

Execution-based Scheduling for VLIW Architectures

Kemal Ebcioglu

Erik R. Altman (*Presenter*)

Sumedh Sathaye

Michael Gschwind

September 2, 1999

Outline

- Overview
- What's new?
- Results
- Conclusions

Overview

- Based on **DAISY**
- **DAISY** = **D**ynamically **A**rchitected **I**nstruction **S**et from **Y**orktown
- **DAISY**: *PowerPC* => *VLIW* via Dynamic Translation

DAISY Strengths

- Strengths of **DAISY** Approach:
 - Compatibility between VLIW generations since distribution format is PowerPC code
 - Architecture just a layer of software
 - Can adapt to changes in program behavior

DAISY Characteristics

- **DAISY** Translation done by Software unlike *DIF* or *Superscalars*
- Architecture Emulated, not OS, unlike *FX!32*, *WABI*, or *VirtualPC*
- Wide VLIW Target up to 16 units

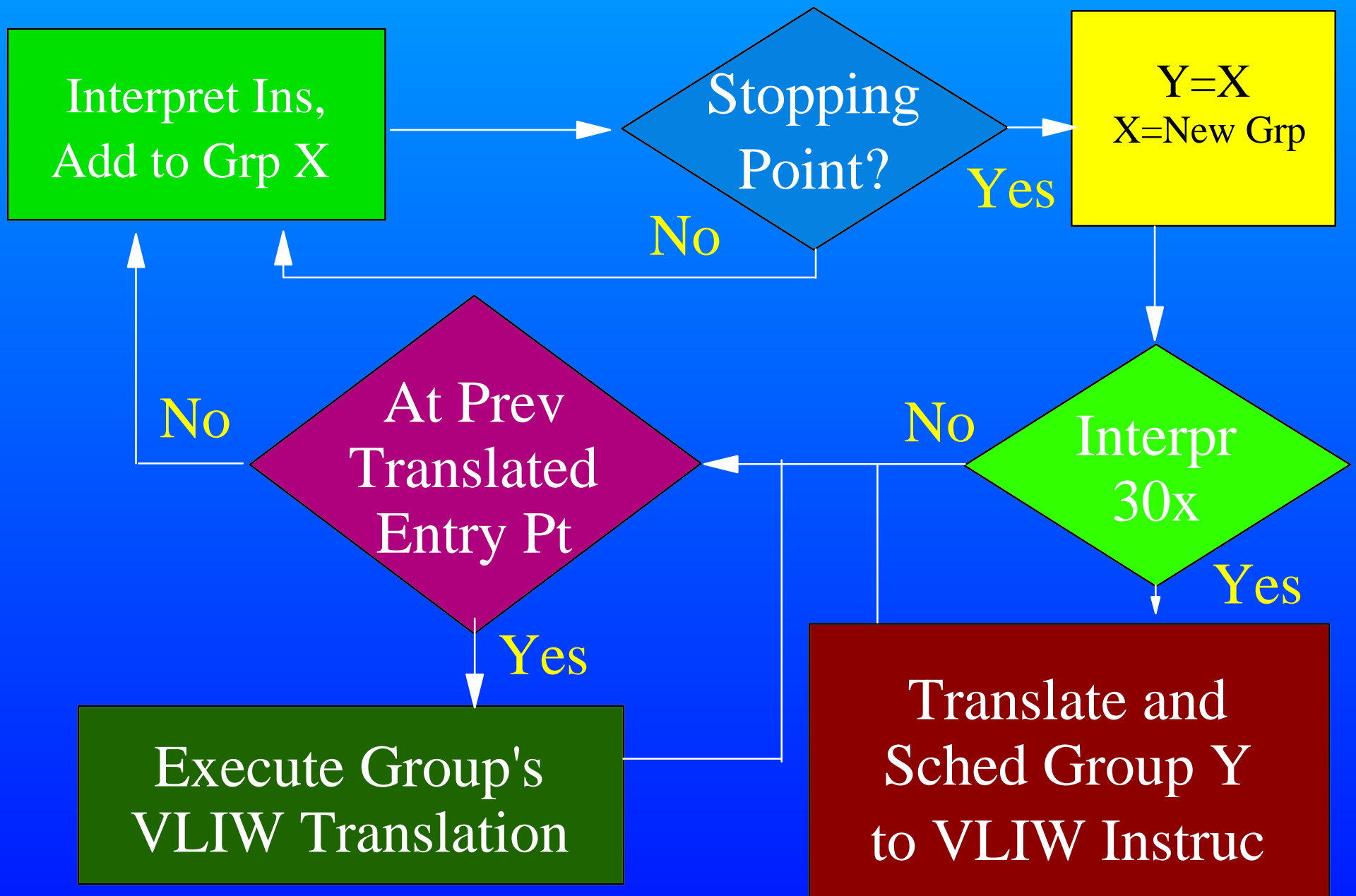
DAISY Problems

- Problems in earlier **DAISY** work:
 - ▶ Serialized on page crossings
 - ▶ Serialized on register branches
 - ▶ Translated all reachable code on a page:
 - Translated unexecuted code paths
 - Arbitrary serializations: Avoid code bloat

