Exploiting Dead Value Information

To appear in MICRO-30

Milo M. Martin, Amir Roth, and Charles N. Fischer

1997 University of Wisconsin-Madison Architecture Affiliates Meeting
October 15, 1997

This document contains the text and graphs from a poster presentation. The pseudo code examples have not been included but are included in the MICRO-30 paper.

Paper available at: `http://www.cs.wisc.edu/~milo/micro30.ps`

Motivation

- **The hardware does not know when a register value is dead**
- **A dead value is no longer needed for continued correct execution**
- **However, the compiler has information about future register usage**

*** An opportunity for compiler/hardware interaction ***

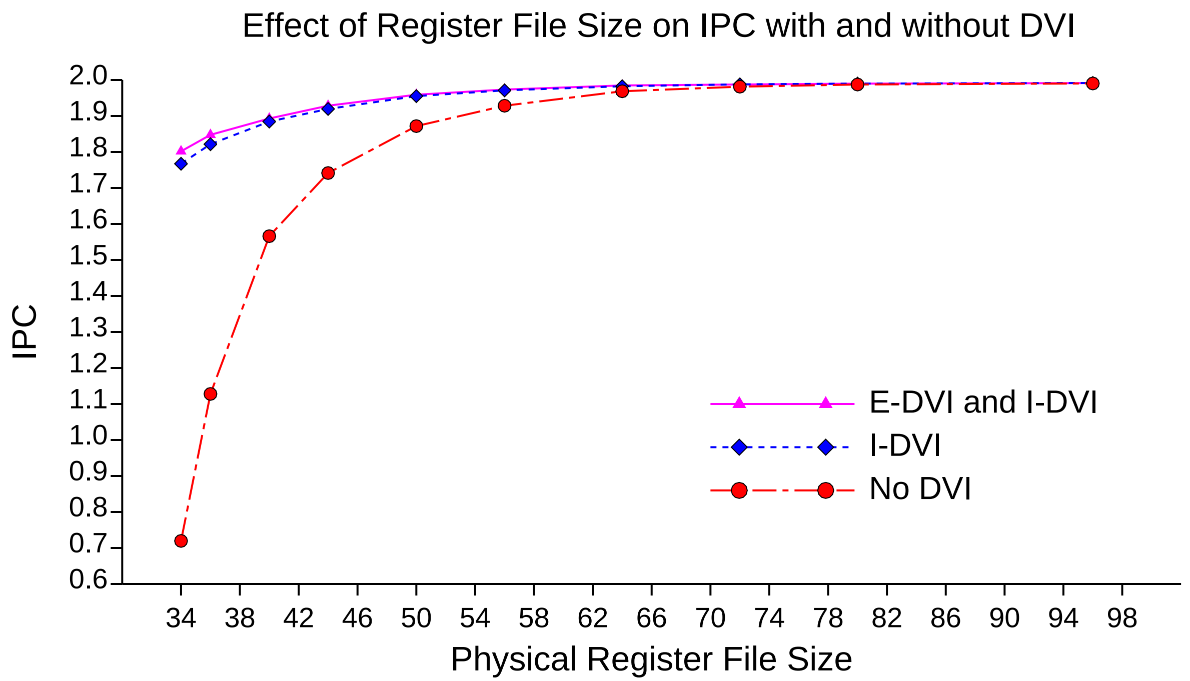
- **Communicating this Dead Value Information (DVI) to the hardware enables optimizations**

