eliminated in software?

Issues

- Due to control-flow and function boundaries
  - ➔ repetition not statically obvious
  - ➔ requires dynamic information
- Inst. repeat with multiple values
  - ➔ specializing for only few may not be sufficient
- In some cases, it may not be prudent
  - ➔ e.g., loop control, function (pro+epi)logue

Hardware Exploitability



Instruction Reuse

Reuse Buffer
- 8k-entry
- 4-way set assoc.

- Still room for improvement
- Manage Reuse Buffer efficiently

Summary

Instruction Repetition

- Inst. execute again with same i/p and produce same o/p
- **Pervasive** (more than 75% dynamic inst repeated)
- More an attribute of program and less of input data
  - Not because of bad compiler
- Detail analysis and statistical characteristics in paper

Analysis useful for better exploitation, like

- develop new mechanisms
- improve existing mechanisms