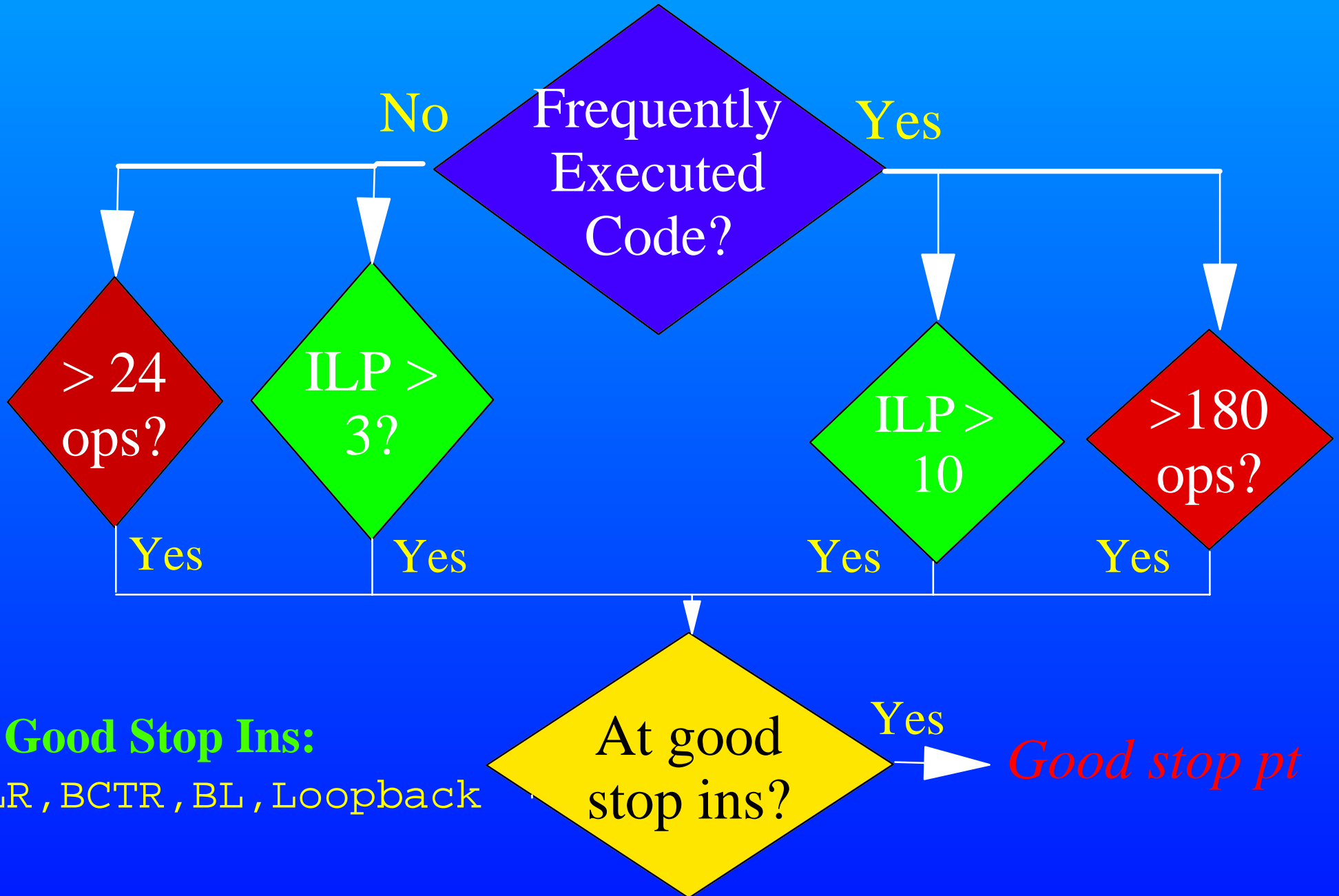
What is new here?

- Method of forming groups so as to have:
 - ▶ Long Window Size
 - ▶ Code spanning page boundaries
 - ▶ Code spanning register branches
 - ▶ Small code windows initially, bigger windows later on frequently executed code
- Simple hardware to identify code where most execution time spent

Group Formation Algorithm



Stopping Points



Extending Groups

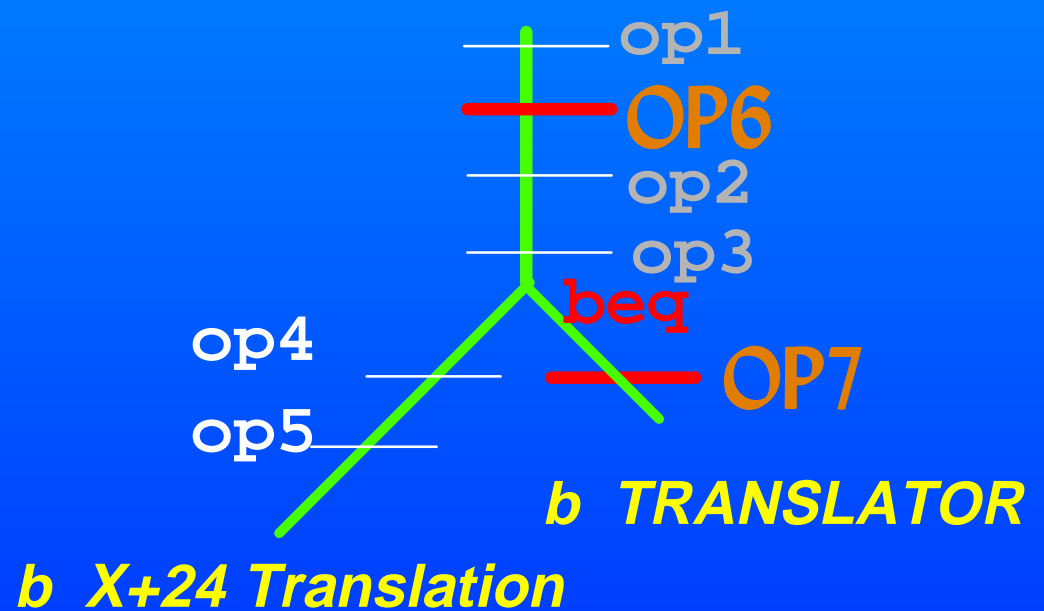
PowerPC

X	op1
X+4	op2
X+8	op3
X+12	beq X+64
X+16	op4
X+20	op5

X+64 OP6

X+68 OP7

VLIW Group



Group Formation Notes

- Translation starts at boot point of base architecture, e.g. FFF00100 on PowerPC
- Groups have no join points => Trees
 - ▶ Group former not build CFG
 - ▶ Looks only at curr instruc and path it is on
 - ▶ Ins past join duplicated on multiple paths

Tree Benefits

- At most one reaching definition
- Register allocation simplified
- Exit point during execution uniquely specifies (long) path through the group

Indirect Branches

V50:

```
cmp cr15,PPC_LR,1000
```

```
cmp cr14,PPC_LR,2000
```

```
cmp cr13,PPC_LR,3000
```

```
b V51
```

```
V51: beq cr15,V1000
```

```
beq cr14,V2000
```

```
beq cr13,V3000
```

```
b V52
```

```
V52: b Group of PPC_LR
```

PPC: BLR

V1000:

*# Translated code from
1000*

V2000:

*# Translated code from
2000*

V3000:

*# Translated code from
3000*

Knowing when to Reoptimize

- **Add profiling code at group exits**
 - ▶ **Code Explosion**
 - ▶ **Pollutes Data Cache**
- **Timer Interrupts**
 - ▶ **Coarse Granularity / Less Accurate**
- **Hardware Array of Cached Counters**
 - ▶ **Indexed by exit point of group**
 - ▶ **Auto-incremented at group exit**
 - ▶ **8K Entries, 8-way associative**