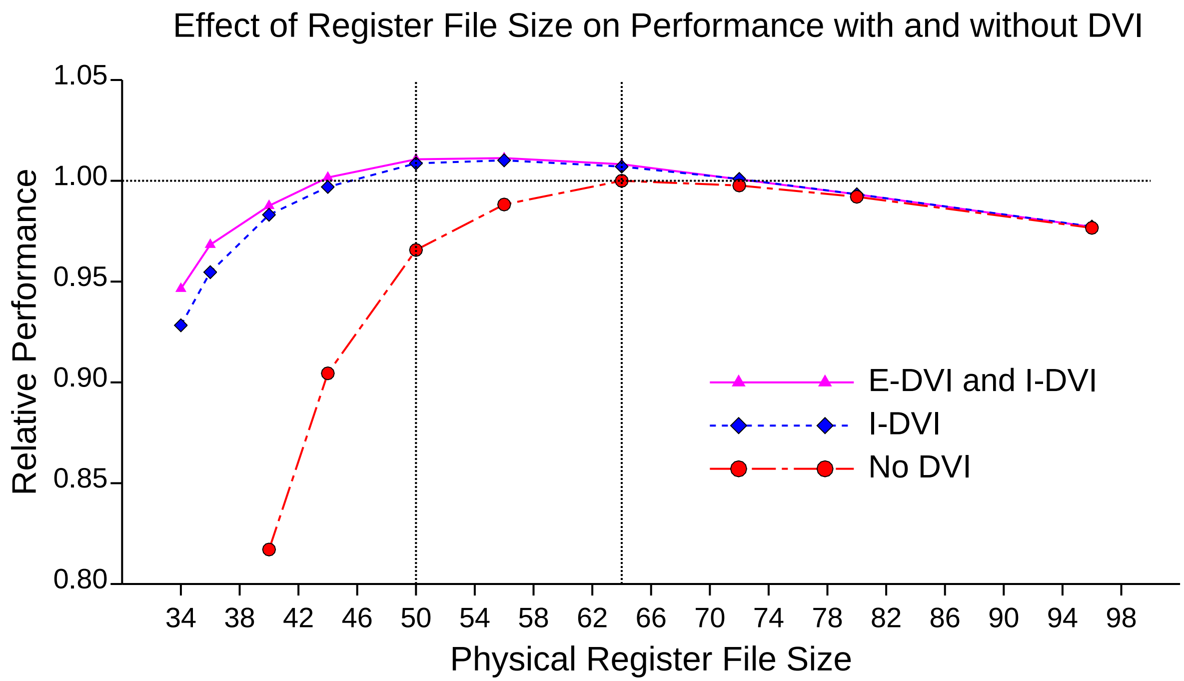
Providing DVI

- **The information can be provided in two ways:**
  - **- Explicitly by new instructions**
  - **- Implicitly by procedure calling conventions**
- **A bit table is used to track the dead/live status of each register**
- **New *live-load* and *live-store* instructions added for saves and restores**

Register File Size Reduction

- Registers holding dead values can be freed and used for register renaming
- This reduces the size requirements of the physical register file
- Allows for a higher IPC for a given register size:
- If the register file is constraining the clock cycle time, reducing the register file size can lead to overall performance gains
- Here we use a register file timing model to calculate overall performance (IPC x clock rate)

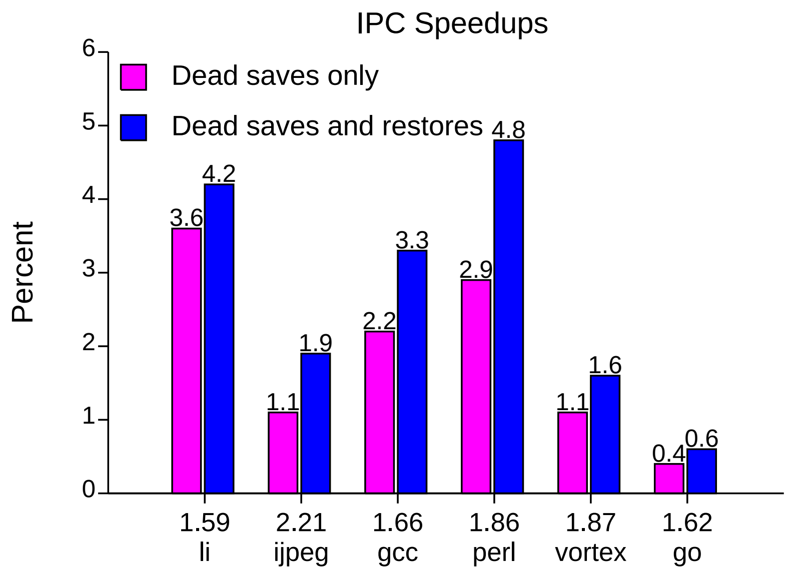

Effect of Register File Size on IPC with and without DVI
IPC
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
34
38
42
46
50
54
58
62
66
70
74
78
82
86
90
94
98
Physical Register File Size
E-DVI and I-DVI
I-DVI
No DVI

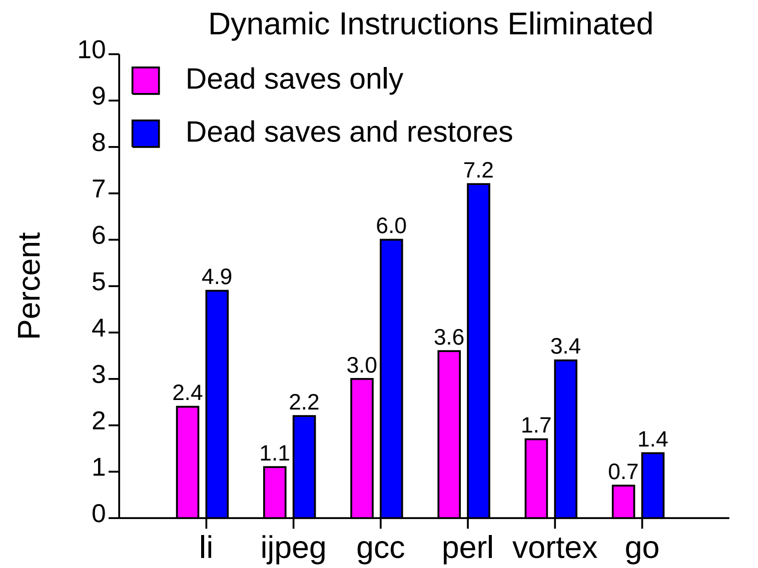


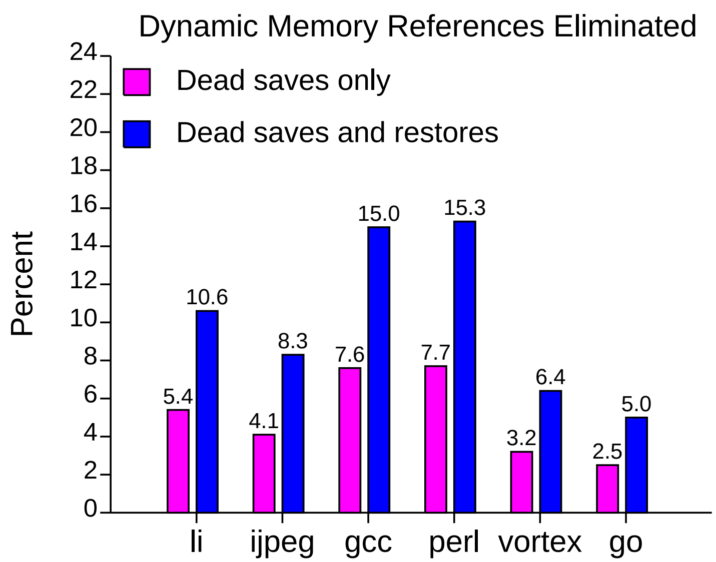
Effect of Register File Size on Performance with and without DVI
Relative Performance
1.05
1.00
0.95
0.90
0.85
0.80
34
38
42
46
50
54
58
62
66
70
74
78
82
86
90
94
98
Physical Register File Size
E-DVI and I-DVI
I-DVI
No DVI