Translator Optimizations

- **Schedule with data & control speculation**
- **Copy propagation**
- **Combining**
- **Unification**
- **Limited Dead Code Elimination**
- **Limited Memory Alias Analysis**
- **Load-Store Telescoping**

Results Configuration (1)

- **Benchmarks**
 - ▶ **SPECint95**
 - ▶ **TPC-C**
- **SPECint95 Sampling Method**
 - ▶ **Uniformly Sampled PowerPC Traces**
 - ▶ **2 million instructions per sample**
 - ▶ **50 samples per benchmark**
- **TPC-C Sampling Method**
 - ▶ **Special-purpose hardware**
 - ▶ **170 million instruction trace**

Cache Parameters

Cache	Size	Linesize	Assoc	Latency
<i>L1-I</i>	64K	1K	8	1
<i>L2-I</i>	1M	2K	8	3
<i>L1-D</i>	32K	256	4	2
<i>L2-D</i>	512K	256	8	4
<i>L3</i>	32M	256	8	42
<i>Memory</i>				150

TLB Parameters

TLB	Entries	Assoc	Latency
<i>DTLB1</i>	128	2	2
<i>DTLB2</i>	1K	8	4
<i>DTLB3</i>	8K	8	10
<i>Page Tbl</i>			90

VLIW Machine Size

- 16 Issue
- 16 ALU's
- 8 Load/Store Units
- 3 Branches per cycle
- 64 integer registers
- Clustered
 - ▶ 4 Clusters of 4 ALU's
 - ▶ 2 Load/Store Units per Cluster
 - ▶ Immediate bypass within Cluster
 - ▶ Extra cycle outside Cluster

Exec Time of Translated Code

- Ignoring Cache Effects, **Cycles for Each Path Through Group** = Number of VLIW Instructions
- Total Cycles Spent in Group:

$$\sum_{\text{All Group Paths } P} (\# \text{ of VLIW Ins on } P) \times (\# \text{ Times } P \text{ Executes})$$

All Group Paths P

ICache Cycles

- **Layout VLIW code for all groups**
- **Index VLIW code by group exit points**
- **Go thru exit points in execution order:**
 - ▶ **Iterate through all VLIW Instruction Addresses corresponding to each exit**
 - ▶ **Feed Addresses to Multilevel ICache Simul**
 - ▶ **Simulator includes history-based prefetch**

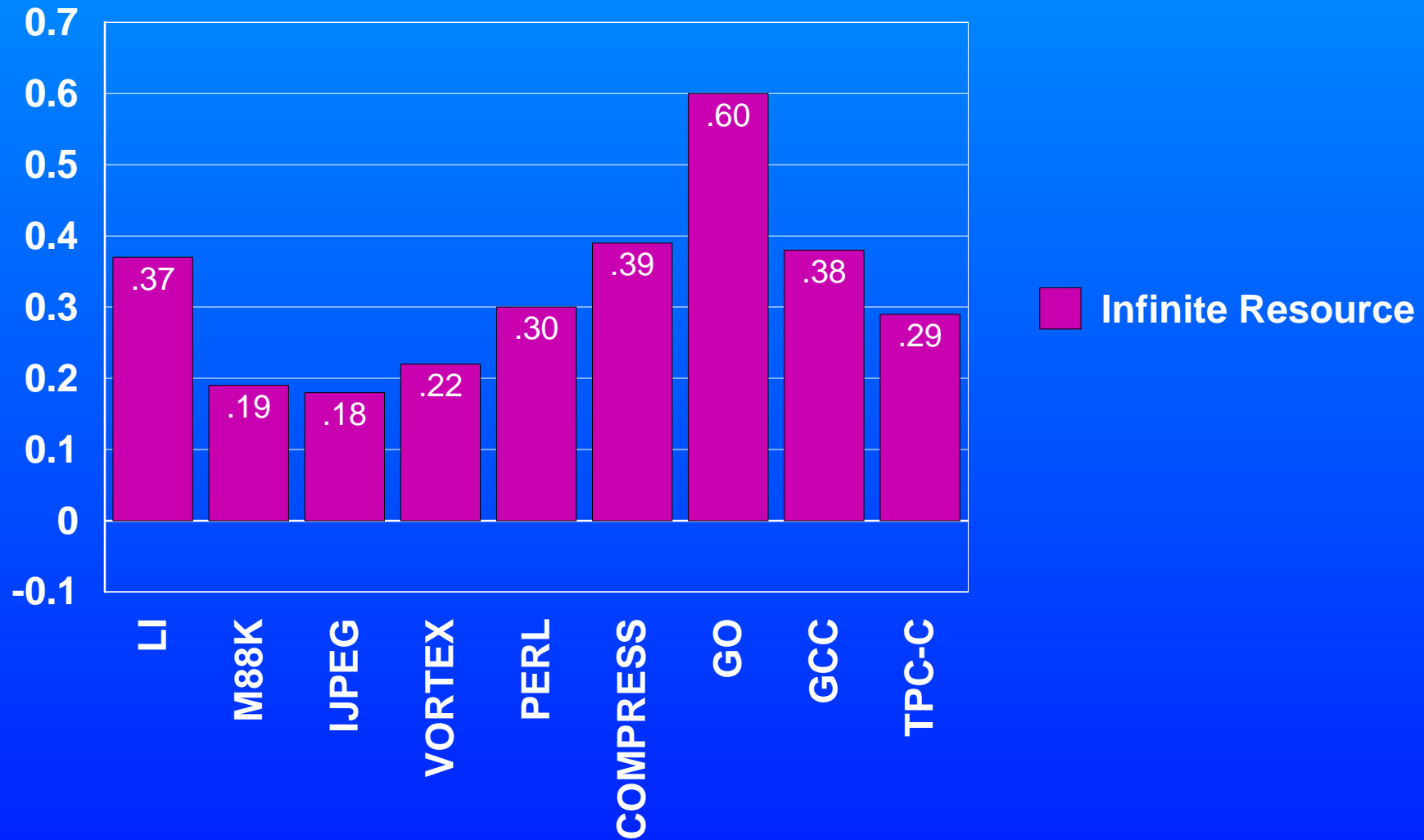
DCache Cycles

- Modeling *Speculative Loads* Difficult in Trace-Based Environment
- Addresses for Specul Ops not on actual Execution Path are Unknown
- => Use LD/ST addresses from PowerPC trace as input to DCache/DTLB Simul
- Mult DCache/DTLB Stall Cycles by 1.7
- 1.7 = Increase in Execution-based DAISY
- DCache *not* lockup-free