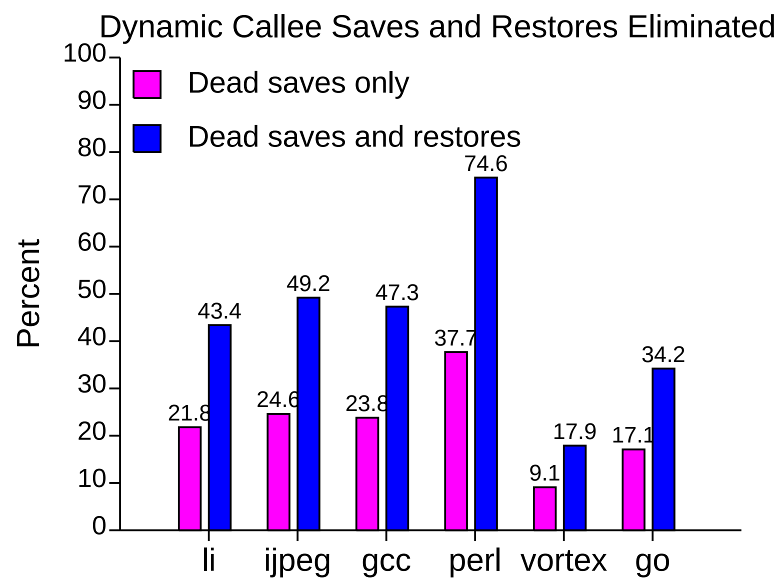
Procedure Call Save/Restore Elimination

- **Dead registers do not need to be saved on procedure calls or restored on procedure returns**
- **Problem: The compiler does not know what registers are live into a procedure**
- **Solution: Use dead value information to track liveness dynamically**


IPC Speedups
Dead saves only
Dead saves and restores
Percent
0
1
2
3
4
5
6
3.6
4.2
1.1
1.9
2.2
3.3
2.9
4.8
1.1
1.6
0.4
0.6
1.59
li
2.21
ijpeg
1.66
gcc
1.86
perl
1.87
vortex
1.62
go

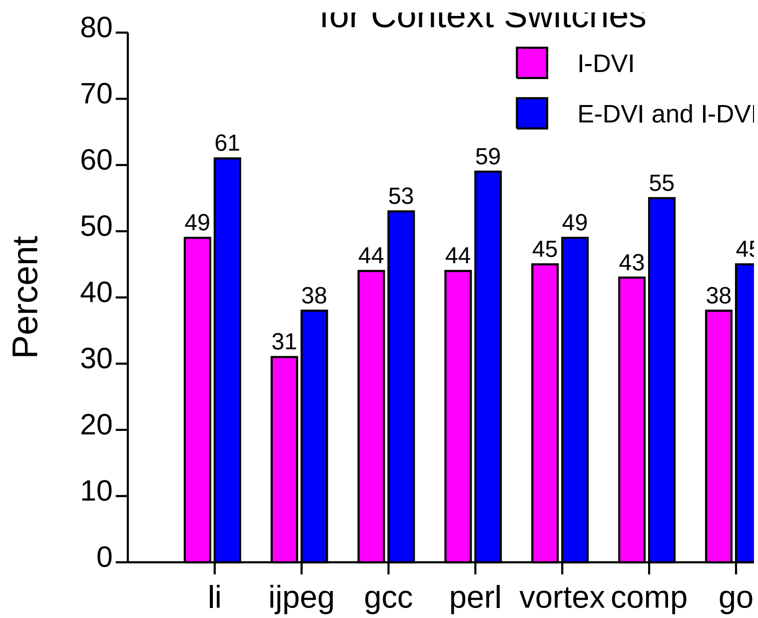

Dynamic Instructions Eliminated
Dead saves only
Dead saves and restores
Percent
0
1
2
3
4
5
6
7
8
9
10
2.4
4.9
1.1
2.2
3.0
6.0
3.6
7.2
1.7
3.4
0.7
1.4
li
ijpeg
gcc
perl
vortex
go


Dynamic Memory References Eliminated
Dead saves only
Dead saves and restores
Percent
0
2
4
6
8
10
12
14
16
18
20
22
24
5.4
10.6
4.1
8.3
7.6
15.0
7.7
15.3
3.2
6.4
2.5
5.0
li
ijpeg
gcc
perl
vortex
go


Dynamic Callee Saves and Restores Eliminated
Dead saves only
Dead saves and restores
Percent
0
10
20
30
40
50
60
70
80
90
100
21.8
43.4
24.6
49.2
23.8
47.3
37.7
74.6
9.1
17.9
17.1
34.2
li
ijpeg
gcc
perl
vortex
go

Context Switch Save/Restore Elimination

- **Dead values do not need to be saved or restored on context switches or thread switches**
- **Register save and restore time can be significant for thread switches**



Results

- **Reduces physical register size by 22%**
- **Removes 51% of context switch save and restores**
- **Eliminates 46% of callee saves which results in IPC improvements of nearly 5%**

Conclusions

- **Our use of DVI requires:**
  - **+ Only a few new instructions**
  - **+ Well known compiler techniques**
  - **+ Limited additional hardware structures**

- **Our optimizations becomes more important as C++ and Java become widespread**
- **Open question: What implications do these results have for future ISAs and calling conventions**