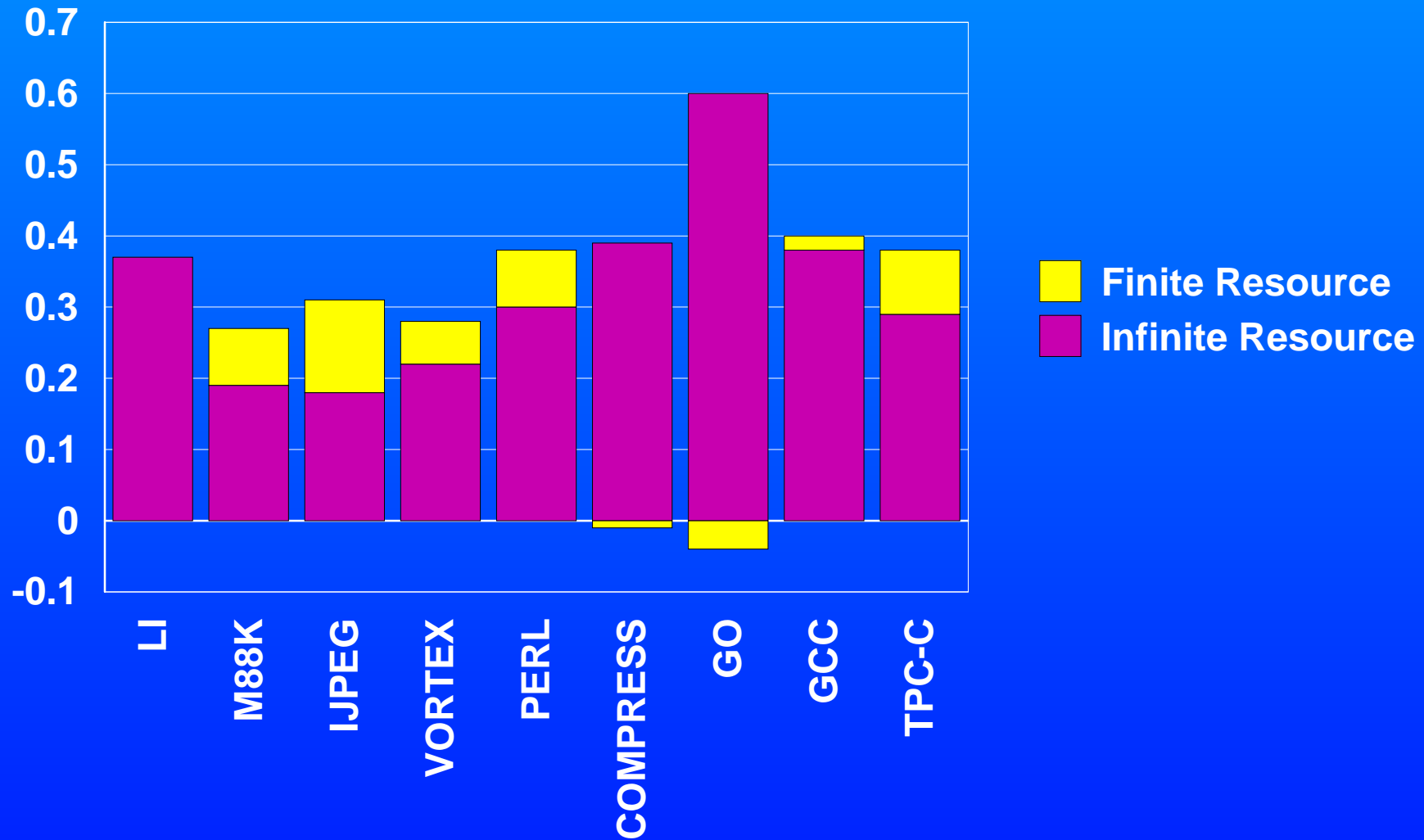
Overall CPI

- **Total Cycles for VLIW Execution:**
 - ▶ Infinite Cache Cycles +
 - ▶ ICache Cycles +
 - ▶ DCache Cycles +
 - ▶ DTLB Cycles +
 - ▶ Translation Overhead
- **CPI = Total VLIW Cycles / Orig PPC Ins**
- **Translation Overhead Negligible --
Details in Paper**

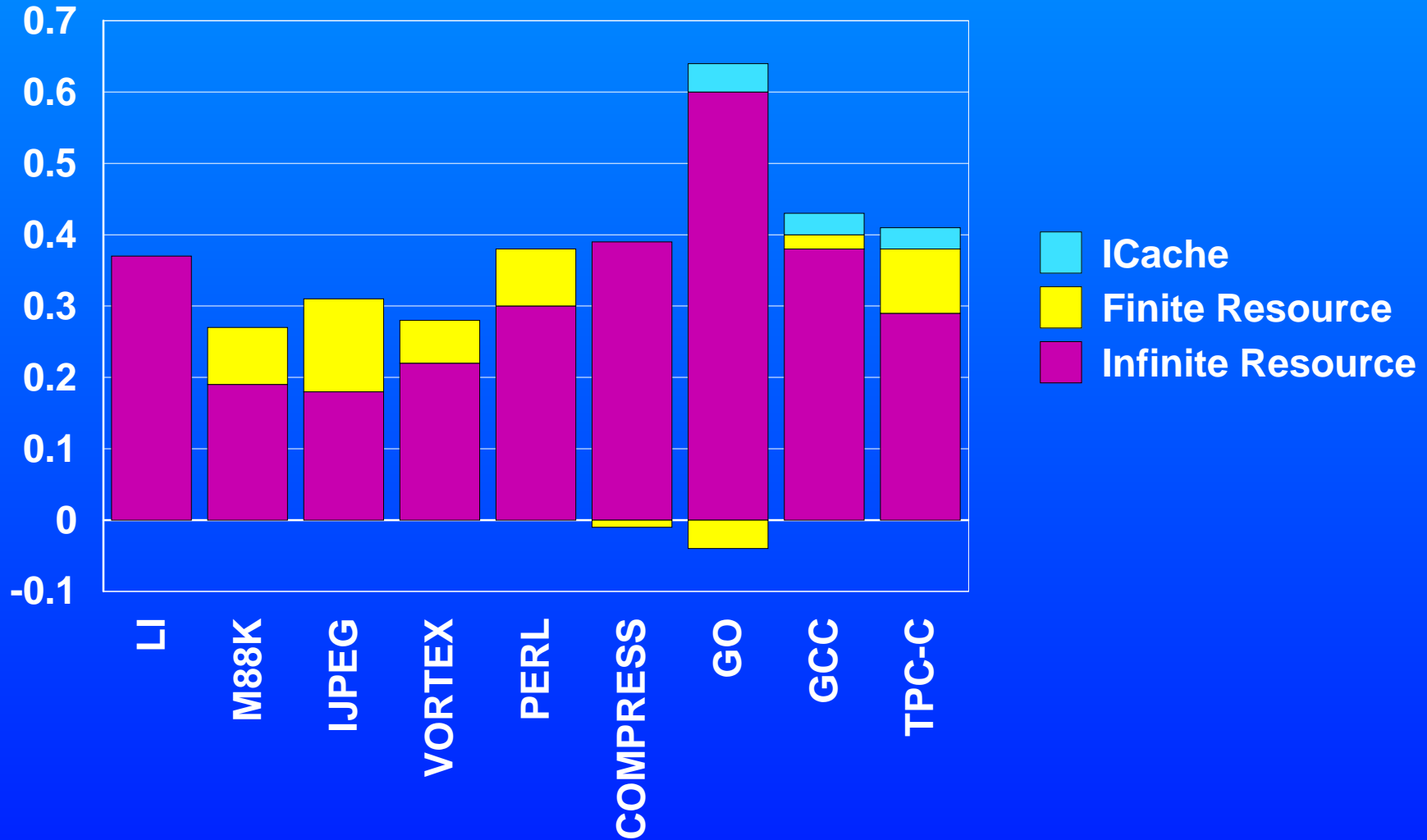
Infinite Resource CPI



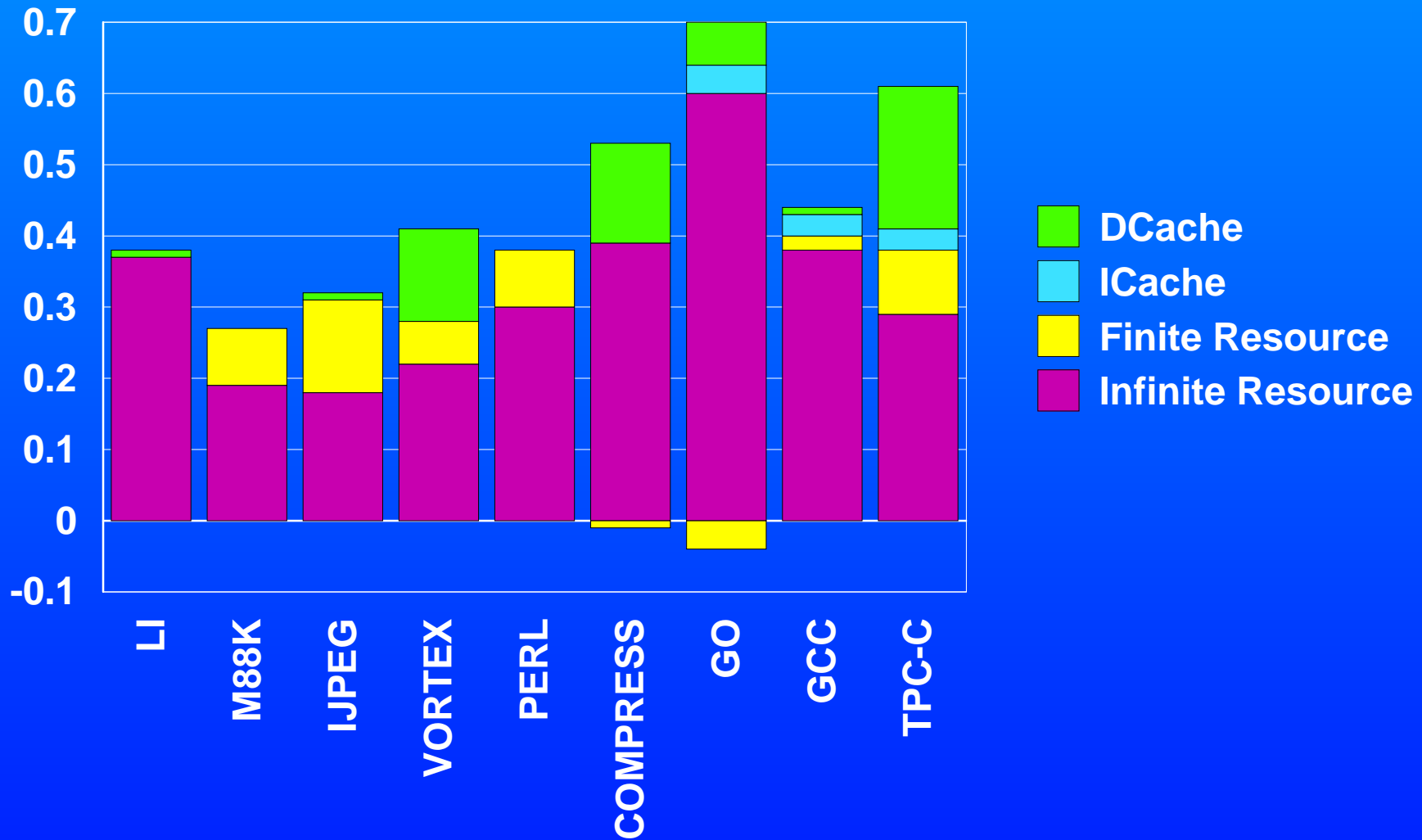
Finite Resource CPI



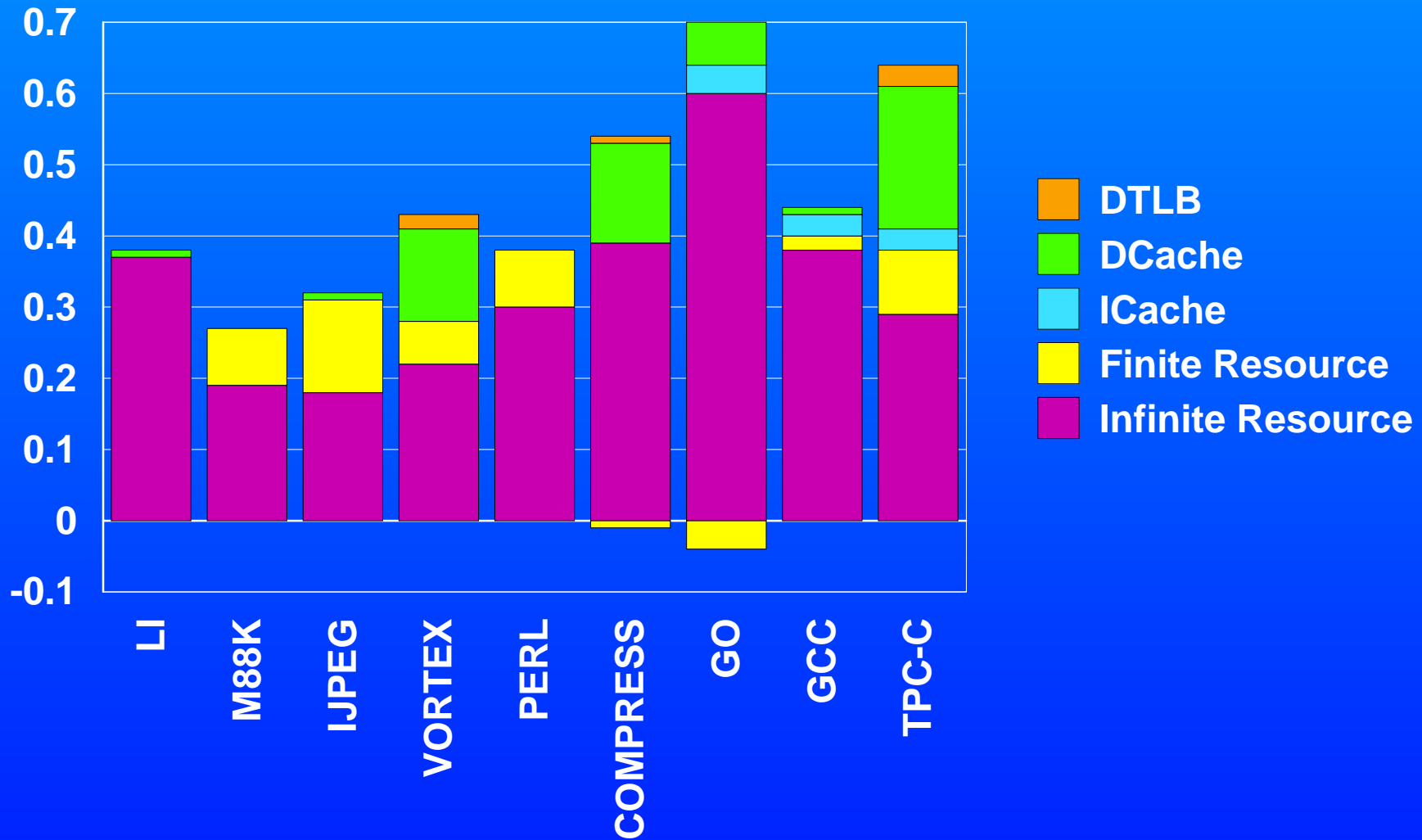
ICache CPI



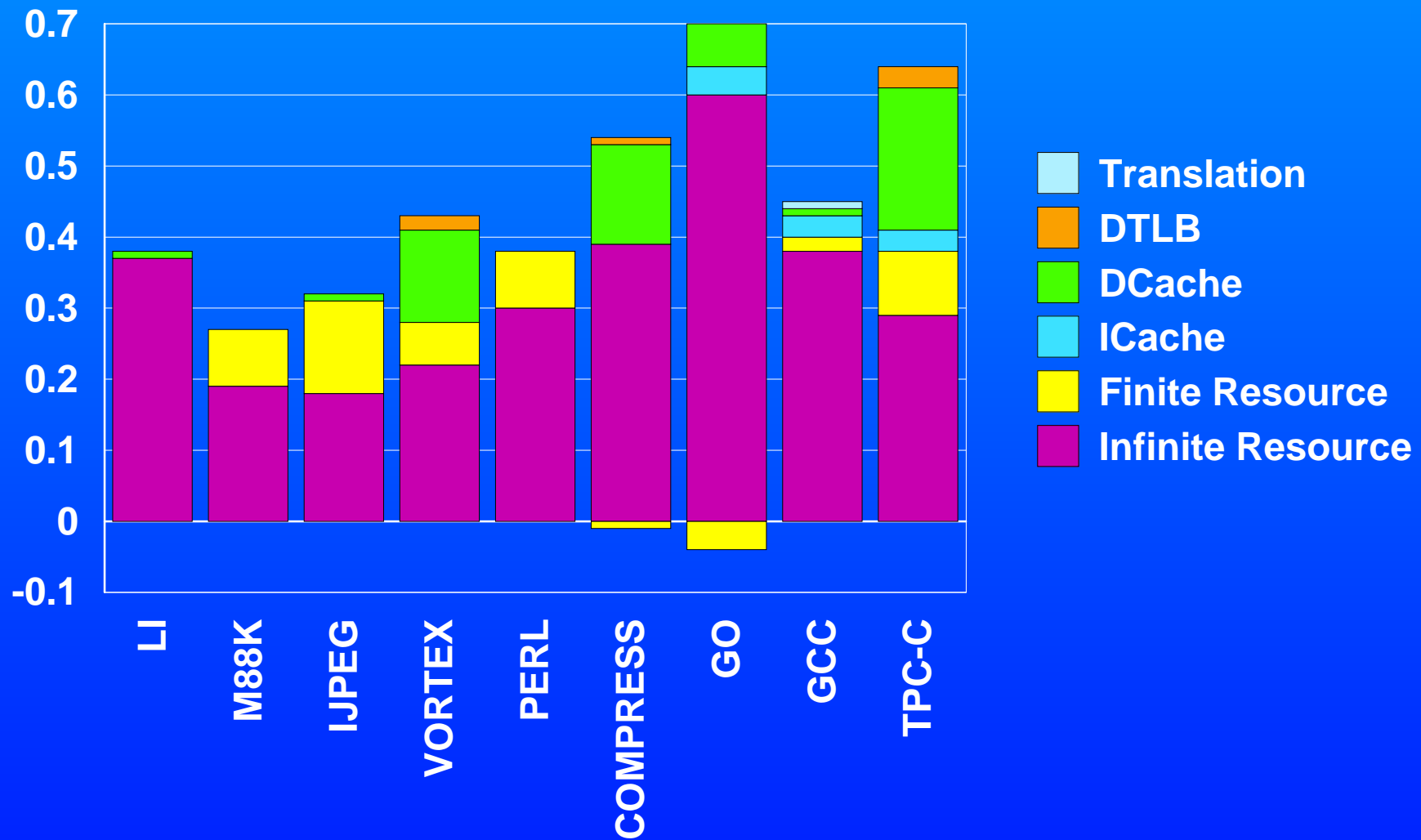
DCache CPI



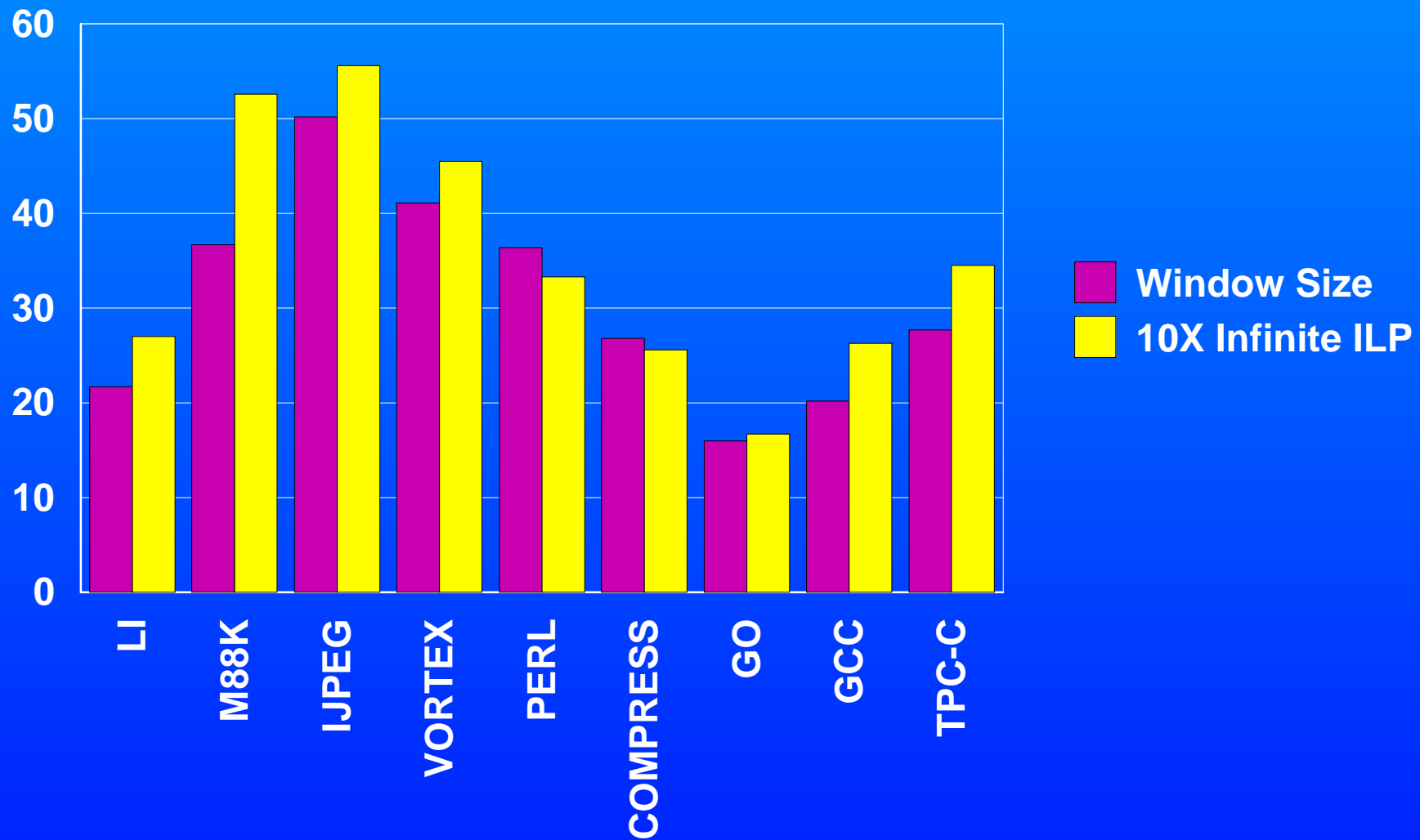
DTLB CPI



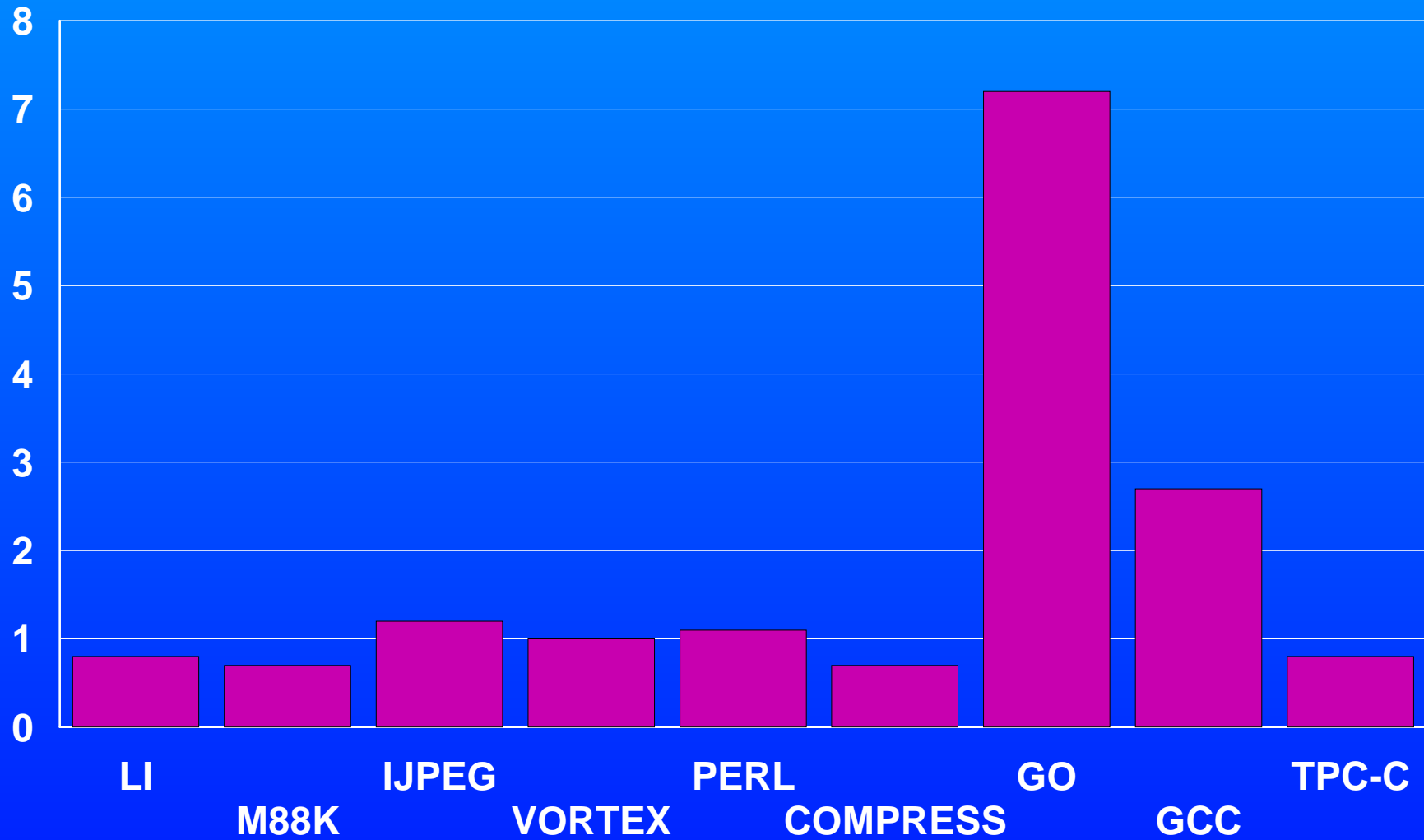
Translation and Overall CPI



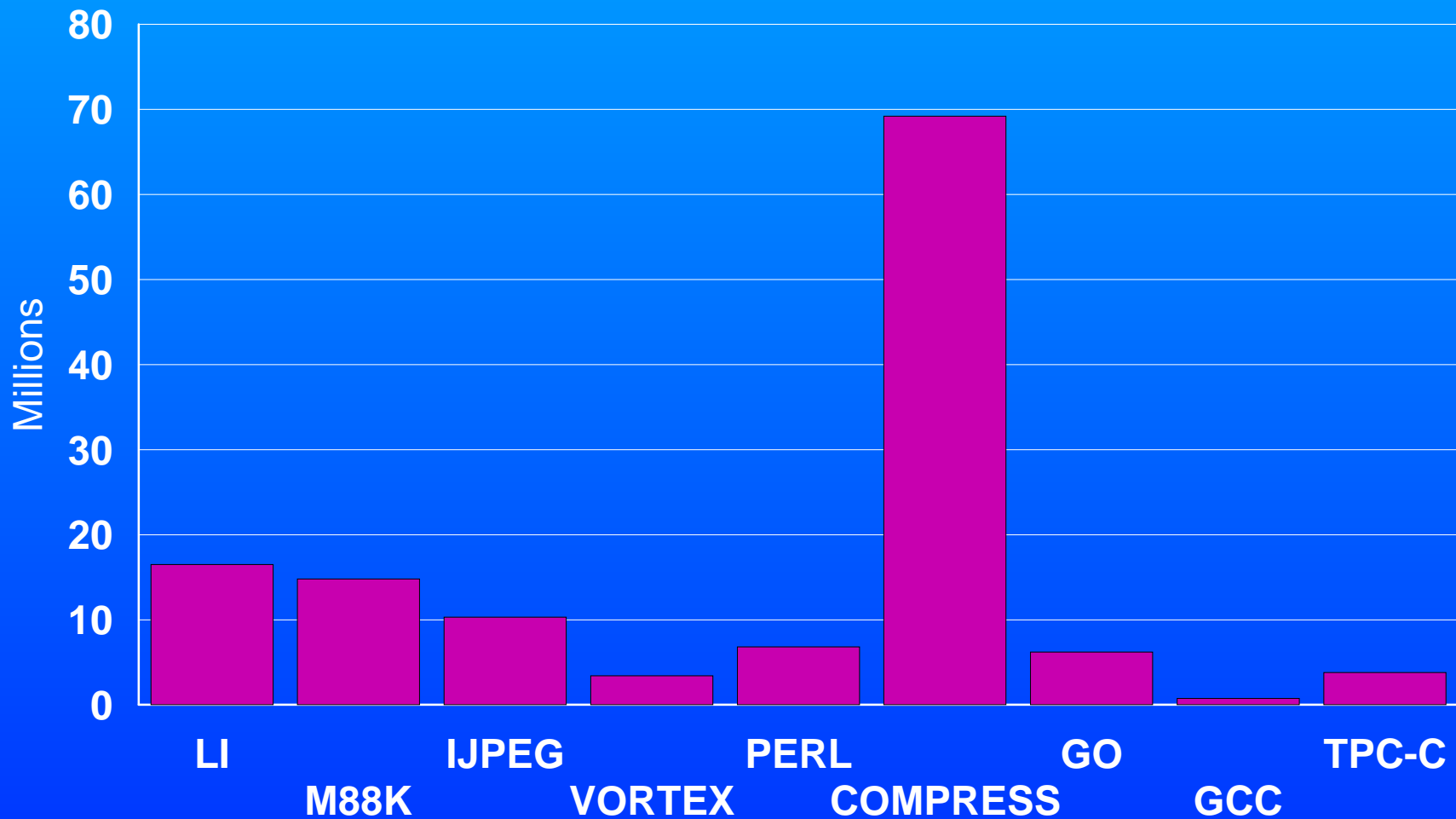
Window Size and ILP



Code Explosion



Instruction Reuse



$$\text{Reuse Rate} = \frac{\text{\# of Dynamic Ins in Trace}}{\text{\# of Unique Ins Addresses in Trace}}$$

Conclusions

- **DAISY** => VLIW Binary Compat w/**PPC**
- Architecture is a layer of software
- Extended Previous **DAISY** Approach:
 - ▶ Cross *Page Boundaries* in Scheduling
 - ▶ Cross *Register Branches*
 - ▶ Two-phase adaptive translation
 - *Good CPI* on frequently executed code
 - *Limit Code Explosion*
- www.research.ibm.com/daisy